Topic 1
CS314 Course Introduction

Chapman: I didn't expect a kind of Spanish Inquisition.

Cardinal Ximinez: NOBODY expects the Spanish Inquisition! Our chief weapon is surprise...surprise and fear...fear and surprise.... Our two weapons are fear and surprise...and ruthless efficiency.... Our **three** weapons are fear, surprise, and ruthless efficiency...and an almost fanatical devotion to the Pope.... Our **four**...no... **Amongst** our weapons.... Amongst our weaponry...are such diverse elements as fear, surprise....

In class: please close laptops and put away mobile devices.

Mike Scott, Gates 6.304
scottm@cs.utexas.edu
www.cs.utexas.edu/~scottm/cs314/

Who Am I?

- Professor of Instruction (lecturer) in CS department since 2000
- Undergrad Stanford, MSCS RPI
- US Navy for 8 years, submarines
- 2 years Round Rock High School prior to coming to UT

Purpose of these Slides

- Discuss
  - course content
  - procedures
  - tools
- For your TO DO list:
  - complete items on the startup page

www.cs.utexas.edu/~scottm/cs314/handouts/startup.htm

Course Goals

- Analyze algorithms and code for efficiency
- Be able to create and use canonical data structures: lists (array and linked), stacks, queues, trees, binary search trees, balanced binary search trees, maps, sets, graphs, hash tables, heaps, tries
- Know and use the following programming tools and techniques: object oriented programming (encapsulation, inheritance, polymorphism), Java Interfaces, iterators, sorting, searching, recursion, dynamic programming, functional programming

Course Goals

- After CS314 you can design and implement medium size programs (several 100's of lines of code split between multiple classes) to solve interesting problems
- Recall, the three core areas of the UTCS undergrad degree:
  Programming, Theory, Systems
- After this class your instructors shall expect you can write complex programs given a specification or problem statement.
  - You have to design the algorithm in many cases.

Prerequisites

- Formal: CS312 with a grade of C- or higher
- Informal: Ability to design and implement programs in Java using the following:

•variables and data types

•expressions, order of operations

•Conditionals (if statements)

  •including boolean logic and boolean expressions

•iteration (loops)

•Methods (functions, procedures)

•Parameters

•structures or records or objects

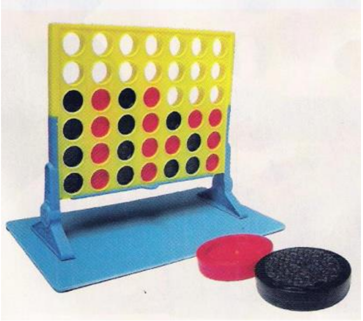
•arrays (vectors, lists)

•top down design (breaking big rocks into little rocks)

  •algorithm and data design

  •create and implement program of at least 200 - 300 loc

•could you write a program to let two people play connect 4?



CS314 Topics

1. Introduction
2. Algorithm Analysis
3. Encapsulation
4. Inheritance
5. Polymorphism
6. Generics
7. Interfaces
8. Iterators
9. Abstract Classes
10. Maps, Sets
11. Linked Lists
12. Recursion
13. Recursive Backtracking
14. Searching, Simple Sorts
15. Stacks
16. Queues
17. Faster Sorting
18. Trees
19. Binary Search Trees
20. Red-Black Trees
21. Huffman Code Trees
22. Graphs
23. Hash tables
24. Tries
25. Heaps
26. Dynamic Programming
27. Functional Programming

Data Structures

- simple definition:
  - a variable that stores other variables, a container
- We will learn a toolbox full of data structures …
- … and how to build them …
- … and how to use new ones.

Clicker Question 1

- Which of the following is a data structure?

A. a method

B. a try / catch block

C. a double

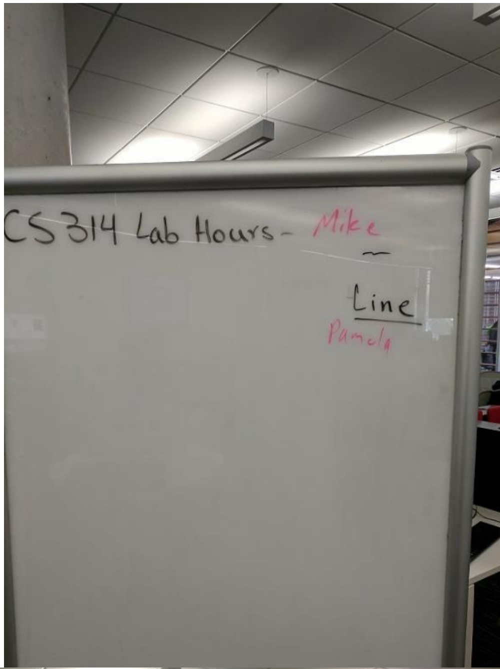
D. an array

E. more than one of A - D

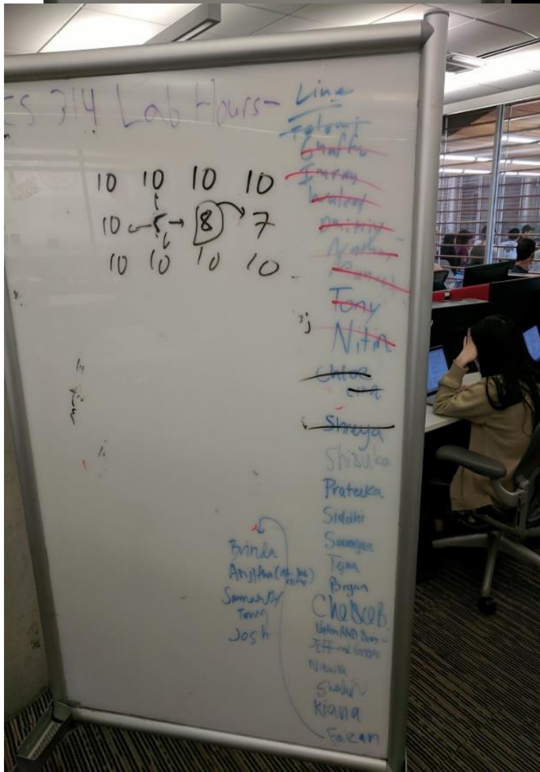
Resources

- Class web site – most course material
- Class discussion group – Ed
- Canvas -> Grades, Program Submissions (GradeScope), Access Zoom Links, Recorded Lectures, Help Videos

Monday ->

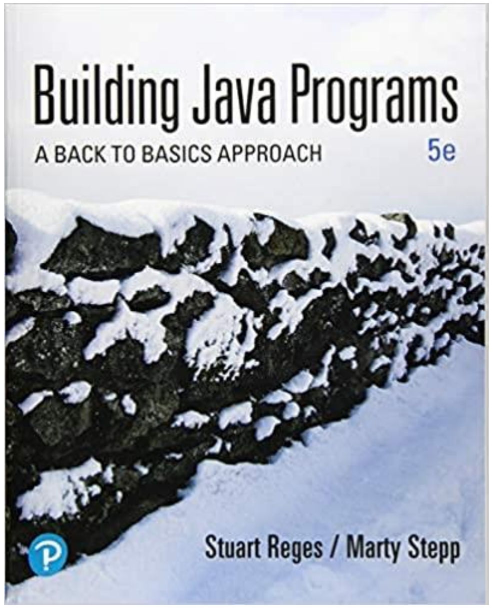


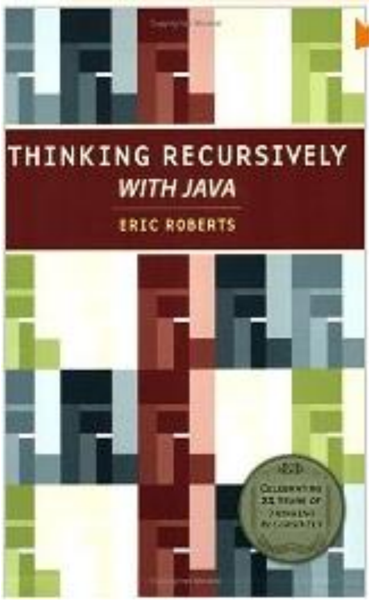
Wednesday ->



Books

- books are recommended, not required
- free alternatives on the web, see schedule
  - BJP ***strongly recommended***
  - Thinking Recursively in Java - recursion





Clicker Question 2

Which of these best describes you?

A. First year at UT and first year college student

B. First year at UT, transferring from another college or university

C. Second year at UT

D. Third year at UT

E. Other

Course Grades

- **Admittedly a complex system**
  - **see syllabus**
  - **complete syllabus quiz**
- **+/- grades used**
- My CS314 Historical Grades
- **82% C- or higher:**
  - **28% A's, 34% B's, 20% C's**
- **8% D or F**
- **10% Q or W (drop)**
- WORK LOAD EVALUATED AS **HIGH (but not EXCESSIVE)** ON COURSE SURVEYS

Programming Assignments

- Individual – **do your own work (no copying or use of generative AIs such as ChatGPT)**
- **Programs checked automatically with plagiarism detection software (MOSS)**
- Turn in the right thing - correct name, correct format or you will lose points / slip days
- Graded on Correctness AND program hygiene

  "Code is read more often than it is written."

  \- *Guido Van Rossum,* Creator of Python
- Slip days: 8 for term, max 2 per assignment, don’t use frivolously

Succeeding in the Course

- Randy Pausch,
  CS Professor at CMU said:

- *"When I got tenure a year early at Virginia, other Assistant Professors would come up to me and say, 'You got tenure early!?!?! What's your secret?!?!?' and I would tell them, 'Call me in my office at 10pm on Friday night and I'll tell you.' "*

- *"A lot of people want a shortcut. I find the best shortcut is the long way, which is basically two words: __**work hard**__."*

Succeeding in the Course - Meta

- "Be the first penguin"
  – Ask questions!!!
  – lecture, section, Ed Discussion, help hours, co-study

- "It is impossible to be perfect"
  – Mistakes are okay.
  – That is how we learn.
  – Trying to be perfect means not taking risks.
  – no risks, no learning

- "Find a Pack"
  – Make friends.
  – Study with them!

How to Get Help

- Ed Discussion Post
- Help Hours
- Class examples
- Examples from book
- Discuss with other students at a high level
- CoStudy Sessions

Succeeding in the Course - Concrete

- Former student:
  - "I really like the boot camp nature of your course."
- do the readings
- start on assignments early
- get help from the teaching staff when you get stuck on an assignment
- attend lecture and discussion sections
- go to the extra study sessions, co-study
- participate on the class discussion group
- **do extra problems** - http://tinyurl.com/pnzp28f
- study for tests using the old tests
- study for tests in groups
- ask questions and get help

Required Software and Hardware

- Java - Oracle or OpenJDK, limit ourselves to Java 8 (see program hygiene requirements for disallowed features)
- IDE such as IntelliJ or Eclipse
  - Do not use a code assist tool such as Copilot on programming assignments
- Device to access Canvas Instapoll during lectures
- Device to take images of section problems and submit to Gradescope
- Zoom, used occasionally

Clicker Question 3

Which computer programming language are you most comfortable with?

A. Java

B. C or C++

C. Python

D. Javascript

E. Other

See: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

and http://lang-index.sourceforge.net/

Topic Number 2
Efficiency – Complexity - Algorithm Analysis

"bit twiddling: 1. (pejorative) An exercise in tuning (see *tune*) in which incredible amounts of time and effort go to produce little noticeable improvement, often with the result that the code becomes incomprehensible."

- The Hackers Dictionary, version 4.4.7

Clicker Question 1

- “A program finds all the prime numbers between 2 and 1,000,000,000 from scratch in 0.37 seconds."
  – Is this a fast solution?

A. no

B. yes

C. it depends

Efficiency

- Computer Scientists don’t just write programs.
- They also ***analyze*** them.
- How efficient is a program?
  – How many computations does it take program to complete?
  – How much memory does a program use?
  – How do these change as the amount of data changes?
  – What is the difference between the average case and worst case efficiency if any?

Technique

- Semi-formal approach for this class
  - more formal techniques in theory classes, CS331
- **How many computations will this program (method, algorithm) perform to get the answer?**
- Many simplifications
  - view algorithms as Java programs
  - **determine by analysis the total number executable statements (computations) in program or method as a function of the amount of data**
  - focus on the *dominant term* in the function

$T(N) = 17N^3 + 25N^2 + 35N + 251$ ***IS ORDER N^3***

Counting Statements

```
int x; // one statement
x = 12; // one statement
int y = z * x + 3 % 5 * x / i; // 1
x++; // one statement
boolean p = x < y && y % 2 == 0 ||
               z >= y * x; // 1
int[] data = new int[100]; // 100
data[50] = x * x + y * y; // 1
```

Clicker 2

- What is output by the following code?

```
int total = 0;
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 11; j++)
        total += 2;
System.out.println(total);
```

A. 24
B. 120
C. 143
D. 286
E. 338

Clicker 3

- What is output when method `sample` is called?

```
// pre: n >= 0, m >= 0
public static void sample(int n, int m) {
    int total = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            total += 5;
    System.out.println(total);
}
```

A. 5

B. n * m

C. n * m * 5

D. n^m

E. $(n * m)^5$

Example

```
public int total(int[] values) {
    int result = 0;
    for (int i = 0; i < values.length; i++)
        result += values[i];
    return result;
}
```

- How many statements are executed by method `total` as a function of `values.length`
- Let N = `values.length`
  - N is commonly used as a variable that denotes the amount of data

Counting Up Statements

- `int result = 0;` 1
- `int i = 0;` 1
- `i < values.length`; N + 1
- `i++` N
- `result += values[i];` N
- `return total;` 1
- T(N) = 3N + 4
- T(N) is the number of executable statements in method `total` as function of `values.length`

Another Simplification

- When determining complexity of an algorithm we want to simplify things
  - ignore some details to make comparisons easier
- Like assigning your grade for course
  - At the end of CS314 your transcript won't list all the details of your performance in the course
  - it won't list scores on all assignments, quizzes, and tests
  - simply a letter grade, B- or A or D+
- So we focus on the dominant term from the function and ignore the coefficient

Big O

- The most common method and notation for discussing the execution time of algorithms is *Big O*, also spoken *Order*
- Big O is a mathematical technique that allows us to take a function and, typically, simplify it.
  - So that when we talk about efficiency in a simpler "language", a language of functions
- Big O is an upper bounds
- Hide a lot of unimportant details by assigning a simple grade (function) to algorithms

Formal Definition of Big O

- T(N) is O(F(N)) if there are positive constants c and N_0 such that $T(N) \leq cF(N)$ when $N \geq N_0$
  - N is the size of the data set the algorithm works on
  - T(N) is a function that characterizes the *actual* running time of the algorithm
  - F(N) is a function that characterizes an upper bounds on T(N). It is a limit on the running time of the algorithm. (The typical Big functions table)
  - c and N_0 are constants

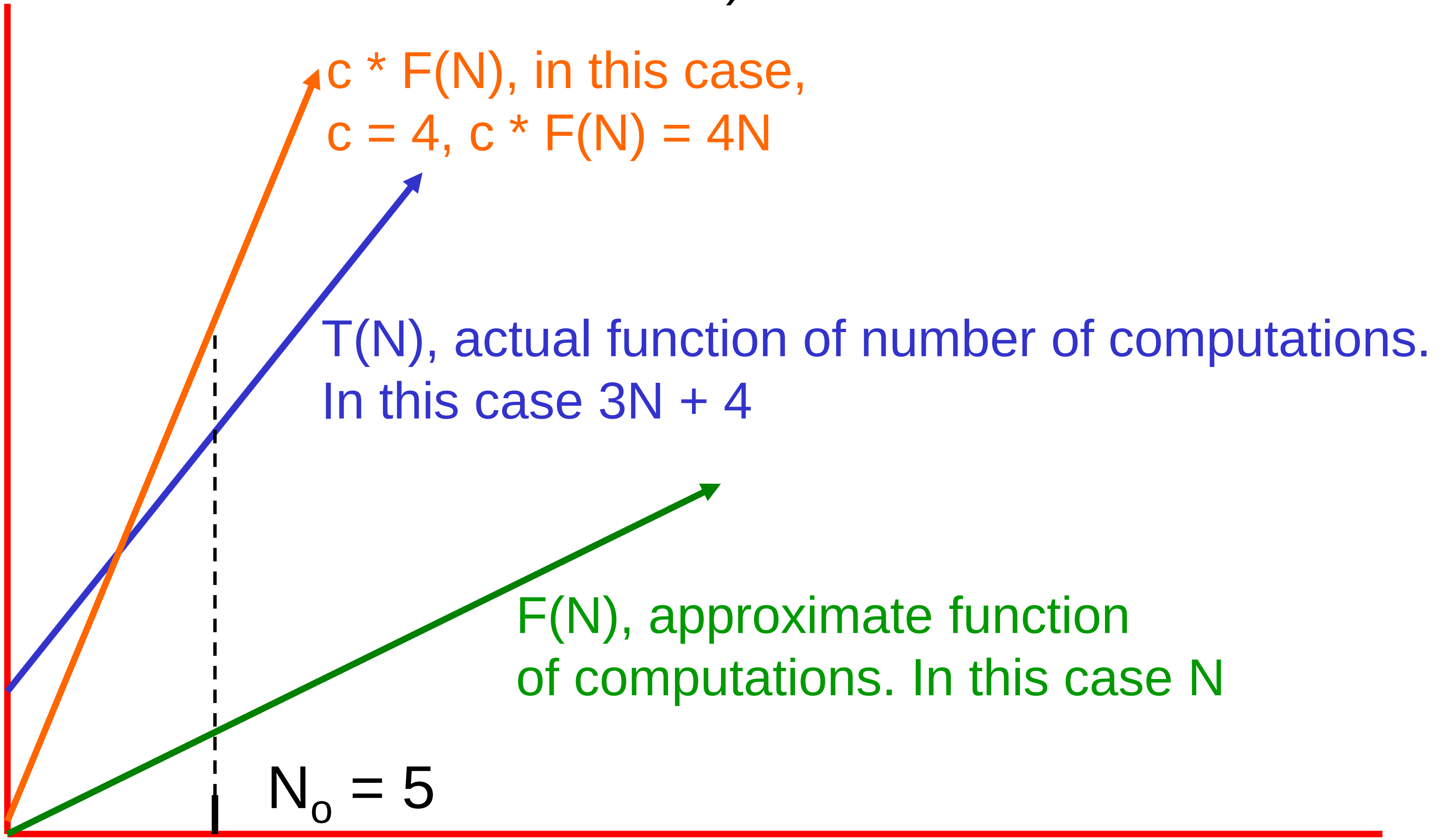
What it Means

- T(N) is the actual growth rate of the algorithm
  - can be equated to the number of executable statements in a program or chunk of code
- F(N) is the function that bounds the growth rate
  - may be upper or lower bound
- T(N) may not necessarily equal F(N)
  - constants and lesser terms ignored because it is a *bounding function*

Showing O(N) is Correct

- Recall the formal definition of Big O
  - T(N) is O(F(N)) if there are positive constants c and N_0 such that $T(N) \leq cF(N)$ when $N > N_0$
- Recall method `total`, T(N) = 3N + 4
  - show method `total` is O(N).
  - F(N) is N
- We need to choose constants c and N_0
- how about c = 4, N_0 = 5 ?

vertical axis: time for algorithm to complete. (simplified to number of executable statements)



horizontal axis: N, number of elements in data set

Typical Big O Functions – "Grades"

Function	Common Name
$N!$	factorial
2^N	Exponential
N^d, $d > 3$	Polynomial
N^3	Cubic
N^2	Quadratic
$N\sqrt{N}$	N Square root N
N log N	N log N
N	Linear
$\sqrt{N}$	Root - n
log N	Logarithmic
1	Constant

Running time grows 'quickly' with more input.

Running time grows 'slowly' with more input.

Clicker 4

- Which of the following is true?
  Recall $T(N)_{total} = 3N + 4$

A. Method `total` is $O(N^{1/2})$

B. Method `total` is $O(N)$

C. Method `total` is $O(N^2)$

D. Two of A – C are correct

E. All of three of A – C are correct

Showing Order More Formally …

- Show $10N^2 + 15N$ is $O(N^2)$
- Break into terms.

- $10N^2 \leq 10N^2$
- $15N \leq 15N^2$ for $N \geq 1$ (Now add)
- $10N^2 + 15N \leq 10N^2 + 15N^2$ for $N \geq 1$
- $10N^2 + 15N \leq 25N^2$ for $N \geq 1$
- $c = 25$, $N_0 = 1$
- Note, the choices for c and N_0 are not unique.

Dealing with other methods

- What do I do about method calls?

```
double sum = 0.0;
for (int i = 0; i < n; i++)
    sum += Math.sqrt(i);
```

- Long way
  - go to that method or constructor and count statements
- Short way
  - substitute the simplified Big O function for that method.
  - **if** Math.sqrt is constant time, O(1), simply count `sum += Math.sqrt(i);` as one statement.

Dealing With Other Methods

```
public int foo(int[] data) {
    int total = 0;
    for (int i = 0; i < data.length; i++)
        total += countDups(data[i], data);
    return total;
}
// method countDups is O(N) where N is the
// length of the array it is passed
```

Clicker 5, What is the Big O of `foo`?

A. O(1) B. O(N) C. O(NlogN)

D. $O(N^2)$ E. O(N!)

Independent Loops

```
// from the Matrix class
public void scale(int factor) {
     for (int r = 0; r < numRows(); r++)
          for (int c = 0; c < numCols(); c++)
               iCells[r][c] *= factor;
}
```

`numRows()` returns number of rows in the matrix iCells

`numCols()` returns number of columns in the matrix iCells

Assume iCells is an N by N square matrix.
Assume numRows and numCols are O(1)

What is the T(N)? Clicker 6, What is the Order?

A. O(1) B. O(N) C. O(NlogN)

D. O(N^2) E. O(N!)

Bonus question. What if numRows is O(N)?

Just Count Loops, Right?

```
// Assume mat is a 2d array of booleans.
// Assume mat is square with N rows,
// and N columns.
public static void count(boolean[][] mat,
                             int row, int col) {
int numThings = 0;
for (int r = row - 1; r <= row + 1; r++)
    for (int c = col - 1; c <= col + 1; c++)
        if (mat[r][c])
            numThings++;
```

Clicker 7, What is the order of the method count?

A. O(1) B. $O(N^{0.5})$ C. O(N) D. $O(N^2)$ E. $O(N^3)$

It is **Not** Just Counting Loops

```
// "Unroll" the loop of method count:
int numThings = 0;
if (mat[r-1][c-1]) numThings++;
if (mat[r-1][c]) numThings++;
if (mat[r-1][c+1]) numThings++;
if (mat[r][c-1]) numThings++;
if (mat[r][c]) numThings++;
if (mat[r][c+1]) numThings++;
if (mat[r+1][c-1]) numThings++;
if (mat[r+1][c]) numThings++;
if (mat[r+1][c+1]) numThings++;
```

Just Count Loops, Right?

```
private static void mystery(int[] data) {
        stopIndex = data.length - 1;
        int j = 1;
        while (stopIndex > 0) {
                if (data[j - 1] > data[j]) {
                        int t = data[j];
                        data[j] = data[j - 1];
                        data[j - 1] = t;
                }
                if (j == stopIndex) {
                        stopIndex--;
                        j = 1;
                } else {
                        j++;
                }
        }
```

N = data.length

Clicker 8, What is the order of method mystery?

A. O(1) B. O($N^{0.5}$) C. O(N) D. O(N^2) E. O(N^3)

Sidetrack, the logarithm

- Thanks to Dr. Math
- $3^2 = 9$
- likewise $\log_3 9 = 2$
  - "The log to the base 3 of 9 is 2."
- The way to think about log is:
  - "the log to the base x of y is the number you can raise x to to get y."
  - Say to yourself "The log is the exponent." (and say it over and over until you believe it.)
  - In CS we work with base 2 logs, a lot
- $\log_2 32 = ?$ $\log_2 8 = ?$ $\log_2 1024 = ?$ $\log_{10} 1000 = ?$

When Do Logarithms Occur

- Algorithms tend to have a logarithmic term when they use a divide and conquer technique
- the size of the data set keeps getting divided by 2

```
public int foo(int n) {
    // pre n > 0
    int total = 0;
    while (n > 0) {
        n = n / 2;
        total++;
    }
    return total;
}
```

- Clicker 9, What is the order of the above code?

A. O(1) B. O(logN) C. O(N)

D. O(Nlog N) E. O(N^2)

The base of the log is typically not included as we can switch from one base to another by multiplying by a constant factor.

Significant Improvement – Algorithm with Smaller Big O function

- Problem: Given an array of ints replace any element equal to 0 with the maximum positive value to the right of that element. (if no positive value to the right, leave unchanged.)

Given:

`[0, 9, 0, 13, 0, 0, 7, 1, -1, 0, 1, 0]`

Becomes:

[**13**, 9, **13**, 13, **7**, **7**, 7, 1, -1, **1**, 1, 0]

Replace Zeros – Typical Solution

```
public void replace0s(int[] data){
  for(int i = 0; i < data.length; i++){
    if (data[i] == 0) {
      int max = 0;
      for(int j = i+1; j<data.length; j++)
        max = Math.max(max, data[j]);
       data[i] = max;
    }
  }
}
```

Assume all values are zeros. (worst case)

Example of a **dependent loops**.

Clicker 10 - Number of times j < data.length evaluated?

A. O(1)

B. O(N)

C. O(NlogN)

D. O(N^2)

E. O(N!)

Replace Zeros – Alternate Solution

```
public void replace0s(int[] data){
    int max =
        Math.max(0, data[data.length - 1]);
    int start = data.length - 2;
    for (int i = start; i >= 0; i--) {
        if (data[i] == 0)
            data[i] = max;
        else
            max = Math.max(max, data[i]);
  }
}
```

Clicker 11 - Big O of this approach?

A. O(1)
B. O(N)
C. O(NlogN)
D. O(N^2)
E. O(N!)

Clicker 12

- Is O(N) really that much faster than O(N^2)?

A. never

B. always

C. typically

- Depends on the actual functions and the value of N.
- 1000N + 250 compared to N^2 + 10
- When do we use mechanized computation?
- N = 100,000
- 100,000,250 < 10,000,000,010 ($10^8 < 10^{10}$)

A VERY Useful Proportion

- Since F(N) is characterizes the running time of an algorithm the following proportion should hold true:

$F(N_0) / F(N_1) \sim= time_0 / time_1$

- An algorithm that is $O(N^2)$ takes 3 seconds to run given 10,000 pieces of data.
  - How long do you expect it to take when there are 30,000 pieces of data?
  - common mistake
  - logarithms?

Why Use Big O?

- As we build data structures Big O is the tool we will use to decide under what conditions one data structure is better than another
- Think about performance when there is a lot of data.
  - "It worked so well with small data sets..."
  - Joel Spolsky, Schlemiel the painter's Algorithm
- Lots of trade offs
  - some data structures good for certain types of problems, bad for other types
  - often able to trade SPACE for TIME.
  - Faster solution that uses more space
  - Slower solution that uses less space

Big O Space

- Big O could be used to specify how much space is needed for a particular algorithm
  – in other words how many variables are needed
- Often there is a *time – space tradeoff*
  – can often take less time if willing to use more memory
  – can often use less memory if willing to take longer
  – truly beautiful solutions take less time and space

The biggest difference between time and space is that you can't reuse time. - Merrick Furst

Quantifiers on Big O

- It is often useful to discuss different cases for an algorithm
- Best Case: what is the best we can hope for?
  – least interesting, but a good exercise
  – **Don't assume no data. Amount of date is still variable, possibly quite large**
- Average Case (a.k.a. expected running time): what usually happens with the algorithm?
- Worst Case: what is the worst we can expect of the algorithm?
  – very interesting to compare this to the average case

Best, Average, Worst Case

- To Determine the best, average, and worst case Big O we must make assumptions about the data set
- Best case -> what are the properties of the data set that will lead to the fewest number of executable statements (steps in the algorithm)
- Worst case -> what are the properties of the data set that will lead to the largest number of executable statements
- Average case -> Usually this means assuming the data is randomly distributed
  – or if I ran the algorithm a large number of times with different sets of data what would the average amount of work be for those runs?

Another Example

```
public double minimum(double[] values) {
    int n = values.length;
    double minValue = values[0];
    for (int i = 1; i < n; i++)
        if (values[i] < minValue)
            minValue = values[i];
    return minValue;
}
```

- T(N)? F(N)? Big O? Best case? Worst Case? Average Case?
- If no other information, assume asking average case

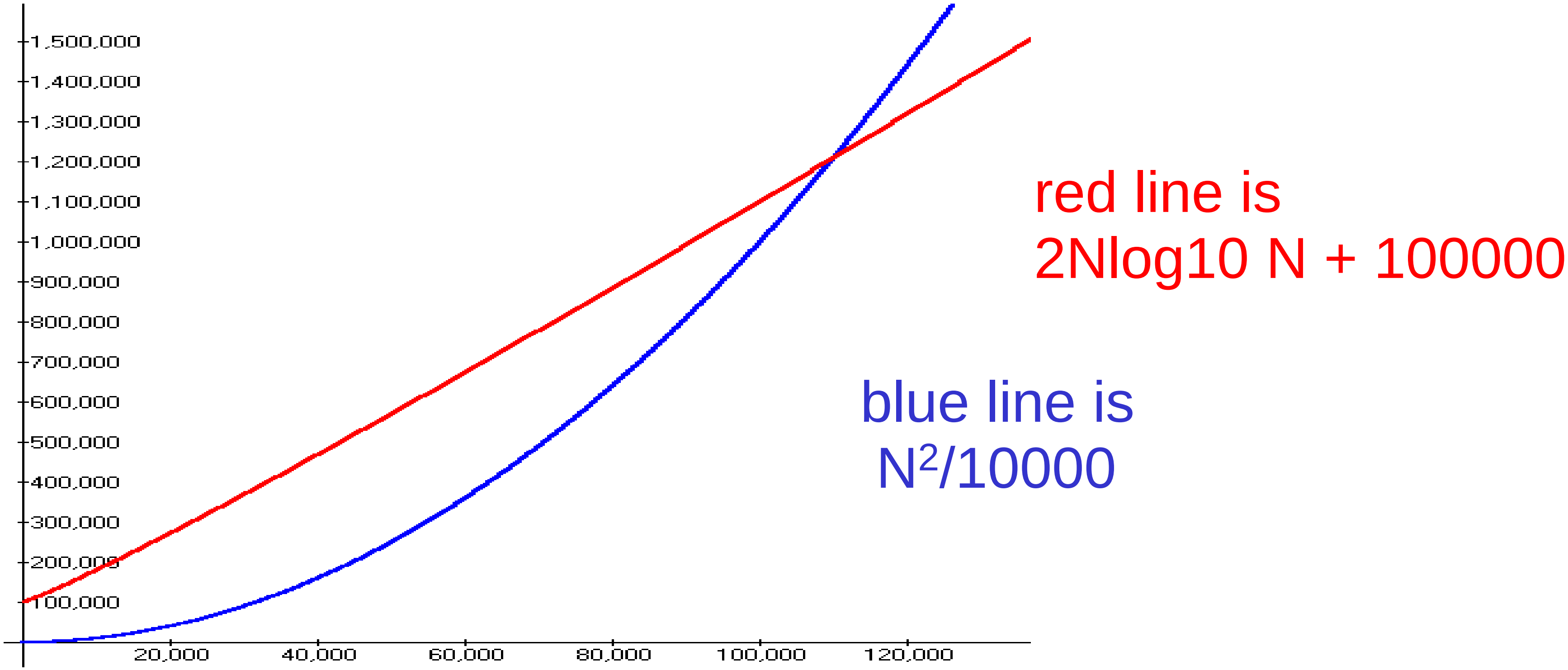
Example of Dominance

- Look at an extreme example. Assume the actual number as a function of the amount of data is:

  $$N^2/10000 + 2N\log_{10} N + 100000$$

- Is it plausible to say the N^2 term dominates even though it is divided by 10000 and that the algorithm is $O(N^2)$?
- What if we separate the equation into $(N^2/10000)$ and $(2N \log_{10} N + 100000)$ and graph the results.

Summing Execution Times



- For large values of N the N^2 term dominates so the algorithm is $O(N^2)$
- When does it make sense to use a computer?

Comparing Grades

- Assume we have a problem
- Algorithm A solves the problem correctly and is $O(N^2)$
- Algorithm B solves the same problem correctly and is $O(N \log_2 N)$
- Which algorithm is faster?
- One of the assumptions of Big O is that the data set is large.
- The "grades" should be accurate tools if this holds true.

Running Times

- Assume N = 100,000 and processor speed is 1,000,000,000 operations per second

Function	Running Time
2^N	$3.2 \times 10^{30,086}$ years
N^4	3171 years
N^3	11.6 days
N^2	10 seconds
$N\sqrt{N}$	0.032 seconds
N log N	0.0017 seconds
N	0.0001 seconds
$\sqrt{N}$	3.2×10^{-7} seconds
log N	1.2×10^{-8} seconds

Theory to Practice OR
Dykstra says: "Pictures are for the Weak."

	1000	2000	4000	8000	16000	32000	64000	128K
O(N)	2.2×10^{-5}	2.7×10^{-5}	5.4×10^{-5}	4.2×10^{-5}	6.8×10^{-5}	1.2×10^{-4}	2.3×10^{-4}	5.1×10^{-4}
O(NlogN)	8.5×10^{-5}	1.9×10^{-4}	3.7×10^{-4}	4.7×10^{-4}	1.0×10^{-3}	2.1×10^{-3}	4.6×10^{-3}	1.2×10^{-2}
$O(N^{3/2})$	3.5×10^{-5}	6.9×10^{-4}	1.7×10^{-3}	5.0×10^{-3}	1.4×10^{-2}	3.8×10^{-2}	0.11	0.30
$O(N^2)$ ind.	3.4×10^{-3}	1.4×10^{-3}	4.4×10^{-3}	0.22	0.86	3.45	13.79	(55)
$O(N^2)$ dep.	1.8×10^{-3}	7.1×10^{-3}	2.7×10^{-2}	0.11	0.43	1.73	6.90	(27.6)
$O(N^3)$	3.40	27.26	(218)	(1745) 29 min.	(13,957) 233 min	(112k) 31 hrs	(896k) 10 days	(7.2m) 80 days

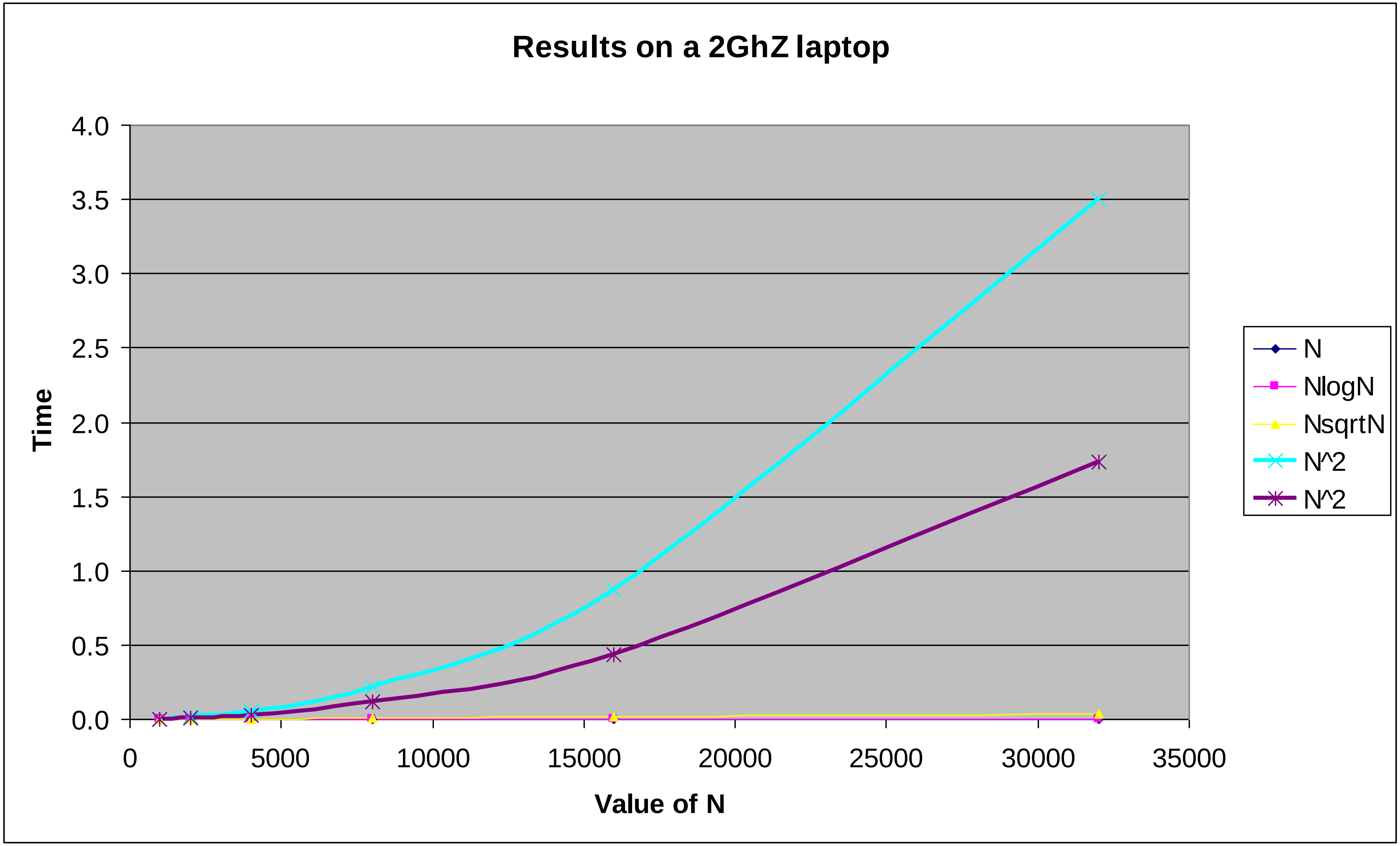
Times in Seconds. Red indicates predicated value.

Change between Data Points

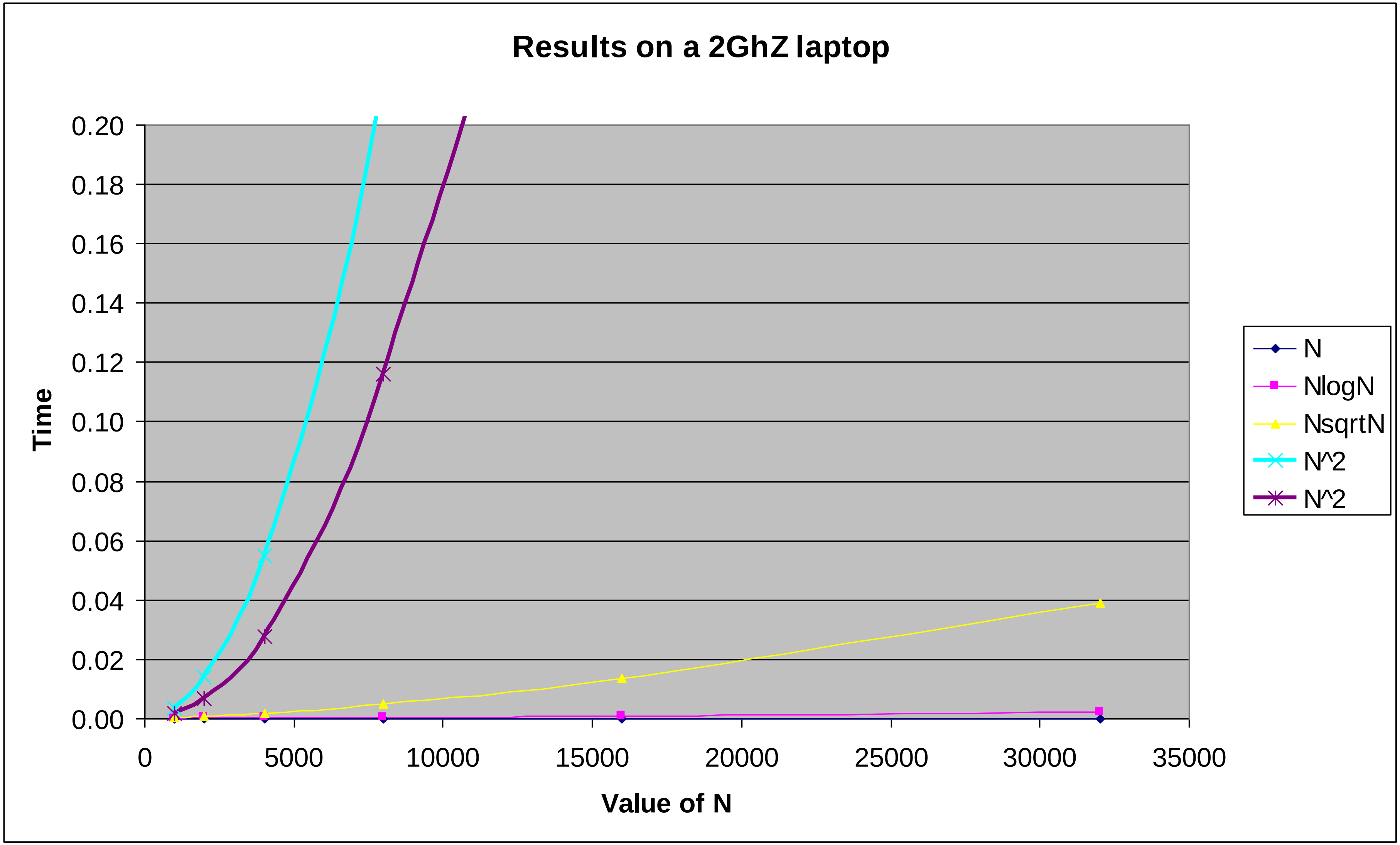
	1000	2000	4000	8000	16000	32000	64000	128K	256k	512k
O(N)	-	1.21	2.02	0.78	1.62	1.76	1.89	2.24	2.11	1.62
O(NlogN)	-	2.18	1.99	1.27	2.13	2.15	2.15	2.71	1.64	2.40
O($N^{3/2}$)	-	1.98	2.48	2.87	2.79	2.76	2.85	2.79	2.82	2.81
O(N^2) ind	-	4.06	3.98	3.94	3.99	4.00	3.99	-	-	-
O(N^2) dep	-	4.00	3.82	3.97	4.00	4.01	3.98	-	-	-
O(N^3)	-	8.03	-	-	-	-	-	-	-	-

Value obtained by Time_x / Time_{x-1}

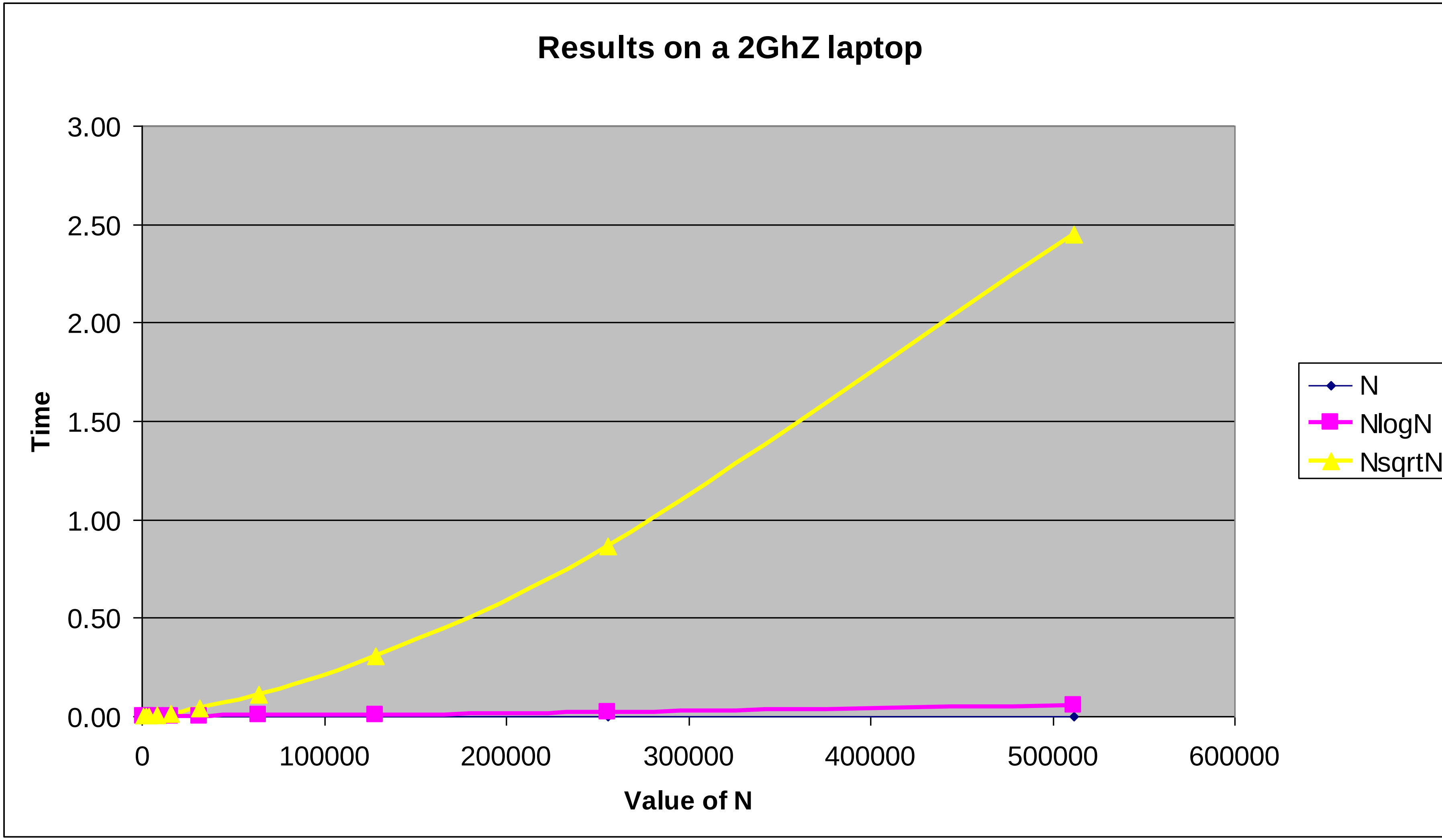
Okay, Pictures



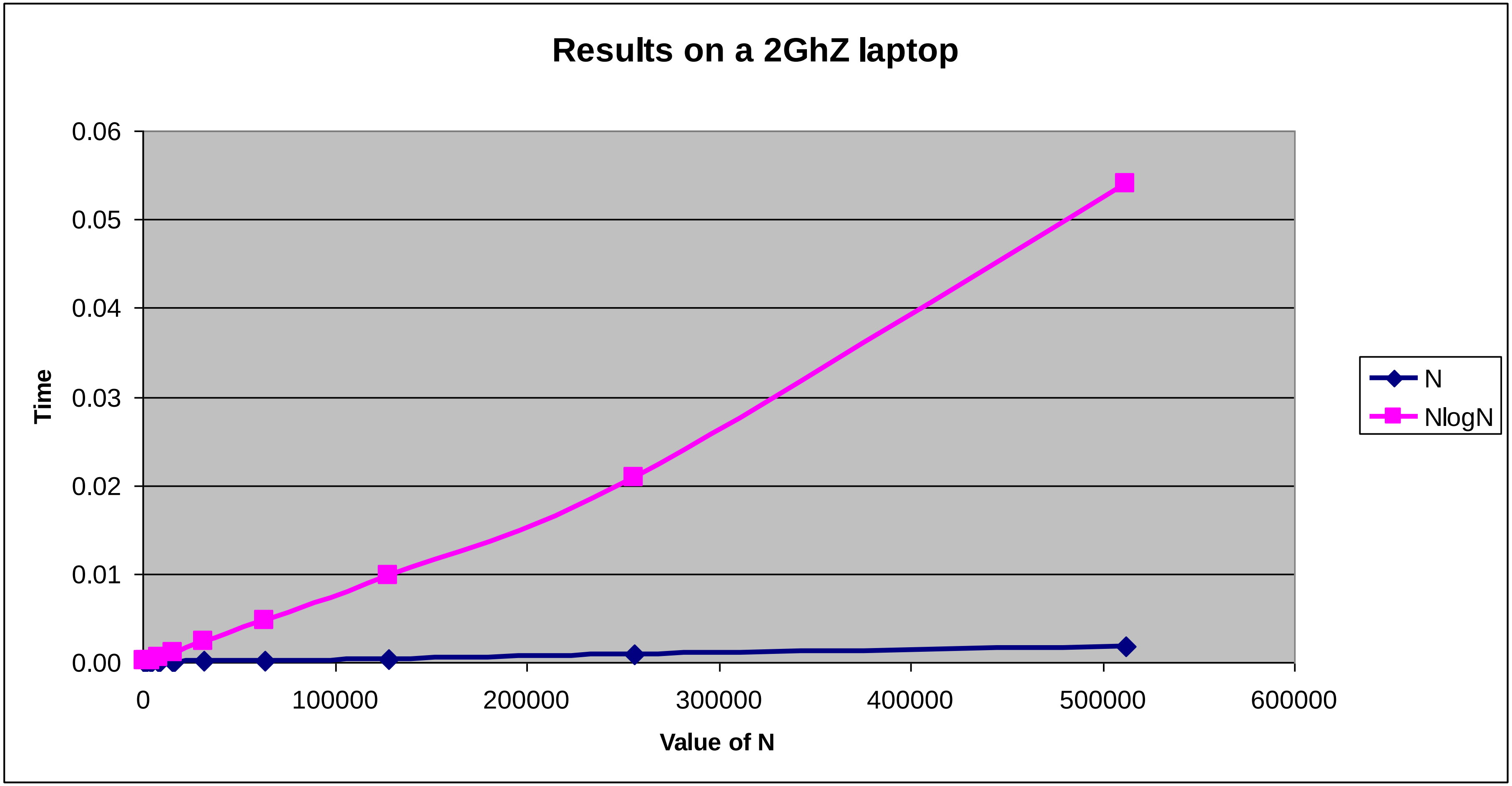
Put a Cap on Time



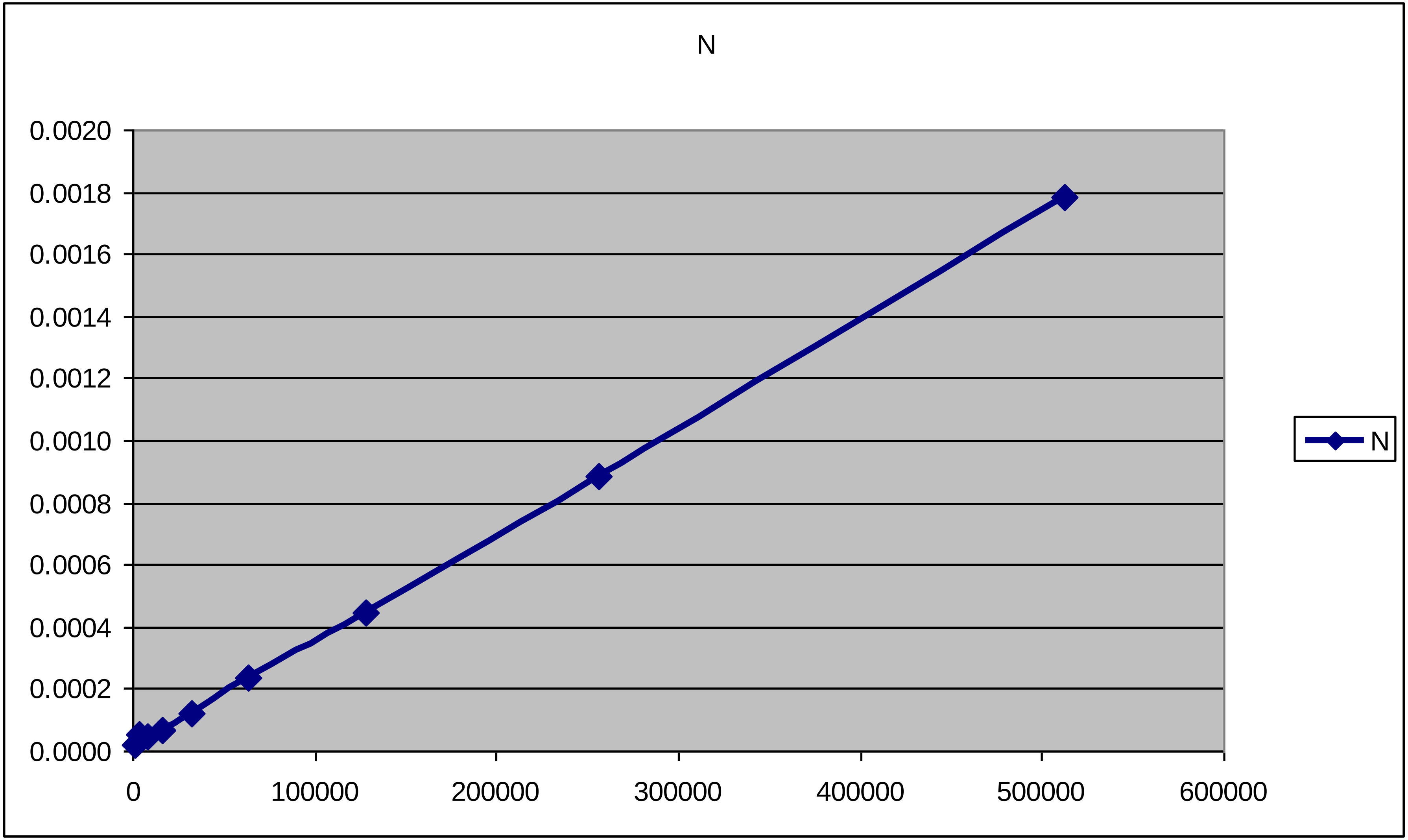
No O(N^2) Data


Results on a 2GhZ laptop
Time
3.00
2.50
2.00
1.50
1.00
0.50
0.00
0
100000
200000
300000
400000
500000
600000
Value of N
N
NlogN
NsqrtN

Just O(N) and O(NlogN)



Just O(N)



N
0.0020
0.0018
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
100000
200000
300000
400000
500000
600000
N

10^9 instructions/sec, runtimes

N	*O(log N)*	*O(N)*	*O(N log N)*	*O(N^2)*
10	0.000000003	0.00000001	0.000000033	0.0000001
100	0.000000007	0.00000010	0.000000664	0.0001000
1,000	0.000000010	0.00000100	0.000010000	0.001
10,000	0.000000013	0.00001000	0.000132900	0.1 min
100,000	0.000000017	0.00010000	0.001661000	10 seconds
1,000,000	0.000000020	0.001	0.0199	16.7 minutes
1,000,000,000	0.000000030	1.0 second	30 seconds	31.7 years

Formal Definition of Big O (repeated)

- T(N) is O(F(N)) if there are positive constants c and N_0 such that $T(N) \leq cF(N)$ when $N \geq N_0$
  - N is the size of the data set the algorithm works on
  - T(N) is a function that characterizes the *actual* running time of the algorithm
  - F(N) is a function that characterizes an upper bounds on T(N). It is a limit on the running time of the algorithm
  - c and N_0 are constants

More on the Formal Definition

- There is a point N_0 such that for all values of N that are past this point, T(N) is bounded by some multiple of F(N)
- Thus if T(N) of the algorithm is O(N^2) then, ignoring constants, at some point we can *bound* the running time by a quadratic function.
- given a *linear* algorithm it is *technically correct* to say the running time is O(N ^ 2). O(N) is a more precise answer as to the Big O of the linear algorithm
  - thus the caveat "pick the most restrictive function" in Big O type questions.

What it All Means

- T(N) is the actual growth rate of the algorithm
  - can be equated to the number of executable statements in a program or chunk of code
- F(N) is the function that bounds the growth rate
  - may be upper or lower bound
- T(N) may not necessarily equal F(N)
  - constants and lesser terms ignored because it is a *bounding function*

Other Algorithmic Analysis Tools

- *Big Omega* T(N) is Ω(F(N)) if there are positive constants c and N_0 such that T(N) $\geq$ cF(N)) when N $\geq$ N_0
  - Big O is similar to less than or equal, an upper bounds
  - Big Omega is similar to greater than or equal, a lower bound
- *Big Theta* T(N) is θ(F(N)) if and only if T(N) is O(F(N))and T(N) is Ω(F(N)).
  - Big Theta is similar to equals

Relative Rates of Growth

Analysis Type	Mathematical Expression	Relative Rates of Growth
Big O	$T(N) = O(F(N))$	$T(N) \le F(N)$
Big Ω	$T(N) = \Omega(F(N))$	$T(N) \ge F(N)$
Big θ	$T(N) = \theta(F(N))$	$T(N) = F(N)$

"In spite of the additional precision offered by Big Theta, Big O is more commonly used, except by researchers in the algorithms analysis field" - Mark Weiss

Topic 3
Encapsulation - Implementing Classes

"And so, from Europe, we get things such as ... object-oriented analysis and design (a clever way of breaking up software programming instructions and data into small, reusable objects, based on certain abstraction principles and design hierarchies.)"

-*Michael A. Cusumano,*
The Business Of Software

Object Oriented Programming

- Creating large programs that work turns out to be very difficult
  - DIA Automated baggage handling system
  - Ariane 5 Flight 501
  - More
- Object oriented programming is one way of *managing the complexity* of programming and software projects
- Break up big problems into smaller, more manageable problems

Object Oriented Programming

- "Object-oriented programming is a method of programming based on a hierarchy of classes, and well-defined and cooperating objects. "
- What is a class?
- "A class is a structure that defines the data and the methods to work on that data. When you write programs in the Java language, all program data is wrapped in a class, whether it is a class you write or a class you use from the Java platform API libraries."
  - a new data type

Object Oriented Programming

- In other words break the problem up based on the things / data types that are part of the problem
- Not the only way
- One of many different kinds of strategies or *paradigms* for software development
  - functional, procedural, event driven, data flow, formal methods, agile or extreme, ...
- In 314 we will do a lot of *object based* programming

Example - Monopoly

If we had to start from scratch what classes would we need to create?

Encapsulation

- One of the features of object oriented languages
- Allows programmers to define **new data types**
- Hide the data of an object (variable)
- Group operations and data together into a new data type
- Usually easier to ***use*** something than understand ***exactly how it works***
  - microwave, car, computer, software, mp3 player

Data Structures

- A data structure is a variable that stores other variables. (overly simplified definition)
  - aka Collection, Container
- May be ordered or unordered (from client's perspective)
  - Order a first element, second element,…
  - Lists are ordered, sets are typically unordered
- May allow duplicate values or not
  - Lists allow duplicates, sets typically do not

The IntList Class

- We will develop a class that models a list of ints
  – initially a pale imitation of the Java ArrayList class
- Improvement on an array of ints
  – resize automatically
  – insert easily
  – remove easily
- A list - our first *data structure*
  – a variable that stores other variables
- Lists maintain elements in a definite order and duplicates are allowed

```
 0    1   2    3    4   <- indices / positions
[5, 12, 5, 17, -5] <- elements
```

Clicker 1

Our `IntList` class has an array of ints instance variable (`int[] container`). What should the length of this internal array be?

A. less than the size of the list
B. equal to the size of the list
C. greater than or equal to the size of the list
D. some fixed amount that never changes
E. 0

Array length less than the number of elements in the list?!?

- What if most elements are all the same value? Only store the elements (and their position) not equal to the default? Sparse List

Clicker 2

When adding a new element to a list, where should the new element be added by default?

A. The beginning
B. The end
C. The middle
D. A random location
E. Don’t bother to actually add

IntList Design

- Create a new, empty IntList

```
new IntList -> []
```

- The above is not code. It is a notation that shows what the results of operations. `[]` is an empty list.
- add to a list.

```
[].add(1) -> [1]
[1].add(5) -> [1, 5]
[1, 5].add(4) -> [1, 5, 4]
```

- elements in a list have a definite order and a position.
  - zero based position or 1 based positioning?

```
IntList aList = new IntList();
aList.add(42);
aList.add(12);
aList.add(37);
```

Abstract view of list of integers

0 1 2
[42, 12, 37]

The wall of abstraction.

Instance Variables

- Internal data
  - also called instance variables because every instance (object) of this class has its own copy of these
  - something to store the elements of the list
  - size of internal storage container?
  - if not what else is needed
- Must be clear on the difference between the internal data of an IntList object and the IntList that is being represented
- Why make internal data private?

Constructors

- For initialization of objects
- IntList constructors
  - default
  - initial capacity?
- redirecting to another constructor

```
this(10);
```

- class constants
  - what `static` means

Default add method

- where to add?
- what if not enough space?

```
[].add(3) -> [3]
[3].add(5) -> [3, 5]
[3, 5].add(3) -> [3, 5, 3]
```

- Testing, testing, testing!
  - a `toString` method would be useful

The IntList Class

- instance variables
- constructors
  - default
  - initial capacity
    - preconditions, exceptions, postconditions, assert
  - meaning of static
- add method
- get method
- size method

toString method

- return a Java String of list
- empty list -> `[]`
- one element -> `[12]`
- multiple elements -> `[12, 0, 5, 4]`

Clicker 3 - Timing Experiment

- Add N elements to an initially empty IntList then call toString. Time both events. How does the time to add compare to the time to complete toString?

```
IntList list = new IntList();
for (int i = 0; i < N; i++)
    list.add(i); // resize, cap * 2
String s = list.toString();
```

A. time to add << time for toString()
B. time to add < time for toString()
C. time to add ~= time for toString()
D. time to add > time for toString()
E. time to add >> time for toString()

The IntList Class

- testing!!!
- toString
  - "beware the performance of String concatenation" – Joshua Bloch
- insert method `(int pos, int value)`
- remove method `(int pos)`
- insertAll method
  `(int pos, IntList other)`
  - queens and kings of all the IntLists!!!

Clicker Question 4

What is output by the following code?

```
IntList list
list = new IntList(25);
System.out.println(list.size());
```

A. `25`

B. `0`

C. `-1`

D. unknown

E. No output due to runtime error.

get and size methods

- get
  - access element from list
  - preconditions?

`[3, 5, 2].get(0)` returns `3`

`[3, 5, 2].get(1)` returns `5`

- size
  - number of elements in the list
  - Do not confuse with the capacity of the internal storage container
  - The array is not the list!

`[4, 5, 2].size()` returns `3`

insert method

- add at someplace besides the end

```
[3, 5].insert(1, 4) -> [3, 4, 5]
```

where what

```
[3, 4, 5].insert(0, 4) -> [4, 3, 4, 5]
```

- preconditions?
- overload add?
- chance for internal loose coupling

Clicker 5

What is output by the following code?

```
IntList list = new IntList();
list.add(3);
list.insert(0, 4); // position, value
list.insert(1, 1);
list.add(5);
list.insert(2, 9);
System.out.println(list);
```

A. `[4, 1, 3, 9, 5]`
B. `[3, 4, 1, 5, 9]`
C. `[4, 1, 9, 3, 5]`
D. `[3, 1, 4, 9, 5]`
E. Something else

remove method

- remove an element from the list based on location

```
[3, 4, 5].remove(0) -> [4, 5]
[3, 5, 6, 1, 2].remove(2) ->
            [3, 5, 1, 2]
```

- preconditions?
- return value?
  – accessor methods, mutator methods, and mutator methods that return a value

Clicker Question 6

What is output by the following code?

```
IntList list = new IntList();
list.add(12);
list.add(15);
list.add(12);
list.add(17);
list.remove(1);
System.out.println(list);
```

A. `[15, 17]`
B. `[12, 17]`
C. `[12, 0, 12, 17]`
D. `[12, 12, 17]`
E. `[15, 12, 17]`

insertAll method

- add all elements of one list to another starting at a specified location

`[5, 3, 7].insertAll(2, [2, 3]) ->`

`[5, 3,` **`2, 3`**`, 7]`

The parameter `[2, 3]` would be unchanged.

- Working with other objects of the same type
  - `this`?
  - where is private private?
  - loose coupling vs. performance
  - queens and kings of all the IntLists!!!

Clicker 7 - InsertAll First Version

- What is the order of the first version of InsertAll? Assume both lists have N elements and that the insert position is halfway through the calling list.

A. O(1)

B. O(logN)

C. $O(N^{0.5})$

D. O(N)

E. $O(N^2)$

Class Design and Implementation – Another Example

This example will not be covered in class.

The Die Class

- Consider a class used to model a die
- What is the interface? What actions should a die be able to perform?

- The methods or behaviors can be broken up into constructors, mutators, accessors

The Die Class Interface

- Constructors (used in creation of objects)
  - default, single int parameter to specify the number of sides, int and boolean to determine if should roll
- Mutators (change state of objects)
  - roll
- Accessors (do not change state of objects)
  - getResult, getNumSides, toString
- Public constants
  - DEFAULT_SIDES

Visibility Modifiers

- All parts of a *class* have visibility modifiers
  - Java keywords
  - **public**, protected, **private,** (no modifier means package access)
  - do not use these modifiers on local variables (syntax error)
- **public** means that constructor, method, or field may be accessed outside of the class.
  - part of the interface
  - constructors and methods are generally public
- **private** means that part of the class is hidden and inaccessible by code outside of the class
  - part of the implementation
  - data fields are generally private

The Die Class Implementation

- Implementation is made up of constructor code, method code, and private data members of the class.
- scope of data members / instance variables
  – *private data members may be used in any of the constructors or methods of a class*
- Implementation is hidden from users of a class and can be changed without changing the interface or affecting clients (other classes that use this class)
  – Example: Previous version of Die class, DieVersion1.java
- Once Die class completed can be used in anything requiring a Die or situation requiring random numbers between 1 and N
  – DieTester class. What does it do?

DieTester method

```
public static void main(String[] args) {
       final int NUM_ROLLS = 50;
       final int TEN_SIDED = 10;
       Die d1 = new Die();
       Die d2 = new Die();
       Die d3 = new Die(TEN_SIDED);
       final int MAX_ROLL = d1.getNumSides() +
              d2.getNumSides() + d3.getNumSides();


       for(int i = 0; i < NUM_ROLLS; i++)
       {      d1.roll();
              d2.roll();
              System.out.println("d1: " + d1.getResult()
                     + " d2: " + d2.getResult() + " Total: "
                     + (d1.getResult() + d2.getResult() ) );
       }
```

DieTester continued

```
int total = 0;
int numRolls = 0;
do
{       d1.roll();
        d2.roll();
        d3.roll();
        total = d1.getResult() + d2.getResult()
                + d3.getResult();
        numRolls++;
}
while(total != MAX_ROLL);

System.out.println("\n\nNumber of rolls to get "
        + MAX_ROLL + " was " + numRolls);
```

Correctness Sidetrack

- When creating the public interface of a class give careful thought and consideration to the *contract* you are creating between yourself and users (other programmers) of your class
- Use *preconditions* to state what you assume to be true before a method is called
  - caller of the method is responsible for making sure these are true
- Use post*conditions* to state what you guarantee to be true after the method is done if the preconditions are met
  - implementer of the method is responsible for making sure these are true

Precondition and Postcondition Example

```
/* pre: numSides > 1
   post: getResult() = 1, getNumSides() = sides
*/
 public Die(int numSides)
 {   assert (numSides > 1) : “Violation of precondition: Die(int)”;
      iMyNumSides = numSides;
      iMyResult = 1;
      assert getResult() == 1 && getNumSides() == numSides;
  }
```

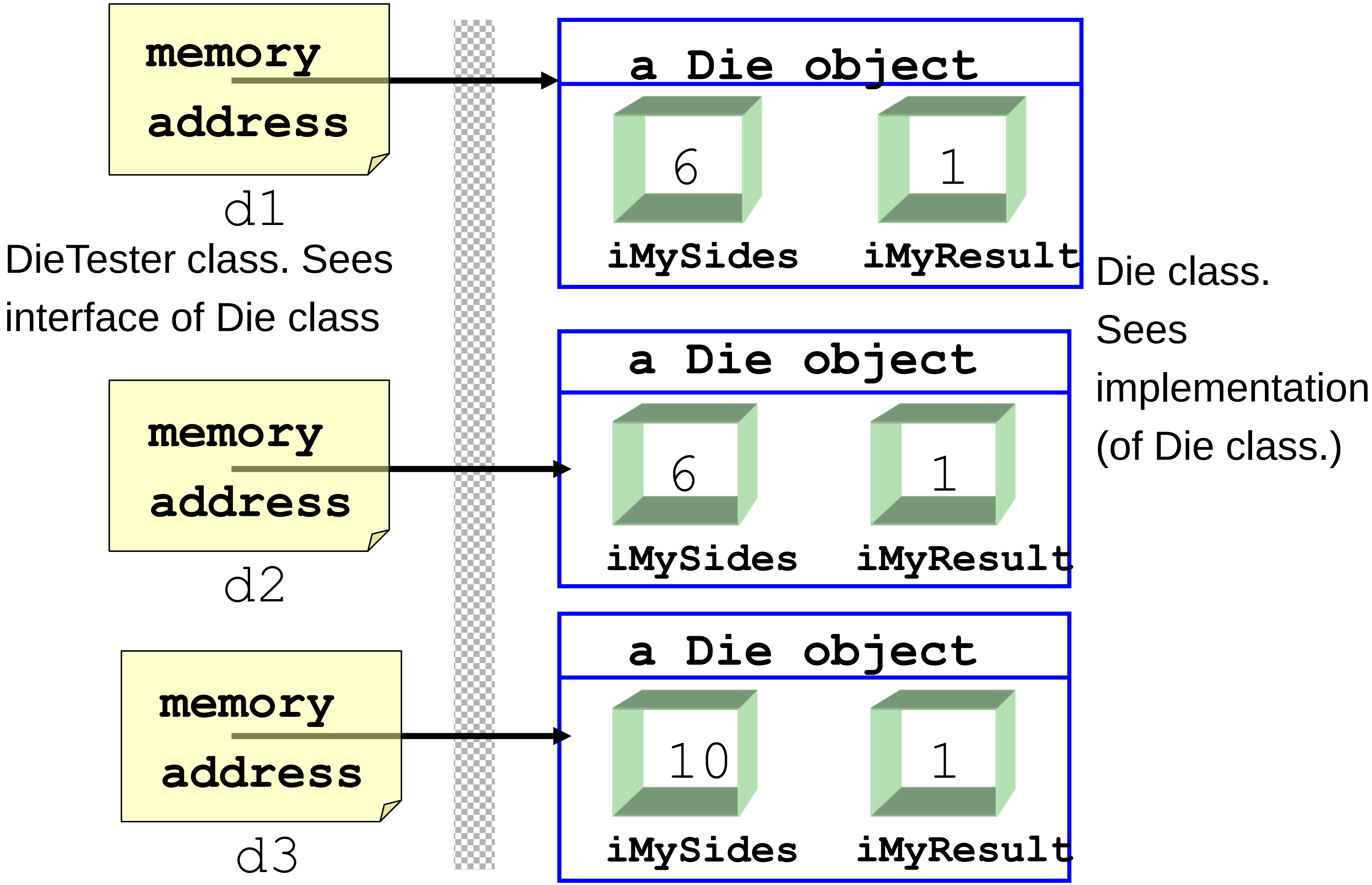
Object Behavior - Instantiation

- Consider the DieTester class

```
Die d1 = new Die();
Die d2 = new Die();
Die d3 = new Die(10);
```

- When the new operator is invoked control is transferred to the Die class and the specified constructor is executed, based on parameter matching
- Space(memory) is set aside for the new object's fields
- The memory address of the new object is passed back and stored in the object variable (pointer)
- After creating the object, methods may be called on it.

Creating Dice Objects



Objects

- Every Die object created has its own instance of the variables declared in the class blueprint

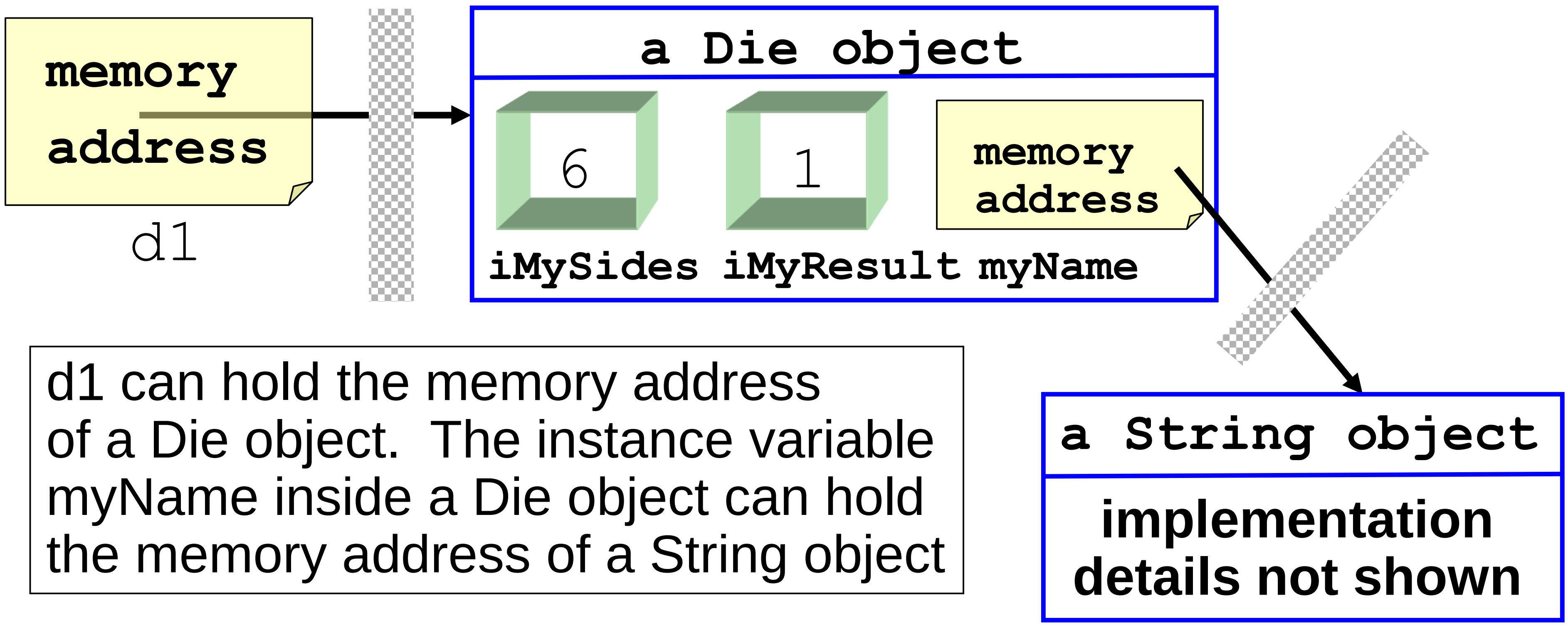
```
private int iMySides;
private int iMyResult;
```

- thus the term *instance variable*
- the instance vars are part of the hidden implementation and may be of *any* data type
  - unless they are public, which is almost always a bad idea if you follow the tenets of information hiding and encapsulation

Complex Objects

- What if one of the instance variables is itself an object?
- add to the Die class

```
private String myName;
```



d1 can hold the memory address of a Die object. The instance variable myName inside a Die object can hold the memory address of a String object

The Implicit Parameter

- Consider this code from the Die class

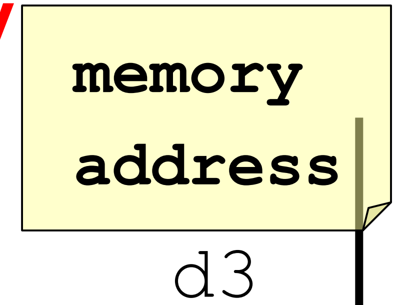
```
public void roll()
{   iMyResult =
        ourRandomNumGen.nextInt(iMySides) + 1;
}
```

- Taken in isolation this code is rather confusing.
- what is this iMyResult thing?
  - It's not a parameter or local variable
  - why does it exist?
  - *it belongs to the Die object that called this method*
  - if there are numerous Die objects in existence
  - Which one is used depends on which object called the method.

The `this` Keyword

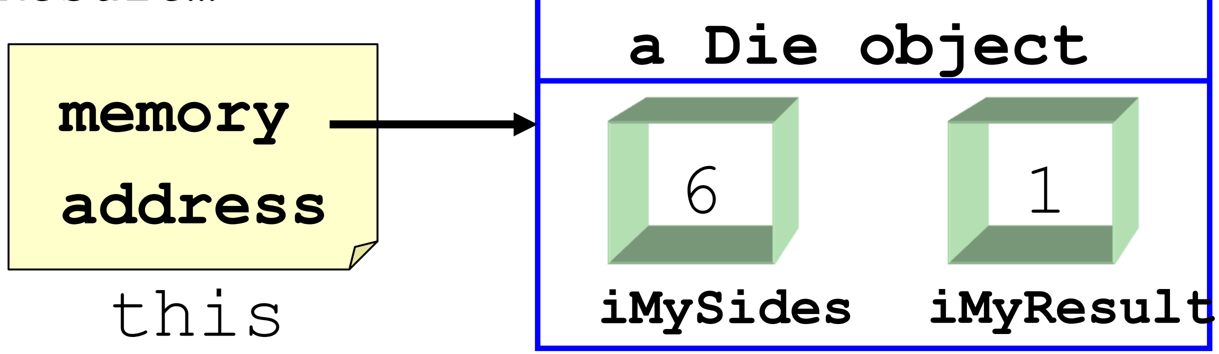
- When a method is called it may be necessary for the calling object to be able to refer to itself
  - most likely so it can pass itself somewhere as a parameter
- when an object calls a method an implicit reference is assigned to the calling object
- the name of this implicit reference is `this`
- `this` is a reference to the current calling object and may be used as an object variable (may not declare it)

this Visually




```
// in some class other than Die
Die d3 = new Die();
d3.roll();
```

```
// in the Die class
public void roll()
{   iMyResult =
        ourRandomNumGen.nextInt(iMySides) + 1;
    /* OR
       this.iMyResult…
    */
}
```



An equals method

- working with objects of the same type in a class can be confusing
- write an equals method for the Die class. assume every Die has a myName instance variable as well as iMyNumber and iMySides

A Possible Equals Method

```
public boolean equals(Object otherObject)
{     Die other = (Die)otherObject;
      return iMySides == other.iMySides
            && iMyResult== other.iMyResult
            && myName.equals( other.myName );
}
```

- Declared Type of Parameter is Object not Die
- override (replace) the equals method instead of overload (present an alternate version)
  - easier to create generic code
- we will see the equals method is *inherited* from the Object class
- access to another object's private instance variables?

Another equals Methods

```
public boolean equals(Object otherObject)
{    // dangerous! Not checking for null or type.
     Die other = (Die)otherObject;
     return this.iMySides == other.iMySides
          && this.iMyNumber == other.iMyNumber
          && this.myName.equals( other.myName );
}
```

Using the this keyword / reference to access the implicit parameters instance variables is unnecessary.

If a method within the same class is called within a method, the original calling object is still the calling object

A "Perfect" Equals Method

- From Cay Horstmann's *Core Java*

```
public boolean equals(Object otherObject)
{       // check if objects identical
        if( this == otherObject)
                return true;
        // must return false if explicit parameter null
        if(otherObject == null)
                return false;
        // if objects not of same type they cannot be equal
        if(getClass() != otherObject.getClass() )
                return false;
        // we know otherObject is a non null Die
        Die other = (Die)otherObject;
        return iMySides == other.iMySides
                && iMyNumber == other.iMyNumber
                && myName.equals( other.myName );
}
```

the instanceof Operator

- `instanceof` is a Java keyword.
- part of a boolean statement

```
public boolean equals(Object otherObj)
{  if otherObj instanceof Die
   {    //now go and cast
        // rest of equals method
   }
}
```

- Should not use instanceof in equals methods.
- `instanceof` has its uses but not in equals because of the contract of the equals method

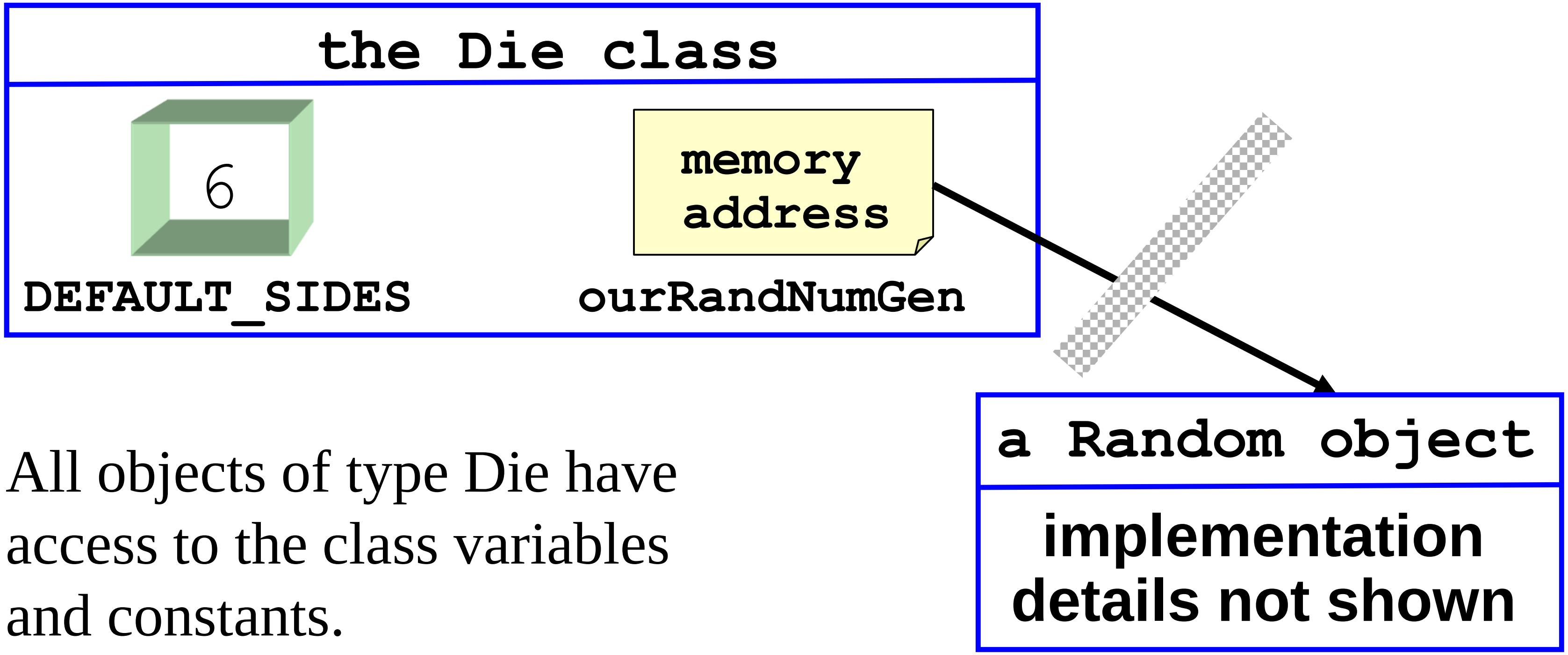
Class Variables and Class Methods

- Sometimes every object of a class does not need its own copy of a variable or constant
- The keyword `static` is used to specify class variables, constants, and methods

```
private static Random ourRandNumGen
    = new Random();
public static final int DEFAULT_SIDES = 6;
```

- The most prevalent use of static is for class constants.
  - if the value can't be changed why should every object have a copy of this non changing value

Class Variables and Constants



All objects of type Die have access to the class variables and constants.

A public class variable or constant may be referred to via the class name.

Syntax for Accessing Class Variables

```
public class UseDieStatic
{   public static void main(String[] args)
    {   System.out.println( "Die.DEFAULT_SIDES "
            + Die.DEFAULT_SIDES );
        // Any attempt to access Die.ourRandNumGen
        // would generate a syntax error

        Die d1 = new Die(10);

        System.out.println( "Die.DEFAULT_SIDES "
            + Die.DEFAULT_SIDES );
        System.out.println( "d1.DEFAULT_SIDES "
            + d1.DEFAULT_SIDES );

        // regardless of the number of Die objects in
        // existence, there is only one copy of DEFAULT_SIDES
        // in the Die class

    } // end of main method
} // end of UseDieStatic class
```

Static Methods

- `static` has a somewhat different meaning when used in a method declaration
- static methods may not manipulate any instance variables
- in non static methods, some object invokes the method

```
d3.roll();
```

- the object that makes the method call is an implicit parameter to the method

Static Methods Continued

- Since there is no implicit object parameter sent to the static method it does not have access to a copy of any objects instance variables
  - unless of course that object is sent as an explicit parameter
- Static methods are normally utility methods or used to manipulate static variables (class variables)
- The Math and System classes are nothing but static methods

static and this

- Why does this work (added to Die class)

```
public class Die
{
      public void outputSelf()
      { System.out.println( this );
      }
}
```

- but this doesn't?

```
public class StaticThis
{
      public static void main(String[] args)
      { System.out.println( this );
      }
}
```

Topic 4
Inheritance

"Question: What is the object oriented way of getting rich?

Answer: Inheritance."

Features of OO Programming

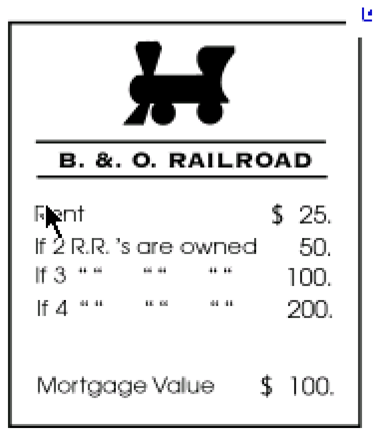
- Encapsulation
  - abstraction, creating new data types
  - information hiding
  - breaking problem up based on data types
- Inheritance
  - code reuse
  - specialization
  - "New code using old code."

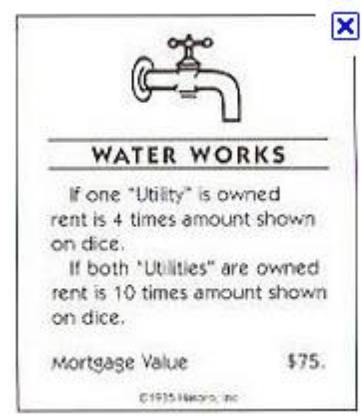
Encapsulation

- Create a program to allow people to play the game Monopoly
  – Create classes for money, dice, players, the bank, the board, chance cards, community chest cards, pieces, etc.
- Some classes use other classes. Are *clients*
  – the board *consists of* spaces
  – a player *has* properties they own
  – a piece *has* a position
- Also referred to as *composition*

Inheritance

- Another kind of relationship exists between things in the world and data types in programs
- There are properties in Monopoly
  - a street *is a kind* of property
  - a railroad *is a kind* of property
  - a utility *is a kind* of property





Inheritance

- In Monopoly there is the concept of a Property
- All properties have some common traits
  - they have a name
  - they have a position on the board
  - they can be owned by players
  - they have a purchase price
- *But* some things are different for each of the three kinds of property
  - determine rent when another player lands on the Property, only Streets can have houses

What to Do?

- If we have a separate class for Street, Railroad, and Utility there is going to be a lot of code copied
  - hard to maintain
  - an *anti-pattern*
- Inheritance is a programming feature to allow data types to build on pre-existing data types without repeating code

Mechanics of Inheritance

1. extends keyword
2. inheritance of instance methods
3. inheritance of instance variables
4. object initialization and constructors
5. calling a parent constructor with **`super()`**
6. overriding methods
7. partial overriding, **`super.parentMethod()`**
8. inheritance requirement in Java
9. the **`Object`** class
10. inheritance hierarchies

Inheritance in Java

- Java is designed to encourage object oriented programming
- all classes, except one, **must** inherit from exactly one other class
- The `Object` class is the *cosmic super class*
  - The `Object` class does not inherit from any other class
  - The `Object` class has several important methods: `toString`, `equals`, `hashCode`, `clone`, `getClass`
- implications:
  - all classes are descendants of `Object`
  - all classes and thus all objects have a `toString`, `equals`, `hashCode`, `clone`, and `getClass` method
    - `toString`, `equals`, `hashCode`, `clone` normally overridden

Nomenclature of Inheritance

- In Java the `extends` keyword is used in the class header to specify which preexisting class a new class is inheriting from

```
public class Student extends Person
```

- Person is said to be
  - the parent class of Student
  - the super class of Student
  - the base class of Student
  - an ancestor of Student
- Student is said to be
  - a child class of Person
  - a sub class of Person
  - a derived class of Person
  - a descendant of Person

Clicker 1

What is the primary reason for using inheritance when programming?

A. To make a program more complicated
B. To copy and paste code between classes
C. To reuse pre-existing code
D. To hide implementation details of a class
E. To ensure pre conditions of methods are met.

Clicker 2

What is output when the `main` method is run?

```
public class Foo {
    public static void main(String[] args) {
        Foo f1 = new Foo();
        System.out.println(f1.toString());
    }
}
```

A. `0`

B. `null`

C. Unknown until code is actually run.

D. No output due to a syntax error.

E. No output due to a runtime error.

Overriding methods

- any method that is not `final` may be overridden by a descendant class
- same signature as method in ancestor
- may not reduce visibility
- may use the original method if simply want to add more behavior to existing
  - super.originalMethod()

Constructors

- Constructors handle initialization of objects
- When creating an object with one or more ancestors (every type except Object) a chain of constructor calls takes place
- The reserved word `super` may be used in a constructor to call a one of the parent's constructors
  - must be first line of constructor
- if no parent constructor is explicitly called the default, 0 parameter constructor of the parent is called
  - if no default constructor exists a syntax error results
- If a parent constructor is called another constructor in the same class may no be called
  - no `super();this();` allowed. One or the other, not both
  - good place for an initialization method

The Keyword `super`

- `super` is used to access something (any protected or public field or method) from the super class that has been overridden
- Rectangle's `toString` makes use of the `toString` in ClosedShape my calling `super.toString()`
- without the super calling `toString` would result in infinite recursive calls
- Java does not allow nested supers

  `super.super.toString()`

  results in a syntax error even though technically this refers to a valid method, `Object`'s toString
- Rectangle *partially* overrides ClosedShapes `toString`

Creating a SortedIntList - A Cautionary Tale of Inheritance

A New Class

- Assume we want to have a list of ints, but that the ints must always be maintained in ascending order

  `[-7, 12, 37, 212, 212, 313, 313, 500]`

  `sortedList.get(0)` returns the min

  `sortedList.get(list.size() - 1)`
  returns the max

Implementing `SortedIntList`

- Do we have to write a whole new class?
- Assume we have an `IntList` class.
- Clicker 3 - Which of the following methods have to be changed?

```
A. add(int value)
B. int get(int location)
C. String toString()
D. int remove(int location)
E. More than one of A - D.
```

Overriding the `add` Method

- First attempt
- Problem?
- solving with insert method
  – double edged sort
- solving with `protected`
  – What `protected` really means

Clicker 4

```
public class IntList {
   private int size
   private int[] con
}
public class SortedIntList extends IntList {
    public SortedIntList() {
        System.out.println(size); // Output?
    }
}
```

A. 0
B. null
C. unknown until code is run
D. no output due to a compile error
E. no output due to a runtime error

Problems

- What about this method?

```
void insert(int location, int val)
```

- What about this method?

```
void insertAll(int location,
    IntList otherList)
```

- `SortedIntList` is **not** a good application of inheritance given **all** the behaviors `IntList` provides.

More Example Code

ClosedShape and Rectangle classes

Simple Code Example

- Create a class named Shape
  - what class does Shape inherit from
  - what methods can we call on Shape objects?
  - add instance variables for a position
  - *override* the toString method
- Create a Circle class that extends Shape
  - add instance variable for radius
  - debug and look at contents
  - try to access instance var from Shape
  - constructor calls
  - use of key word *super*

Shape Classes

- Declare a class called `ClosedShape`
  - assume all shapes have x and y coordinates
  - override `Object`'s version of `toString`
- Possible sub classes of `ClosedShape`
  - `Rectangle`
  - `Circle`
  - `Ellipse`
  - `Square`
- Possible hierarchy

`ClosedShape` <- `Rectangle` <- `Square`

A ClosedShape class

```
public class ClosedShape {

    private double myX;
    private double myY;

    public ClosedShape() {
        this(0,0);
    }

    public ClosedShape (double x, double y) {
        myX = x;
        myY = y;
    }

    public String toString() {
        return "x: " + getX() + " y: " + getY();  }

    public double getX(){ return myX; }
    public double getY(){ return myY; }
}
// Other methods not shown
```

A Rectangle Constructor

```
public class Rectangle extends ClosedShape {
   private double myWidth;
   private double myHeight;

   public Rectangle( double x, double y,
          double width, double height ) {
       super(x,y);
       // calls the 2 double constructor in
       // ClosedShape
       myWidth = width;
       myHeight = height;

   }

    // other methods not shown
}
```

A Rectangle Class

```
public class Rectangle extends ClosedShape {
   private double myWidth;
   private double myHeight;

   public Rectangle() {
      this(0, 0);
   }

   public Rectangle(double width, double height) {
      myWidth = width;
      myHeight = height;
   }

   public Rectangle(double x, double y,
               double width, double height) {
       super(x, y);
       myWidth = width;
       myHeight = height;
   }

   public String toString() {
      return super.toString() + " width " + myWidth
         + " height " + myHeight;
   }
}
```

Initialization method

```
public class Rectangle extends ClosedShape {
   private double myWidth;
   private double myHeight;

   public Rectangle() {
      init(0, 0);
   }

   public Rectangle(double width, double height) {
       init(width, height);
   }

   public Rectangle(double x, double y,
             double width, double height) {
      super(x, y);
      init(width, height);
   }

   private void init(double width, double height) {
      myWidth = width;
      myHeight = height;
   }
```

Result of Inheritance

Do any of these cause a syntax error?
What is the output?

```
Rectangle r = new Rectangle(1, 2, 3, 4);
ClosedShape s = new CloseShape(2, 3);
System.out.println(s.getX());
System.out.println(s.getY());
System.out.println(s.toString());
System.out.println(r.getX());
System.out.println(r.getY());
System.out.println(r.toString());
System.out.println(r.getWidth());
```

The Real Picture

A Rectangle object

Available methods are all methods from Object, ClosedShape, and Rectangle

Fields from Object class
Instance variables declared in Object
Fields from ClosedShape class
Instance Variables declared in ClosedShape
Fields from Rectangle class
Instance Variables declared in Rectangle

Access Modifiers and Inheritance

- public
  - accessible to all classes
- private
  - accessible only within that class. Hidden from all sub classes.
- protected
  - accessible by classes within the same *package* and all descendant classes
- Instance variables are *typically* private
- protected methods are used to allow descendant classes to modify instance variables in ways other classes can't

Why private Vars and not protected?

- In ***general*** it is good practice to make instance variables private
  - hide them from your descendants
  - if you think descendants will need to access them or modify them provide protected methods to do this
- Why?
- Consider the following example

Required update

```
public class GamePiece {
    private Board myBoard;
    private Position myPos;

    // whenever my position changes I must
    // update the board so it knows about the change

    protected void alterPos(Position newPos) {
        Position oldPos = myPos;
        myPos = newPos;
        myBoard.update(oldPos, myPos);
    }
```

Topic 5 - Polymorphism

"“Inheritance is new code that reuses old code. Polymorphism is old code that reuses new code.” - OOP [Koan]

NO!!!!

Polymorphism

- Another feature of OOP
- “having many forms”
- object variables in Java are *polymorphic*
- object variables can refer to objects of their declared type AND any objects that are descendants of the declared type

```
Property p = new Property();
p = new Railroad(); // legal!
p = new Utility(); //legal!
p = new Street();
Object obj1; // = what?
```

Data Type

- object variables and objects:
  - *declared type*. Also called the static type.
  - *dynamic type*. What is the actual type of the object the variable is referring to at run time or when a particular statement is executed.
- Method calls are syntactically legal if the method is in the declared type *or any ancestor* of the declared type
- **The actual method that is executed at runtime is based on the dynamic type**
  - dynamic dispatch

Clicker Question 1

Consider the following class declarations:

```
public class BoardSpace
public class Property extends BoardSpace
public class Street extends Property
public class Railroad extends Property
```

Which of the following statements would cause a syntax error? (Assume all classes have a zero argument constructor.)

A. `Object obj = new Railroad();`

B. `Street s = new BoardSpace();`

C. `BoardSpace b = new Street();`

D. `Railroad r = new Street();`

E. More than one of these

Method LookUp

- To determine if a method is legal **the compiler** looks in the class of the declared type
  - if it finds it great, if not go to the super class and look there
  - continue until the method is found, or the Object class is reached and the method was never found. (Compile error)
- To determine which method is actually executed **the run time system** (abstractly):
  - starts with the actual run time class of the object that is calling the method
  - search the class for that method
  - if found, execute it, otherwise go to the super class and keep looking
  - repeat until a version is found
- Is it possible the runtime system won't find a method?

Clicker Question 2

What is output by the code to the right when run?

A. `!!live`

B. `!eggegg`

C. `!egglive`

D. `!!!`

E. Something else

```
public class Animal {
  public String bt(){ return "!"; }
}

public class Mammal extends Animal {
  public String bt(){ return "live"; }
}

public class Platypus extends Mammal {
  public String bt(){ return "egg";}
}

Animal a1 = new Animal();
Animal a2 = new Platypus();
Mammal m1 = new Platypus();
System.out.print( a1.bt() );
System.out.print( a2.bt() );
System.out.print( m1.bt() );
```

Clicker Question 3

What is output by the code to the right when run? Think carefully about the dynamic type.

A. `MeowWoof`

B. `MeowEm`

C. `EmWoof`

D. `EmEm`

E. Something else

```
public class Animal {
    public void show() {
        System.out.print(this.speak());
    }
    public String speak() { return "Em"; }
}

public class Dog extends Animal {
  public String speak() { return "Woof"; }
}

public class Cat extends Animal {
    public void show(int x) {
        System.out.print("Meow");
    }
}

Cat patches = new Cat();
Dog velvet = new Dog();
patches.show();
velvet.show();
```

Why Bother?

- Inheritance allows programs to model relationships in the real world
  - if the program follows the model it may be easier to write
- Inheritance allows code reuse
  - complete programs faster (especially large programs)
- Polymorphism allows code reuse in another way
- Inheritance and polymorphism allow programmers to create *generic algorithms*

Genericity

- One of the goals of OOP is the support of code reuse to allow more efficient program development
- If a algorithm is essentially the same, but the code would vary based on the data type genericity allows only a single version of that code to exist
- in Java, there are 2 ways of doing this
  1. polymorphism and the inheritance requirement
  2. generics

A Generic List Class

Back to `IntList`

- We may find `IntList` useful, but what if we want a List of `String`s? `Rectangle`s? `List`s?
  - What if I am not sure?
- Are the List algorithms different if I am storing `String`s instead of `int`s?
- How can we make a generic List class?

Generic List Class

- required changes
- How does `toString` have to change?
  - why?!?!
  - A good example of why keyword `this` is necessary from `toString`
- What can a `List` hold now?
- How many `List` classes do I need?

Clicker 4

- After altering the data type of the elements to Object in our list class, how many lines of code in the toString method, originally from the IntList class, need to be changed?

A. 0

B. 1

C. 2

D. 3

E. >= 4

Writing an `equals` Method

- How to check if two objects are equal?

```
if(objA == objA)
    // does this work?
```

- Why not this

```
public boolean equals(List other)
```

- Because

```
public void foo(List a, Object b)
    if( a.equals(b) )
        System.out.println( same )
```

  – what if b is really a `List`?

`equals` method

- read the javadoc carefully!
- Must handle `null`
- Parameter must be `Object`
  - otherwise overloading instead of overriding
  - causes
- must handle cases when parameter is not same data type as calling object
  - `instanceof` or `getClass()`
- don't rely on `toString` and then `String`'s equals (efficiency)

the createASet example

```
public Object[] createASet(Object[] items)
{    /*
     pre: items != null, no elements
     of items = null
     post: return an array of Objects
     that represents a set of the elements
     in items. (all duplicates removed)
     */
```

```
{5, 1, 2, 3, 2, 3, 1, 5} -> {5, 1, 2, 3}
```

createASet examples

```
String[] sList = {"Texas", "texas", "Texas",
                  "Texas", "UT", "texas"};
Object[] sSet = createASet(sList);
for(int i = 0; i < sSet.length; i++)
    System.out.println( sSet[i] );

Object[] list = {"Hi", 1, 4, 3.3, true,
                 new ArrayList(), "Hi", 3.3, 4};
Object[] set = createASet(list);
for(int i = 0; i < set.length; i++)
    System.out.println( set[i] );
```

Topic 6
Generic Type Parameters

"Get your data structures correct first, and the rest of the program will write itself."
- *David Jones*

Back to our Array Based List

- Started with a list of ints
- Don't want to have to write a new list class for every data type we want to store in lists
- Moved to an array of `Object`s to store the elements of the list

```
// from array based list
private Object[] con;
```

Using Object

- In Java, all classes inherit from exactly one other class except Object which is at the top of the class hierarchy
  - therefore all classes are descendants of Object
- object variables can refer to objects of their declared type and any descendants
  - polymorphism
- Thus, if the internal storage container is of type Object it can hold anything
  - primitives handled by *wrapping* them in objects. int – Integer, char - Character

Difficulties with Object

- *Creating* generic data structures using the Object data type and polymorphism is relatively straight forward
- Using these generic data structrues leads to some difficulties
  - Casting
  - Type checking
- Code examples on the following slides

Clicker 1

- What is output by the following code?

```
GenericList list = new GenericList(); // 1
Street s = new Street("Boardwalk", 400,
                              Color.BLUE);
list.add(s); // 2
System.out.print(list.get(0).getPrice());// 3
```

A. 400

B. No output due to syntax error at line // 1

C. No output due to syntax error at line // 2

D. No output due to syntax error at line // 3

E. No output due to runtime error.

Code Example - Casting

- Assume a list class

```
GenericList li = new GenericList();
li.add("Hi");
System.out.println(li.get(0).charAt(0));
// previous line has syntax error
// return type of get is Object
// Object does not have a charAt method
// compiler relies on declared type
System.out.println(
      ((String) li.get(0)).charAt(0) );
// must cast to a String
```

Code Example – type checking

```
//pre: all elements of li are Monopoly Properties
public void printPrices(GenericList li) {
     for (int i = 0; i < li.size(); i++) {
          Property temp = (Property) li.get(i);
          System.out.println(temp.getPrice());
     }
}
// what happens if pre condition not met?
```

"Fixing" the Method

```
//pre: all elements of li are Monopoly Properties

public void printPrices(GenericList li) {
     for(int i = 0; i < li.size(); i++) {
          // GACK!!!!
          if (li.get(i) instanceof Property) {
            Property temp = (Property) li.get(i);
            System.out.println(temp.getPrice());
          }
     }
}
```

Clicker 2 - Too Generic?

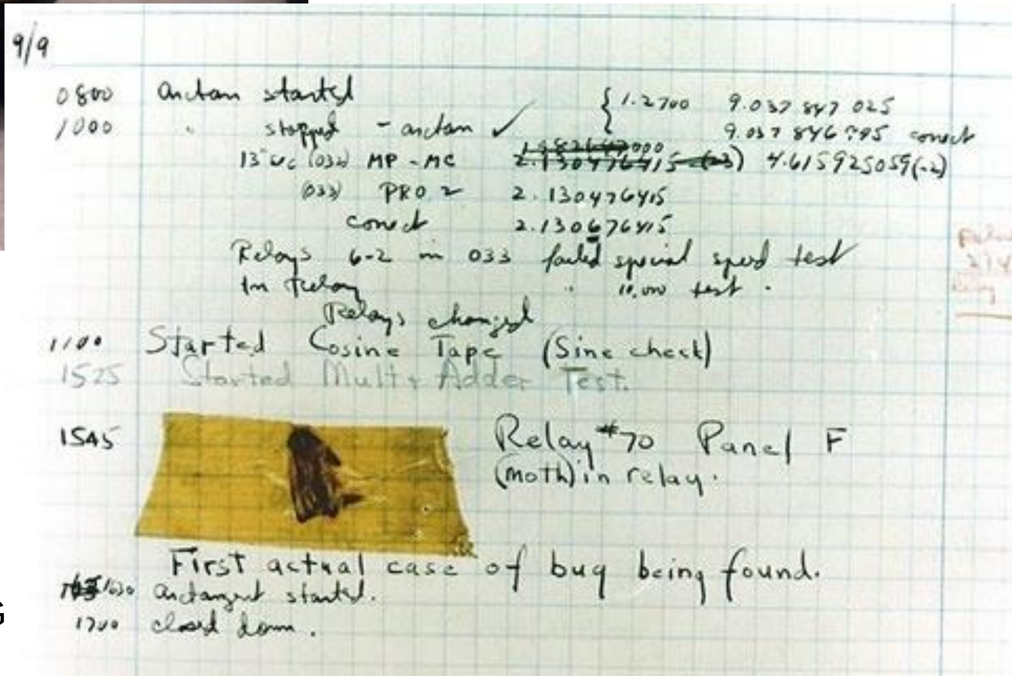
- Does this code compile?

```
GenericList list = new GenericList();
list.add("Olivia");
list.add(Integer.valueOf(12));
list.add(12); // autobox aka autowrap
list.add(new Rectangle(1, 2, 3, 4));
list.add(new GenericList());
```

A. No

B. Yes

Is this a bug or a feature?



Generic Types

- Java has syntax for *parameterized data types*
- Referred to as *Generic Types* in most of the literature
- A traditional parameter *has* a data type and can store various values just like a variable

```
public void foo(int x)
```

- Generic Types are ***like*** parameters, but the data type for the parameter is *data type*
  – like a variable that stores a data type
  – **this is an abstraction**. Actually, all data type info is erased at compile time and replaced with casts and, typically, variables of type Object

Making our Array List Generic

- Data type variables declared in class header

```
public class GenericList<E> {
```

- The `<E>` is the declaration of a data type parameter for the class
  - any legal identifier: `Foo`, `AnyType`, `Element`, `DataTypeThisListStores`
  - Java style guide recommends terse identifiers
- The value E stores will be filled in whenever a programmer creates a new `GenericList`

```
GenericList<String> li =
                new GenericList<>();
```

Modifications to GenericList

- instance variable

  `private E[] myCon;`

- Parameters on
  - add, insert, remove, insertAll, mayber contains?
- Return type on `get(int index)`
- Changes to creation of internal storage container

  `myCon = (E[]) new Object[DEFAULT_SIZE];`

- Equals method. Careful, do not make unsafe cast

Modifications to GenericList

- Careful with the equals method
- Recall type information is actually erased at compile time.
  - At runtime not sure what data type of elements are. (Unless we get into reflection.)
- use of wildcard
- rely on the elements equals methods

Using Generic Types

- Back to Java's ArrayList

```
ArrayList list1 = new ArrayList();
```

  - still allowed, a "raw" ArrayList
  - works just like our first pass at GenericList
  - casting, lack of type safety

Using Generic Types

```
ArrayList<String> list2 =
                new ArrayList<String>();
```

– for list2 `E` stores `String`

```
list2.add( "Isabelle" );
System.out.println(
    list2.get(0).charAt(2) ); //ok
list2.add( new Rectangle() );
// syntax error
```

Parameters and Generic Types

- Old version

```
//pre: all elements of li are Strings
public void printFirstChar(ArrayList li){
```

- New version

```
//pre: none
public void printFirstChar(ArrayList<String> li){
```

- Elsewhere

```
ArrayList<String> list3 = new ArrayList<String>();
printFirstChar( list3 ); // ok
ArrayList<Integer> list4 = new ArrayList<Integer>();
printFirstChar( list4 ); // syntax error
```

Generic Types and Subclasses

```
ArrayList<Shape> list5 =
          new ArrayList<Shape>();
list5.add(new Rectangle());
list5.add(new Square());
list5.add(new Circle());
// all okay
```

- `list5` can store `Shape` objects and any descendants of `Shape`

Topic 7

Interfaces

I once attended a Java user group meeting where James Gosling (one of Java's creators) was the featured speaker. During the memorable Q&A session, someone asked him: "If you could do Java over again, what would you change?" "**I'd leave out classes**," he replied. After the laughter died down, he explained that the real problem wasn't classes per se, but rather implementation inheritance (the extends relationship). Interface inheritance (the implements relationship) is preferable.

- Allen Holub

Clicker 1

- How many sorts do you want to have to write?

```
public static void selSort(double[] data) {
   for (int i = 0; i < data.length; i++) {
      int small = i;
      for(int j = i + 1; j < data.length; j++) {
         if (data[j] < data[small])
            small = j;
      }
      double temp = data[i];
      data[i] = data[small];
      data[small] = temp;
   }
}
```

A. 0
B. 1
C. 2
D. 3
E. >= 4

Why interfaces?

- Interfaces allow the creation of *abstract types*
  - "A set of data values and associated operations that are precisely specified independent of any particular implementation. "
  - multiple implementations allowed
- Interfaces allow a data type to be specified without worrying about the implementation
  - do design first
  - What will this data type do?
  - Don’t worry about implementation until design is done.
  - separation of concerns.
  - allow us to create *generic algorithms*

Interfaces

```
public interface List<E> {
```

- No constructors
- No instance variables
- abstract instance methods

```
public void add(E val);
```

- default instance methods
- static methods
- class constants (prefer enums)

```
public static final int DEFAULT_CAP = 10;
```

- an interface can (but does not have to) extends other interfaces

Implementing Interfaces

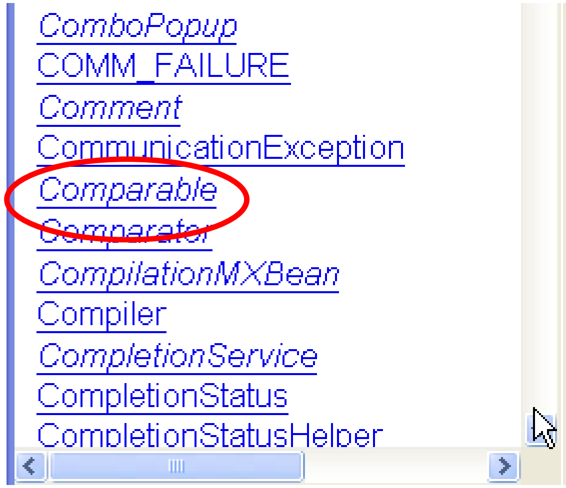
- In Java, a class inherits (extends) exactly one other class, but …
- A class can *implement* as many interfaces as it likes

```
public class ArrayList implements List,
  Serializable
```

- A class that implements an interface must provide implementations of all non default method declared in the interface
  or the class must be abstract
- interfaces can extend other interfaces
  – multiple in fact, unlike Java classes

The Comparable Interface

- The Java Standard Library contains a number of interfaces
  - names are italicized in the class listing
- One of the most important interfaces is the Comparable interface



Comparable Interface

```
package java.lang;

public interface Comparable<T> {
     public int compareTo(T other);
}
```

- compareTo must return
  - an int <0 if the calling object is less than the parameter,
  - 0 if they are equal
  - an int >0 if the calling object is greater than the parameter other
- compareTo should be *consistent with equals* but this isn't required.

Interfaces

- "Use interfaces to ensure a class has methods that ***other*** classes or methods will use." (In other words, clients of your class.)
  - Anthony, Spring 2013
- The other classes or methods may already be written.
- The other methods or classes use interface type for the parameters of methods.
- POLYMORPHISM
  - old code using new code

Clicker Question 2

- What is output by the following code?

```
Comparable c1 = new Comparable();
Comparable c2 = new Comparable();
System.out.println(c1.compareTo(c2));
```

A. A value < 0

B. 0

C. A value > 0

D. Unknown until program run

E. Compile error

Example `compareTo`

- Suppose we have a class to model playing cards
  - Ace of Spades, King of Hearts, Two of Clubs
- each card has a suit and a value, represented by ints
- this version of `compareTo` will compare values first and then break ties with suits

`compareTo` in a `Card` class

```
public class Card implements Comparable<Card> {

      public int compareTo(Card otherCard) {
            return this.rank - other.rank;
      }
      // other methods not shown
}
```

Assume ints for ranks (2, 3, 4, 5, 6,...) and suits (0 is clubs, 1 is diamonds, 2 is hearts, 3 is spades).

Interfaces and Polymorphism

- Interfaces may be used as the data type for object variables
- Can't simply create objects of that type
- Can refer to any objects that implement the interface or descendants
- Assume `Card` implements `Comparable`

```
Card c = new Card();
Comparable comp1 = new Card();
Comparable comp2 = c;
```

Clicker Question 3

- Which of the following lines of code causes a syntax error?

```
Comparable c1; // A
c1 = "Ann"; // B
Comparable c2 = "Kelly"; // C
int x = c2.compareTo(c1); // D
// E No syntax errors.

// what is x after statement?
```

Why Make More Work?

- Why bother implementing an interface such as Comparable
  - objects can use method that expect an interface type
- Example if I implement Comparable:

  Arrays.sort(Object[] a)

  public static void sort(Object[] a)

  All elements in the array must implement the Comparable interface. Furthermore, all elements in the array must be *mutually comparable*
- objects of my type can be stored in data structures that accept Comparables

A List Interface

- What if we wanted to specify the operations for a List, but no implementation?
- Allow for multiple, different implementations.
- Provides a way of creating *abstractions.*
  - a central idea of computer science and programming.
  - specify "what" without specifying "how"
  - "Abstraction is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time. "

List Interface

```
public interface List <E> {
  public void add(E val);
  public int size();
  public E get(int location);
  public void insert(int location, E val);
  public E remove(int location);
}
```

One Sort

```
public static void sort(Comparable[] data) {
   final int LIMIT = data.length - 1;
   for(int i = 0; i < LIMIT; i++) {
      int small = i;
      for(int j = i + 1; j < data.length; j++) {
         int d = data[j].compareTo(data[small]);
         if (d < 0)
           small = j;
      }
      Comparable temp = data[i];
      data[i] = data[small];
      data[small] = temp;
   } // end of i loop
}
```

Topic 8
Iterators

"First things first, but not necessarily in that order "

-Dr. Who

Iterators

- ArrayList is part of the *Java Collections Framework*
- *Collection* is an interface that specifies the basic operations every collection (data structure) shall have
- Some Collections don't have a definite order
  - Sets, Maps, Graphs
- How to access all the items in a Collection with no specified order?

Iterator Interface

- An iterator object is a "one shot" object
  - it is designed to go through all the elements of a Collection once
  - if you want to go through the elements of a Collection again you have to get another iterator object
- Iterators are obtained by calling a method from the Collection

Iterator Iterface Methods

- The Iterator interface 3 methods we will use:

```
boolean hasNext()
```

//returns true if this iteration has more elements

```
E next()
```

//returns the next element in this iteration

//pre: hastNext()

```
void remove()
```

/*Removes from the underlying collection the last element returned by the iterator.

pre: This method can be called only once per call to next. After calling, must call next again before calling remove again.

*/

Clicker 1

- Which of the following produces a syntax error?

```
ArrayList<String> list = new ArrayList<>();
Iterator<String> it1 = new Iterator(); // I
Iterator<String> it2 = new Iterator(list); // II
Iterator<String> it3 = list.iterator(); // III
```

A. `I`

B. `II`

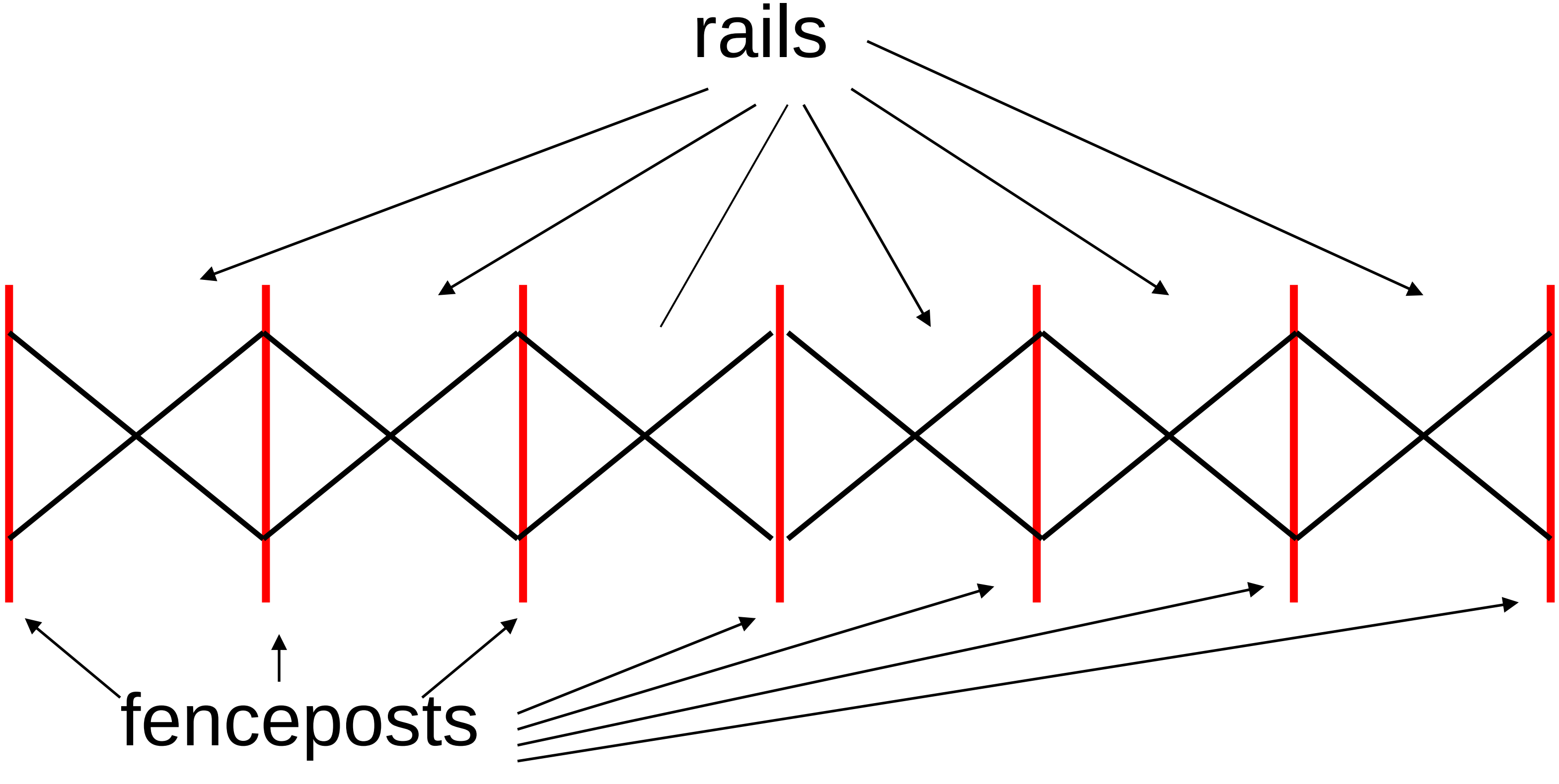
C. `III`

D. `I and II`

E. `II and III`

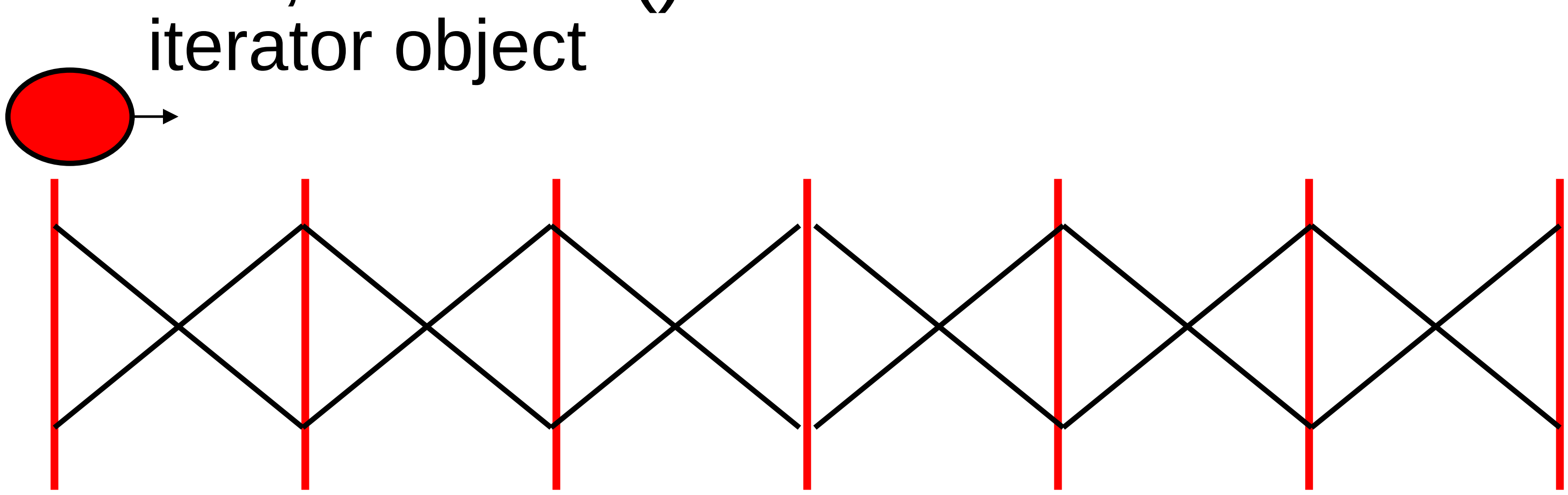
Iterator

- Imagine a fence made up of fence posts and rail sections



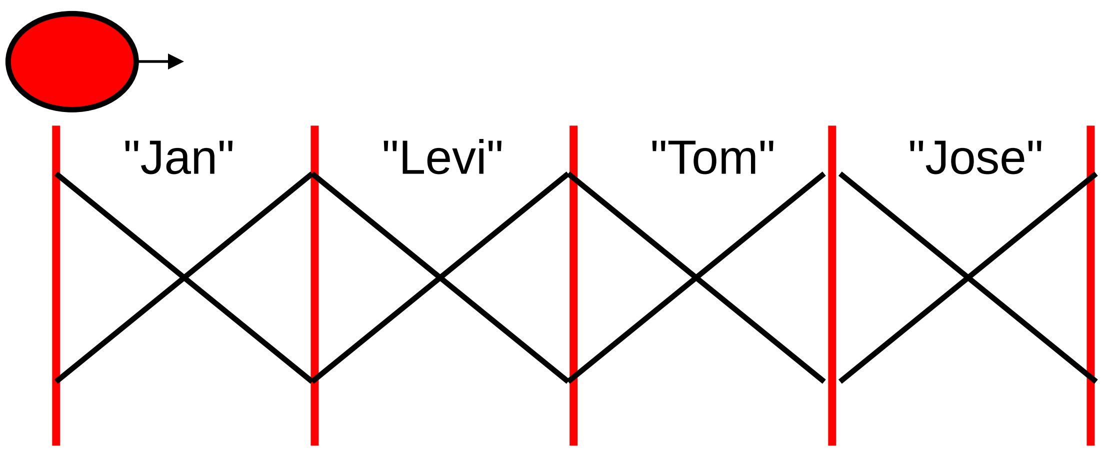
Fence Analogy

- The iterator lives on the fence posts
- The data in the collection are the rails
- Iterator created at the far left post
- As long as a rail exists to the right of the Iterator, hasNext() is true



Fence Analogy

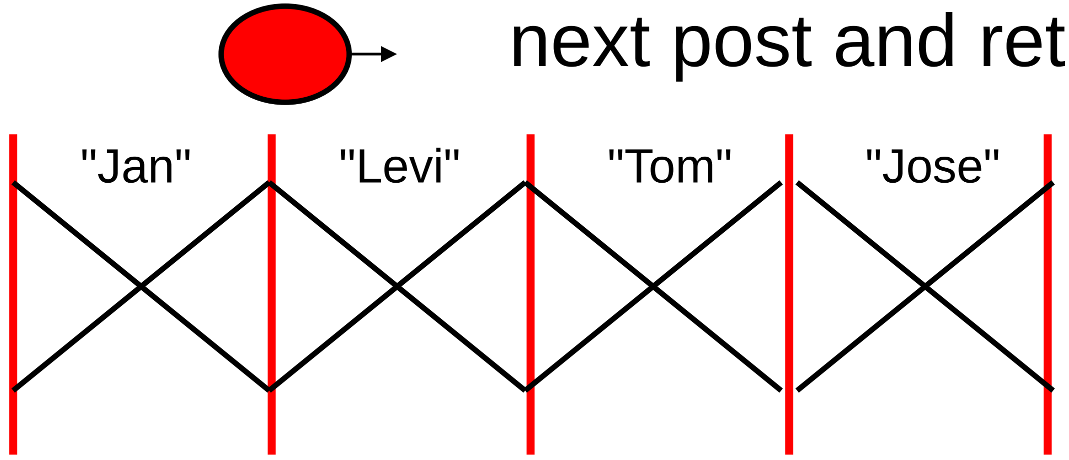
```
ArrayList<String> names = new ArrayList<>();
names.add("Jan");
names.add("Levi");
names.add("Tom");
names.add("Jose");
Iterator<String> it = names.iterator();
int i = 0;
```



Fence Analogy

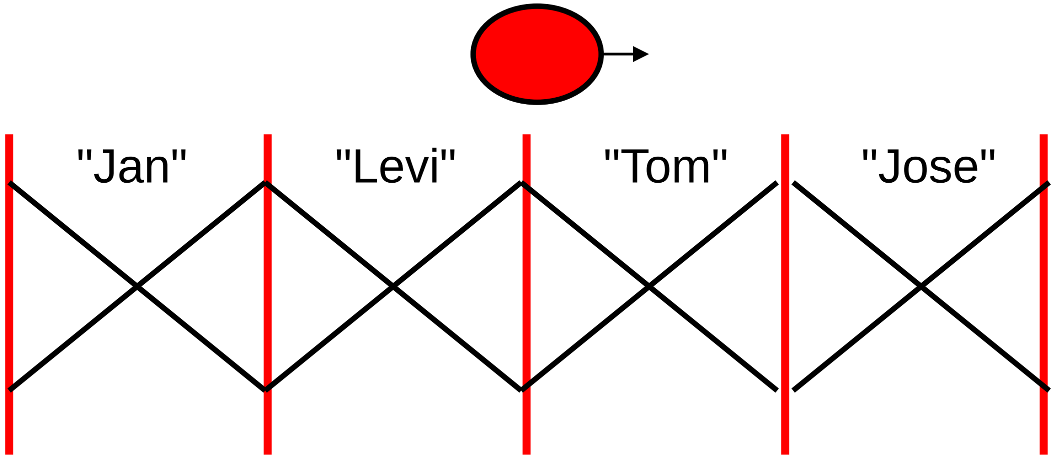
```
while(it.hasNext()) {
    i++;
    System.out.println(it.next());
}
// when i == 1, prints out Jan
```

first call to next moves iterator to next post and returns "Jan"



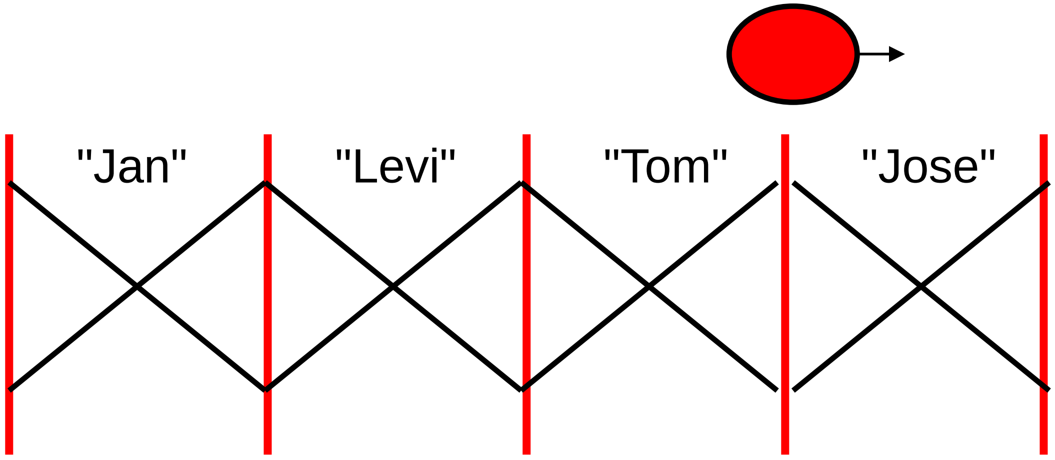
Fence Analogy

```
while(it.hasNext()) {
    i++;
    System.out.println(it.next());
}
// when i == 2, prints out Levi
```



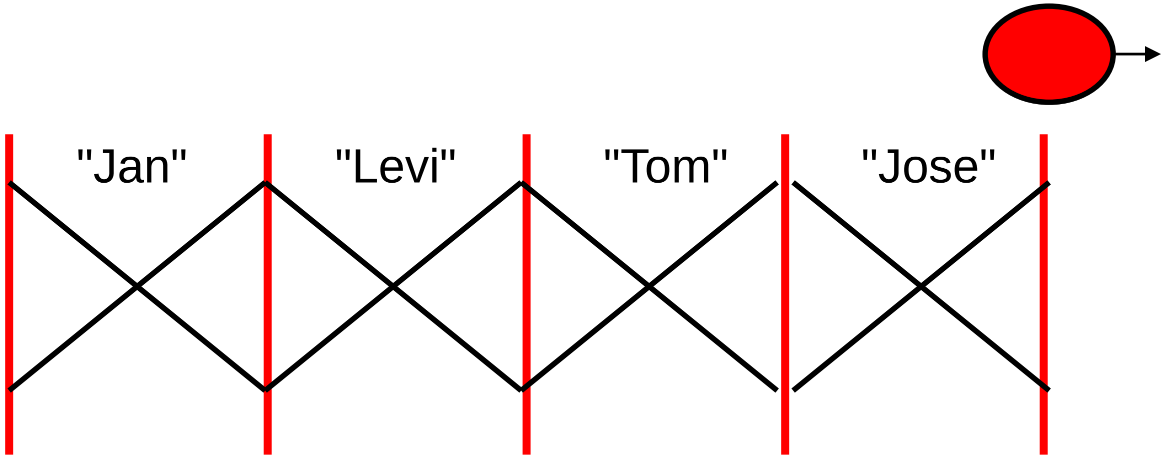
Fence Analogy

```
while(it.hasNext()) {
    i++;
    System.out.println(it.next());
}
// when i == 3, prints out Tom
```



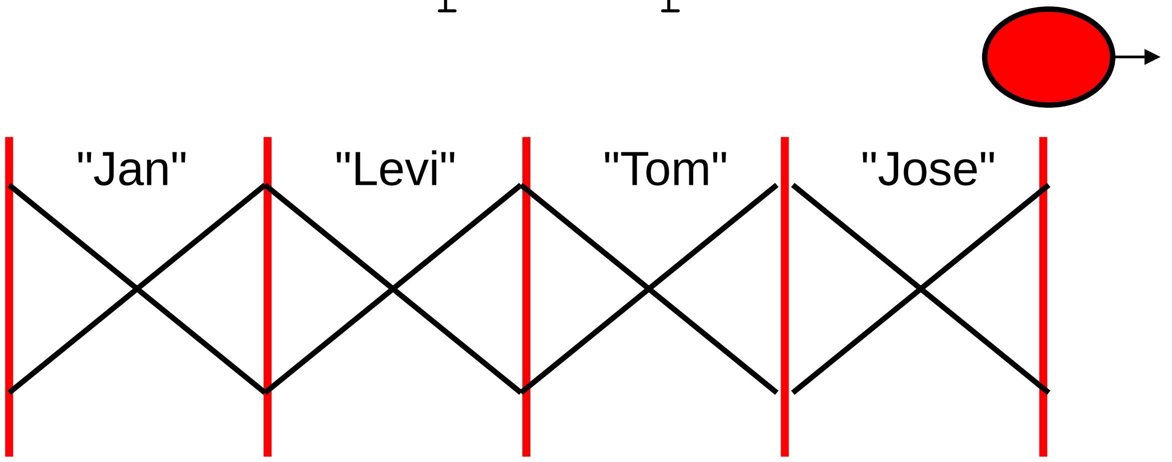
Fence Analogy

```
while(it.hasNext()) {
    i++;
    System.out.println(it.next());
}
// when i == 4, prints out Jose
```



Fence Analogy

```
while(it.hasNext()) {
    i++;
    System.out.println(it.next());
}
// call to hasNext returns false
// while loop stops
```



Typical Iterator Pattern

```
public void printAll(Collection<String> col) {
    Iterator<String> it = col.iterator();
    while (it.hasNext()) {
        String temp = it.next();
        System.out.println(temp);
    }
}
```

OR......

```
for (String temp : col) {
    System.out.println(temp);
}
```

Clicker Question 2

- What is output by the following code?

```
ArrayList<Integer> list = new ArrayList<>();
list.add(3);
list.add(3);
list.add(5);
Iterator<Integer> it = list.iterator();
System.out.print(it.next() + " ");
System.out.print(it.next() + " ");
System.out.print(it.next());
```

A. `3`　　B. `3 5`　　C. `3 3 5`

D. `3 3`　　E. `3 3` then a runtime error

remove method

- An `Iterator` can be used to remove things from the `Collection`
- Can only be called once per call to `next()`

```
public void removeWordsOfLength(int len) {
  Iterator<String> it = myList.iterator
  while(it.hasNext()) {
     String temp = it.next();
     if (temp.length() == len) {
          it.remove();
     }
   }
}
// original list = ["dog", "cat", "hat", "sat"]
// resulting list after removeWordsOfLength(3) ?
```

Clicker 3

```
public void printTarget(Collection<String>
                                          names, int len) {
    Iterator<String> it = names.iterator();
    while(it.hasNext())
        if(it.next().length() == len)
            System.out.println(it.next());
}
```

Given names = ["Jan", "Ivan", "Tom", "George"] and len = 3 what is output by the printTarget method?

A. Jan Ivan Tom George
B. Jan Tom
C. Ivan George
D. No output due to syntax error
E. No output due to runtime error

The Iterable Interface

- A related interface is `Iterable`
- The method of interest to us in the interface:

```
public Iterator<T> iterator()
```

- Why?
- Anything that implements the `Iterable` interface can be used in the for each loop.

```
ArrayList<Integer> list;
//code to create and fill list
int total = 0;
for (int x : list) {
    total += x;
}
```

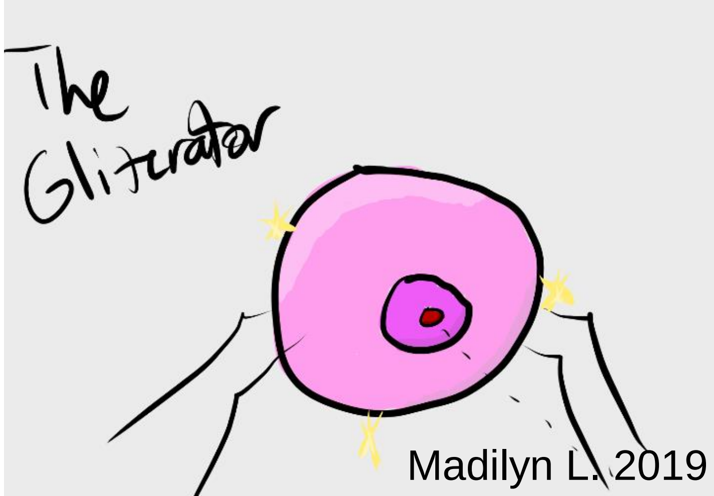
Iterable

- If you simply want to go through all the elements of a Collection (or Iterable thing) use the for each loop
  - hides creation of the Iterator

```
public void printAllOfLength(ArrayList<String> names,
                                             int len){
    //pre: names != null, names only contains Strings
    //post: print out all elements of names equal in
    // length to len
   for (String s : names)
      if (s.length() == len)
            System.out.println(s);
}
```

Implementing an Iterator

- Implement an Iterator for our GenericList class
  - Nested Classes
  - Inner Classes
  - Example of encapsulation
  - checking precondition on remove
  - does our GenricList *need* an Iterator?



Madilyn L. 2019

Comodification

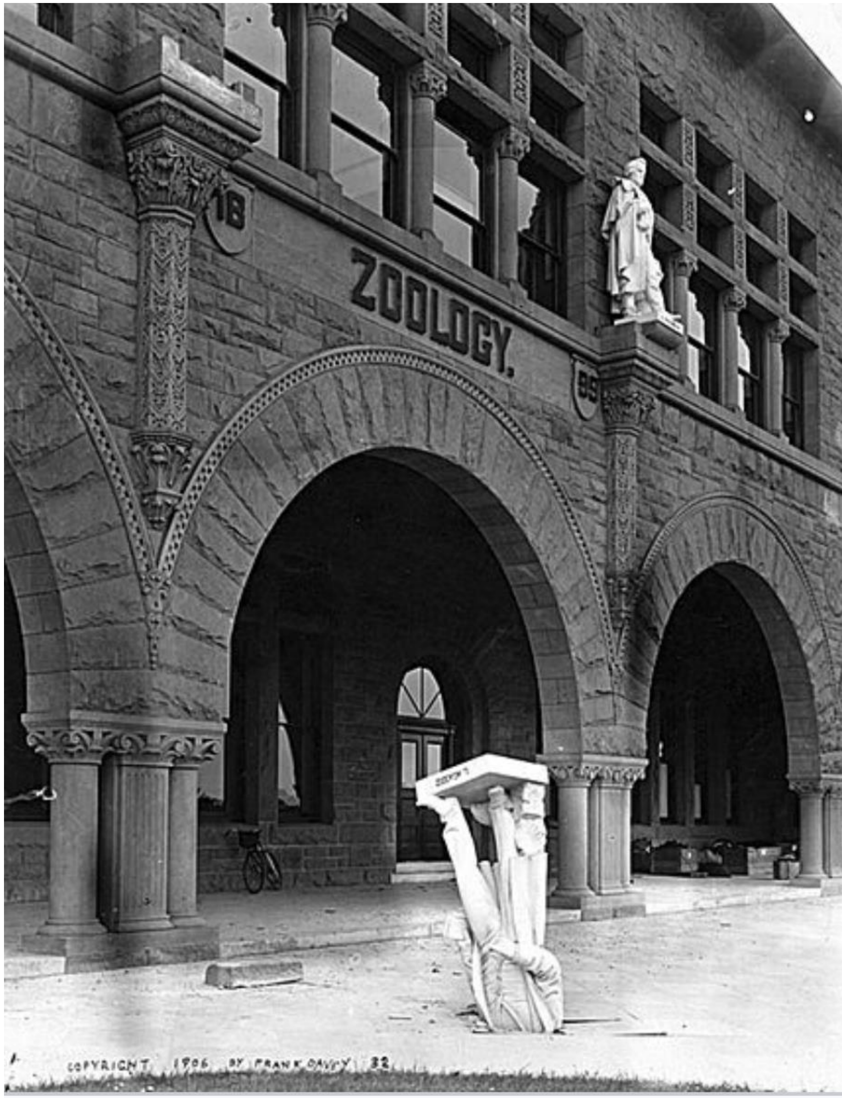
- If a `Collection` (`ArrayList`) is changed while an iteration via an iterator is in progress an Exception will be thrown the next time the `next()` or `remove()` methods are called via the iterator

```
ArrayList<String> names = new ArrayList<>();
names.add("Jan");
Iterator<String> it = names.iterator();
names.add("Andy");
it.next(); // exception occurs here
```

Topic 9
Abstract Classes

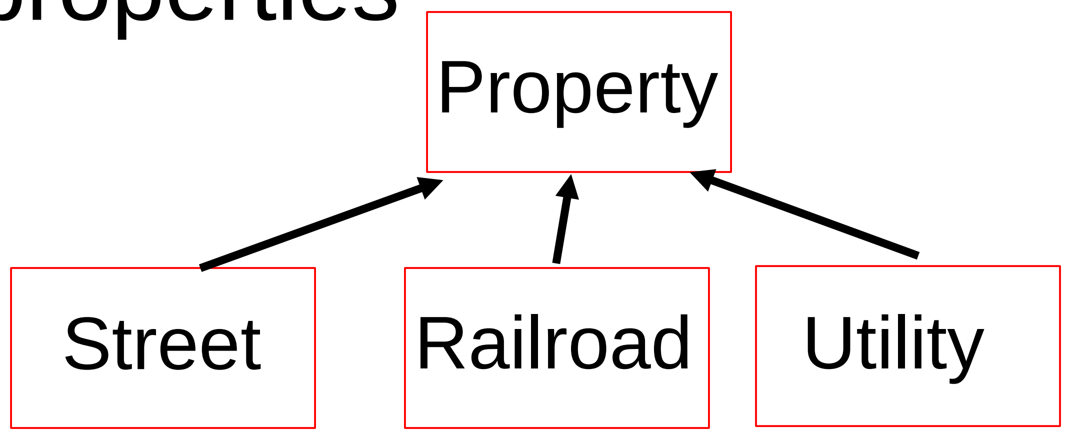
"I prefer Agassiz in the abstract, rather than in the concrete."

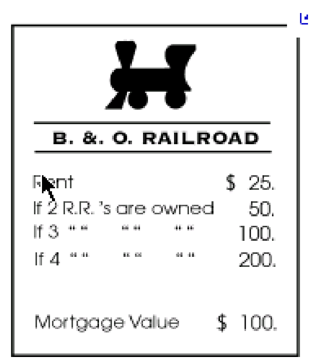
- Statue of Biologist Louis Agassiz that fell from a ledge on the Stanford Quad during the 1906 San Francisco earthquake.

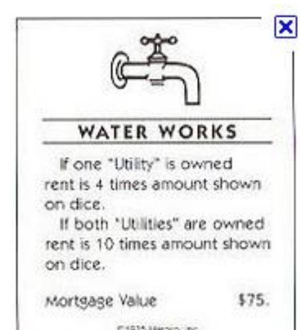


Back to the Monopoly Property Example

- There are properties on a monopoly board
- Railroads, Utilities, and Streets are kinds of properties







A getRent Behavior

- One behavior we want in Property is the getRent method
- problem: How do I get the rent of something that is “just a Property”?

The Property class

```
public class Property {

    private int cost;
    private String name;

    public int getRent() {
        return hmmmmm??????;
    }
```

Doesn't seem like we have enough information to get the rent if all we know is it is a Property.

Potential Solutions

1. Just leave it for the sub classes.
   - Have each sub class define getRent()
2. Define getRent() in Property and simply return -1.
   - Sub classes override the method with more meaningful behavior.

Leave it to the Sub - Classes

```
// no getRent() in Property
// Railroad and Utility DO have getRent() methods

public void printRents(Property[] props) {
      for (Property p : props)
            System.out.println(p.getRent());

}

Property[] props = new Property[2];
props[0] = new Railroad("NP", 200, 1);
props[1] = new Utility("Electric", 150, false);
printRents(props);
```

Clicker 1 - What is result of above code?

A. 200150

B. different every time

C. Syntax error

D. Class Cast Exception

E. Null Pointer Exception

"Fix" by Casting

```
// no getRent() in Property
public void printRents(Property[] props) {
   for (Property p : props) {
      if (p instanceof Railroad)
         System.out.println(((Railroad) p).getRent());
      else if (p instanceof Utility)
         System.out.println(((Utility) p).getRent());
      else if (p instanceof Street)
          System.out.println(((Street) p).getRent())
   } // GACK!!!!
}
Property[] props= new Property[2];
props[0] = new Railroad("NP", 200, 1);
props[1] = new Utility("Electric", 150, false);
printRents( props);
```

What happens as we add more sub classes of `Property`?

What happens if one of the objects is just a `Property`?

Fix with Placeholder Return

```
// getRent() in Property returns -1

public void printRents(Property[] props) {
     for (Property p : props)
          System.out.println(p.getRent());
}

Property[] props= new Property[2];
props[0] = new Railroad("NP", 200, 1);
props[1] = new Utility("Electric", 150, false);
printRents(props);
```

What happens if sub classes don't override getRent()?

Is that a good answer?

A Better Fix

- We know we want to be able to get the rent of objects that are instances of `Property`
- The problem is we don't know how to do that if all we know is it a `Property`
- Make `getRent` an `abstract` method
- Java keyword

Making getRent Abstract

```
public class Property {

    private int cost;
    private String name;

    public abstract int getRent();
    // I know I want it.
    // Just don’t know how, yet…

}
```

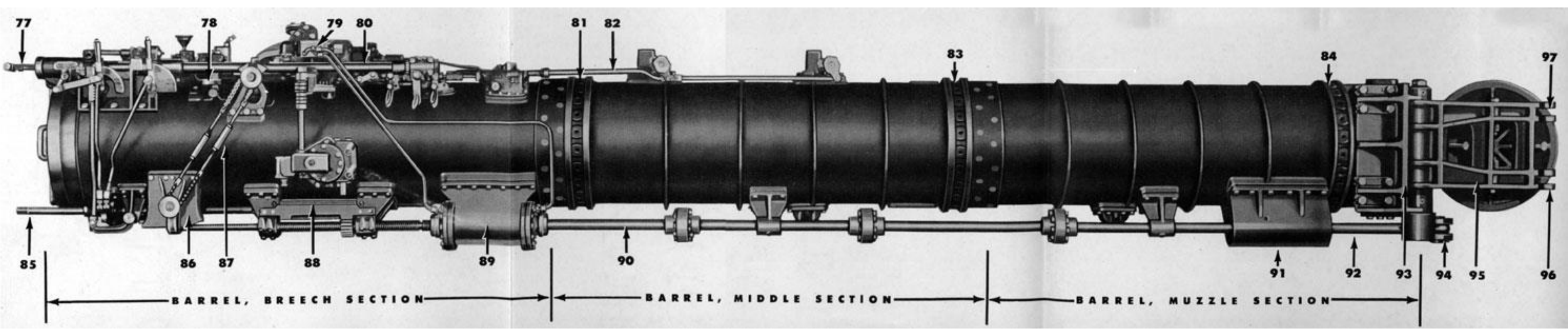
Methods that are declared abstract have no body an undefined behavior.

All non-default methods in a Java interface are abstract.

Problems with Abstract Methods

Given `getRent()` is now an abstract method what is wrong with the following code?

```
Property p = new Property();
System.out.println(p.getRent());
```



If things can go wrong with a tool, provide safeguards to prevent that from happening.

Undefined Behavior = Bad

- Not good to have undefined behaviors
- If a class has 1 or more abstract methods, the class must also be declared abstract.
  - version of `Property` shown would cause a compile error
- Even if a class has zero abstract methods a programmer can still choose to make it abstract
  - if it models some abstract thing
  - is there anything that is just a "Mammal"?

Abstract Classes Safety

1. A class with one or more abstract methods must be declared abstract.
   - Syntax error if not done.
   - Can still decide to make class abstract even if no abstract methods.
2. Objects of an abstract type cannot be instantiated.
   - Just like interfaces
   - Can still declare variables of this type
3. A subclass must implement all inherited abstract methods or be abstract itself.

Abstract Classes

```
public abstract class Property {

    private int cost;
    private String name;

    public abstract double getRent();
    // I know I want it.
    // Just don't know how, yet…

}
// Other methods not shown
```

if a class is abstract the compiler will not allow constructors of that class to be called

```
Property s = new Property(1, 2);
//syntax error
```

Abstract Classes

- In other words you can’t create instances of objects where the lowest or most specific class type is an abstract class
- Prevents having an object with an undefined behavior
- Why would you still want to have constructors in an abstract class?
- Object variables of classes that are abstract types may still be declared

```
Property p; //okay
```

Sub Classes of Abstract Classes

- Classes that extend an abstract class must provided a working version of any and all abstract methods from the parent class
  - or they must be declared to be abstract as well
  - could still decide to keep a class abstract regardless of status of abstract methods

Implementing getRent()

```
public class Railroad extends Property {

   private static int[] rents
          = {25, 50, 100, 200};

   private int numOtherRailroadsOwned;

   public double getRent() {
     return rents[numOtherRailroadsOwned];}

   // other methods not shown
}
```

A Utility Class

```java
public class Utility extends Property {

    private static final int ONE_UTILITY_RENT = 4;
    private static final int TWO_UTILITY_RENT = 10;

    private boolean ownOtherUtility;

    public Utility(String n, int c, boolean other) {
        super(n, c);
    }

    public String toString() {
        return "Utility. own other utility? " + ownOtherUtility;
    }

    public int getRent(int roll) {
        return ownOtherUtility ? roll * TWO_UTILITY_RENT  :
            roll * TWO_UTILITY_RENT;
    }
}
```

Polymorphism in Action

```
// getRent() in Property is abstract

public void printRents(Property[] props) {
    for (Property p : props)
        System.out.println(p.getRent());
}
```

- Add the Street class. What needs to change in printRents method?
- Inheritance is can be described as new code using old code.
- **Koan of Polymorphism: Polymorphism can be described as old code reusing new code.**

Comparable in Property

```
public abstract class Property
              implements Comparable<Property> {
    private int cost;
    private String name;

    public abstract int getRent();

    public int compareTo(Property other) {
        return this.getRent()
               - otherProperty.getRent();
    }
}
```

Back to Lists

- We suggested having a list interface

```
public interface IList<E> extends Iterable<E> {
  public void add(E value);
  public int size();
  public E get(int location);
  public E remove(int location);
  public boolean contains(E value);
  public void addAll(IList<E> other);
  public boolean containsAll(IList<E> other);
}
```

Data Structures

When implementing data structures:

- Specify an interface
- Create an abstract class that is *skeletal implementation* interface
- Create classes that extend the skeletal interface

```
public boolean contains(E val) {
    for (E e : this)
        if val.equals(e)
            return true;
return false
```

Topic 10
Using Maps

"He's off the map!"

-Stan (Mark Ruffalo) *Eternal Sunshine of the Spotless Mind*

Data Structures

- More than arrays and lists
- Write a program to determine the frequency of all the "words" in a file.

Performance using ArrayList

Title	Size (kb)	Total Words	Distinct Words	Time (sec)
small sample	0.6	89	25	0.001
2BR02B	34	5,638	1,975	0.051
Alice in Wonderland	120	29,460	6,017	0.741
Adventures of Sherlock Holmes	581	107,533	15,213	4.144
2008 CIA Factbook	10,030	1,330,100	74,042	173.000

Order?

- Express change of value as factor of previous file

Title	Size	Total Words	Distinct Words	Time
small sample	0.6	89	25	0.001
2BR02B	57x	63x	79x	51x
Alice in Wonderland	3.5x	5.2x	3.0x	14.5x
Adventures of Sherlock Holmes	4.8x	3.7x	2.5x	6.0x
2008 CIA Factbook	17x	12.3x	5x	42x

O(Total Words * Distinct Words) ??

Clicker 1

- Given 10 seconds for the 2008 CIA Factbook with 1,330,100 total words and 74,042 distinct words, how long for 1,000x total words and 100x distinct words?

A. an hour

B. a day

C. a week

D. a month

E. half a year

Why So Slow??

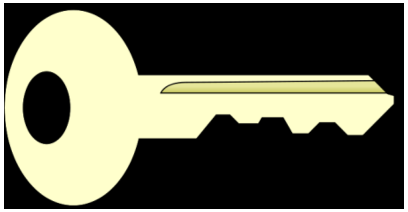
- Write a contains method for an array based list

```
public boolean indexOf(Object o) {
```

A Faster Way - Maps

- Also known as:
  - table, search table, dictionary, associative array, or associative container
- A data structure optimized for a very specific kind of search / access
- In a *map* we access by asking "give me the *value* associated with this *key.*"

Keys and Values



- Dictionary Analogy:
  - The *key* in a dictionary is a word: *foo*
  - The *value* in a dictionary is the definition: *First on the standard list of metasyntactic variables used in syntax examples*
- A key and its associated value form a pair that is stored in a map
- To retrieve a value the key for that value must be supplied
  - A List can be viewed as a Map with integer keys

More on Keys and Values

- Keys must be unique, meaning a given key can only represent one value
  – but one value may be represented by multiple keys
  – like synonyms in the dictionary.
  Example:
  factor: n.See coefficient of X
  – *factor* is a key associated with the same value (definition) as the key *coefficient of X*

Clicker 2

- Is it required that the keys and values of a map be the same data type?

A. No

B. Yes

C. It Depends

Map <String, List<String>>

Movie	Characters
Wizard of Oz	Dorothy, Toto, Scarecrow, Tin Man, Cowardly Lion
Iron Man	Tony Stark, Pepper Potts, Phil Coulson, Obadiah Stane
Pride and Prejudice	Elizabeth Bennet, Jane Bennet, Mr. Darcy, Mr. Bingley
The Avengers	Tony Stark, Pepper Potts, Steve Rogers, Bruce Banner, Phil Coulson
Sense and Sensibility	Elinor Dashwood, Marianne Dashwood, Edward Ferrars, John Willoughby, Colonel Brandon

The Map<K, V> Interface in Java

- `void clear()`
  - Removes all mappings from this map (optional operation).
- `boolean containsKey(Object key)`
  - Returns true if this map contains a mapping for the specified key.
- `boolean containsValue(Object value)`
  - Returns true if this map maps one or more keys to the specified value.
- `Set<K> keySet()`
  - Returns a Set view of the keys contained in this map.

The Map Interface Continued

- `V get(Object key)`
  - Returns the value to which this map maps the specified key. Returns null if key not present.
- `boolean isEmpty()`
  - Returns true if this map contains no key-value mappings.
- `V put(K key, V value)`
  - Associates the specified value with the specified key in this map

The Map Interface Continued

- `V remove(Object key)`
  - Removes the mapping for this key from this map if it is present
- `int size()`
  - Returns the number of key-value mappings in this map.
- `Collection<V> values()`
  - Returns a collection view of the values contained in this map.

Results with HashMap

Title	Size (kb)	Total Words	Distinct Words	Time List	Time Map
small sample	0.6	89	25	0.001	0.0008
2BR02B	34	5,638	1,975	0.051	0.0140
Alice in Wonderland	120	29,460	6,017	0.741	0.0720
Adventures of Sherlock Holmes	581	107,533	15,213	4.144	0.2500
2008 CIA Factbook	10,030	1,330,100	74,042	173.000	4.0000

Order?

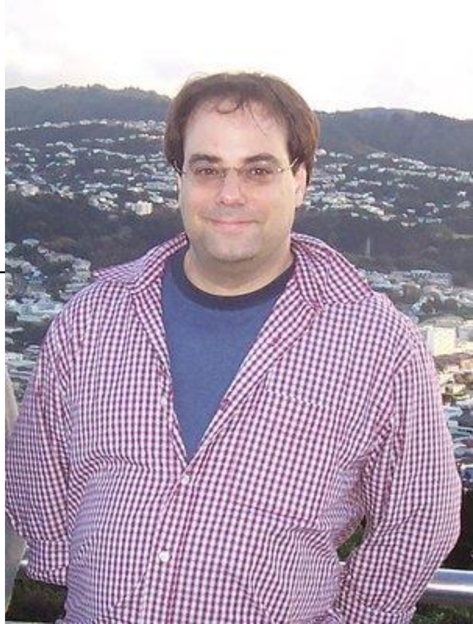
Title	Size	Total Words	Distinct Words	Time List	Time Map
small sample	0.6	89	25	0.001	0.0008
2BR02B	57x	63x	79x	51x	18x
Alice in Wonderland	3.5x	5.2x	3.0x	14.5x	5x
Adventures of Sherlock Holmes	4.8x	3.7x	2.5x	5.6x	3.5x
2008 CIA Factbook	17x	12.3x	5x	42x	16x

O(Total Words)?

Topic 11
Linked Lists

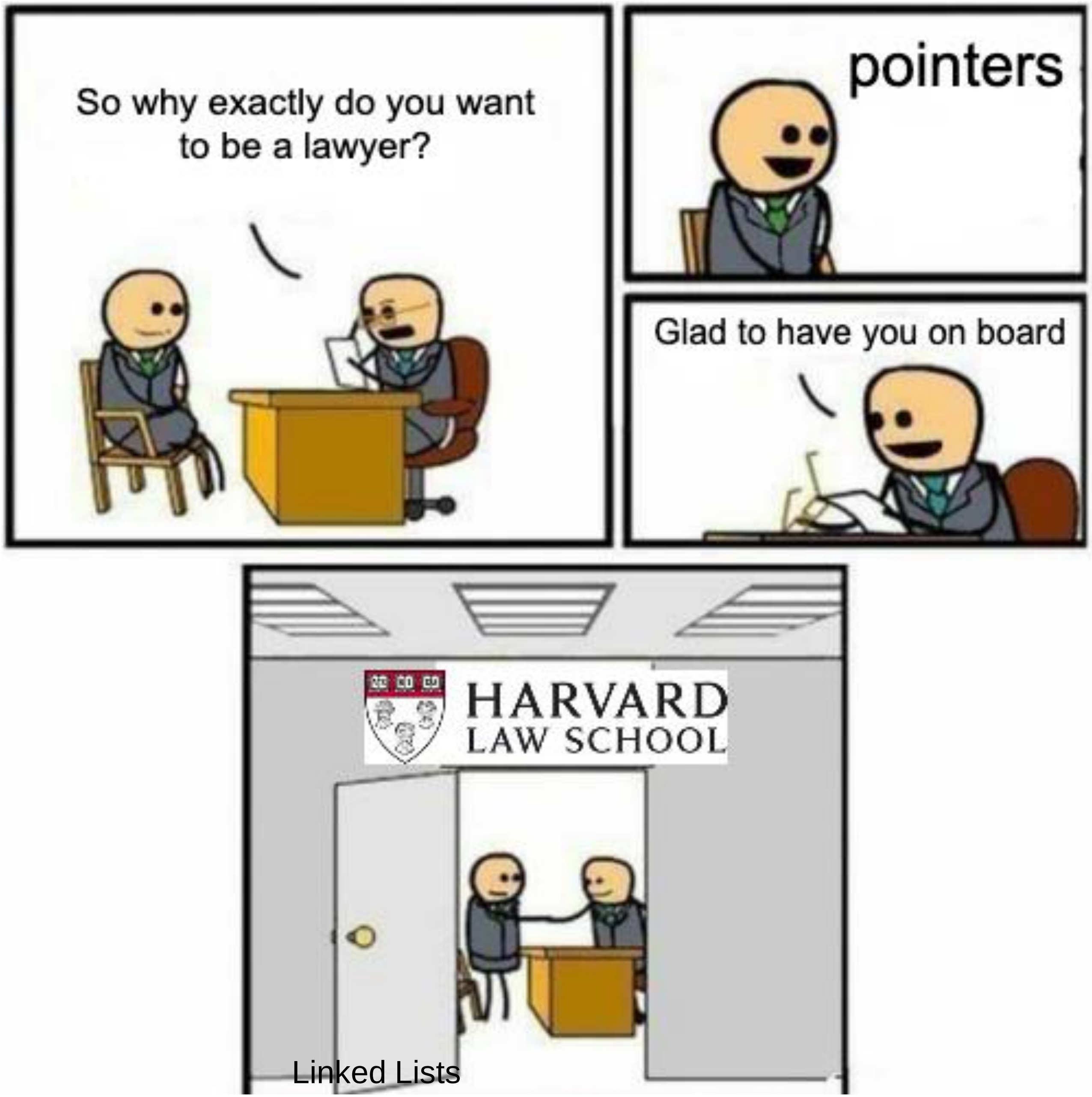
"All the kids who did great in high school writing pong games in BASIC for their Apple II would get to college, take CompSci 101, a data structures course, and when they hit the pointers business their brains would just totally explode, and the next thing you knew, they were majoring in Political Science because law school seemed like a better idea."

-Joel Spolsky



Thanks to Don Slater of CMU for use of his slides.

Sam G. - Fall 2023



Clicker 1

- What is output by the following code?

```
ArrayList<Integer> a1 = new ArrayList<>();
ArrayList<Integer> a2 = new ArrayList<>();
a1.add(12);
a2.add(12);
System.out.println(a1 == a2);
```

A. `false`

B. `true`

C. No output due to syntax error

D. No output due to runtime error

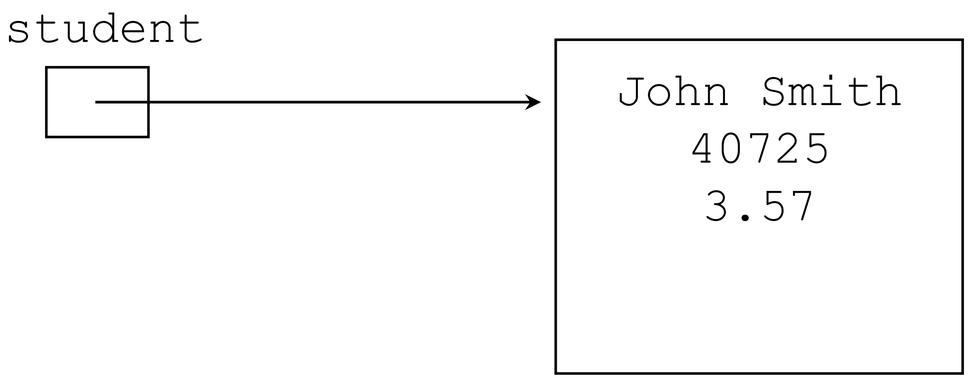
E. Varies from one run of the program to the next

Dynamic Data Structures

- *Dynamic* data structures
  - They grow and shrink one element at a time, normally without some of the inefficiencies of arrays
  - as opposed to a static container such as an array
- Big O of Array Manipulations
  - Access the kth element
  - Add or delete an element in the middle of the array while maintaining relative order
  - adding element at the end of array? space avail? no space avail?
  - add element at beginning of an array

Object References

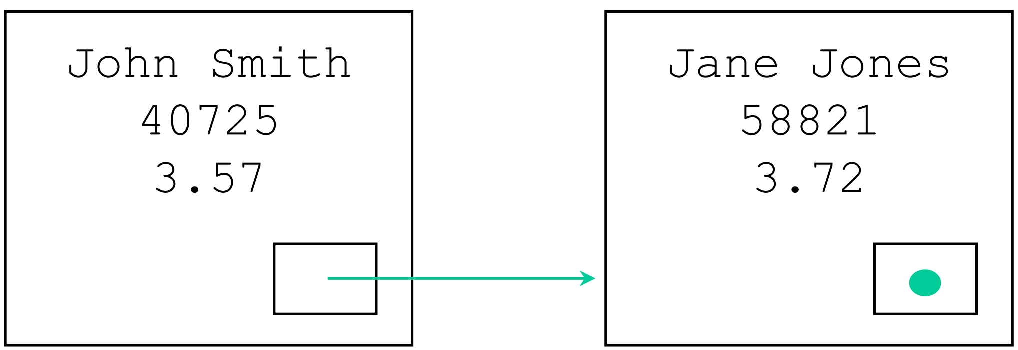
- Recall that an *object reference* is a variable that stores the address of an object
- A reference can also be called a *pointer*
- They are often depicted graphically:



References as Links

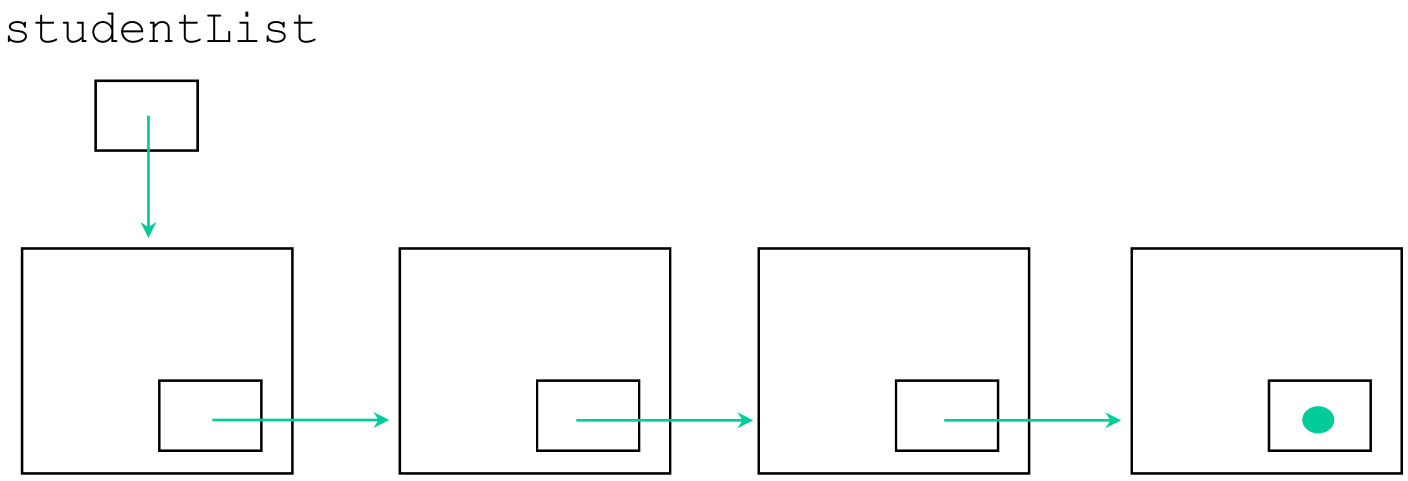
- Object references can be used to create *links* between objects

- Suppose a `Student` class contained a reference to another `Student` object



References as Links

- References can be used to create a variety of linked structures, such as a *linked list*:



Linked Lists

- A **linear** collection of self-referential objects, typically called nodes, connected by other links
  - linear: **for every node in the list, there is one and only one node that precedes it (except for possibly the first node, which may have no predecessor,) and there is one and only one node that succeeds it, (except for possibly the last node, which may have no successor)**
  - self-referential: **a node that has the ability to refer to another node of the same type, or even to refer to itself**
  - node: **contains data of any type, including a reference to another node of the same data type, or to nodes of different data types**
  - **Usually a list will have a beginning and an end; the first element in the list is accessed by a reference to that class, and the last node in the list will have a reference that is set to `null`**

Advantages of linked lists

- Linked lists are dynamic, they can grow or shrink as necessary

- Linked lists are *non-contiguous;* the logical sequence of items in the structure is decoupled from any physical ordering in memory

Nodes and Lists

- A different way of implementing a list
- Each element of a Linked List is a separate Node object.
- Each Node tracks a single piece of data plus a reference (pointer) to the next
- Create a new Node very time we add something to the List
- Remove nodes when item removed from list and allow garbage collector to reclaim that memory

A Node Class

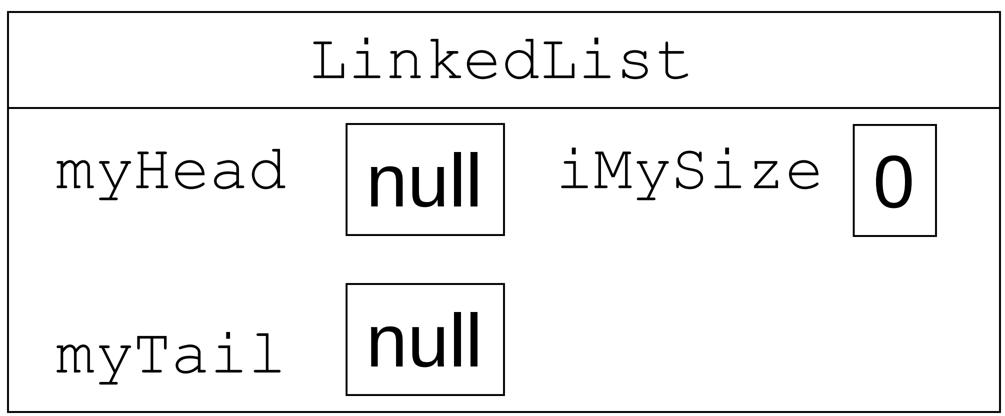
```
public class Node<E> {
        private E myData;
        private Node<E> myNext;

        public Node()
        {       myData = null; myNext = null;      }

        public Node(E data, Node<E> next)
        {       myData = data; myNext = next; }

        public E getData()
        {       return myData;        }

        public Node<E> getNext()
        {       return myNext;        }

        public void setData(E data)
        {       myData = data;        }

        public void setNext(Node<E> next)
        {       myNext = next;        }
}
```

One Implementation of a Linked List

- The Nodes show on the previous slide are *singly linked*
  – a node refers only to the next node in the structure
  – it is also possible to have *doubly linked* nodes.
  – The node has a reference to the next node in the structure and the *previous* node in the structure as well
- How is the end of the list indicated
  – myNext = null for last node
  – a separate dummy node class / object

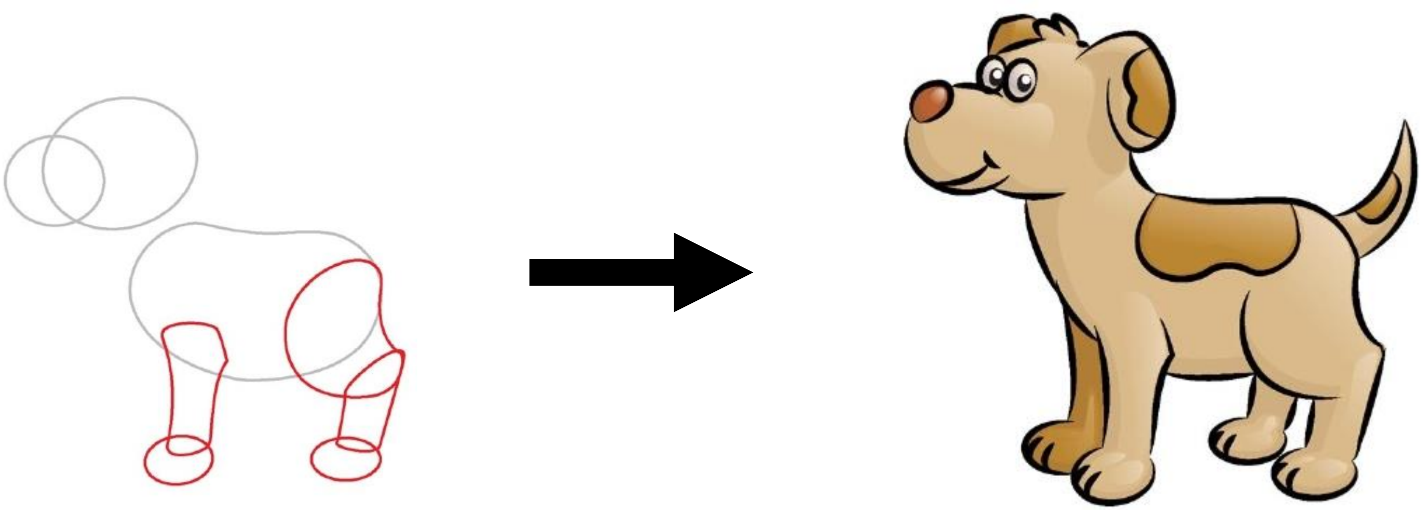
A Linked List Implementation

```
public class LinkedList<E> implements IList<E>
      private Node<E> head;
      private Node<E> tail;
      private int size;

      public LinkedList(){
            head = null;
            tail = null;
            size = 0;
      }
}
LinkedList<String> list = new LinkedList<String>();
```



Writing Methods

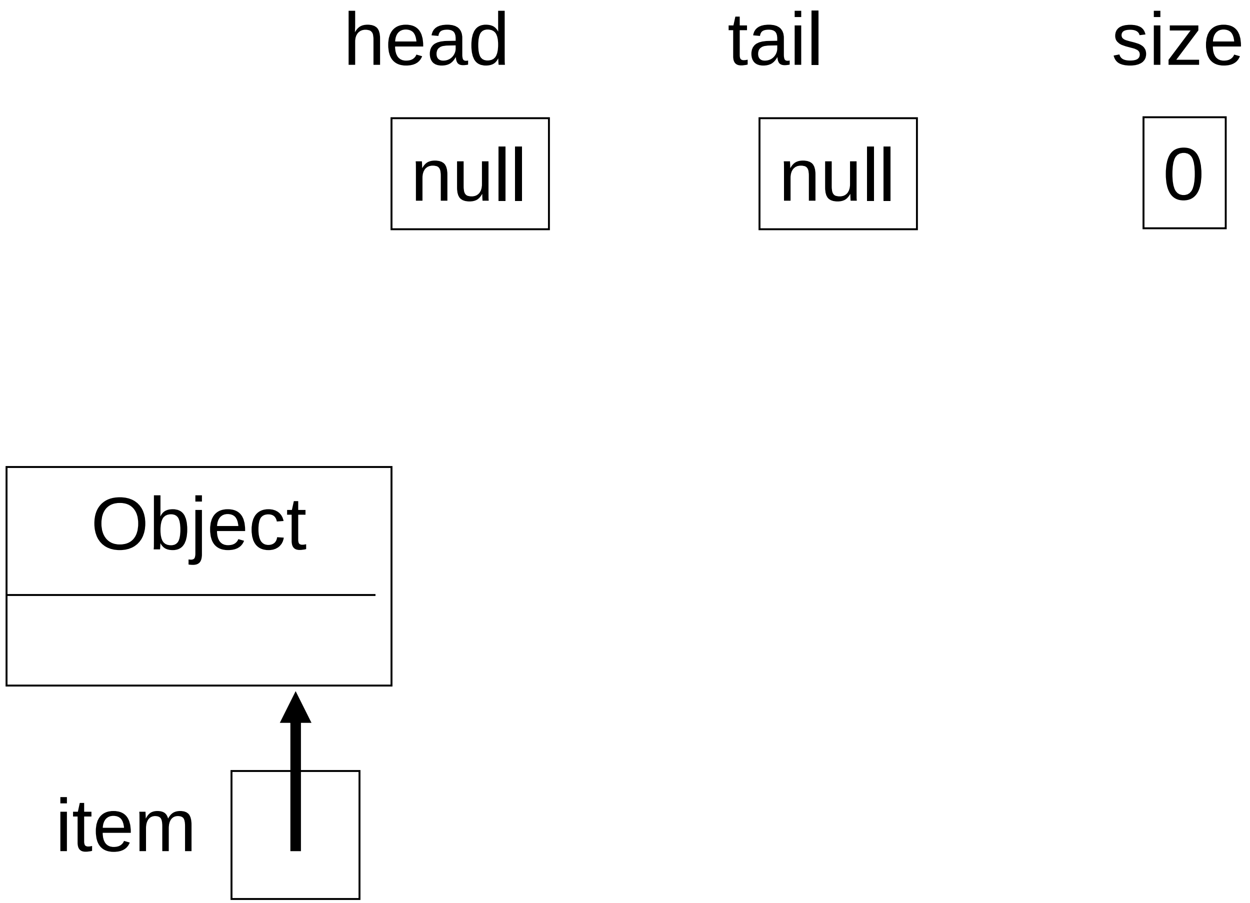
- When trying to code methods for Linked Lists ***draw pictures!***
  - If you don't draw pictures of what you are trying to do it is very easy to make mistakes!



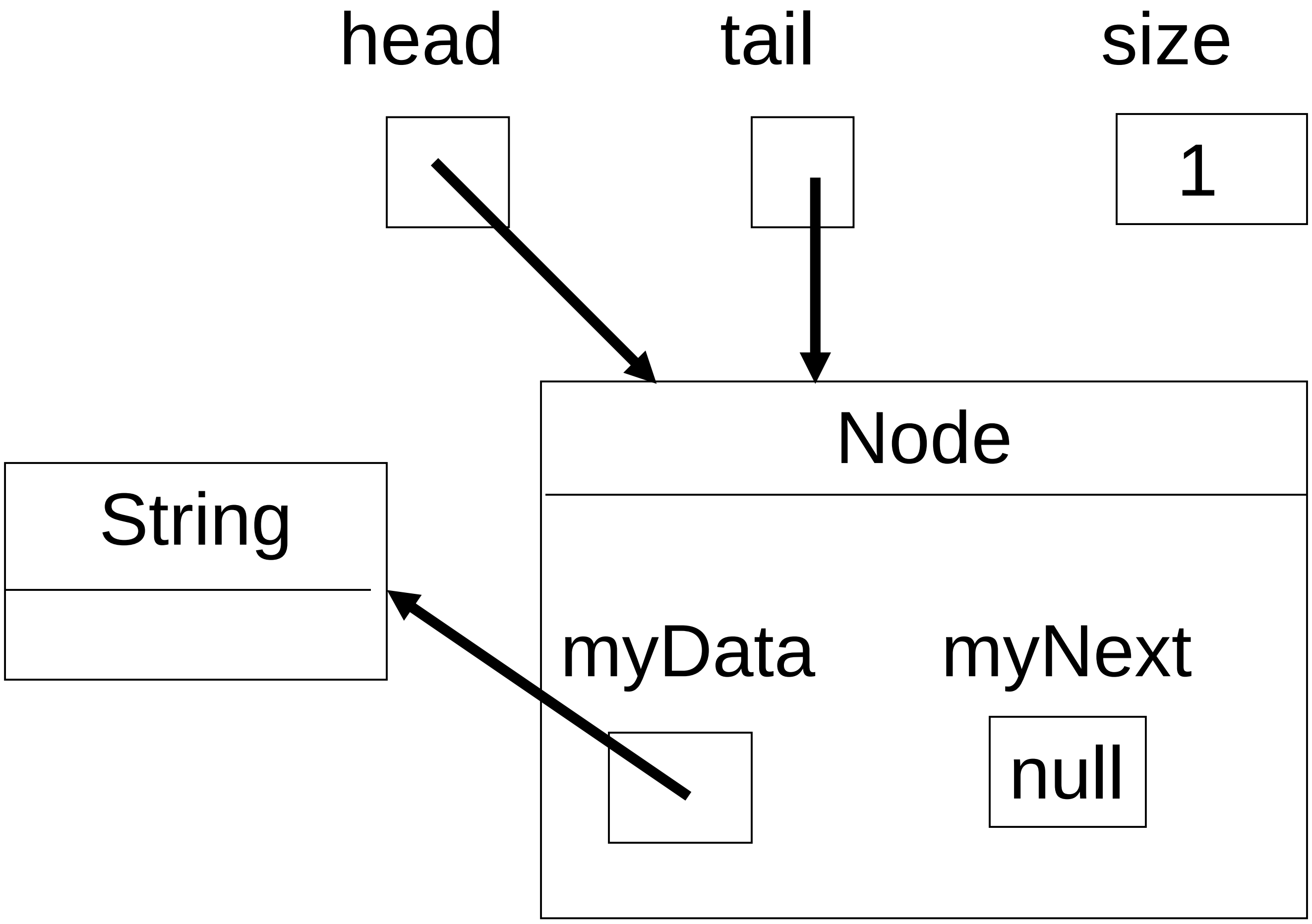
add method

- add to the end of list
- special case if empty
- steps on following slides
- public void add(E obj)

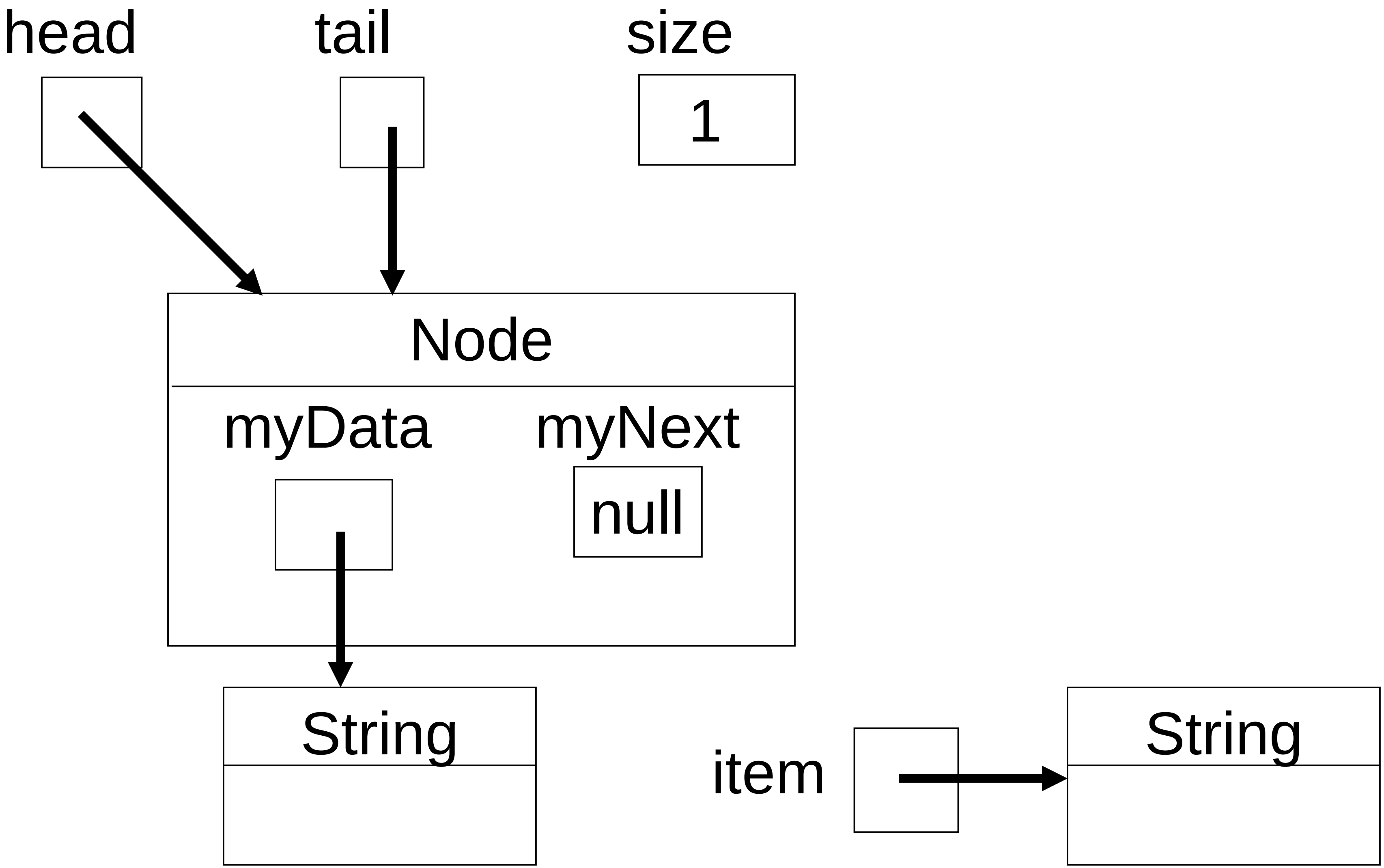
Add Element - List Empty (Before)



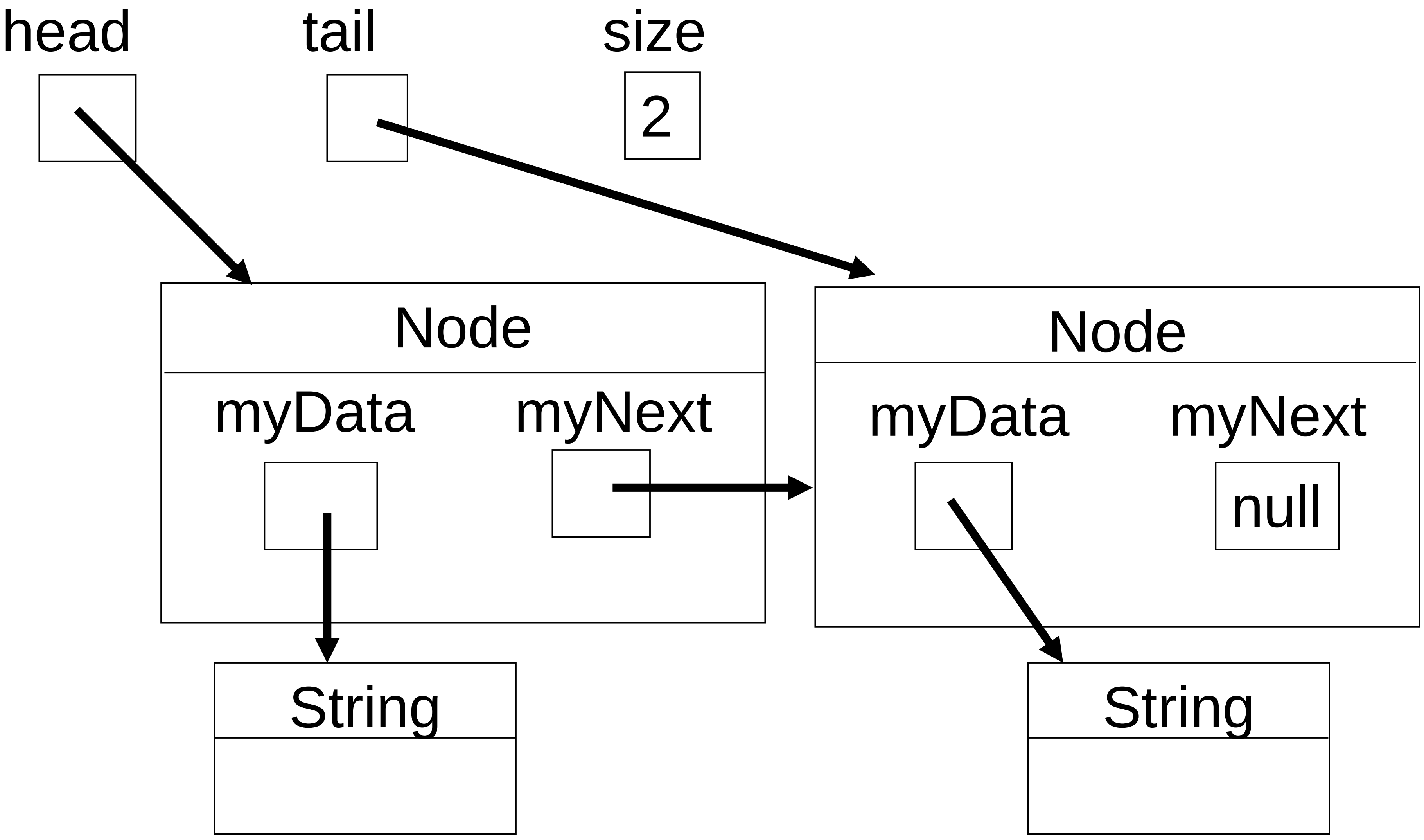
Add Element - List Empty (After)



Add Element - List Not Empty (Before)



Add Element - List Not Empty (After)



Code for default add

- public void add(E obj)

Clicker 2

- What is the worst case Big O for adding to the end of an array based list and our LinkedList314 class? The lists already contain N items.

Array based	Linked
A. O(1)	O(1)
B. O(N)	O(N)
C. O(logN)	O(1)
D. O(1)	O(N)
E. O(N)	O(1)

Contains method

- Implement a contains method for our Linked List class

public boolean contains(E val) // val != null

Code for addFront

- add to front of list
- public void addFront(E obj)
- How does this compare to adding at the front of an array based list?

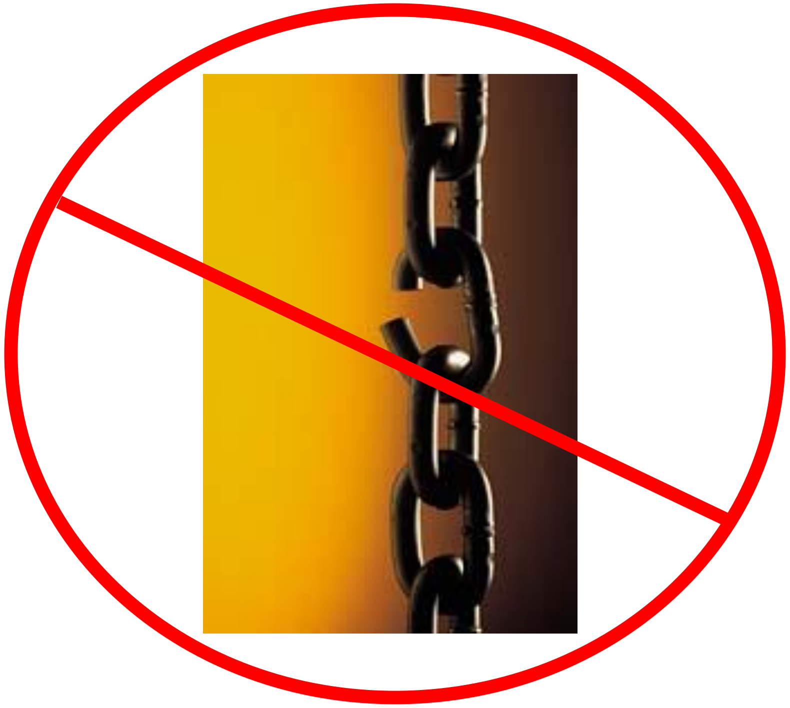
Clicker 3

- What is the Big O for adding to the front of an array based list and a linked list? The lists already contain N items.

	Array based	Linked
A.	O(1)	O(1)
B.	O(N)	O(1)
C.	O(logN)	O(1)
D.	O(1)	O(N)
E.	O(N)	O(N)

Code for Insert

- public void insert(int pos, E obj)
- Must be careful not to break the chain!
- Where do we need to go?
- Special cases?



Clicker 4

- What is the Big O for inserting an element into the middle of an array based list and into the middle of a linked list? Each list already contains N items.

Array based	Linked
A. O(1)	O(1)
B. O(1)	O(N)
C. O(N)	O(1)
D. O(N)	O(N)
E. O(N)	O(logN)

Clicker Question 5

- What is the Big O for getting an element based on position from an array based list and from a linked list? Each list contains N items. In other words `E get(int pos)`

Array based	Linked
A. O(1)	O(1)
B. O(1)	O(N)
C. O(N)	O(1)
D. O(logN)	O(N)
E. O(N)	O(N)

Code for get

- public E get(int pos)
- The downside of Linked Lists

Code for remove

- public E remove(int pos)

Clicker 6

- What is the order to remove the last element of a singly linked list with references to the first and last nodes of the linked structure of nodes?
  The list contains N elements

A. O(1)

B. O(logN)

C. O(N^0.5)

D. O(N)

E. O(NlogN))

Why Use Linked List

- What operations with a Linked List faster than the version from ArrayList?

Clicker 7 - Getting All Elements in Order From a Linked List

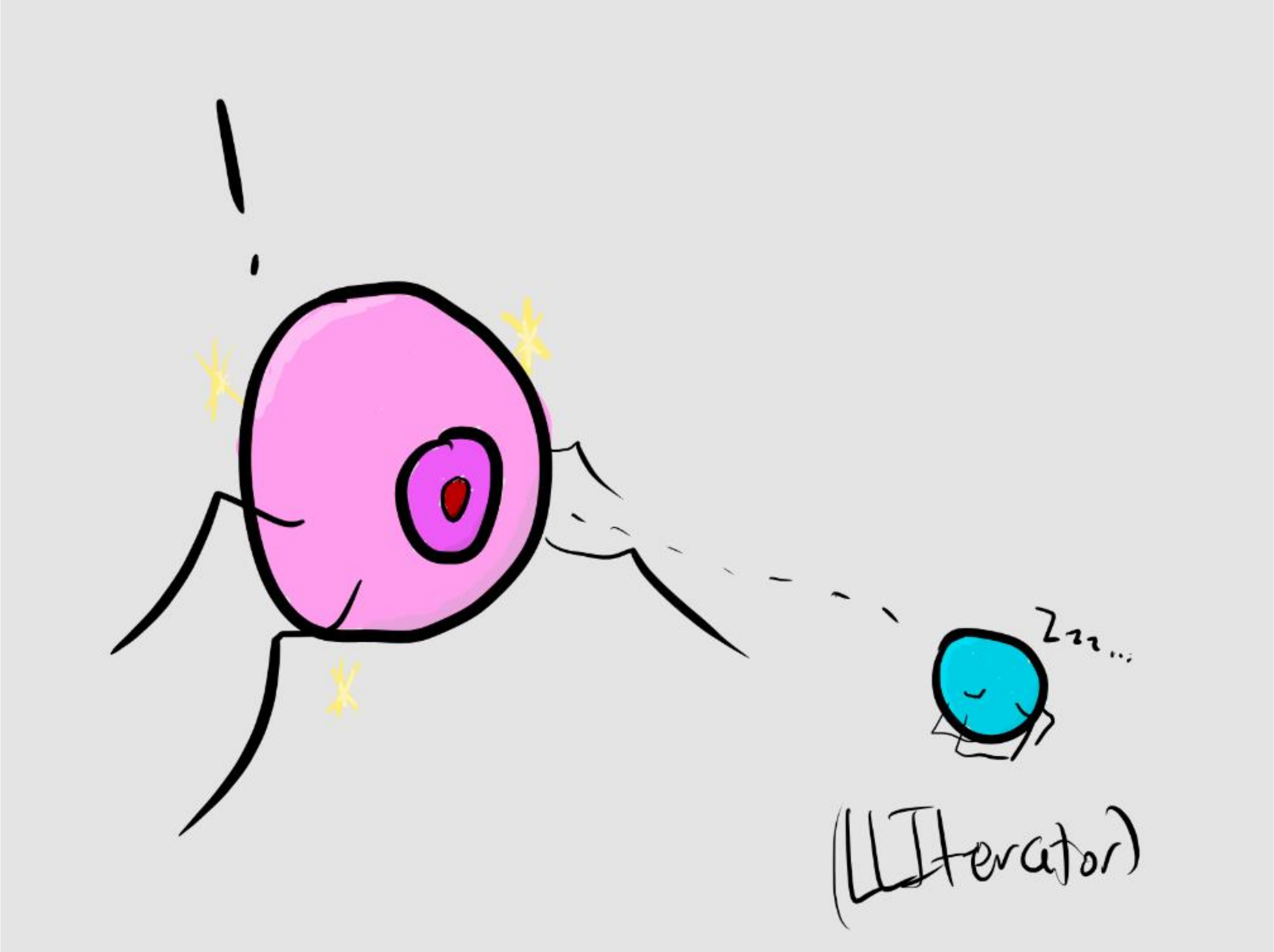
- What is the Order (Big O) of the following code?

```
LinkedList314<Integer> list;
list = new LinkedList314<Integer>();
// code to fill list with N elements
int total = 0;
//Big O of following code?
for (int i = 0; i < list.size(); i++)
    total += list.get(i);
```

A. O(N) B. $O(2^N)$ C. O(NlogN)

D. $O(N^2)$ E. $O(N^3)$

Iterators to the Rescue



Other Possible Features of Linked Lists

- Doubly Linked
- Circular
- Dummy Nodes for first and last node in list

```
public class DLNode<E> {
        private E myData;
        private DLNode<E> myNext;
        private DLNode<E> myPrevious;

}
```

Dummy Nodes

- Use of Dummy Nodes for a Doubly Linked List removes most special cases
- Also could make the Double Linked List circular

Doubly Linked List add

- public void add(E obj)

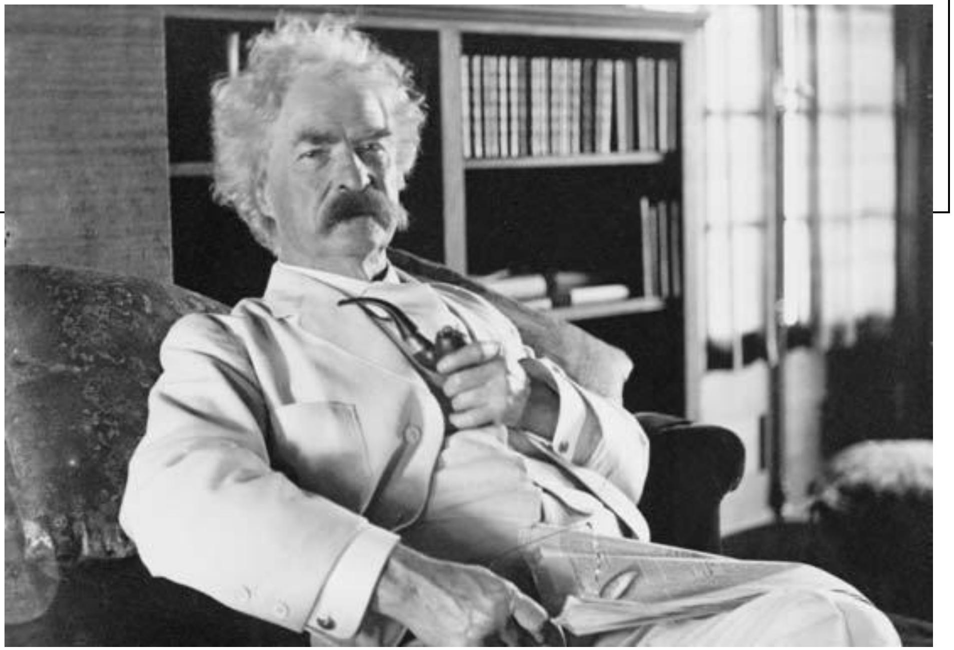
Insert for Doubly Linked List

- public void insert(int pos, E obj)

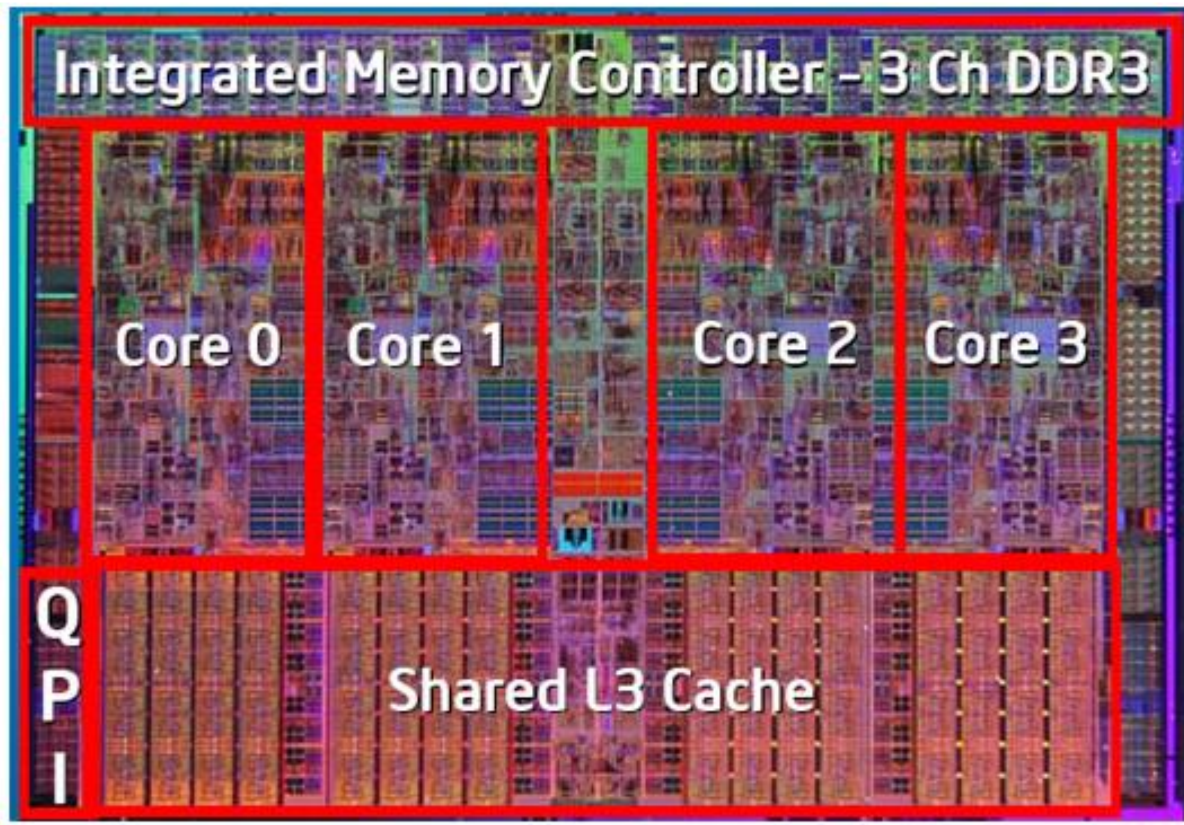
Topic 12
Introduction to Recursion

"To a man with a hammer, everything looks like a nail"

-Mark Twain



Underneath the Hood.



The Program Stack

- When you invoke a method in your code what happens when that method is done?

```
public class Mice {
    public static void main(String[] args) {
        int x = 37;
        int y = 12;
        method1(x, y);
        int z = 73;
        int m1 = method1(z, x);
        method2(x, x);
    }

    // method1 and method2
    // on next slide
```

method1 and method2

```
// in class Mice
public static int method1(int a, int b) {
      int r = 0;
      if (b != 0) {
            int x = a / b;
            int y = a % b;
            r = x + y;
      }
      return r;
}
public static void method2(int x, int y) {
      x++;
      y--;
      int z = method1(y, x);
      System.out.print(z);
}
```

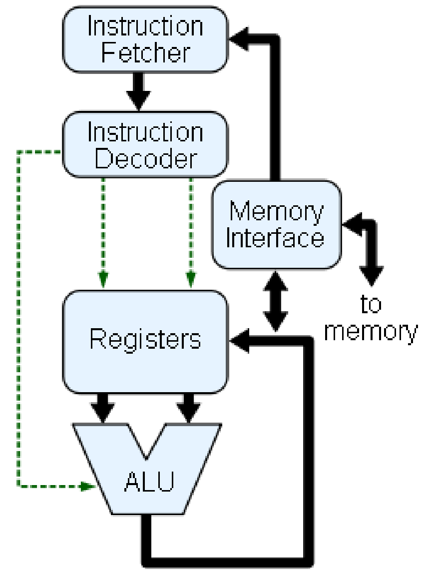
The Program Stack

- When your program is run on a processor, the commands are converted into another set of instructions and assigned memory locations.
  – normally a great deal of expansion takes place

```
public static void main(String[] args) {
    int x = 37;  // 0
    int y = 12;  // 1
    method1(x, y);  // 2
    int z = 73;  // 3
    int m1 = method1(z, x);  // 4
    method2(x, x);  // 7
}
```

Basic CPU Operations

- A CPU works via a fetch command / execute command loop and a program counter



- Instructions stored in memory (Instructions are data!)

```
int x = 37;  // 0
int y = 12;  // 1
method1(x, y);  // 2
int z = 73;  // 3
int m1 = method1(z, x);  // 4
method2(x, x);  // 5
```

- What if the first instruction of the method1 is stored at memory location 50?

```
// in class Mice
public static int method1(int a, int b) {
     int r = 0; // 51
     if (b != 0) { // 52
          int x = a / b; // 53
          int y = a % b; // 54
          r = x + y; // 55
     }
     return r; // 56
}
public static void method2(int x, int y) {
     x++; // 60
     y--; // 61
     int z = method1(y, x); // 62
     System.out.print(z); // 63
}
```

Clicker 1 - The Program Stack

```
int x = 37;   // 1
int y = 12;   // 2
method1(x, y);  // 3
int z = 73;   // 4
int m1 = method1(z, x);  // 5
method2(x, x);  // 6
```

- Instruction 3 is really saying *jump to instruction 50 with parameters x and y*
- ***In general*** what happens when method1 finishes?

A. program ends   B. goes to instruction 4

C. goes back to *whatever* method called it

Activation Records and the Program Stack

- When a method is invoked all the relevant information about the current method (variables, values of variables, next line of code to be executed) is placed in an *activation record*
- The activation record is *pushed* onto the *program stack*
- A *stack* is a data structure with a single access point, the *top.*

The Program Stack

- Data may either be added (*pushed*) or removed *(popped)* from a stack but it is always from the top.
  - A stack of dishes
  - which dish do we have easy access to?

top

Using Recursion

A Problem

- Write a method that determines how much space is take up by the files in a directory
- A directory can contain files and directories
- How many directories does our code have to examine?
- How would you add up the space taken up by the files in a single directory
  - Hint: don't worry about any sub directories at first

Clicker 2

- How many levels of directories have to be visited?

A. 0

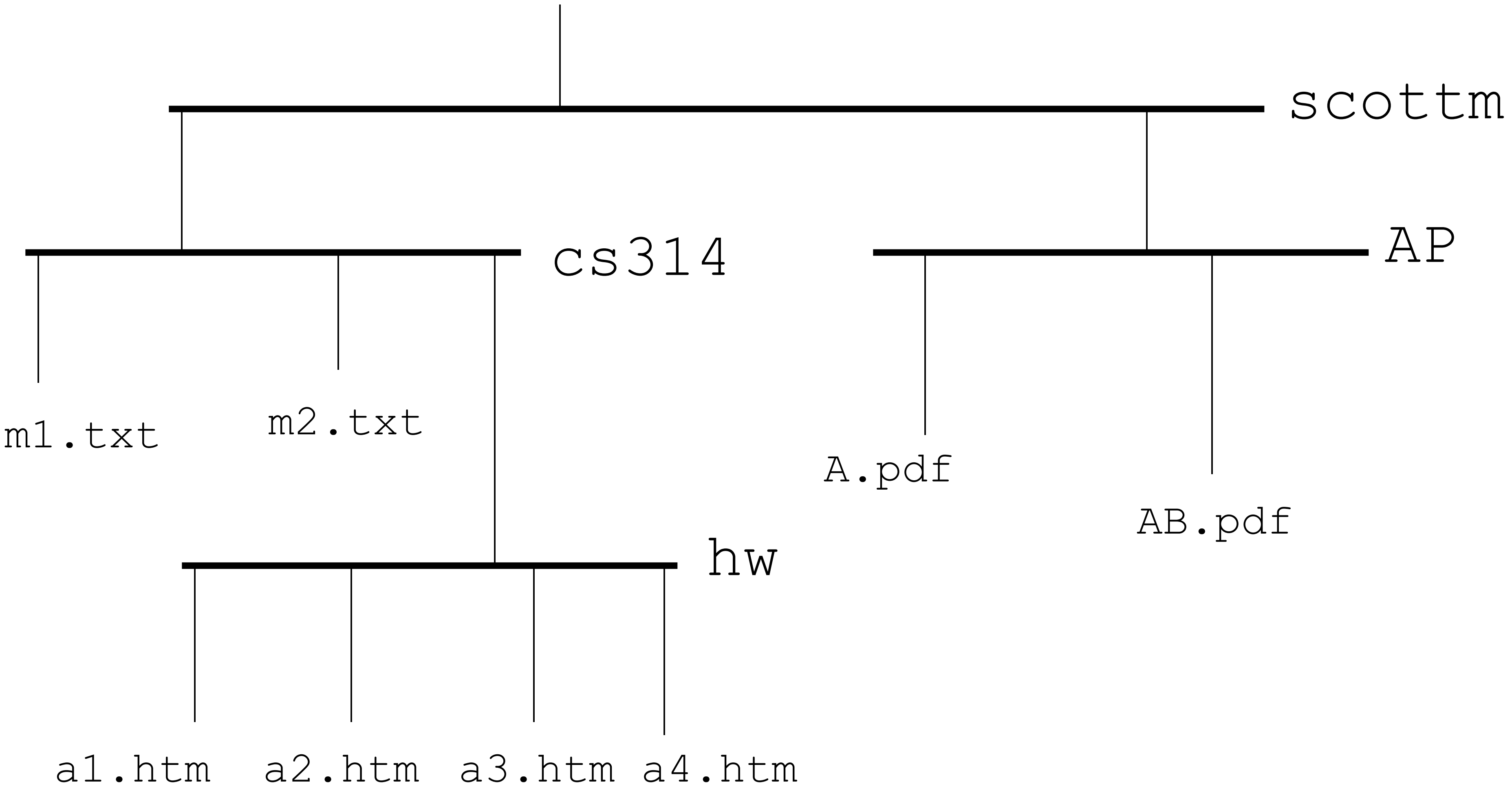
B. 1

C. 8

D. Infinite

E. Unknown

Sample Directory Structure



Java File Class

- **`File`**`(String pathname)` Creates a new File instance by converting the given pathname.
- `boolean isDirectory()` Tests whether the file denoted by this abstract pathname is a directory.
- `File[] listFiles()` Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.

Code for `getDirectorySpace()`

```
// pre: dir is a directory and dir != null
public static long spaceUsed(File dir) {
  if( dir == null || !dir.isDirectory())
      throw new IllegalArgumentException();
  long spaceUsed = 0;
  File[] subFilesAndDirs = dir.listFiles();
  if(subFilesAndDirs != null)
    for(File sub : subFilesAndDirs)
      if(sub != null)
        if(sub.isFile()) // sub is a plain old file
          spaceUsed += sub.length();
        else if (sub.isDirectory())
            // else sub is a directory
          spaceUsed += spaceUsed(sub);
  return spaceUsed;
}
```

Clicker 3

- Is it possible to write a non recursive method to determine space taken up by files in a directory, including its subdirectories, and their subdirectories, and their subdirectories, and so forth?

A. No

B. Yes

C. It Depends

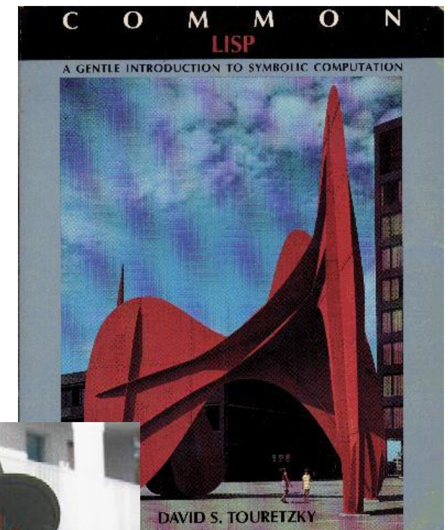
Iterative `getDirectorySpace()`

```
public long getDirectorySpace(File d) {
  ArrayList<File> dirs = new ArrayList<>();
  dirs.add(d);
  long total = 0;
  while (dirs.size() > 0) {
     File temp = dirs.remove(dirs.size() - 1);
     File[] filesAndSubs = temp.listFiles();
     if (filesAndSubs != null) {
         for (File f : filesAndSubs) {
             if (f != null) {
                 if (f.isFile())
                     total += f.length();
                 else if (f.isDirectory())
                     dirs.add(f);
             }
         }
  }
  return total;
}
```

Wisdom for Writing Recursive Methods

The 3 plus 1 rules of Recursion

1. Know when to stop
2. Decide how to take one step
3. Break the journey down into that step and a smaller journey
4. Have faith



From *Common Lisp: A Gentle Introduction to Symbolic Computation* by David Touretzky

Writing Recursive Methods

- Rules of Recursion
  1. Base Case: Always have at least one case that can be solved without using recursion
  2. Make Progress: Any recursive call must progress toward a base case.
  3. "You gotta believe." Always assume that the recursive call works. (Of course you will have to design it and test it to see if it works or prove that it always works.)

A recursive solution solves a small part of the problem and leaves the rest of the problem in the same form as the original

N!

- the classic first recursion problem / example
- N!

  5! = 5 * 4 * 3 * 2 * 1 = 120

```
 int res = 1;
for(int i = 2; i <= n; i++)
     res *= i;
```

Factorial Recursively

- Mathematical Definition of Factorial
- for N >= 0, N! is:

```
0! = 1
N! = N * (N - 1)! (for N > 0)
```

The definition is recursive.

```
// pre n >= 0
public int fact(int n) {
  if(n == 0)
    return 1;
  else
     return n * fact(n-1);
} // return (n == 0) ? 1 : n * fact(n - 1);
```

Tracing Fact With the Program Stack

System.out.println(fact(4));

top → System.out.println(fact(4));

Calling fact with 4

n 4 in method fact

partial result = n * fact(n-1)

top → System.out.println(fact(4));

Calling fact with 3

n 3 in method fact

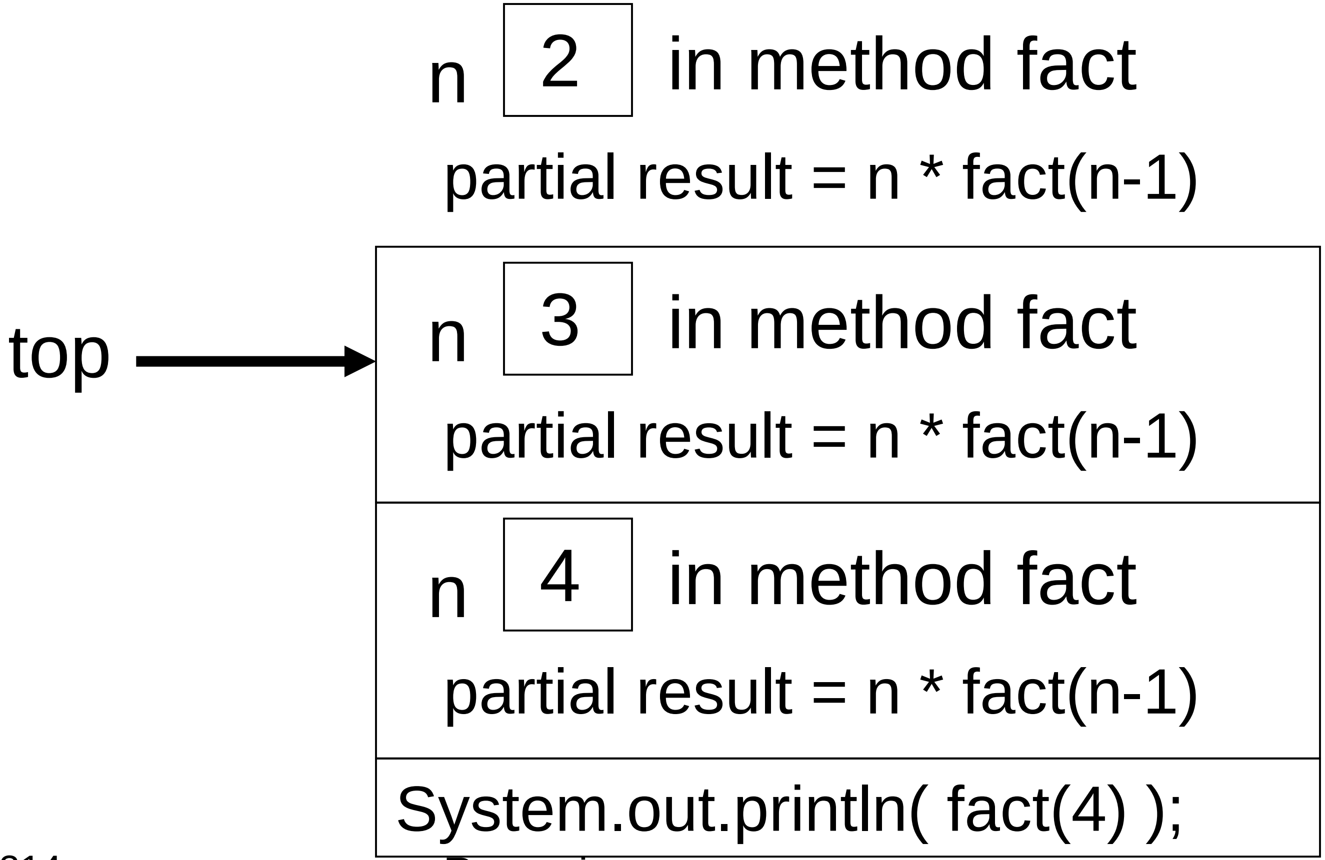
partial result = n * fact(n-1)

top → n 4 in method fact

partial result = n * fact(n-1)

System.out.println(fact(4));

Calling fact with 2



Calling fact with 1

n 1 in method fact

partial result = n * fact(n-1)

top

n 2 in method fact

partial result = n * fact(n-1)

n 3 in method fact

partial result = n * fact(n-1)

n 4 in method fact

partial result = n * fact(n-1)

System.out.println(fact(4));

Calling fact with 0 and returning 1

n 0 in method fact

returning 1 to whatever method called me

top

n 1 in method fact partial result = n * fact(n-1)
n 2 in method fact partial result = n * fact(n-1)
n 3 in method fact partial result = n * fact(n-1)
n 4 in method fact partial result = n * fact(n-1)
System.out.println(fact(4));

Returning 1 from fact(1)

n $\boxed{1}$ in method fact

partial result = n * 1,

return 1 to whatever method called me

top →

n $\boxed{2}$ in method fact partial result = n * fact(n-1)
n $\boxed{3}$ in method fact partial result = n * fact(n-1)
n $\boxed{4}$ in method fact partial result = n * fact(n-1)
System.out.println(fact(4));

Returning 2 from fact(2)

n | 2 | in method fact

partial result = 2 * 1,
return 2 to whatever method called me

top →

n | 3 | in method fact

partial result = n * fact(n-1)

n | 4 | in method fact

partial result = n * fact(n-1)

System.out.println(fact(4));

Returning 6 from fact(3)

n | 3 | in method fact

partial result = 3 * 2,
return 6 to whatever method called me

top

n | 4 | in method fact

partial result = n * fact(n-1)

System.out.println(fact(4));

Returning 24 from fact(4)

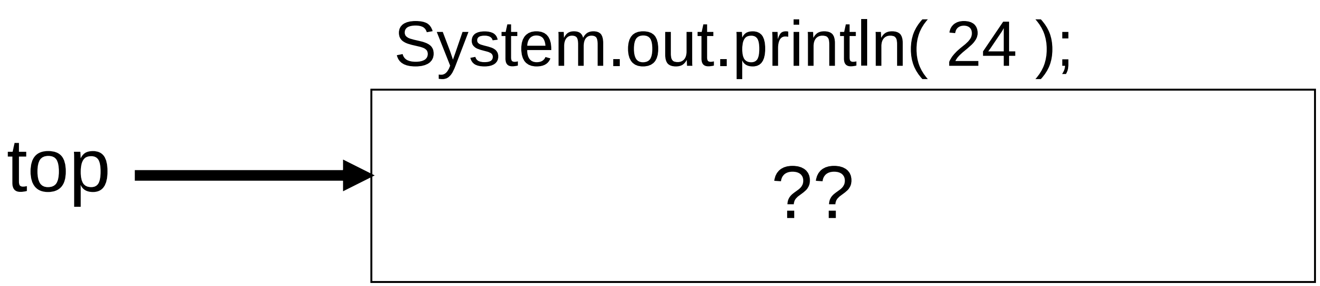
n | 4 | in method fact

partial result = 4 * 6,

return 24 to whatever method called me

top → System.out.println(fact(4));

Calling System.out.println



Evaluating Recursive Methods

Evaluating Recursive Methods

- you must be able to evaluate recursive methods

```
public static int mystery (int n){
    if(n == 0)
        return 2;
    else
        return 3 * mystery(n-1);
}
// what is returned by mystery(3)
```

Evaluating Recursive Methods

- Draw the program stack!

  m(3) = 3 * m(2) -> 3 * 18 = 54

  m(2) = 3 * m(1) -> 3 * 6 = 18

  m(1) = 3 * m(0) -> 3 * 2 = 6

  m(0) = 2

  -> 54

- with practice you can see the result

Clicker 4

- What is returned by `fact(-3)` ?

A. 0

B. 1

C. Infinite loop

D. Syntax error

E. Runtime error

```
public static int fact(int n) {
        if (n == 0) {
                return 1;
        } else {
                return n * fact(n - 1);
        }
}
```

Evaluating Recursive Methods

- What about multiple recursive calls?

```
public static int bar(int n){
    if (n <= 0)
        return 2;
    else
      return 3 + bar(n-1) + bar(n-2);
}
```

- Clicker 5 - What does bar(4) return?

A. 2 B. 3 C. 12 D. 22 E. 37

Evaluating Recursive Methods

- What is returned by `bar(4)`?

b(4) = 3 + b(3) + b(2)

b(3) = 3 + b(2) + b(1)

b(2) = 3 + b(1) + b(0)

b(1) = 3 + b(0) + b(-1)

b(0) = 2

b(-1) = 2

Evaluating Recursive Methods

- What is returned by `bar(4)`?

b(4) = 3 + b(3) + b(2)

b(3) = 3 + b(2) + b(1)

b(2) = 3 + b(1) + b(0) //substitute in results

b(1) = 3 + 2 + 2 = 7

b(0) = 2

b(-1) = 2

Evaluating Recursive Methods

- What is returned by `bar(4)`?

b(4) = 3 + b(3) + b(2)

b(3) = 3 + b(2) + b(1)

b(2) = 3 + 7 + 2 =12

b(1) = 7

b(0) = 2

b(-1) = 2

Evaluating Recursive Methods

- What is returned by `bar(4)`?

b(4) = 3 + b(3) + b(2)

b(3) = 3 + 12 + 7 = 22

b(2) = 12

b(1) = 7

b(0) = 2

b(-1) = 2

Evaluating Recursive Methods

- What is returned by `bar(4)`?

b(4) = 3 + 22 + 12 = 37

b(3) = 22

b(2) = 12

b(1) = 7

b(0) = 2

b(-1) = 2

Recursion Practice

- Write a method raiseToPower(int base,
  int power)
- //pre: power >= 0

- Simple recursion (also called tail recursion)

Finding the Maximum in an Array

- public int max(int[] data) {
- Helper method or create smaller arrays each time

Clicker 6

- When writing recursive methods what should be done first?

A. Determine recursive case
B. Determine recursive step
C. Make a recursive call
D. Determine base case(s)
E. Determine the Big O

Your Meta Cognitive State

- Remember we are learning to use a tool.
- It is not a good tool for ***all*** problems.
  - In fact we will implement several algorithms and methods where an iterative (looping without recursion) solution would work just fine
- After learning the mechanics and basics of recursion the real skill is knowing what problems or class of problems to apply it to

Big O and Recursion

- Determining the Big O of recursive methods can be tricky.
- A *recurrence relation* exits if the function is defined recursively.
- The T(N), actual running time, for N! is recursive
- $T(N)_{fact} = T(N-1)_{fact} + O(1)$
- This turns out to be O(N)
  - There are N steps involved

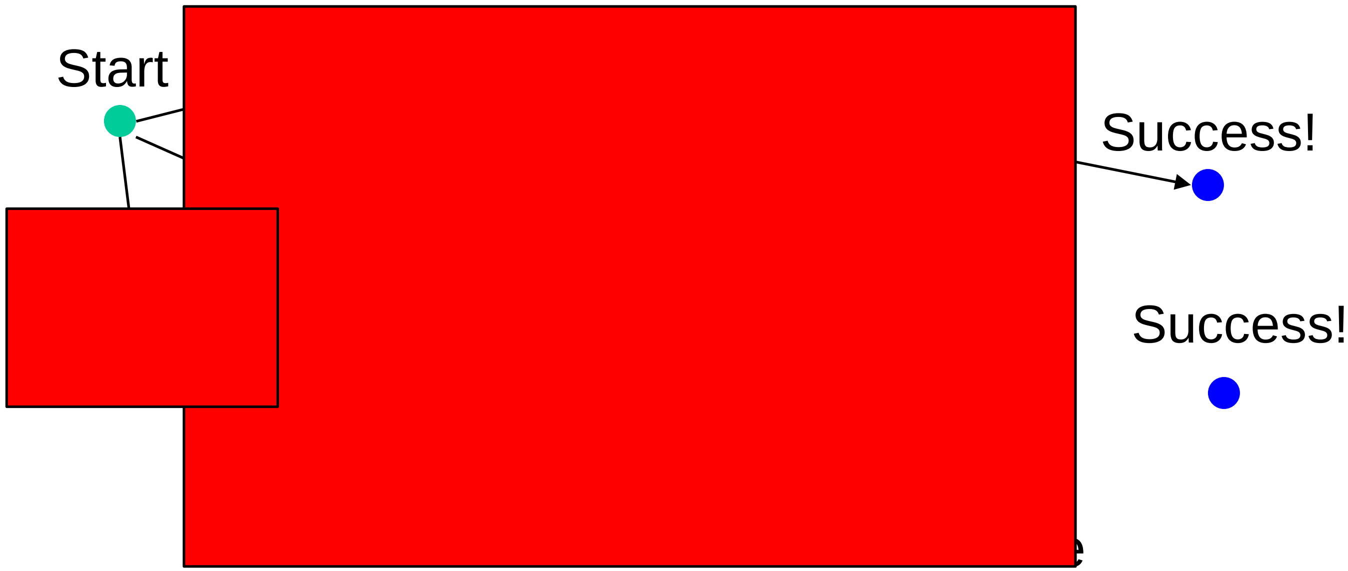
Common Recurrence Relations

- T(N) = T(N/2) + O(1) -> O(logN)
  - binary search
- T(N) = T(N-1) + O(1) -> O(N)
  - sequential search, factorial
- T(N) = T(N/2) + T(N/2) + O(1) -> O(N),
  - tree traversal
- T(N) = T(N-1) + O(N) -> O(N^2)
  - selection sort
- T(N) = T(N/2) + T(N/2) + O(N) -> O(NlogN)
  - merge sort
- T(N) = T(N-1) + T(N-1) + O(1) -> O(2^N)
  - Fibonacci

Topic 13
Recursive Backtracking

Devon, 2022

Backtracking

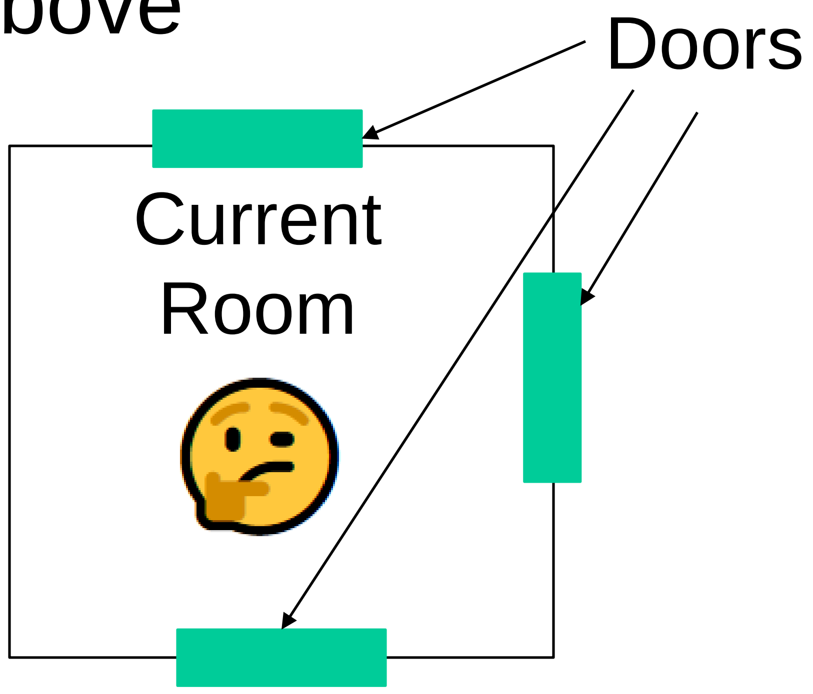


Problem space consists of states (nodes) and actions (paths that lead to new states). When in a node can can only see paths to connected nodes

If a node only leads to failure go back to its "parent" node. Try other alternatives. If these all lead to failure then more backtracking may be necessary.

Escaping a Maze

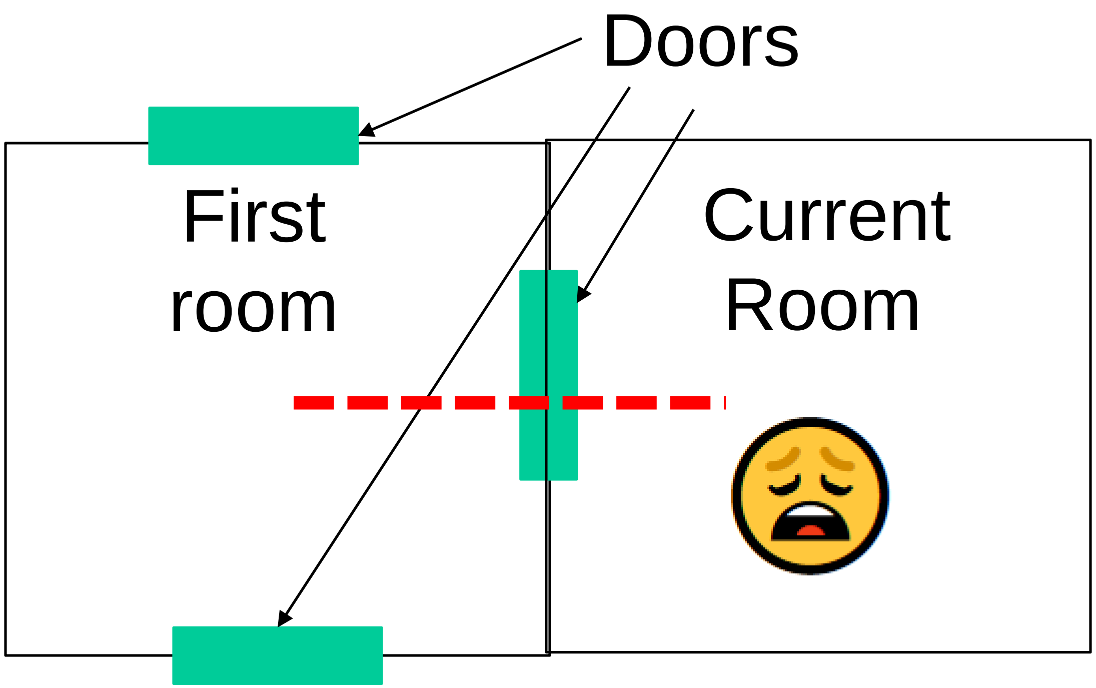
- Which door should we take?
- A view from above



Exit out there,
some where …
we hope

Escaping a Maze

- Try door to the east

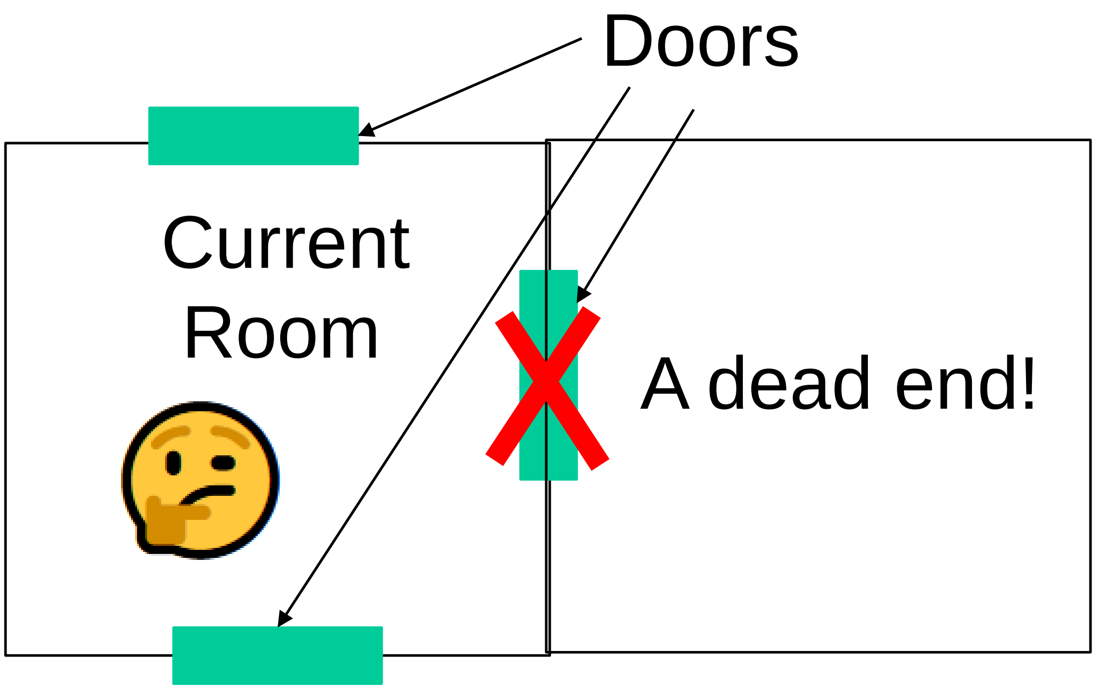


Exit out there, some where … we hope

A dead end!

Escaping a Maze

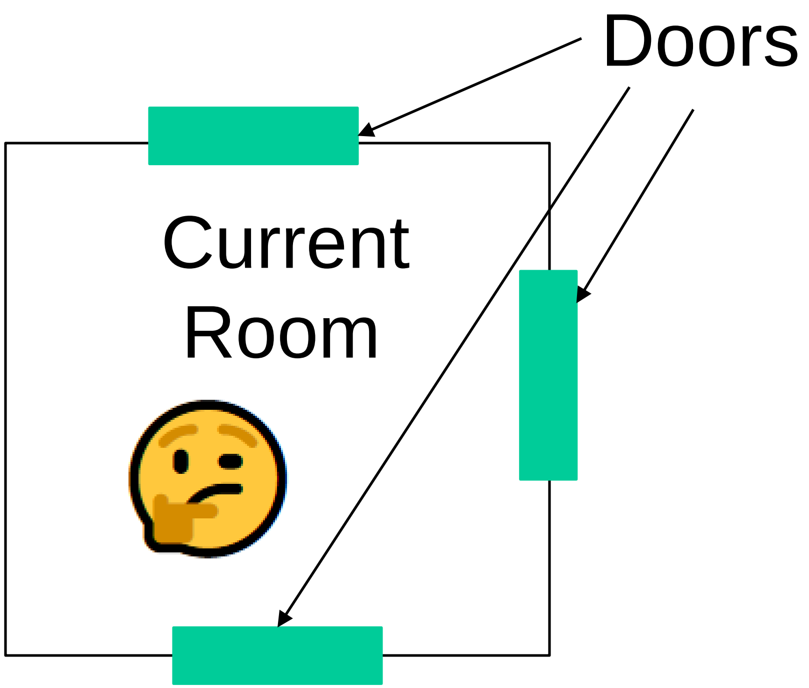
- Back we go



Exit out there,
some where …
we hope

Escaping a Maze

- What if we knew the exit was to the south?

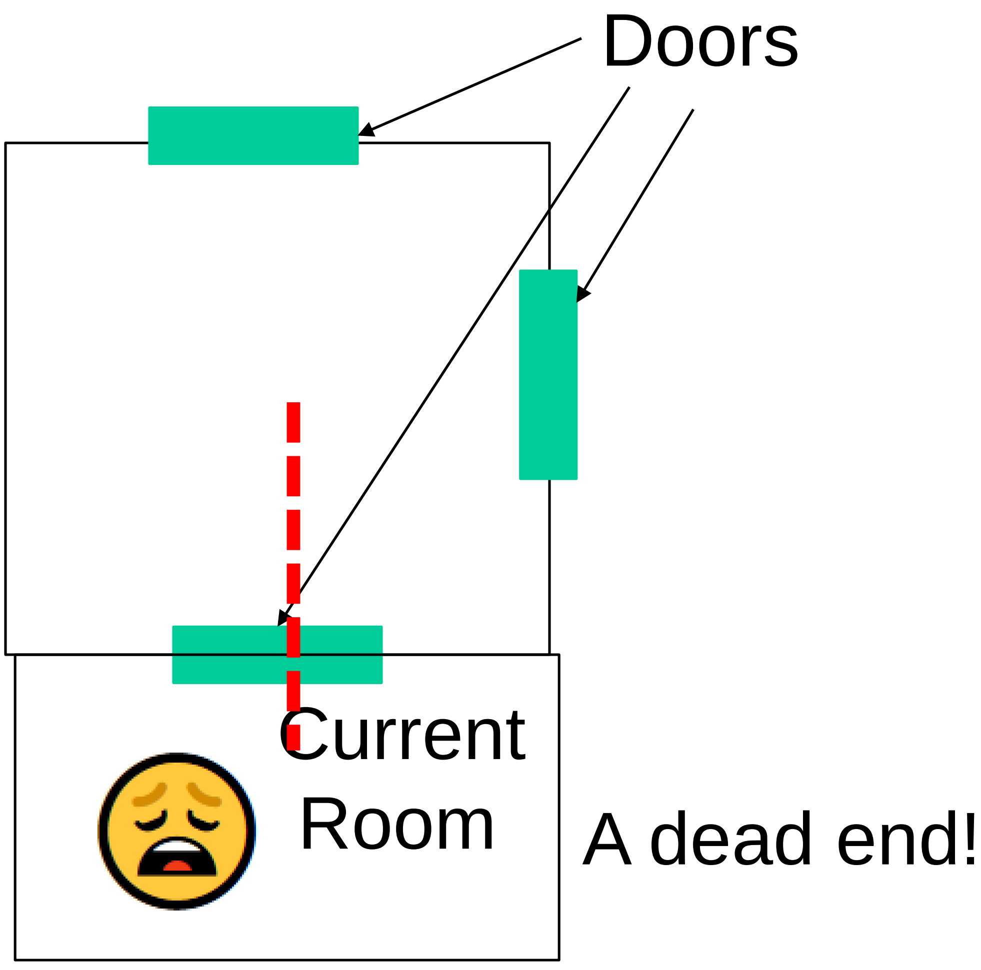


Exit out there,
some where
to the south!

Escaping a Maze

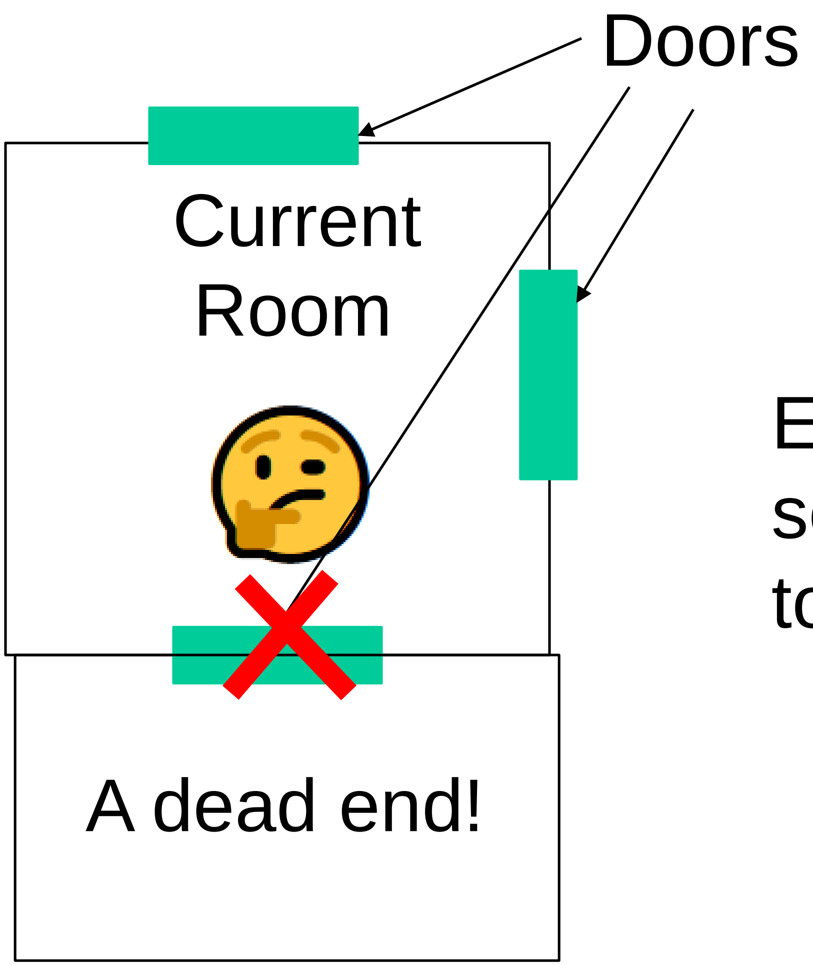
- Start over. What if we knew the exit was to the south?

Exit out there,
some where
to the south!



Escaping a Maze

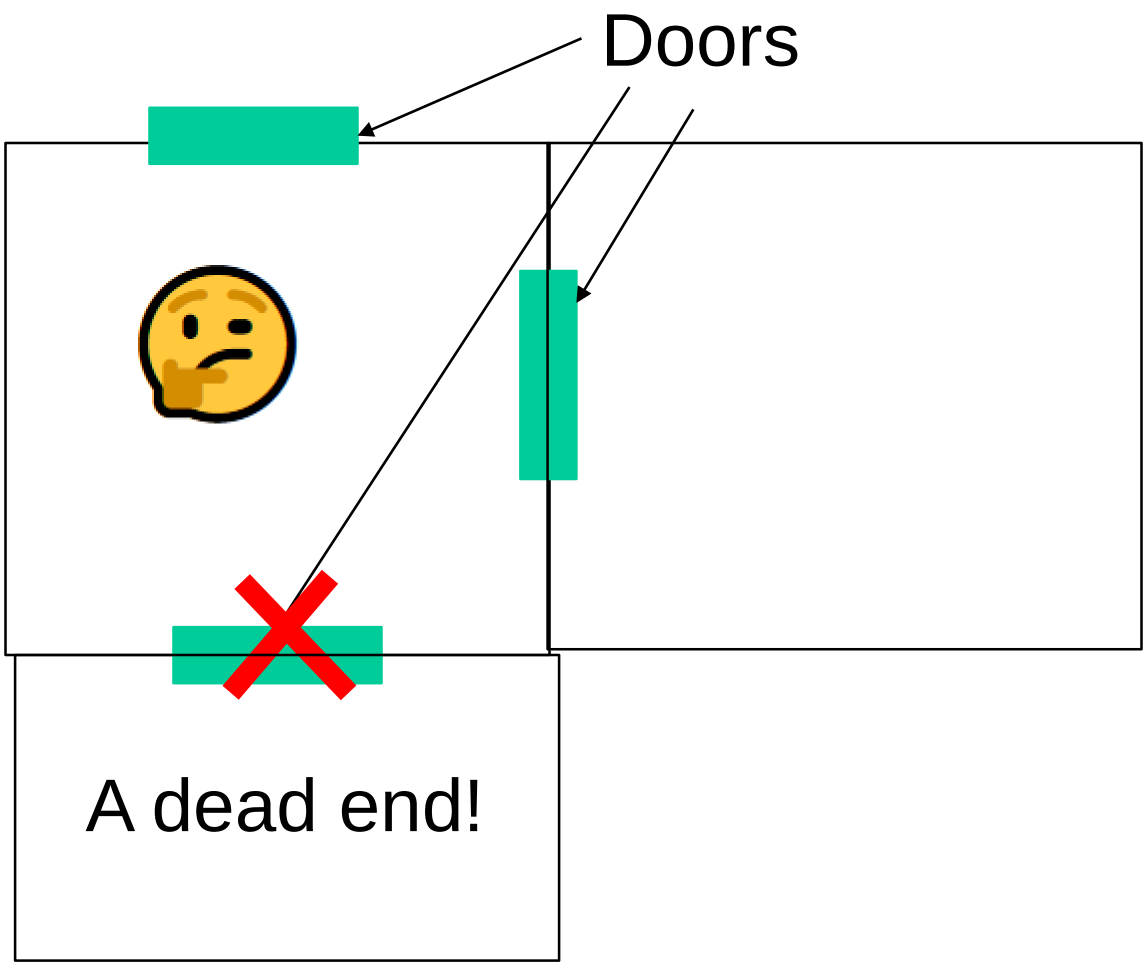
- What if we knew the exit was to the south?



Exit out there, some where to the south!

Escaping a Maze

- What if we knew the exit was to the south?

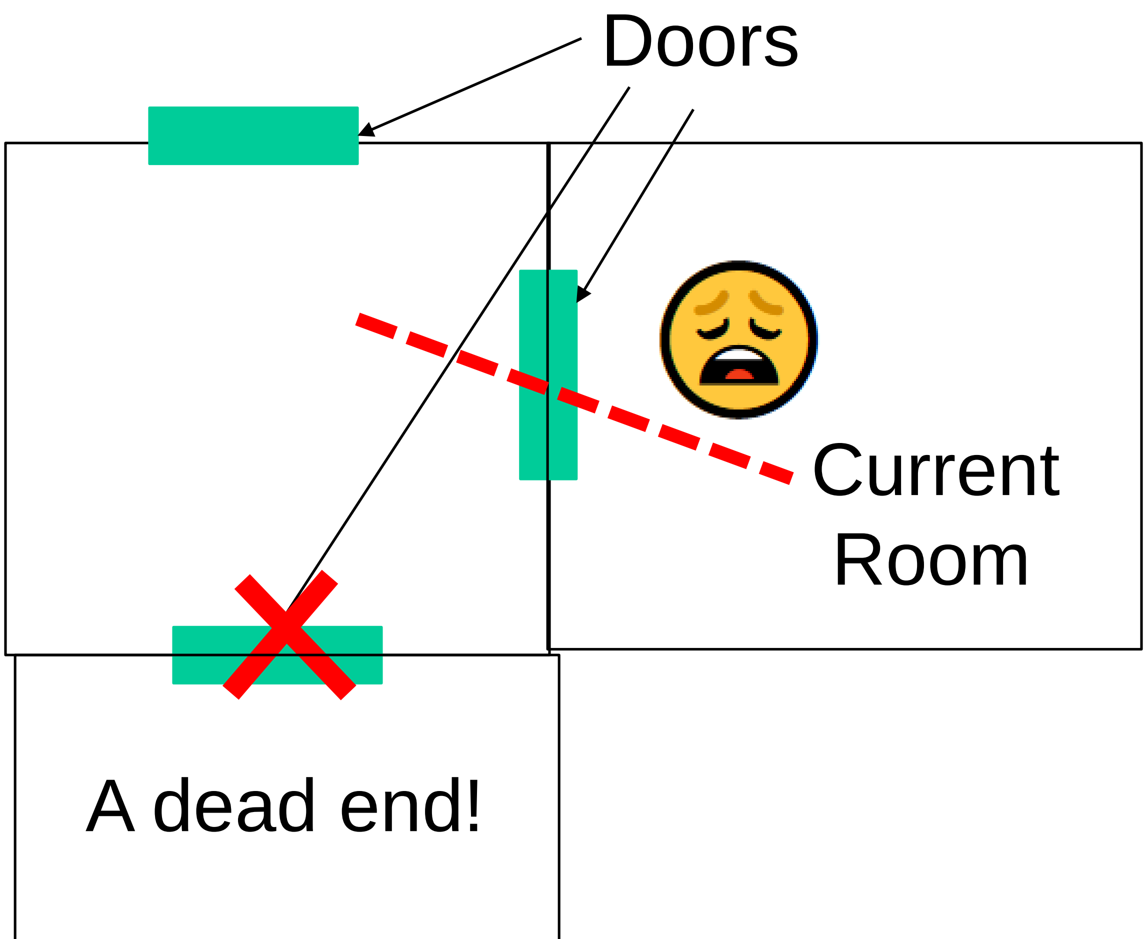


Exit out there,
some where
to the south!

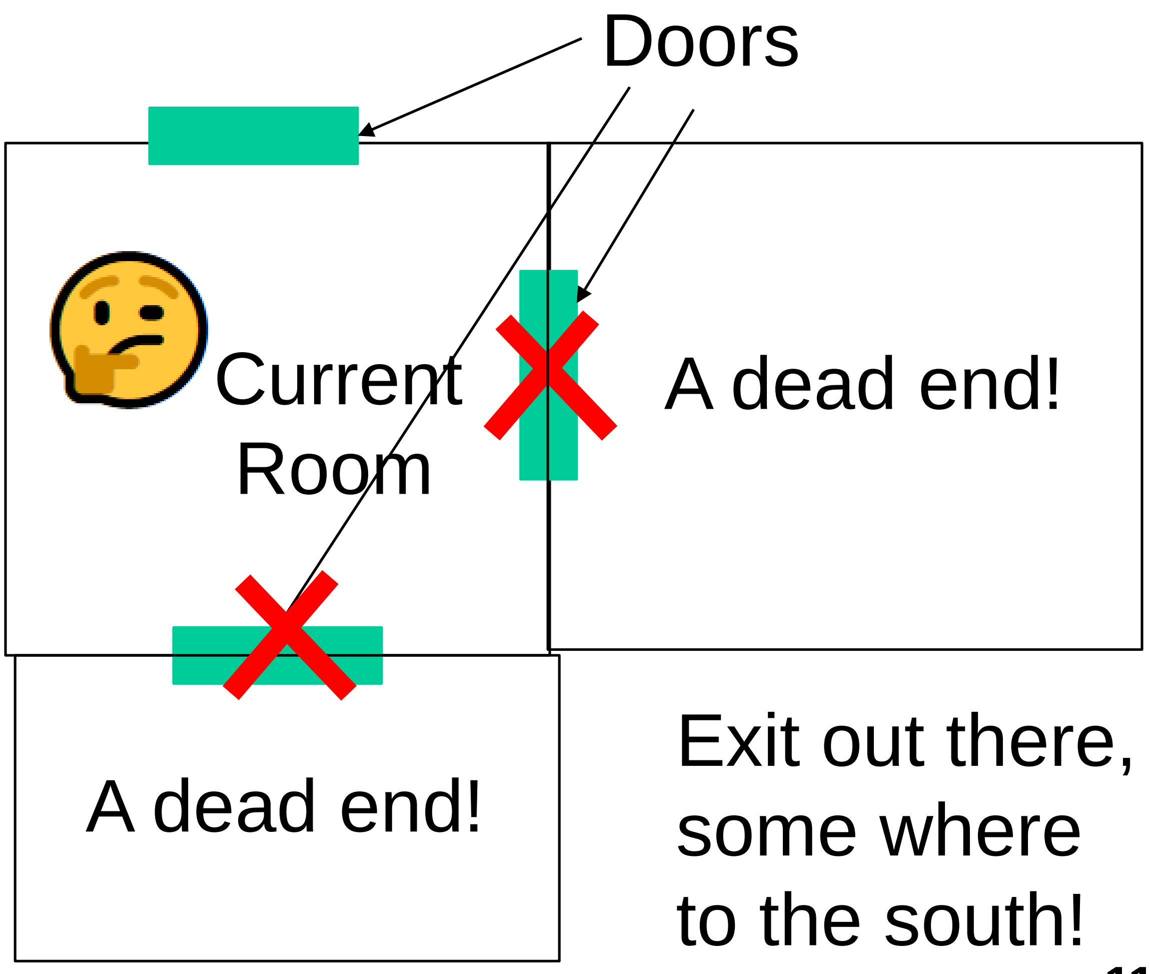
Escaping a Maze

- What if we knew the exit was to the south?

Exit out there, some where to the south!

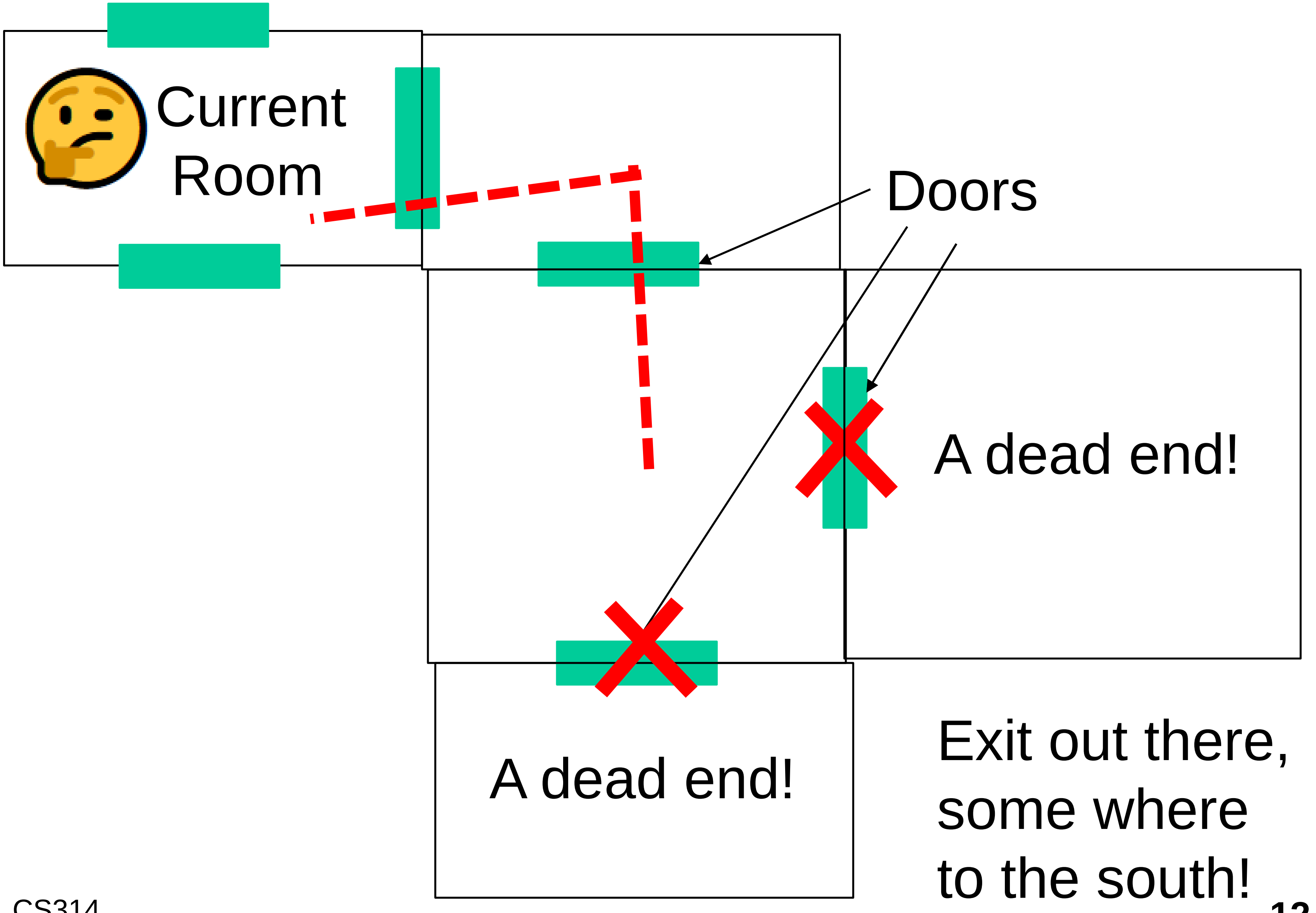


Escaping a Maze



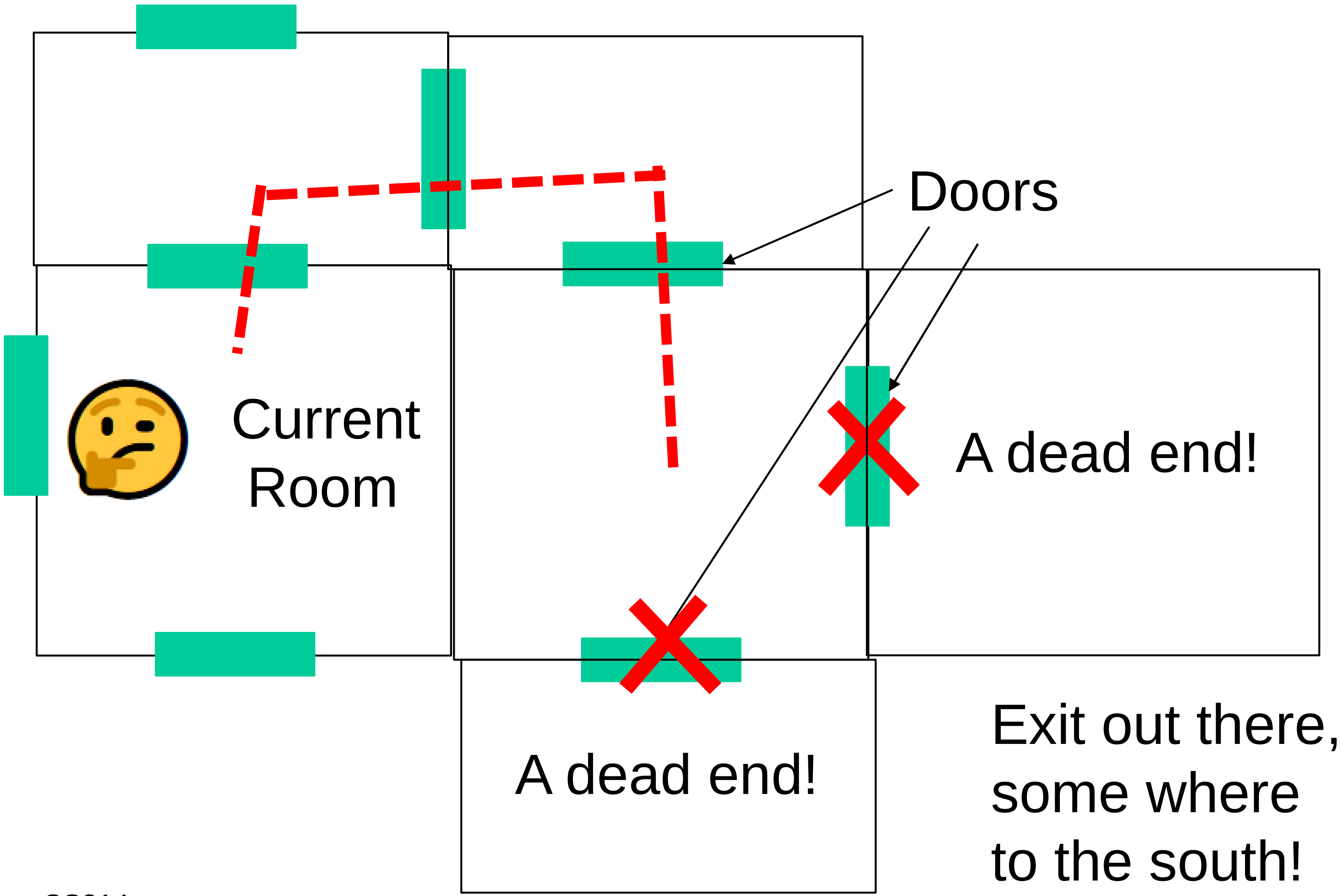
Exit out there, some where to the south!

Escaping a Maze

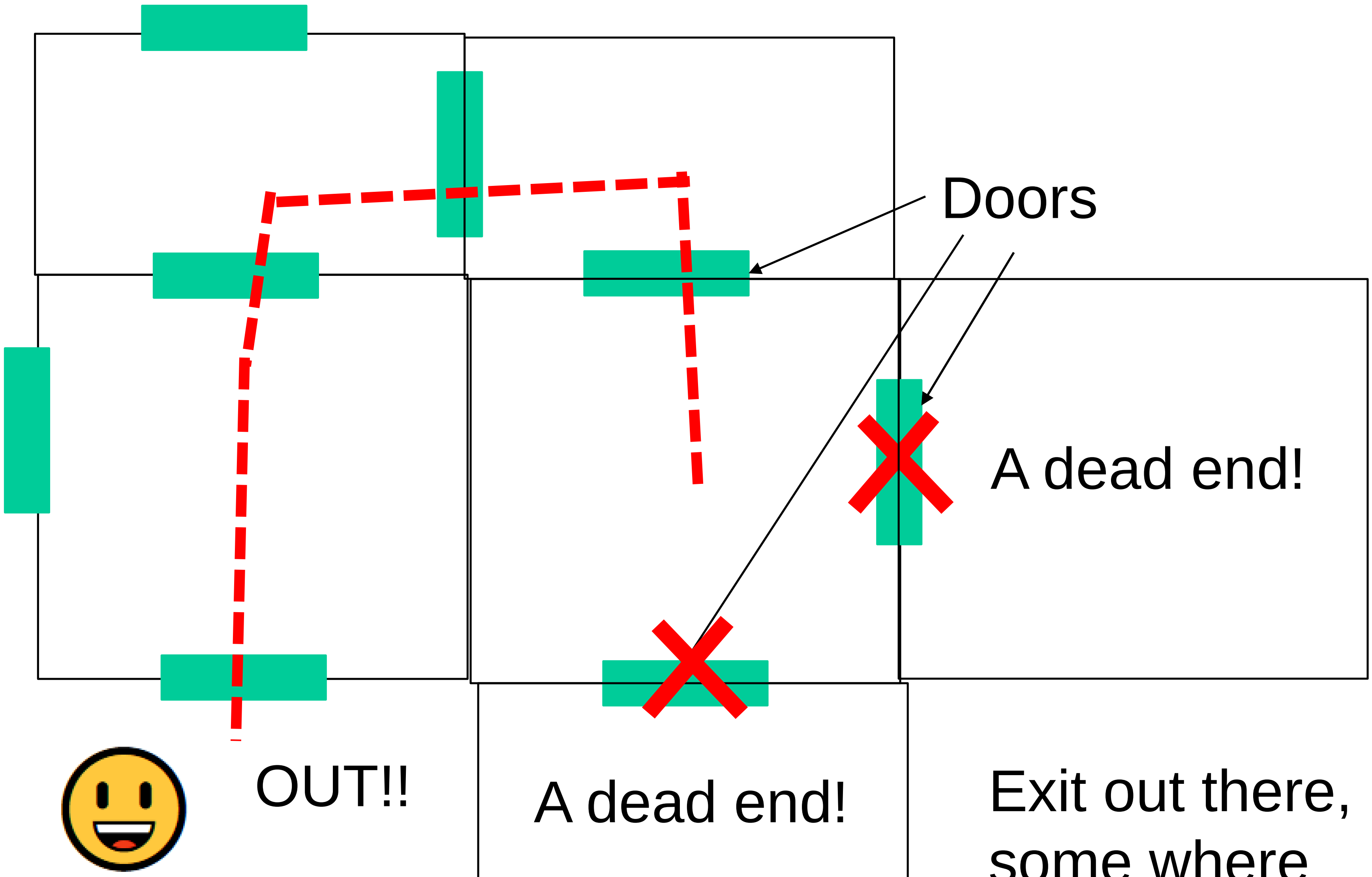


Exit out there,
some where
to the south!

Escaping a Maze



Escaping a Maze



Exit out there, some where to the south!

Recursive Backtracking

Pseudo code for recursive backtracking algorithms – looking for ***a*** solution

If at a solution, report success
for (every possible choice from current state)
    Make that choice and take one step along path
    Use recursion to **try** to solve the problem for the new state
    If the recursive call succeeds, report the success to the previous level
    Otherwise Back out of the current choice to restore the state at the start of the loop.
Report failure

Another Concrete Example

- Sudoku
- 9 by 9 matrix with some numbers filled in
- all numbers must be between 1 and 9
- Goal: Each row, each column, and each mini matrix must contain the numbers between 1 and 9 once each
  - no duplicates in rows, columns, or mini matrices

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Solving Sudoku – Brute Force

- A *brute force* algorithm is a simple but generally inefficient approach
- Try all combinations until you find one that works
- This approach isn't clever, but computers are fast
- Then try and improve on the brute force results

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Solving Sudoku

- Brute force Sudoku Soluton
  - – if not open cells, solved
  - – scan cells from left to right, top to bottom for first open cell
  - – When an open cell is found start cycling through digits 1 to 9.
  - – When a digit is placed check that the set up is legal
  - – now solve the board

5	3	1		7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Clicker 1

- After placing a number in a cell is the remaining problem very similar to the original problem?

A. No

B. Yes

Solving Sudoku – Later Steps

5	3	**1**		7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	**1**	**2**	7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	**1**	**2**	7	**4**			
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	**1**	**2**	7	**4**	**8**		
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	**1**	**2**	7	**4**	**8**	**9**	
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

uh oh!

Sudoku – A Dead End

- We have reached a dead end in our search

5	3	1	2	7	4	8	9	
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- With the current set up none of the nine digits work in the top right corner

Backing Up

- When the search reaches a dead end in ***backs up*** to the previous cell it was trying to fill and goes onto to the next digit
- We would back up to the cell with a 9 and that turns out to be a dead end as well so we back up again
  – so the algorithm needs to remember what digit to try next
- Now in the cell with the 8. We try and 9 and move forward again.

5	3	1	2	7	4	8	9	
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	1	2	7	4	9		
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Characteristics of Brute Force and Backtracking

- Brute force algorithms are slow
- The first pass attempts typically don't employ a lot of logic
- But, brute force algorithms are fairly easy to implement as a first pass solution
  - many backtracking algorithms are brute force algorithms

Key Insights

- After trying placing a digit in a cell we want to solve the new sudoku board
  - Isn't that a smaller (or simpler version) of the same problem we started with?!?!?!?
- After placing a number in a cell the we need to remember the next number to try in case things don't work out.
- We need to know if things worked out (found a solution) or they didn't, and if they didn't try the next number
- If we try all numbers and none of them work in our cell we need to *report back* that things didn't work

Clicker 2

- Grace 2019 Asked: When we reach the base case in the solveSudoku method (9 x 9 board) and before we return true, how many stack frames are on the program stack of the solveSudoku method? Pick the closest answer.

A. <= 9

B. 82

C. 81^9

D. 9^{81}

E. cannot determine

Recursive Backtracking

- Problems such as Suduko can be solved using recursive backtracking
- recursive because later versions of the problem are just slightly simpler versions of the original
- backtracking because we may have to try different alternatives

Recursive Backtracking - Repeated

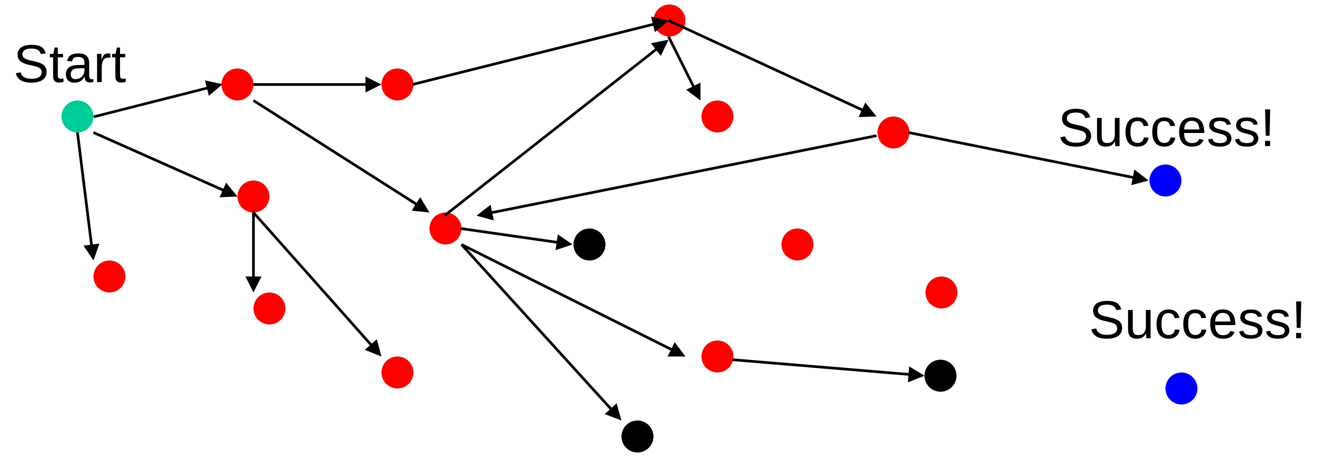
Pseudo code for recursive backtracking algorithms – looking for ***a*** solution

If at a solution, report success
for (every possible choice from current state)
- Make that choice and take one step along path
- Use recursion to try to solve the problem for the new state
- ***If*** the recursive call succeeds, report the success to the previous level
- ***Otherwise*** Back out of the current choice to restore the state at the start of the loop.

Report failure

Goals of Backtracking

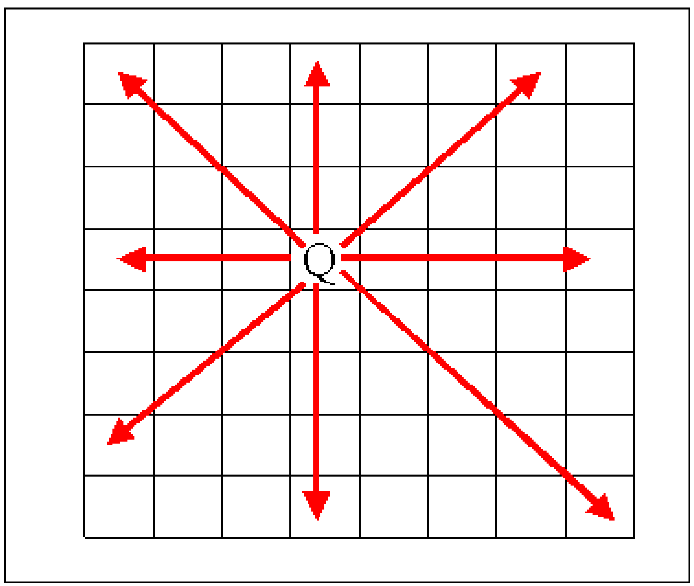
- Possible goals
  - Find a path to success
  - Find all paths to success
  - Find the best path to success
- Not all problems are exactly alike, and finding one success node may not be the end of the search

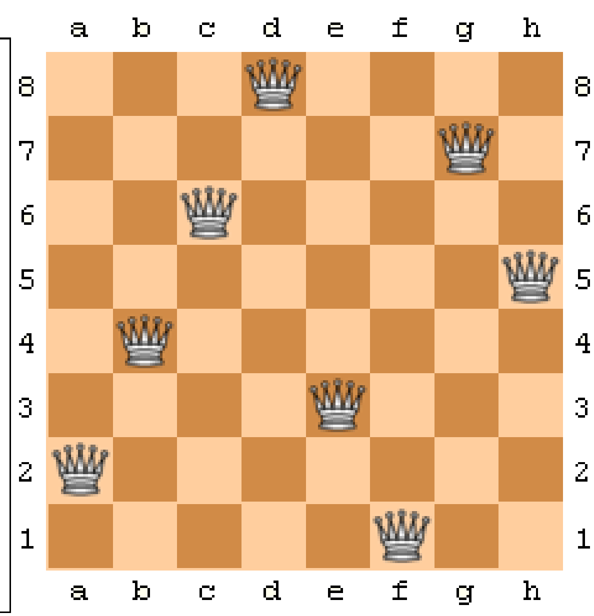


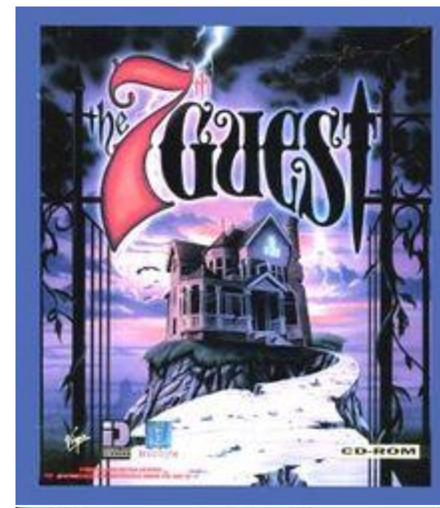
The ~~8~~ N Queens Problem

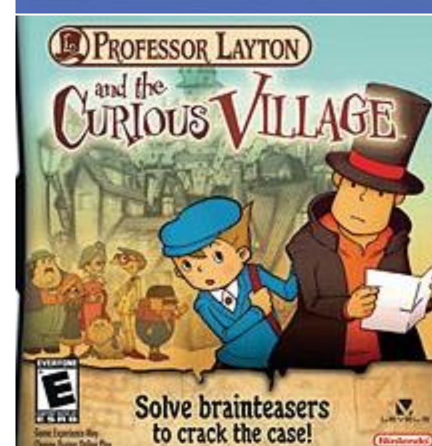
The 8 Queens Problem

- A classic chess puzzle
  - Place 8 queen pieces on a chess board so that none of them can attack one another









The N Queens Problem

- Place N Queens on an N by N chessboard so that none of them can attack each other
- Number of possible placements?
- In 8 x 8

64 * 63 * 62 * 61 * 60 * 59 * 58 * 57

= 178,462, 987, 637, 760 / 8!

= 4,426,165,368

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdots (n-k+1)}{k \cdot (k-1) \cdots 1} = \frac{n!}{k!(n-k)!} \quad \text{if } 0 \leq k \leq n \qquad (1)$$

n choose k

– How many ways can you choose k things from a set of n items?

– In this case there are 64 squares and we want to choose 8 of them to put queens on

Clicker 3

- For a safe solution, how many queens can be placed in a given column?

A. 0

B. 1

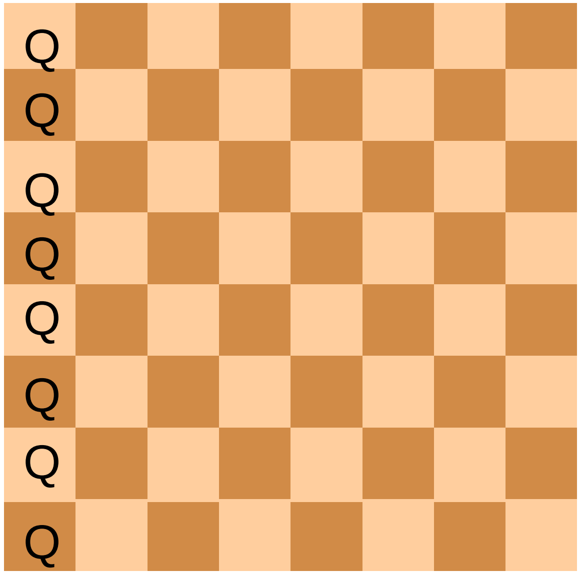
C. 2

D. 3

E. Any number

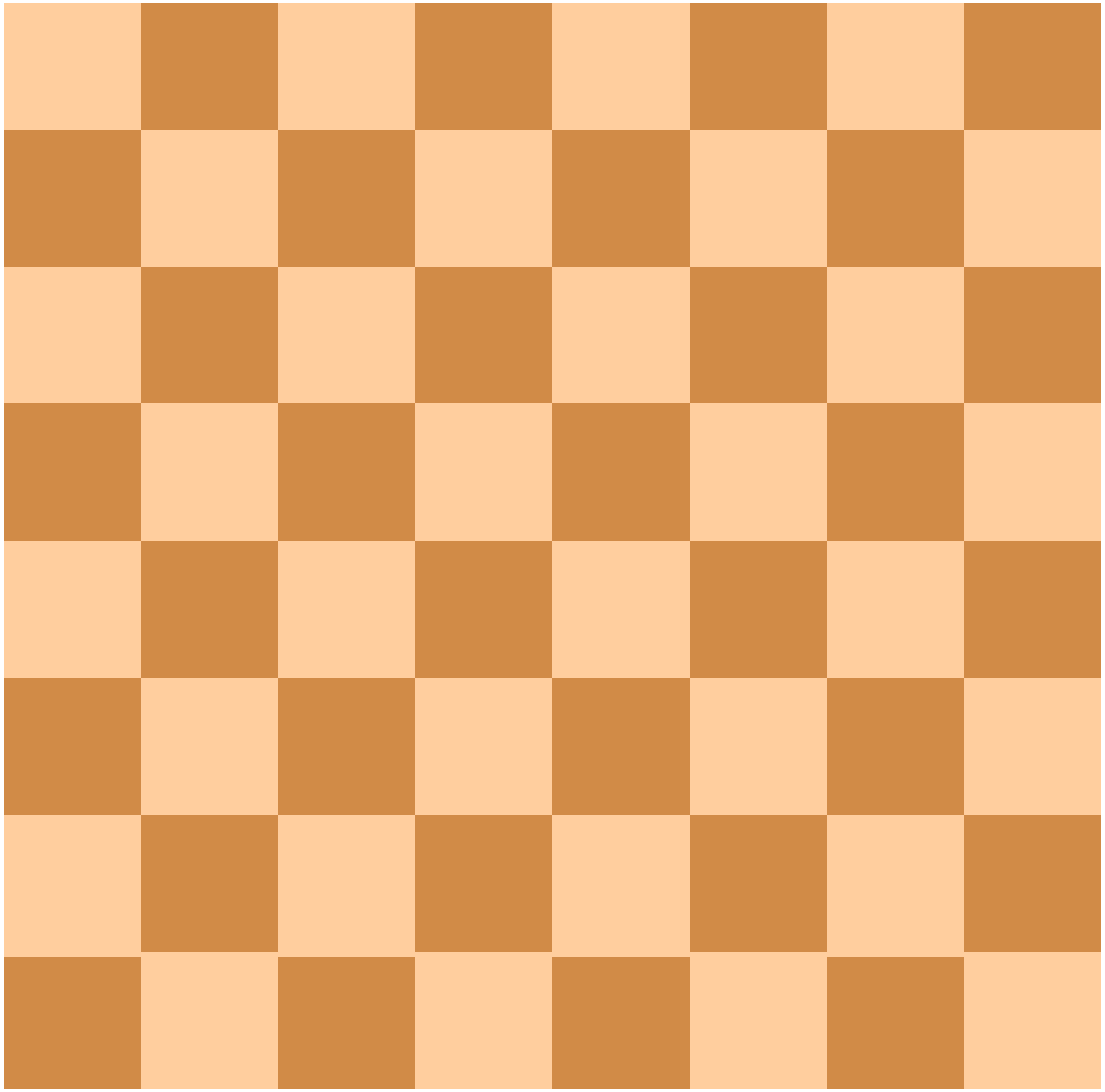
Reducing the Search Space

- The previous calculation includes set ups like this one



- Includes lots of set ups with multiple queens in the same column
- How many queens can there be in one column?
- Number of set ups
  8 * 8 * 8 * 8 * 8 * 8 * 8 * 8 = 16,777,216
- We have reduced search space by two orders of magnitude by applying some logic

Solving N Queens Approach



A Solution to 8 Queens

- If number of queens is fixed and I realize there can't be more than one queen per column I can iterate through the rows for each column

```
for(int r0 = 0; r0 < 8; r0++){
      board[r0][0] = 'q';
       for(int r1 = 0; r1 < 8; r1++){
            board[r1][1] = 'q';
            for(int r2 = 0; r2 < 8; r2++){
                  board[r2][2] = 'q';
                  // a little later
                  for(int r7 = 0; r7 < 8; r7++){
                        board[r7][7] = 'q';
                        if( queensAreSafe(board) )
                              printSolution(board);
                        board[r7][7] = ' '; //pick up queen
                  }
                  board[r6][6] = ' '; // pick up queen
```

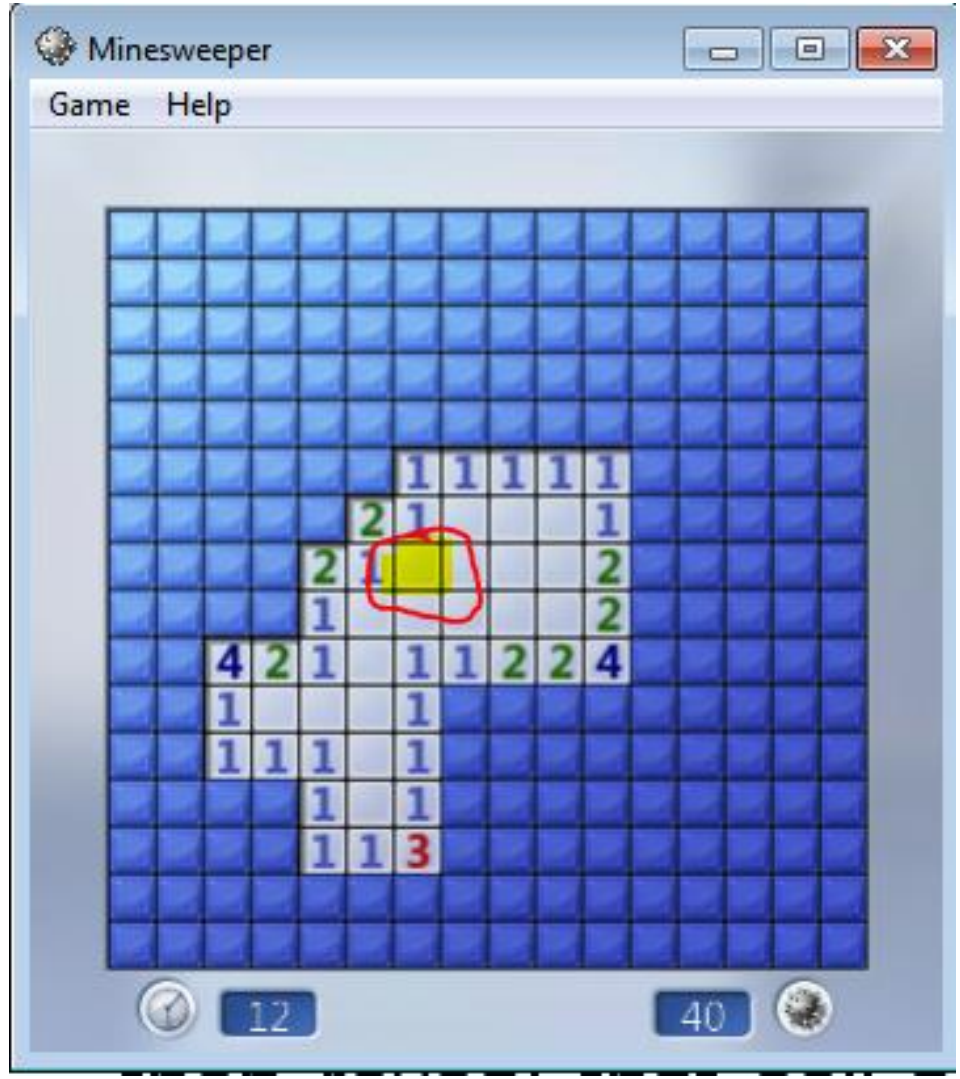
N Queens

- The *problem* with N queens is you don't know how many for loops to write.
- Do the problem recursively
- Write recursive code with class and demo
  - show backtracking with breakpoint and debugging option

Recursive Backtracking

- You must practice!!!
- Learn to recognize problems that fit the pattern
- Is a *kickoff* method needed?
- All solutions or a solution?
- Reporting results and acting on results

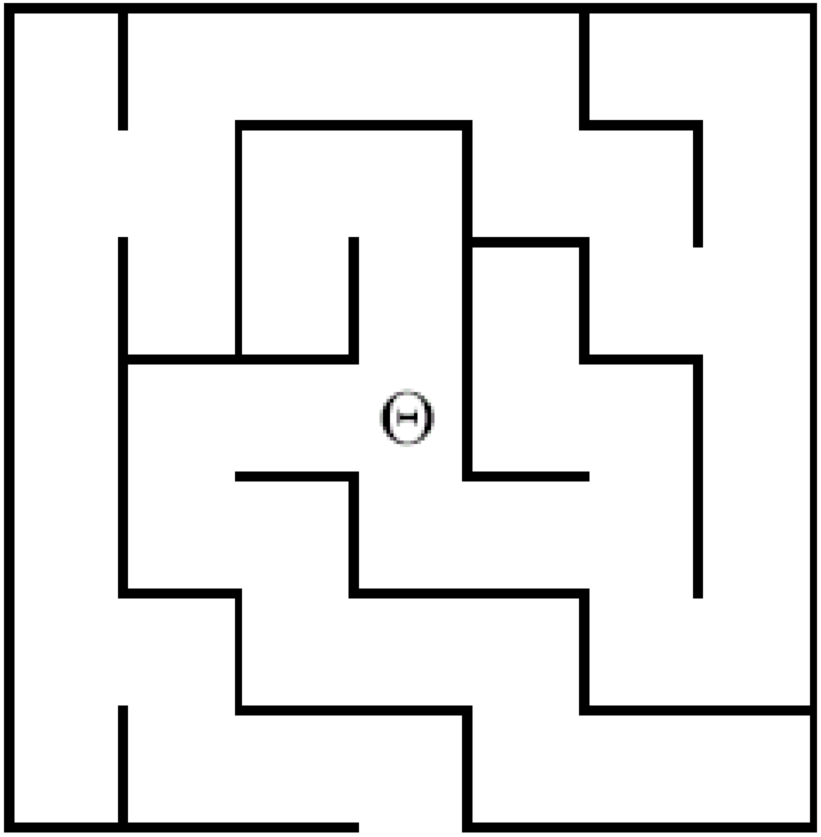
Minesweeper



Minesweeper Reveal Algorithm

- Minesweeper
- click a cell
  - if bomb game over
  - if cell that has 1 or more bombs on border
    then reveal the number of bombs that border cell
  - if a cell that has 0 bombs on border
    then reveal that cell as a blank and click on the 8 surrounding cells

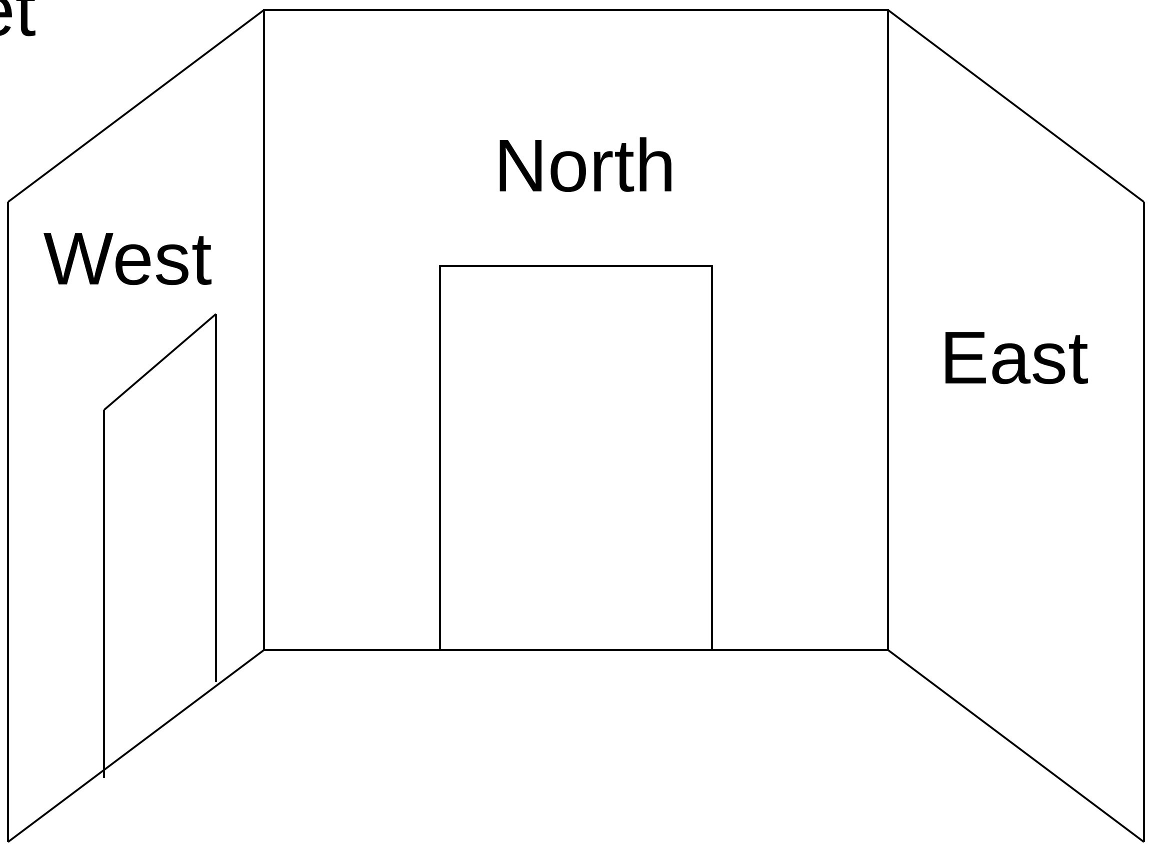
Another Backtracking Problem A Simple Maze



Search maze until way out is found. If no way out possible report that.

The Local View

Which way do
I go to get
out?



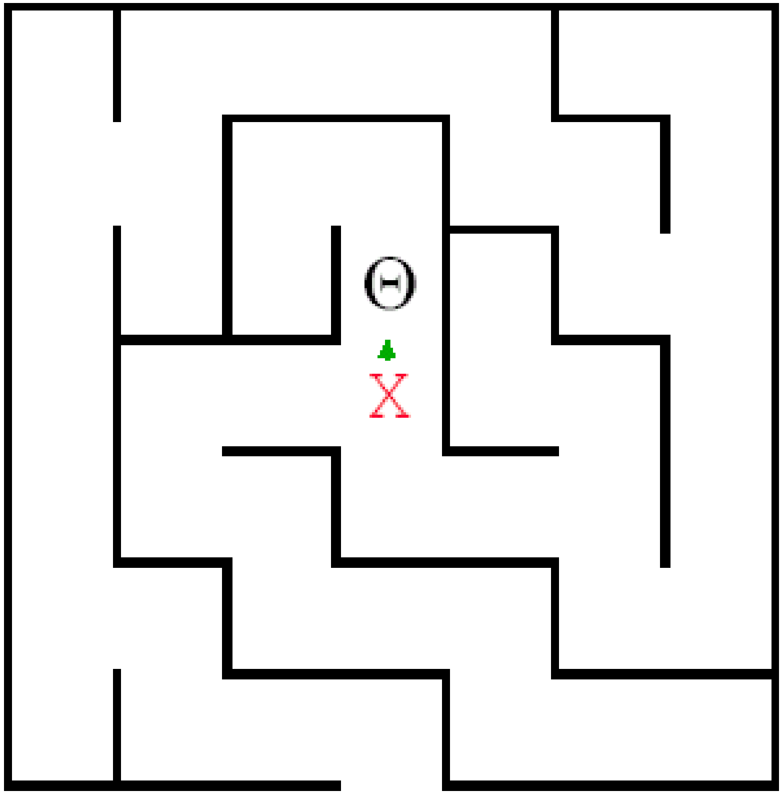
Behind me, to the South
is a door leading South

Modified Backtracking Algorithm for Maze

- If the current square is outside, return TRUE to indicate that a solution has been found.
  If the current square is marked, return FALSE to indicate that this path has been tried.
  Mark the current square.
  for **(**each of the four compass directions**)**
  { if **(** this direction is not blocked by a wall **)**
  { Move one step in the indicated direction from the current square.
  Try to solve the maze from there by making a recursive call.
  If this call shows the maze to be solvable, return TRUE to indicate that fact.
  }
  }
  Unmark the current square.

  Return FALSE to indicate that none of the four directions led to a solution.

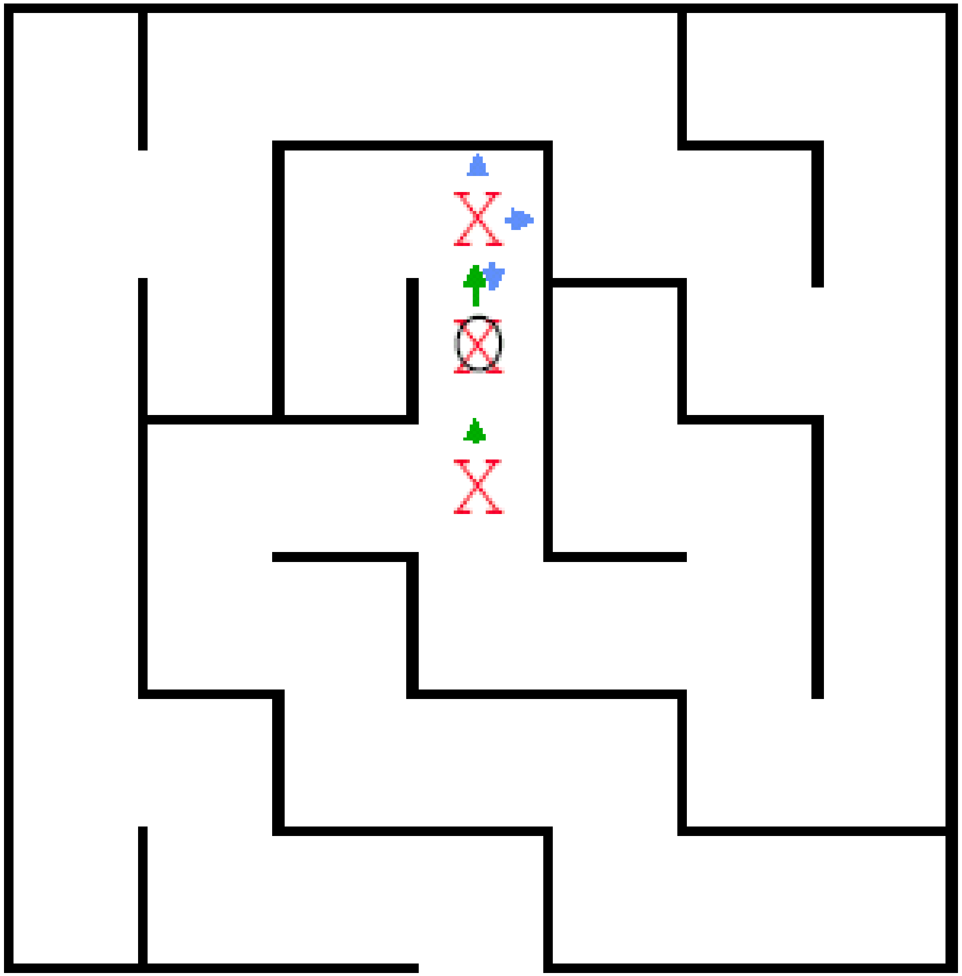
Backtracking in Action



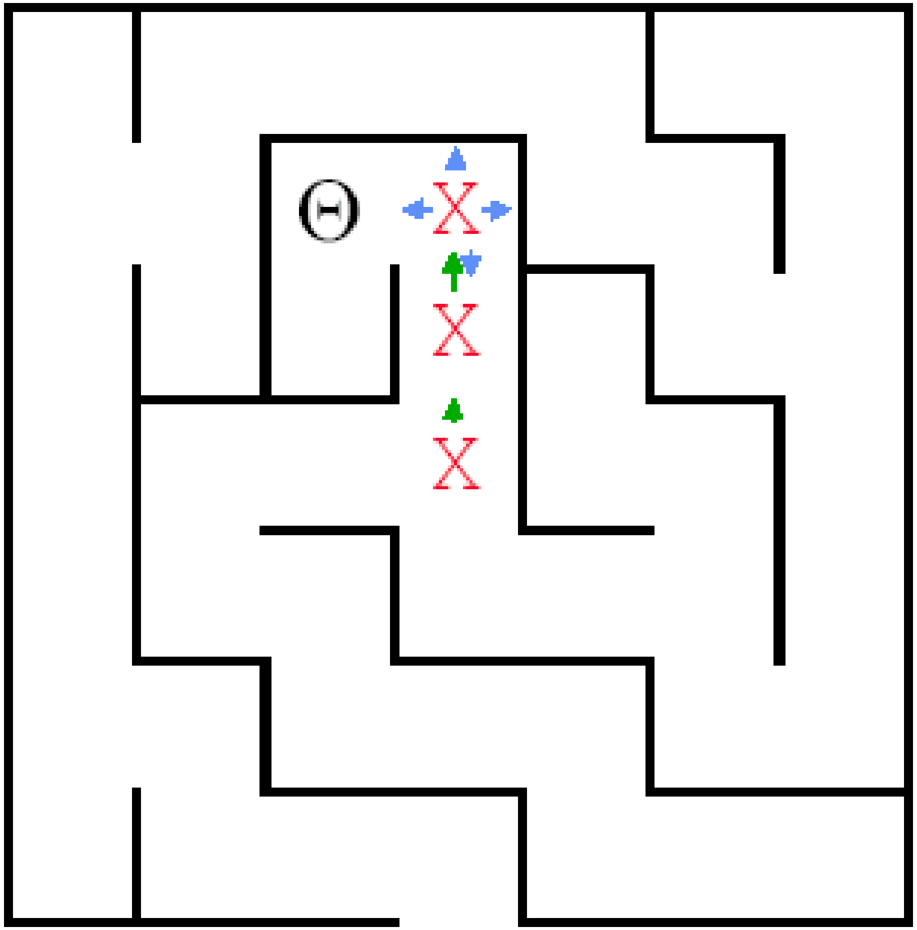
The crucial part of the algorithm is the for loop that takes us through the alternatives from the current square. Here we have moved to the North.

```
for (dir = North; dir <= West; dir++)
{     if (!WallExists(pt, dir))
      {if (SolveMaze(AdjacentPoint(pt, dir)))
            return(TRUE);
}
```

Backtracking in Action

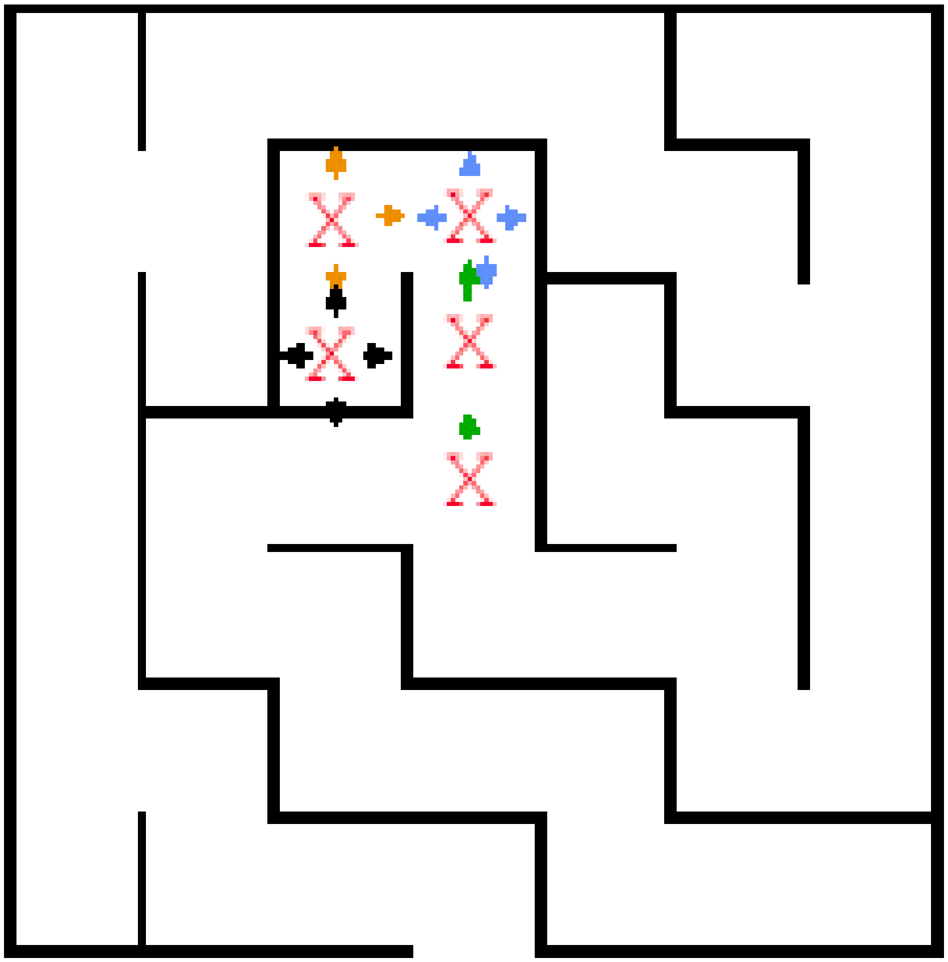


Here we have moved North again, but there is a wall to the North . East is also blocked, so we try South. That call discovers that the square is marked, so it just returns.



So the next move we can make is West.

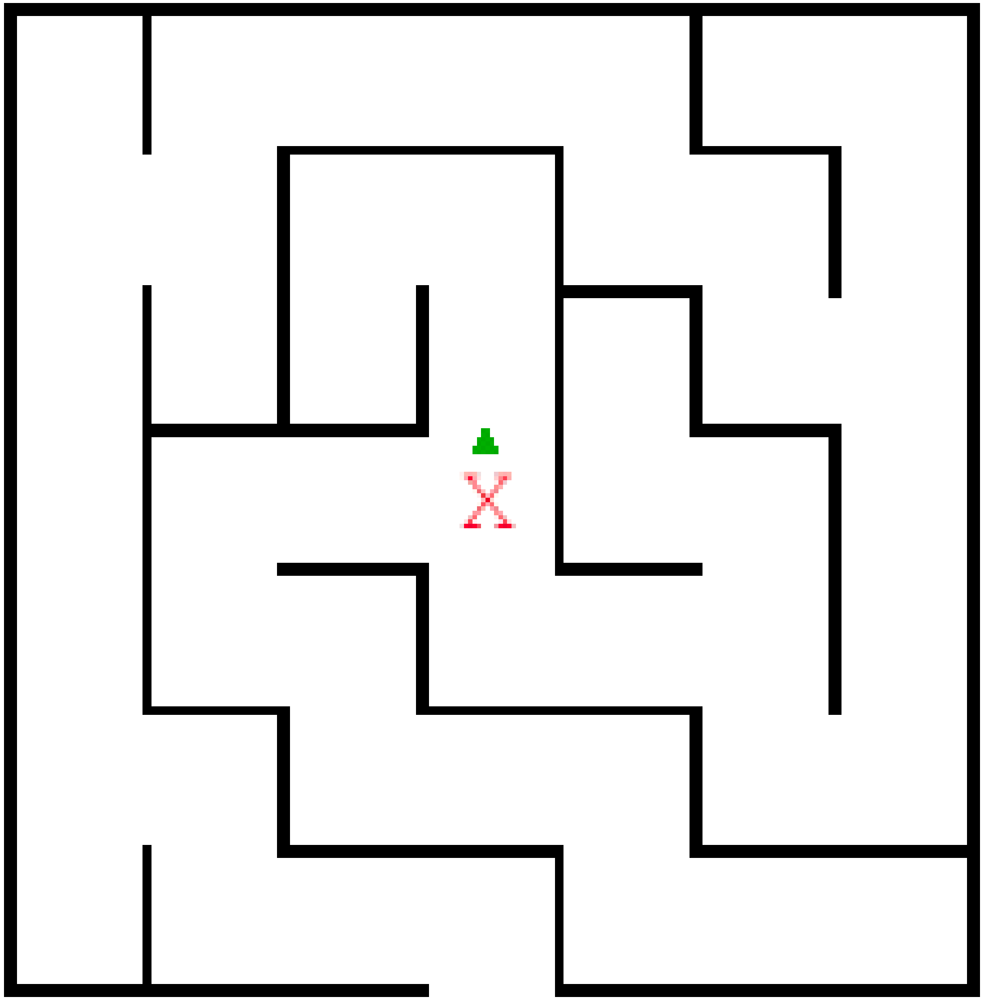
Where is this leading?



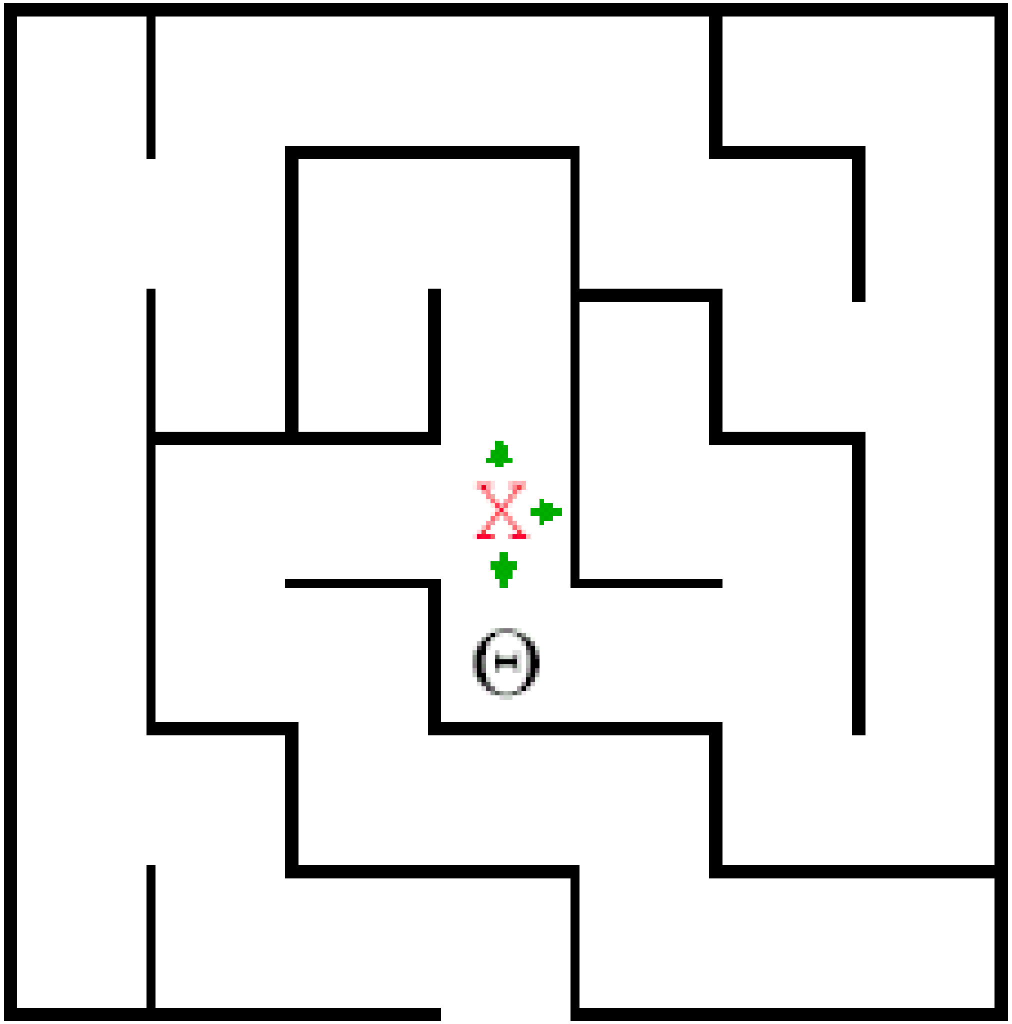
This path reaches a dead end.

Time to backtrack!

Remember the program stack!

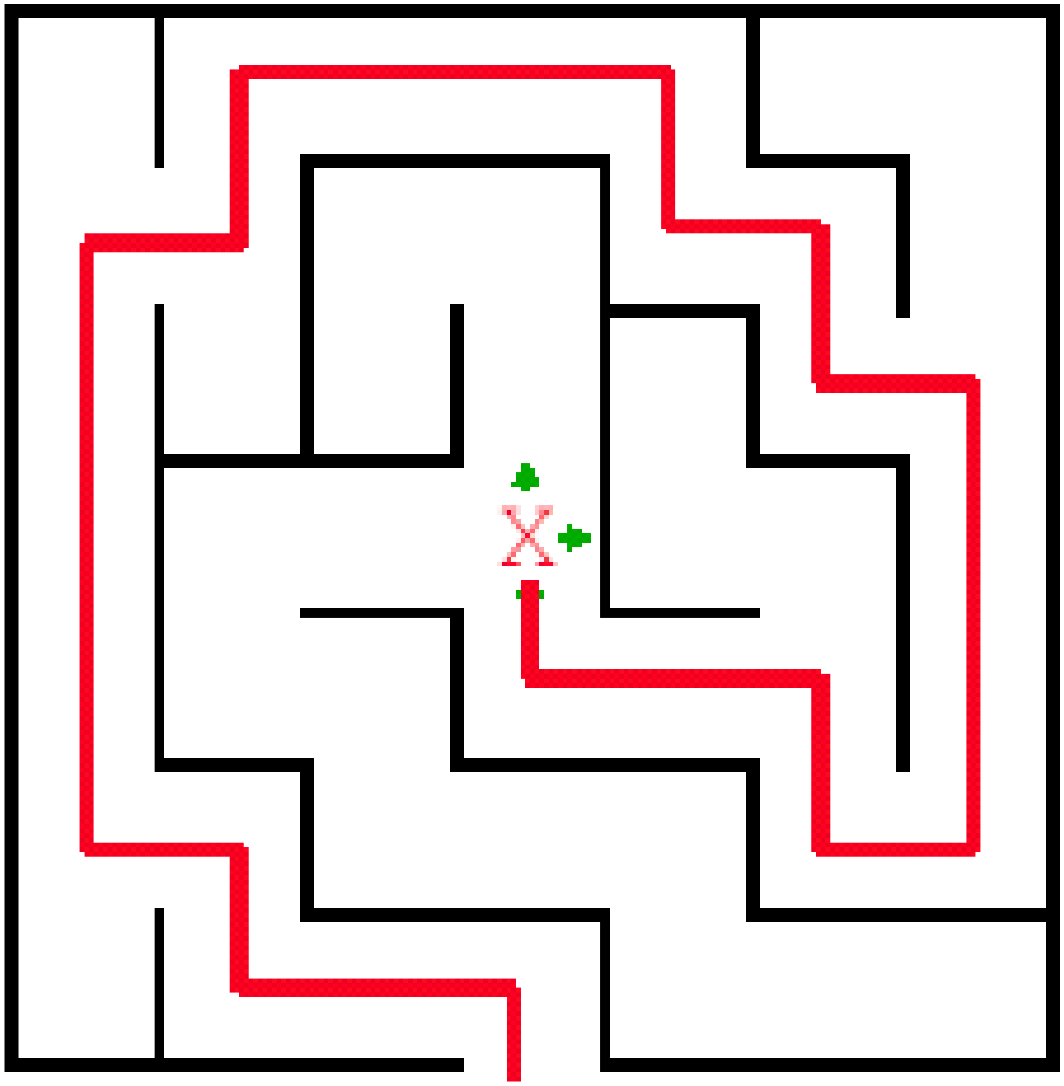


The recursive calls end and return until we find ourselves back here.



And now we try South

Path Eventually Found



More Backtracking Problems

Other Backtracking Problems

- Knight's Tour
- Regular Expressions
- Knapsack problem / Exhaustive Search
  - Filling a knapsack. Given a choice of items with various weights and a limited carrying capacity find the optimal load out. 50 lb. knapsack. items are 1 40 lb, 1 32 lb. 2 22 lbs, 1 15 lb, 1 5 lb. A greedy algorithm would choose the 40 lb item first. Then the 5 lb. Load out = 45lb. Exhaustive search 22 + 22 + 5 = 49.

The CD problem

- We want to put songs on a Compact Disc. 650MB CD and a bunch of songs of various sizes.

If there are no more songs to consider return result

else{

Consider the next song in the list.

Try not adding it to the CD so far and use recursion to evaluate best without it.

Try adding it to the CD, and use recursion to evaluate best with it

Whichever is better is returned as absolute best from here

}

Another Backtracking Problem

- Airlines give out frequent flier miles as a way to get people to always fly on their airline.
- Airlines also have partner airlines. Assume if you have miles on one airline you can redeem those miles on any of its partners.
- Further assume if you can redeem miles on a partner airline you can redeem miles on any of its partners and so forth...
  - Airlines don't usually allow this sort of thing.
- Given a list of airlines and each airlines partners determine if it is possible to redeem miles on a given airline A on another airline B.

Airline List – Part 1

- Delta
  - partners: Air Canada, Aero Mexico, OceanAir
- United
  - partners: Aria, Lufthansa, OceanAir, Quantas, British Airways
- Northwest
  - partners: Air Alaska, BMI, Avolar, EVA Air
- Canjet
  - partners: Girjet
- Air Canda
  - partners: Areo Mexico, Delta, Air Alaska
- Aero Mexico
  - partners: Delta, Air Canda, British Airways

Airline List - Part 2

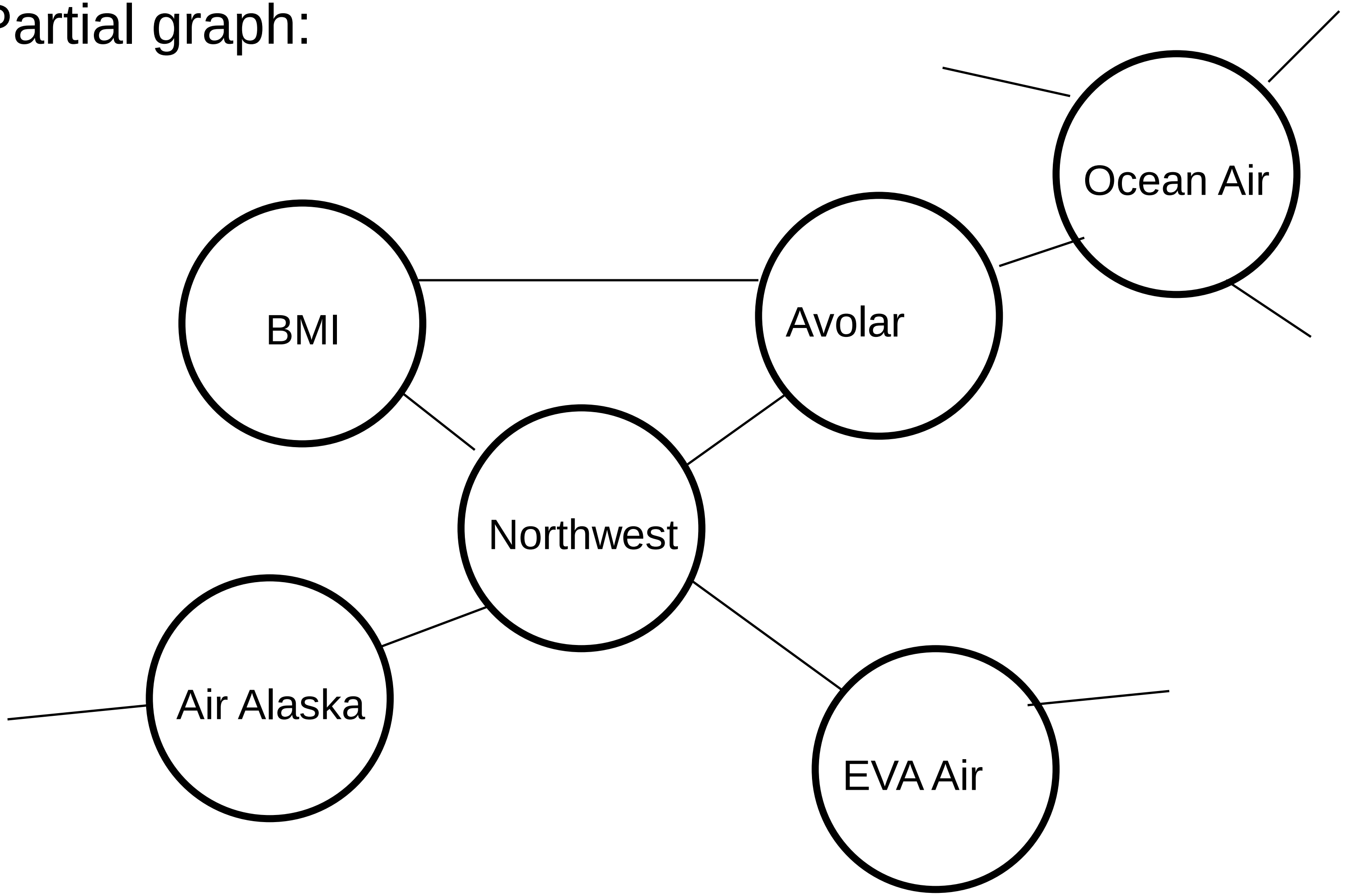
- Ocean Air
  - partners: Delta, United, Quantas, Avolar
- AlohaAir
  - partners: Quantas
- Aria
  - partners: United, Lufthansa
- Lufthansa
  - partners: United, Aria, EVA Air
- Quantas
  - partners: United, OceanAir, AlohaAir
- BMI
  - partners: Northwest, Avolar
- Maxair
  - partners: Southwest, Girjet

Airline List - Part 3

- Girjet
  - partners: Southwest, Canjet, Maxair
- British Airways
  - partners: United, Aero Mexico
- Air Alaska
  - partners: Northwest, Air Canada
- Avolar
  - partners: Northwest, Ocean Air, BMI
- EVA Air
  - partners: Northwest, Luftansa
- Southwest
  - partners: Girjet, Maxair

Problem Example

- If I have miles on Northwest can I redeem them on Aria?
- Partial graph:



Topic 14
Searching and Simple Sorts

"There's nothing in your head the sorting hat can't see. So try me on and I will tell you where you ought to be."

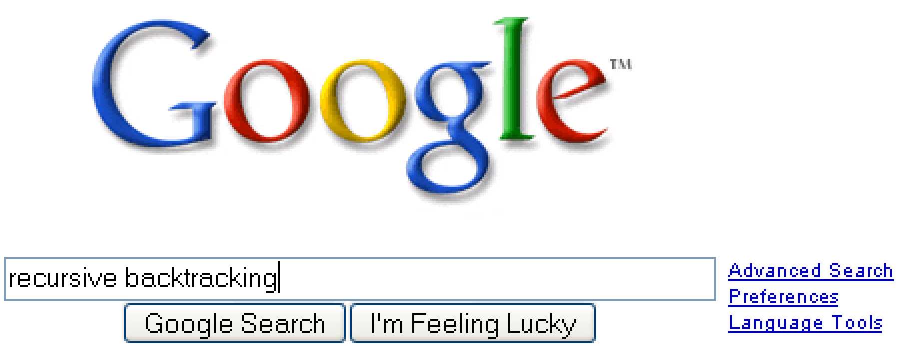
-The Sorting Hat, *Harry Potter and the Sorcerer's Stone*

Sorting and Searching

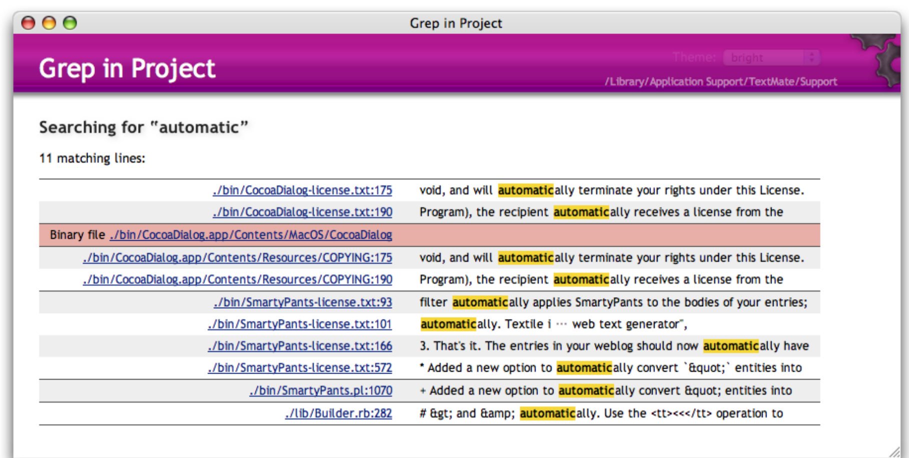
- Fundamental problems in computer science and programming
- Sorting done to make searching easier
- Multiple different algorithms to solve the same problem
  – How do we know which algorithm is "better"?
- Look at searching first
- Examples use arrays of ints to illustrate algorithms

Searching

iTunes
on!science
View
Search iTunes Store
Search Results
badmonkey0001



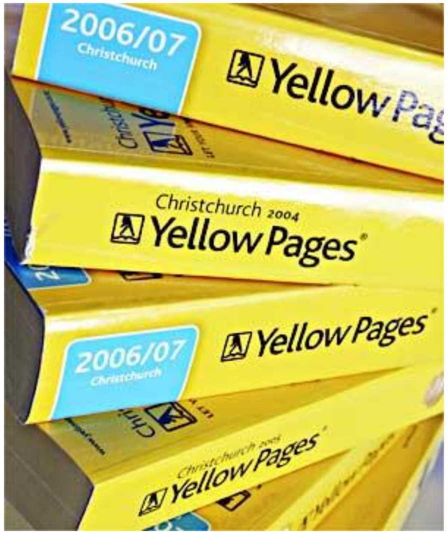
Google™
recursive backtracking
Google Search
I'm Feeling Lucky
Advanced Search
Preferences
Language Tools



Grep in Project
Grep in Project
/Library/Application Support/TextMate/Support
Searching for "automatic"
11 matching lines:
./bin/CocoaDialog-license.txt:175 void, and will automatically terminate your rights under this License.
./bin/CocoaDialog-license.txt:190 Program), the recipient automatically receives a license from the
Binary file ./bin/CocoaDialog.app/Contents/MacOS/CocoaDialog
./bin/CocoaDialog.app/Contents/Resources/COPYING:175 void, and will automatically terminate your rights under this License.
./bin/CocoaDialog.app/Contents/Resources/COPYING:190 Program), the recipient automatically receives a license from the
./bin/SmartyPants-license.txt:93 filter automatically applies SmartyPants to the bodies of your entries;
./bin/SmartyPants-license.txt:101 automatically. Textile i ··· web text generator",
./bin/SmartyPants-license.txt:166 3. That's it. The entries in your weblog should now automatically have
./bin/SmartyPants-license.txt:572 * Added a new option to automatically convert `"` entities into
./bin/SmartyPants.pl:1070 + Added a new option to automatically convert " entities into
./lib/Builder.rb:282 # > and & automatically. Use the <tt><<</tt> operation to

Searching

- Given an array or list of data find the location of a particular value or report that value is not present
- linear search
  - intuitive approach?
  - start at first item
  - is it the one I am looking for?
  - if not go to next item
  - repeat until found or all items checked
- If items not sorted or unsortable this approach is necessary



Linear Search

```
/*    pre: data != null
      post: return the index of the first occurrence
      of target in data or -1 if target not present in
      data
*/
public int linearSearch(int[] data, int target) {
    for (int i = 0; i < data.length; i++) {
        if (data[i] == target) {
            return i;
        }
    }
    return -1;
}
```

Linear Search, Generic

```
/*    pre: data != null, no elements of data == null
           target != null
      post: return the index of the first occurrence
      of target in data or -1 if target not present in
      data
*/
public int linearSearch(Object[] data, Object target) {
    for (int i = 0; i < data.length; i++)
        if (target.equals(data[i]))
            return i;
    return -1;
}
```

T(N)? Big O? Best case, worst case, average case?

Clicker 1

- What is the average case Big O of linear search in an array with N items, if an item is present once?

A. O(1)

B. O(logN)

C. O(N)

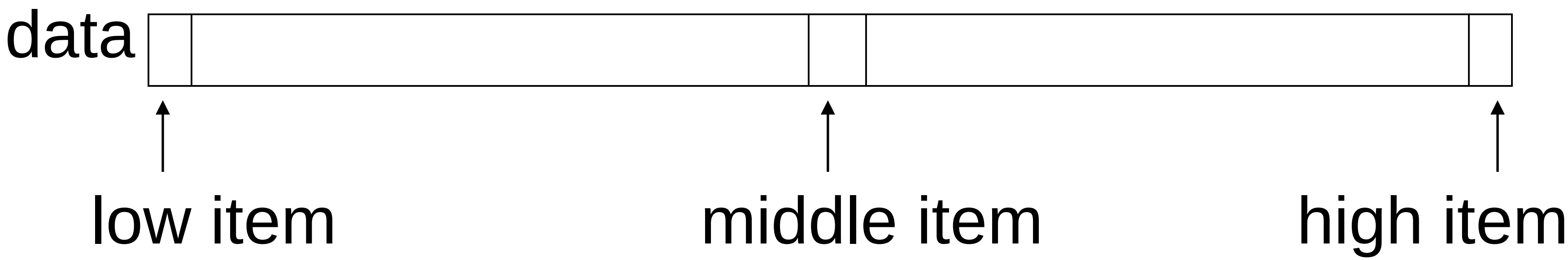
D. O(NlogN)

E. O(N^2)

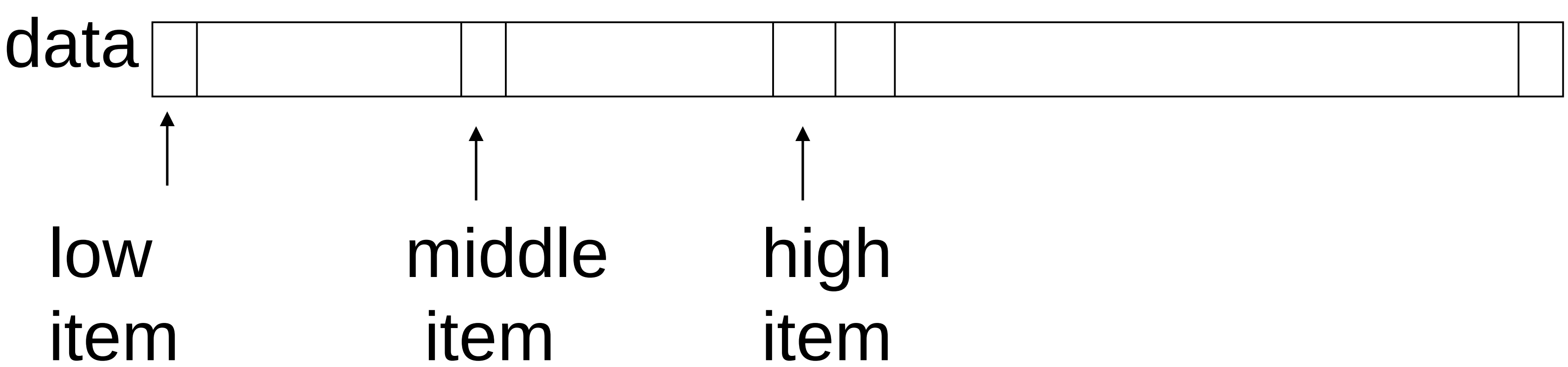
Searching in a Sorted Array or List

- If items are sorted then we can *divide and conquer*
- dividing your work in half with each step
  - generally a good thing
- The Binary Search with array in ascending order
  - Start at middle of list
  - is that the item?
  - If not is it less than or greater than the item?
  - less than, move to second half of list
  - greater than, move to first half of list
  - repeat until found or sub list size = 0

Binary Search



Is middle item what we are looking for? If not is it more or less than the target item? (Assume lower)



and so forth…

Binary Search in Action

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53

```
public static int bsearch(int[] data, int target) {
      int indexOfTarget = -1;
      int low = 0;
      int high = data.length - 1;
      while(indexOfTarget == -1 && low <= high ) {
            int mid = low + ((high - low) / 2);
            if( data[mid] == target )
                  indexOfTarget = mid;
            else if( data[mid] < target)
                  low = mid + 1;
            else.
                  high = mid - 1;
      }
      return indexOfTarget;
}
// mid = (low + high) / 2; // may overflow!!!
// or mid = (low + high) >>> 1; using bitwise op
```

Trace When Key == 3

Trace When Key == 30

Variables of Interest?

Clicker 2

What is the worst case Big O of binary search in an array with N items, if an item is present?

A. O(1)
B. O(logN)
C. O(N)
D. O(NlogN)
E. O(N^2)

Generic Binary Search

```
public static <T extends Comparable<? super T>> int
             bsearch(T[] data, T target) {

       int result = -1;
       int low = 0;
       int high = data.length - 1;
       while( result == -1 && low <= high ) {
              int mid = low + ((high - low) / 2);
              int compareResult = target.compareTo(data[mid]);
              if(compareResult == 0)
                     result = mid;
              else if(compareResult > 0)
                     low = mid + 1;
              else
                     high = mid - 1; // compareResult < 0
       }
       return result;
}
```

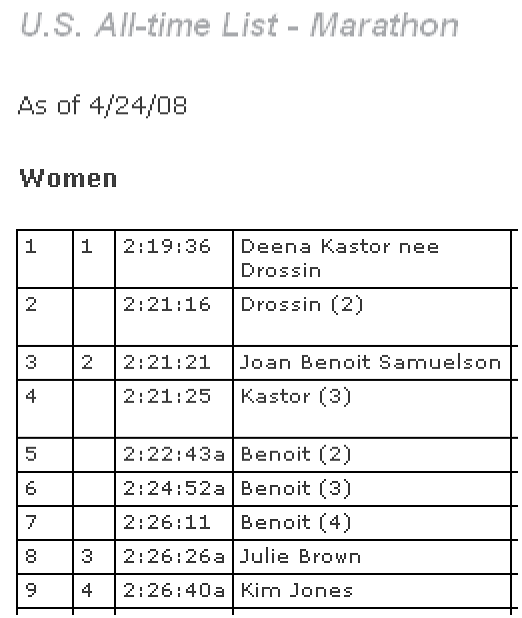
Recursive Binary Search

```
public static int bsearch(int[] data, int target) {
    return bsearch(data, target, 0, data.length - 1);
}

public static int bsearch(int[] data, int target,
                                      int low, int high) {
    if(low <= high){
        int mid = low + ((high - low) / 2);
        if( data[mid] == target )
            return mid;
        else if(data[mid] > target)
            return bsearch(data, target, low, mid - 1);
        else
            return bsearch(data, target, mid + 1, high);
    }
    return -1;
}
// Clicker 3 Is this a recursive backtracking algorithm?
A. NO
B. YES
```

Other Searching Algorithms

- Interpolation Search
  - more like what people really do
- Indexed Searching
- Binary Search Trees
- Hash Table Searching
- best-first
- A*

Sorting


U.S. All-time List - Marathon

As of 4/24/08

Women

1	1	2:19:36	Deena Kastor nee Drossin
2		2:21:16	Drossin (2)
3	2	2:21:21	Joan Benoit Samuelson
4		2:21:25	Kastor (3)
5		2:22:43a	Benoit (2)
6		2:24:52a	Benoit (3)
7		2:26:11	Benoit (4)
8	3	2:26:26a	Julie Brown
9	4	2:26:40a	Kim Jones

Song Name	Time	Track # ▲	Artist	Album
Letters from the Wasteland	4:29	1 of 10	The Wallflowers	Breach
When You're On Top	3:54	1 of 13	The Wallflowers	Red Letter Days
Hand Me Down	3:35	2 of 10	The Wallflowers	Breach
How Good It Can Get	4:11	2 of 13	The Wallflowers	Red Letter Days
Sleepwalker	3:31	3 of 10	The Wallflowers	Breach
Closer To You	3:17	3 of 13	The Wallflowers	Red Letter Days
I've Been Delivered	5:01	4 of 10	The Wallflowers	Breach
Everybody Out Of The Water	3:42	4 of 13	The Wallflowers	Red Letter Days
Witness	3:34	5 of 10	The Wallflowers	Breach
Three Ways	4:19	5 of 13	The Wallflowers	Red Letter Days
Some Flowers Bloom Dead	4:43	6 of 10	The Wallflowers	Breach
Too Late to Quit	3:54	6 of 13	The Wallflowers	Red Letter Days
Mourning Train	4:04	7 of 10	The Wallflowers	Breach
If You Never Got Sick	3:44	7 of 13	The Wallflowers	Red Letter Days
Up from Under	3:38	8 of 10	The Wallflowers	Breach
Health and Happiness	4:03	8 of 13	The Wallflowers	Red Letter Days
Murder 101	2:31	9 of 10	The Wallflowers	Breach
See You When I Get There	3:09	9 of 13	The Wallflowers	Red Letter Days
Birdcage	7:42	10 of 10	The Wallflowers	Breach
Feels Like Summer Again	3:48	10 of 13	The Wallflowers	Red Letter Days
Everything I Need	3:37	11 of 13	The Wallflowers	Red Letter Days
Here in Pleasantville	3:40	12 of 13	The Wallflowers	Red Letter Days
Empire in My Mind (Bonus Track)	3:31	13 of 13	The Wallflowers	Red Letter Days

Sorting

- A fundamental application for computation
- Done to make finding data (searching) faster
- Many different algorithms for sorting
- One of the difficulties with sorting is working with a fixed size storage container (array)
  – if resize, that is expensive (slow)
- The simple sorts are slow
  – bubble sort
  – selection sort
  – insertion sort

Selection sort

- Algorithm
  - Search through the data and find the smallest element
  - swap the smallest element with the first element
  - repeat starting at second element and find the second smallest element

```
public static void selectionSort(int[] data) {

      for (int i = 0; i < data.length - 1; i++) {
            int min = i;
            for (int j = i + 1; j < data.length; j++)
                  if (data[j] < data[min])
                        min = j;
            int temp = data[i];
            data[i] = data[min];
            data[min] = temp;
       }
}
```

Insertion Sort in Practice

44	68	191	119	119	37	83	82

What is the T(N), *actual* number of statements executed, of the selection sort code, given an array of N elements? What is the Big O?

Generic Selection Sort

```
public static <T extends Comparable<? super T>>
      void selectionSort(T[] data) {

    for(int i = 0; i < data.length - 1; i++) {
        int min = i;
        for(int j = i + 1; j < data.length; j++)
            if( data[min].compareTo(data[j]) > 0 )
                min = j;
        T temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}
```

Insertion Sort

- Another of the $O(N^2)$ sorts
- The first item is sorted
- Compare the second item to the first
  – if smaller swap
- Third item, compare to item next to it
  – need to swap
  – after swap compare again
- And so forth…

Insertion Sort Code

```
public void insertionSort(int[] data) {
    for (int i = 1; i < data.length; i++) {
        int temp = data[i];
        int j = i;
        while (j > 0 && temp < data[j - 1]) {
            // swap elements
            data[j] = data[j - 1];
            data[j - 1] = temp;
            j--;
        }
    }
}
```

- Best case, worst case, average case Big O?

Clicker 4 - Comparing Algorithms

- Which algorithm do you think has a smaller T(N) given random data, selection sort or insertion sort?

A. Insertion Sort

B. Selection Sort

C. About the same

Topic 15
Implementing and Using Stacks

"stack n.
The set of things a person has to do in the future. "I haven't done it yet because every time I pop my stack something new gets pushed." If you are interrupted several times in the middle of a conversation, "My stack overflowed" means "I forget what we were talking about."

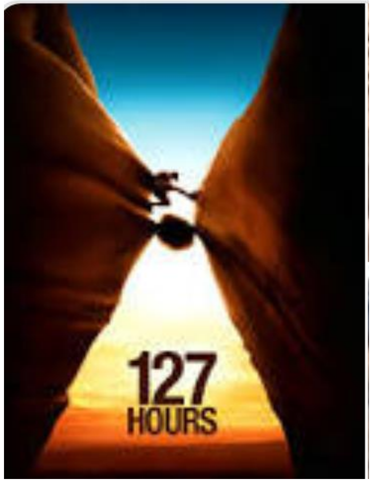
-The Hacker's Dictionary

Friedrich L. Bauer
German computer scientist who proposed "stack method of expression evaluation" in 1955.

Sharper Tools

Stacks

Lists

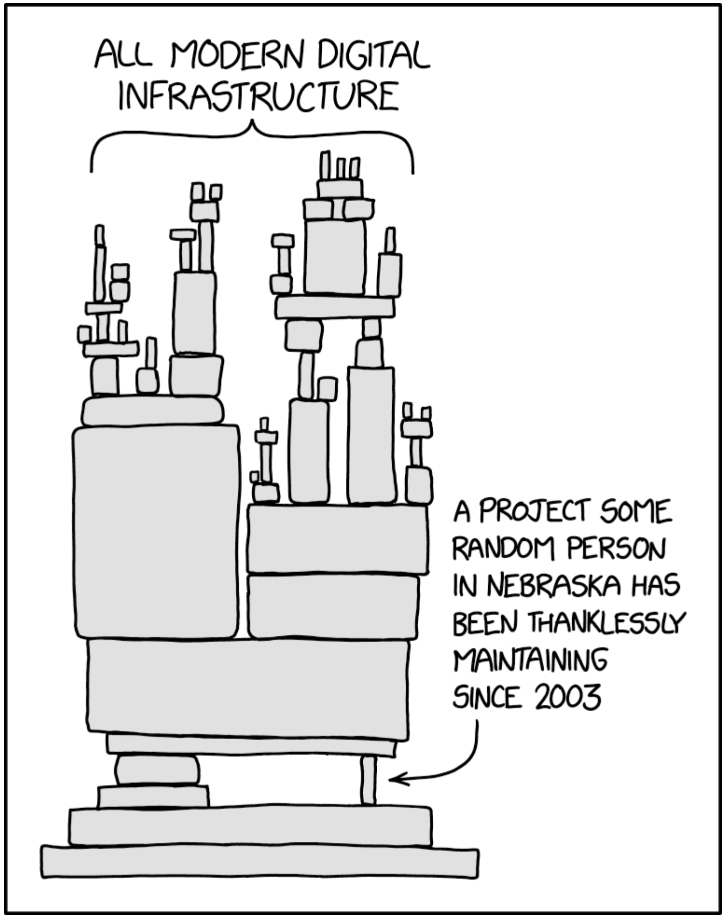


Stacks

- Access is allowed only at one point of the structure, normally termed the *top* of the stack
  - access to the most recently added item only
- Operations are limited:
  - push (add item to stack)
  - pop (remove top item from stack)
  - top (get top item without removing it)
  - isEmpty
- Described as a "Last In First Out" (LIFO) data structure

Implementing a stack

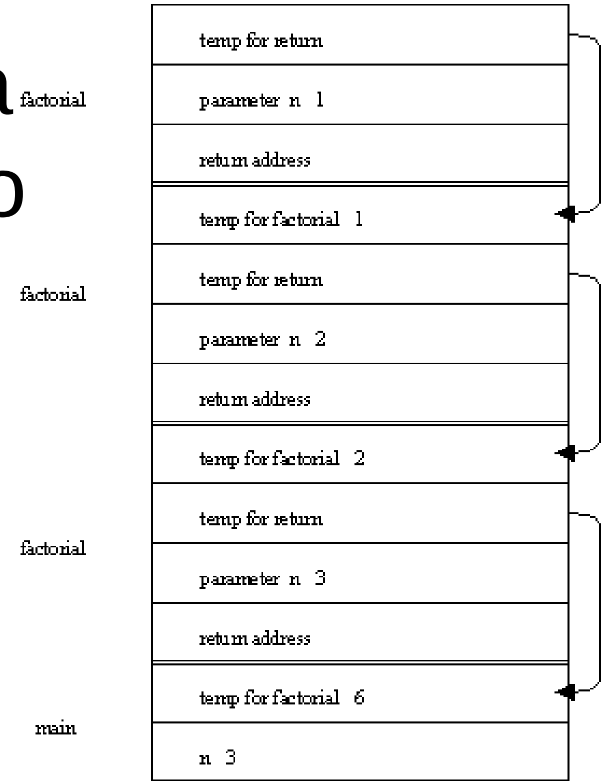
- need an underlying collection to hold the elements of the stack
- 3 obvious choices?
  - native array
  - linked structure of nodes
  - a list!!!
- Adding a *layer of abstraction*. A HUGE idea.
- array implementation
- linked list implementation

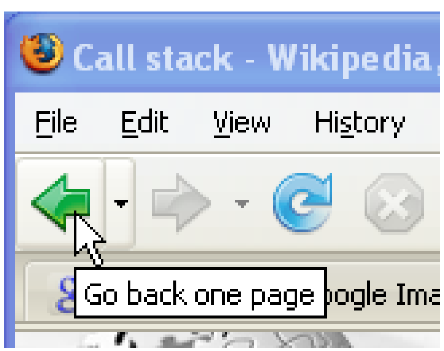


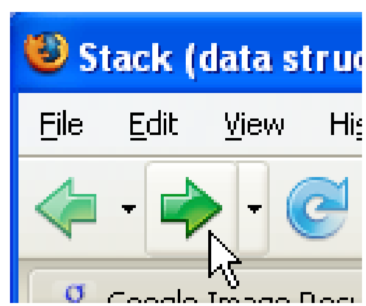
https://xkcd.com/2347/

Uses of Stacks

- The runtime stack used by a process (running program) to keep track of methods in progress
- Search problems
- Undo, redo, back, forward







Stack Operations

Assume a simple stack for integers.

Stack<Integer> s = new Stack<>();

s.push(12);

s.push(4);

s.push(s.top() + 2);

s.pop();

s.push(s.top());

//what are contents of stack?

Clicker 1 - What is Output?

```
Stack<Integer> s = new Stack<>();
// put stuff in stack
for (int i = 0; i < 5; i++)
    s.push(i);
// Print out contents of stack.
// Assume there is a size method.
for (int i = 0; i < s.size(); i++)
    System.out.print(s.pop() + " ");
```

A `0 1 2 3 4`

B `4 3 2 1 0`

C `4 3 2`

D `2 3 4`

E No output due to runtime error

Corrected Version

```
Stack<Integer> s = new Stack<Integer>();
// put stuff in stack
for (int i = 0; i < 5; i++)
    s.push(i);
// print out contents of stack
// while emptying it
final int LIMIT = s.size();
for (int i = 0; i < LIMIT; i++)
    System.out.print(s.pop() + " ");
//or
//  while (!s.isEmpty())
//      System.out.println(s.pop());
```

Stack Operations

Write a method to print out contents of stack in reverse order.

Applications of Stacks

Mathematical Calculations

- What does 3 + 2 * 4 equal?
  2 * 4 + 3?   3 * 2 + 4?
- The precedence of operators affects the order of operations.
- A mathematical expression cannot simply be evaluated left to right.
- A challenge when evaluating a program.
- *Lexical analysis* is the process of interpreting a program.

What about 1 - 2 - 4 ^ 5 * 3 * 6 / 7 ^ 2 ^ 3

Infix and Postfix Expressions

- The way we are use to writing expressions is known as infix notation
- Postfix expression does not
- require any precedence rules
- 3 2 * 1 + is postfix of 3 * 2 + 1
- evaluate the following postfix expressions and write out a corresponding infix expression:

2 3 2 4 * + * 1 2 3 4 ^ * +

1 2 - 3 2 ^ 3 * 6 / + 2 5 ^ 1 -

Clicker Question 2

- What does the following postfix expression evaluate to?

  6 3 2 + *

A. 11

B. 18

C. 24

D. 30

E. 36

Evaluation of Postfix Expressions

- Easy to do with a stack
- given a proper postfix expression:
  - get the next token
  - if it is an operand push it onto the stack
  - else if it is an operator
    - pop the stack for the right hand operand
    - pop the stack for the left hand operand
    - apply the operator to the two operands
    - push the result onto the stack
  - when the expression has been exhausted the result is the top (and only element) of the stack

Infix to Postfix

- Convert the following equations from infix to postfix:

  2 ^ 3 ^ 3 + 5 * 1

  11 + 2 - 1 * 3 / 3 + 2 ^ 2 / 3

  Problems:

    Negative numbers?

    parentheses in expression

Infix to Postfix Conversion

- Requires operator precedence parsing algorithm
  - parse v. To determine the syntactic structure of a sentence or other utterance

Operands: add to expression

Close parenthesis: pop stack symbols until an open parenthesis appears

Operators:

Have an on stack and off stack precedence

Pop all stack symbols until a symbol of lower precedence appears. Then push the operator

End of input: Pop all remaining stack symbols and add to the expression

Simple Example

Infix Expression: 3 + 2 * 4

PostFix Expression:

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Simple Example

Infix Expression: + 2 * 4

PostFix Expression: 3

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Simple Example

Infix Expression: 2 * 4

PostFix Expression: 3

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Simple Example

Infix Expression: * 4

PostFix Expression: 3 2

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Simple Example

Infix Expression: 4

PostFix Expression: 3 2

Operator Stack: + *

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Simple Example

Infix Expression:

PostFix Expression: 3 2 4

Operator Stack: + *

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Simple Example

Infix Expression:

PostFix Expression: 3 2 4 *

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Simple Example

Infix Expression:

PostFix Expression: 3 2 4 * +

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

Example

11 + 2 ^ 4 ^ 3 - ((4 + 5) * 6) ^ 2

Show algorithm in action on above equation

Balanced Symbol Checking

- In processing programs and working with computer languages there are many instances when symbols must be balanced

  { } , [ ] , ( )

  A stack is useful for checking symbol balance. When a closing symbol is found it must match the most recent opening symbol of the same type.
- Applicable to checking html and xml tags!

Algorithm for Balanced Symbol Checking

- Make an empty stack
- read symbols until end of file
  - if the symbol is an opening symbol push it onto the stack
  - if it is a closing symbol do the following
    - if the stack is empty report an error
    - otherwise pop the stack. If the symbol popped does not match the closing symbol report an error
- At the end of the file if the stack is not empty report an error

Algorithm in practice

- list[i] = 3 * (44 - method(foo(list[2 * (i + 1) + foo(list[i - 1])) / 2 *) - list[method(list[0])];

- Complications
  - when is it not an error to have non matching symbols?

- Processing a file
  - *Tokenization*: the process of scanning an input stream. Each independent chunk is a token.
- Tokens may be made up of 1 or more characters

Topic 16
Queues

"FISH queue: n.

[acronym, by analogy with FIFO (First In, First Out)] ‘First In, Still Here’. A joking way of pointing out that processing of a particular sequence of events or requests has stopped dead. Also FISH mode and FISHnet; the latter may be applied to any network that is running really slowly or exhibiting extreme flakiness."

-The Jargon File 4.4.7

Queues

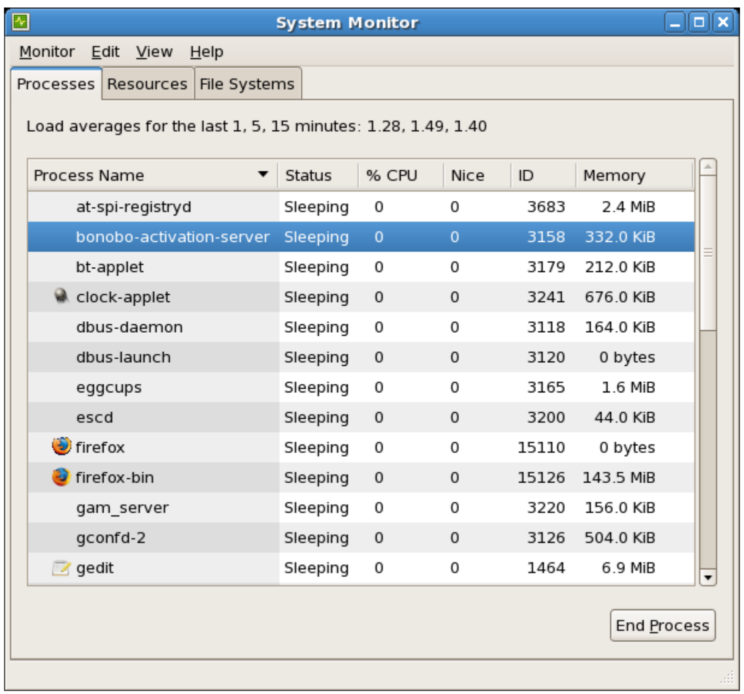
- A sharp tool, like stacks
- A line
  - In England people don't "get in line" they "queue up".

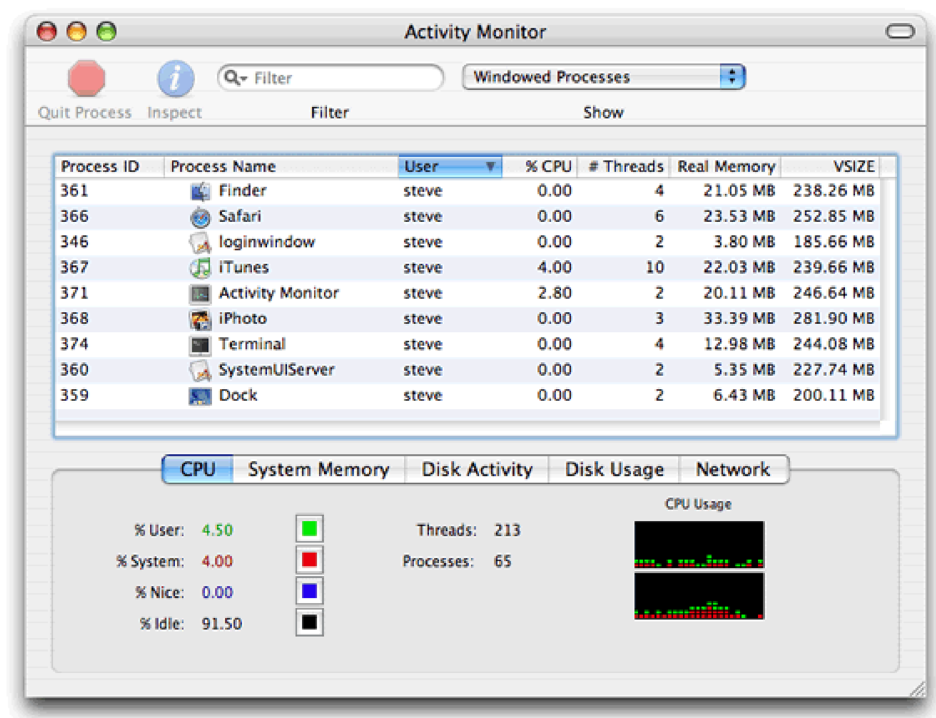
Queue Properties

- Queues are a first in first out data structure
  - FIFO (or LILO, but I guess that sounds a bit silly)
- Add items to the end of the queue
- Access and remove from the front
  - Access to the element that has been in the structure the ***longest*** amount of time
- Used extensively in operating systems
  - Queues of processes, I/O requests, and much more

Queues in Operating Systems

- On a computer with N cores on the CPU, but more than N processes, how many processes can actually be executing at one time?
- One job of OS, schedule the processes for the CPU





Queue operations

- `void enqueue(E item)`
  - a.k.a. `add(E item)`
- `E front()`
  - a.k.a. `E peek()`
- `E dequeue()`
  - a.k.a. `E remove()`
- `boolean isEmpty()`
- Specify methods in an interface, allow multiple implementations.

Queue interface, version 1

```
public interface Queue314<E> {
      //place item at back of this queue
      public void enqueue(E item);

      //access item at front of this queue
      //pre: !isEmpty()
      public E front();

      //remove item at front of this queue
      //pre: !isEmpty()
      public E dequeue();

      public boolean isEmpty();
}
```

Implementing a Queue

- Given the internal storage container and choice for front and back of queue what are the Big O of the queue operations?

	ArrayList	LinkedList (Singly Linked)	LinkedList (Doubly Linked)
enqueue			
front			
dequeue			
isEmpty			

Clicker 1

- If implementing a queue with a singly linked list with references to the first and last nodes (head and tail) which end of the list should be the front of the queue in order to have all queue operations O(1)?

A. The front of the list should be the front of the queue.

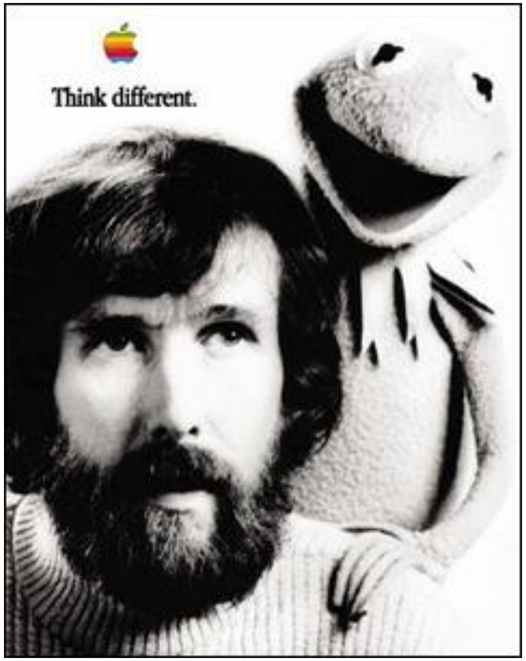
B. The back of the list should be the front of the queue.

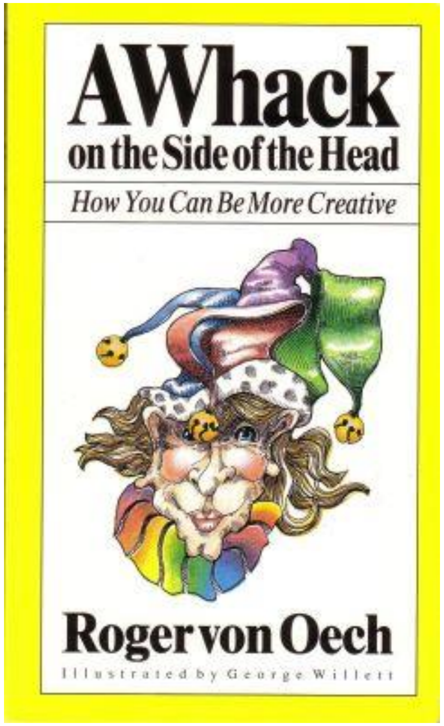
C. Either end will work to make all ops O(1).

D. Neither end will allow all ops to be O(1).

Alternate Implementation

- How about implementing a Queue with a native array?
  - Seems like a step backwards





Application of Queues

- Radix Sort
  - radix is a synonym for *base.* base 10, base 2
- Multi pass sorting algorithm that ***only*** looks at individual digits during each pass
- Use queues as *buckets* to store elements
- Create an array of 10 queues
- Starting with the least significant digit place value in queue that matches digit
- empty queues back into array
- repeat, moving to next least significant digit

Radix Sort in Action: 1s place

- original values in array

  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Look at ones place

  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Array of Queues (all empty initially):

0	5
1	6
2	7
3	8
4	9

Radix Sort in Action: 1s

- original values in array

  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Look at ones place

  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Queues:

0	70, 40	5	
1		6	86
2	12, 252, 12	7	37, 7
3	113, 93	8	
4		9	9, 79

Radix Sort in Action: 10s

- Empty queues in order from 0 to 9 back into array

  70, 40, 12, 252, 12, 113, 93, 86, 37, 7, 9, 79

- Now look at 10's place

  70, 40, 12, 252, 12, 113, 93, 86, 37, 7, 9, 79

- Queues:

0 7, 9	5 252
1 12, 12, 113	6
2	7 70, 79
3 37	8 86
4 40	9 93

Radix Sort in Action: 100s

- Empty queues in order from 0 to 9 back into array

  7, 9, 12, 12, 113, 37, 40, 252, 70, 79, 86, 93

- Now look at 100's place

  __7, __9, _12, _12, 113, _37, _40, 252, _70, _79, _86, _93

- Queues:

0	_7, _9, _12, _12, _37, _40, _70, _79, _86, _93	5	
1	113	6	
2	252	7	
3		8	
4		9	

Radix Sort in Action: Final Step

- Empty queues in order from 0 to 9 back into array

  7, 9, 12, 12, 40, 70, 79, 86, 93, 113, 252

Radix Sort Code

```
public static void sort(int[] data){
      ArrayList<Queue<Integer>> queues
           = new ArrayList<Queue<Integer>>();
      for(int i = 0; i < 10; i++)
          queues.add(new LinkedList<Integer>());
      int passes = numDigits(getMax(data));// helper methods

      for(int i = 0; i < passes; i++){
          for(int j = 0; j < data.length; j++) {
              int digit = valueOfDigit(data[j], i);
              queues.get(digit).add(data[j]);
          }
          int pos = 0;
          for(Queue<Integer> q : queues){
              while(!q.isEmpty())
                  data[pos++] = q.remove();
          }
      }
}
```

Topic 17
Faster Sorting

"The bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems."

- Don Knuth

Previous Sorts

- Insertion Sort and Selection Sort are both average case $O(N^2)$
- Today we will look at two faster sorting algorithms.
  - quicksort
  - mergesort

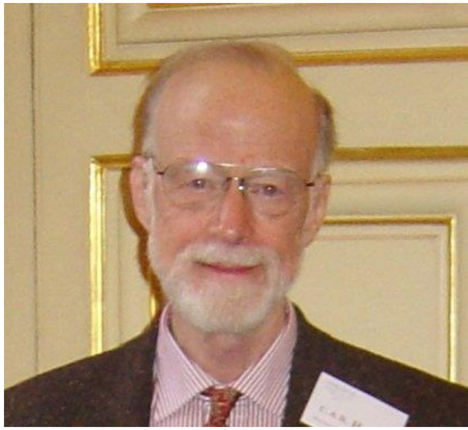
Properties of Sorting Algorithms

- In place?
  - Do we use another data structure or not?
  - Program stack *typically* not considered another data structure if only using O(log N) space
- Comparison?
  - Works by comparing the items to be sorted to each other?
  - *How could we not?*
- Stable?
  - Next slide!

Stable Sorting

- A property of sorts
- If a sort guarantees the relative order of equal items stays the same then it is a *stable sort*
- $[7_1, 6, 7_2, 5, 1, 2, 7_3, -5]$ original data
  - subscripts added for clarity
- $[-5, 1, 2, 5, 6, 7_1, 7_2, 7_3]$ sorted data
  - result of stable sort
- Real world example:
  - sort a table in Wikipedia by one criteria, then another
  - sort by country, then by major wins

Quicksort



- Created by C.A.R. (Tony) Hoare
- A divide and conquer approach

1. If the list has 0 or 1 elements it is sorted
2. otherwise, pick any element p in the list. This is called the ***pivot*** value - If we are unlucky with the chosen pivot, things can go poorly.
3. ***Partition*** the list minus the pivot into two sub lists according to values less than or greater than the pivot. (equal values go to either)
4. return the quicksort of the first list followed by the quicksort of the second list

Quicksort in Action

39 23 17 90 33 72 46 79 11 52 64 5 71

Pick middle element as pivot: 46

Partition list

23 17 5 33 39 11   46   79 72 52 64 90 71

quick sort the less than list

Pick middle element as pivot: 33

23 17 5 11   33   39

quicksort the less than list, pivot now 5

{}   5   23 17 11

quicksort the less than list, base case

quicksort the greater than list

Pick middle element as pivot: 17

and so on….

Quicksort on Another Data Set

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
44	68	191	119	119	37	83	95	191	45	158	130	76	153	39	25

Big O of Quicksort?

```java
private static void swapReferences(Object[] a, int index1, int index2) {
    Object tmp = a[index1];
    a[index1] = a[index2];
    a[index2] = tmp;
}

 private void quicksort(Comparable[] data, int start, int stop) {
     if(start < stop) {
          int pivotIndex = (start + stop) / 2;

          // Place pivot at start position
          swapReferences(data, pivotIndex, start);
          Comparable pivot = data[start];

          // Begin partitioning
          int j = start;

          // from first to j are elements less than or equal to pivot
          // from j to i are elements greater than pivot
          // elements beyond i have not been checked yet
          for(int i = start + 1; i <= stop; i++ ) {
              //is current element less than or equal to pivot
              if (data[i].compareTo(pivot) <= 0) {
                  // if so move it to the less than or equal portion
                  j++;
                  swapReferences(data, i, j);
              }
          }

          //restore pivot to correct spot
          swapReferences(data, start, j);
          quicksort( data, start, j - 1 );    // Sort small elements
          quicksort( data, j + 1, stop );   // Sort large elements
      } // else start >= stop, 0 or 1 element, base case, do nothing
  }
```

Clicker 1

- What are the best case and worst case Orders (Big O) for quicksort?

	Best	Worst
A.	$O(N\log N)$	$O(N^2)$
B.	$O(N^2)$	$O(N^2)$
C.	$O(N^2)$	$O(N!)$
D.	$O(N\log N)$	$O(N\log N)$
E.	$O(N)$	$O(N\log N)$

Clicker 2

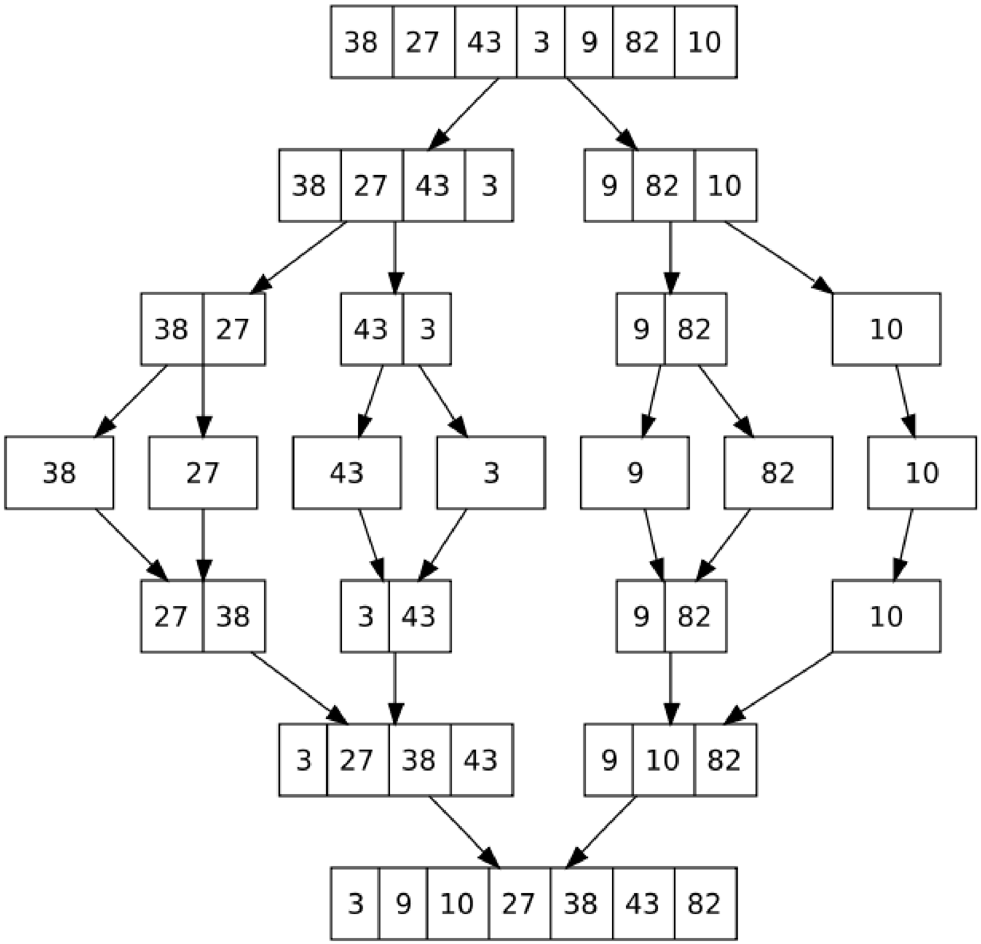
- Is quicksort always stable?
  A. No
  B. Yes

Merge Sort Algorithm

Don Knuth cites John von Neumann as the creator of this algorithm

1. If a list has 1 element or 0 elements it is sorted
2. If a list has more than 1 split into 2 separate lists
3. Perform this algorithm on each of those smaller lists
4. Take the 2 sorted lists and merge them together

Merge Sort



When implementing one temporary array is used instead of multiple temporary arrays.

Why?

Merge Sort code

```
/**
 * perform a merge sort on the elements of data
 * @param data data != null, all elements of data
 * are the same data type
 */
public static void mergeSort(Comparable[] data) {
       Comparable[] temp = new Comparable[data.length];
       sort(data, temp, 0, data.length - 1);
}

private static void sort(Comparable[] data, Comparable[] temp,
                           int low, int high) {
       if(low < high) {
              int center = (low + high) / 2;
              sort(data, temp, low, center);
              sort(data, temp, center + 1, high);
              merge(data, temp, low, center + 1, high);
       }
}
```

Merge Sort Code

```
private static void merge( Comparable[] data, Comparable[] temp,
                              int leftPos, int rightPos, int rightEnd) {
          int leftEnd = rightPos - 1;
          int tempPos = leftPos;
          int numElements = rightEnd - leftPos + 1;
          //main loop
          while(leftPos <= leftEnd && rightPos <= rightEnd){
                    if( data[leftPos].compareTo(data[rightPos]) <= 0) {
                              temp[tempPos] = data[leftPos];
                              leftPos++;
                    } else {
                              temp[tempPos] = data[rightPos];
                              rightPos++;
                    }
                    tempPos++;
          }
          //copy rest of left half
          while (leftPos <= leftEnd) {
                    temp[tempPos] = data[leftPos];
                    tempPos++;
                    leftPos++;
          }
          //copy rest of right half
          while (rightPos <= rightEnd) {
                    temp[tempPos] = data[rightPos];
                    tempPos++;
                    rightPos++;
          }
          //Copy temp back into data
          for (int i = 0; i < numElements; i++, rightEnd--)
                    data[rightEnd] = temp[rightEnd];
}
```

Clicker 3

- What are the best case and worst case Orders (Big O) for mergesort?

	Best	Worst
A.	$O(N\log N)$	$O(N^2)$
B.	$O(N^2)$	$O(N^2)$
C.	$O(N^2)$	$O(N!)$
D.	$O(N\log N)$	$O(N\log N)$
E.	$O(N)$	$O(N\log N)$

Clicker 4

- Is mergesort always stable?

  A. No

  B. Yes

Clicker 5

- You have 1,000,000 distinct items in random order that you will be searching. How many searches need to be performed before the data is changed to make it worthwhile to sort the data before searching?

A. ~40

B. ~100

C. ~500

D. ~2,000

E. ~500,000

Comparison of Various Sorts (2001)

Num Items	Selection	Insertion	Quicksort
1000	0.016	0.005	0 ??
2000	0.059	0.049	0.006
4000	0.271	0.175	0.005
8000	1.056	0.686	0??
16000	4.203	2.754	0.011
32000	16.852	11.039	0.045
64000	expected?	expected?	0.068
128000	expected?	expected?	0.158
256000	expected?	expected?	0.335
512000	expected?	expected?	0.722
1024000	expected?	expected?	1.550

times in seconds

Comparison of Various Sorts (2011)

Num Items	Selection	Insertion	Quicksort	Merge	Arrays.sort
1000	0.002	0.001	-	-	-
2000	0.002	0.001	-	-	-
4000	0.006	0.004	-	-	-
8000	0.022	0.018	-	-	-
16000	0.086	0.070	0.002	0.002	0.002
32000	0.341	0.280	0.004	0.005	0.003
64000	1.352	1.123	0.008	0.010	0.007
128000	5.394	4.499	0.017	0.022	0.015
256000	21.560	18.060	0.035	0.047	0.031
512000	86.083	72.303	0.072	0.099	0.066
1024000	???	???	0.152	0.206	0.138
2048000			0.317	0.434	0.287
4096000			0.663	0.911	0.601
8192000			1.375	1.885	1.246

Comparison of Various Sorts (2020)

Num Items	Selection	Insertion	Quicksort	Mergesort	Arrays. sort(int)	Arrays.sort(Integer)	Arrays. parallelSort
1,000	<0.001	<0.001	-	-	-	-	-
2,000	0.001	<0.001	-	-	-	-	-
4,000	0.004	0.003	-	-	-	-	Speeds
8,000	0.017	0.010	-	-	-	-	up????
16,000	0.065	0.040	0.002	0.002	0.003	0.011	0.007
32,000	0.258	0.160	0.002	0.003	0.002	0.008	0.003
64,000	1.110	0.696	0.005	0.008	0.004	0.011	0.001
128,000	4.172	2.645	0.011	0.015	0.009	0.024	0.002
256,000	16.48	10.76	0.024	0.034	0.018	0.051	0.004
512,000	70.38	47.18	0.049	0.068	0.040	0.114	0.008
1,024,000	-	-	0.098	0.143	0.082	0.259	0.017
2,048,000	-	-	0.205	0.296	0.184	0.637	0.035
4,096,000	-	-	0.450	0.659	0.383	1.452	0.079
8,192,000	-	-	0.941	1.372	0.786	3.354	0.148

Concluding Thoughts

- Language libraries often have sorting algorithms in them
  - Java Arrays and Collections classes
  - C++ Standard Template Library
  - Python sort and sorted functions
- Hybrid sorts
  - when size of unsorted list or portion of array is small use insertion sort, otherwise use O(N log N) sort like Quicksort or Mergesort

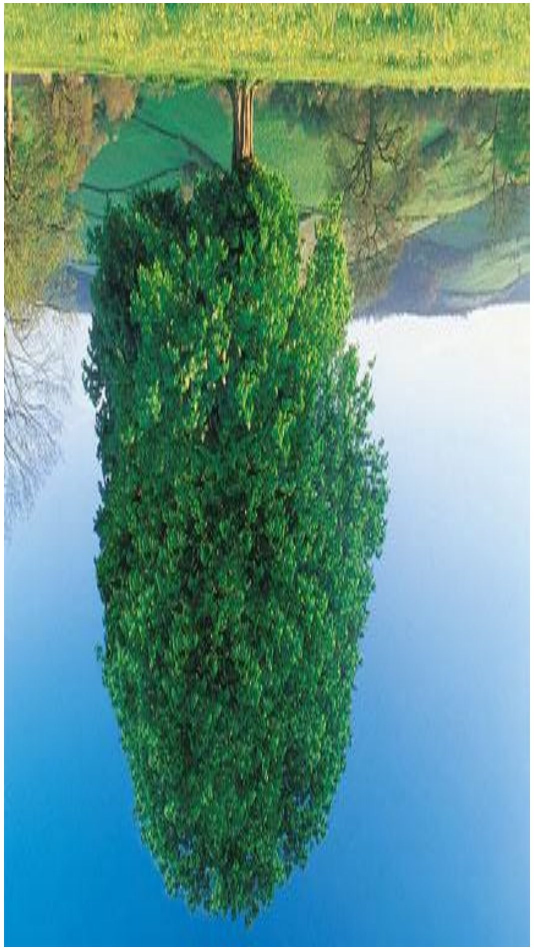
Concluding Thoughts

- Sorts still being created!
- Timsort (2002)
  - created for python version 2.3
  - now used in Java version 7.0+
  - takes advantage of real world data
  - real world data is usually partially sorted, not totally random
- Library Sort (2006)
  - Like insertion sort, but leaves gaps for later elements

Topic 18
Binary Trees

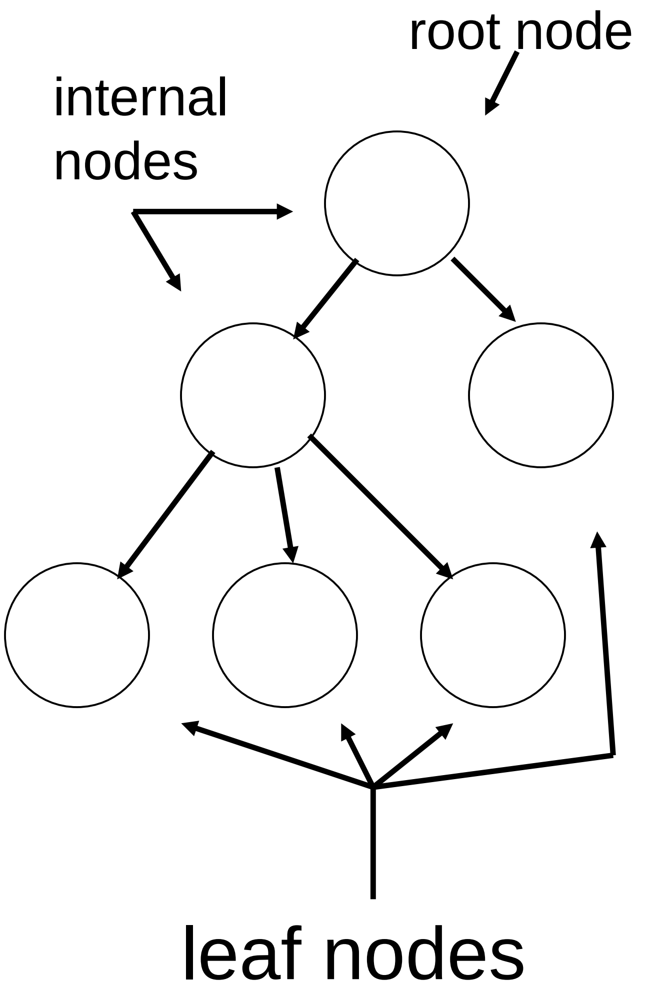
"A tree may grow a thousand feet tall, but its leaves will return to its roots."

-Chinese Proverb



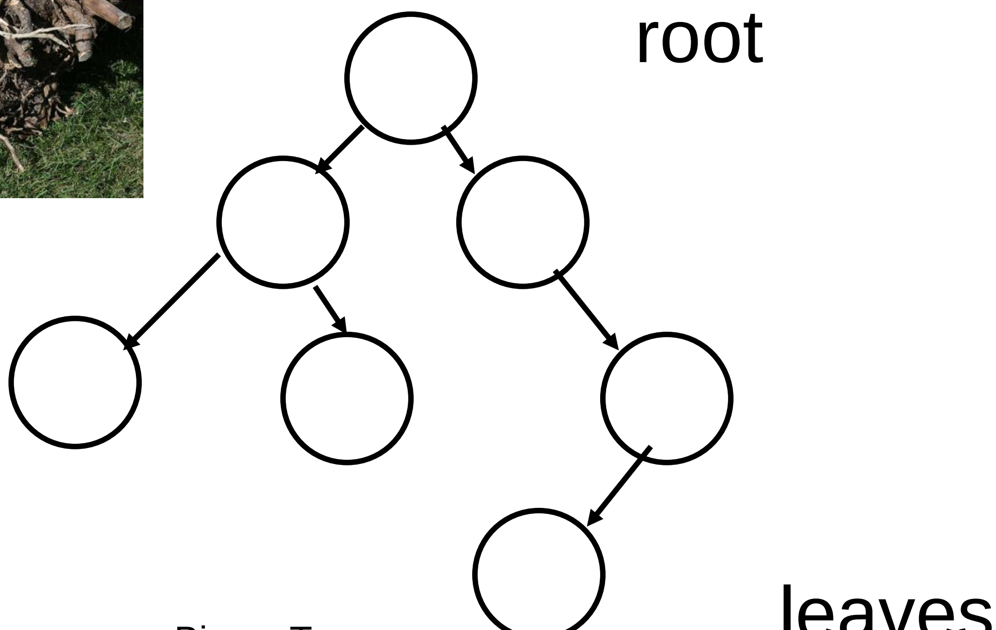
Definitions

- A *tree* data structure
  – one entry point, the ***root***
  – Each node is either a ***leaf*** or an *internal node*
  – An internal node has 1 or more ***children***, nodes that can be reached directly from that internal node.
  – The internal node is said to be the ***parent*** of its child nodes



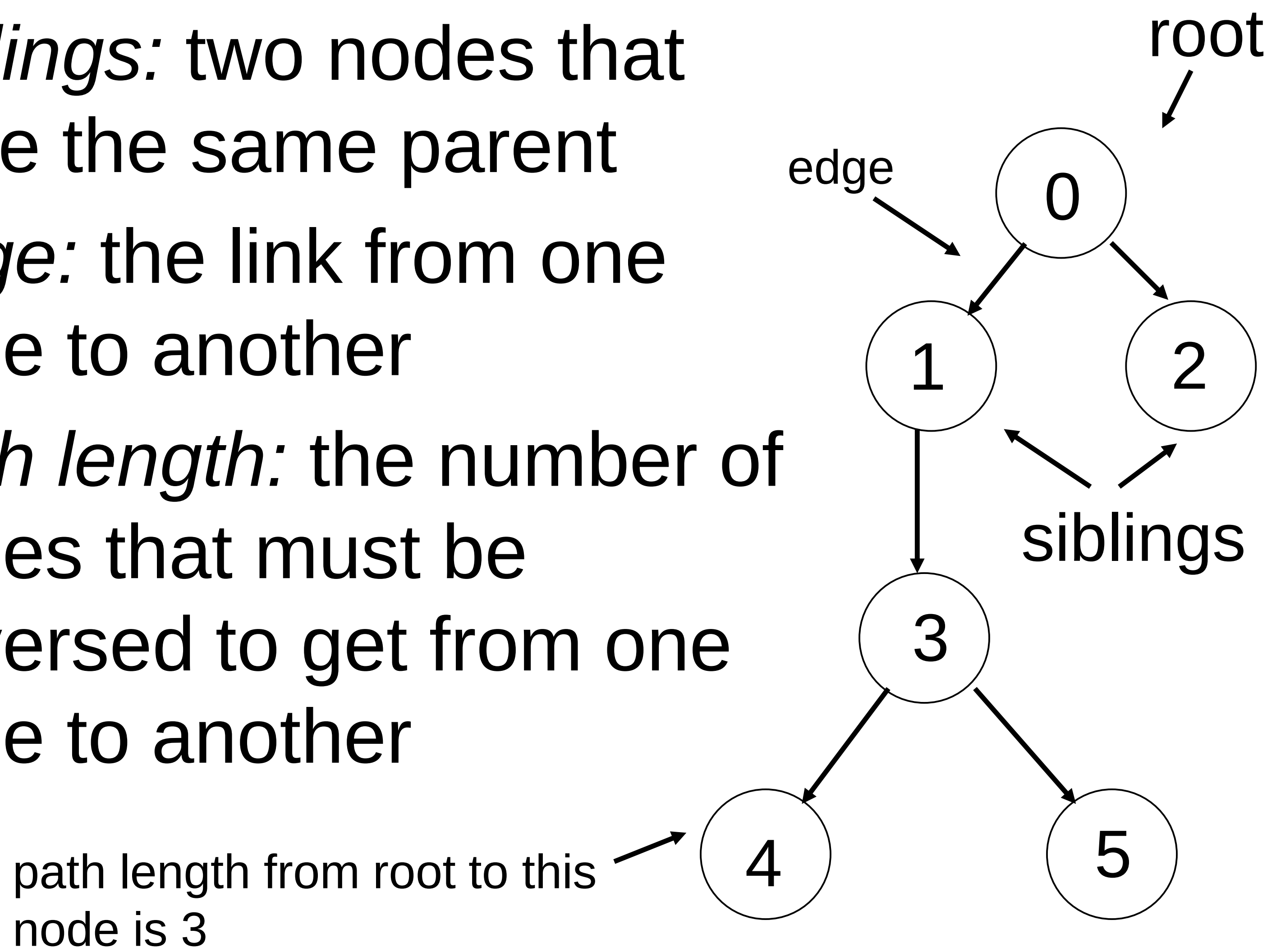
Properties of Trees

- Only access point is the root
- All nodes, except the root, have one parent
  – like the inheritance hierarchy in Java
- Traditionally trees drawn upside down



Properties of Trees and Nodes

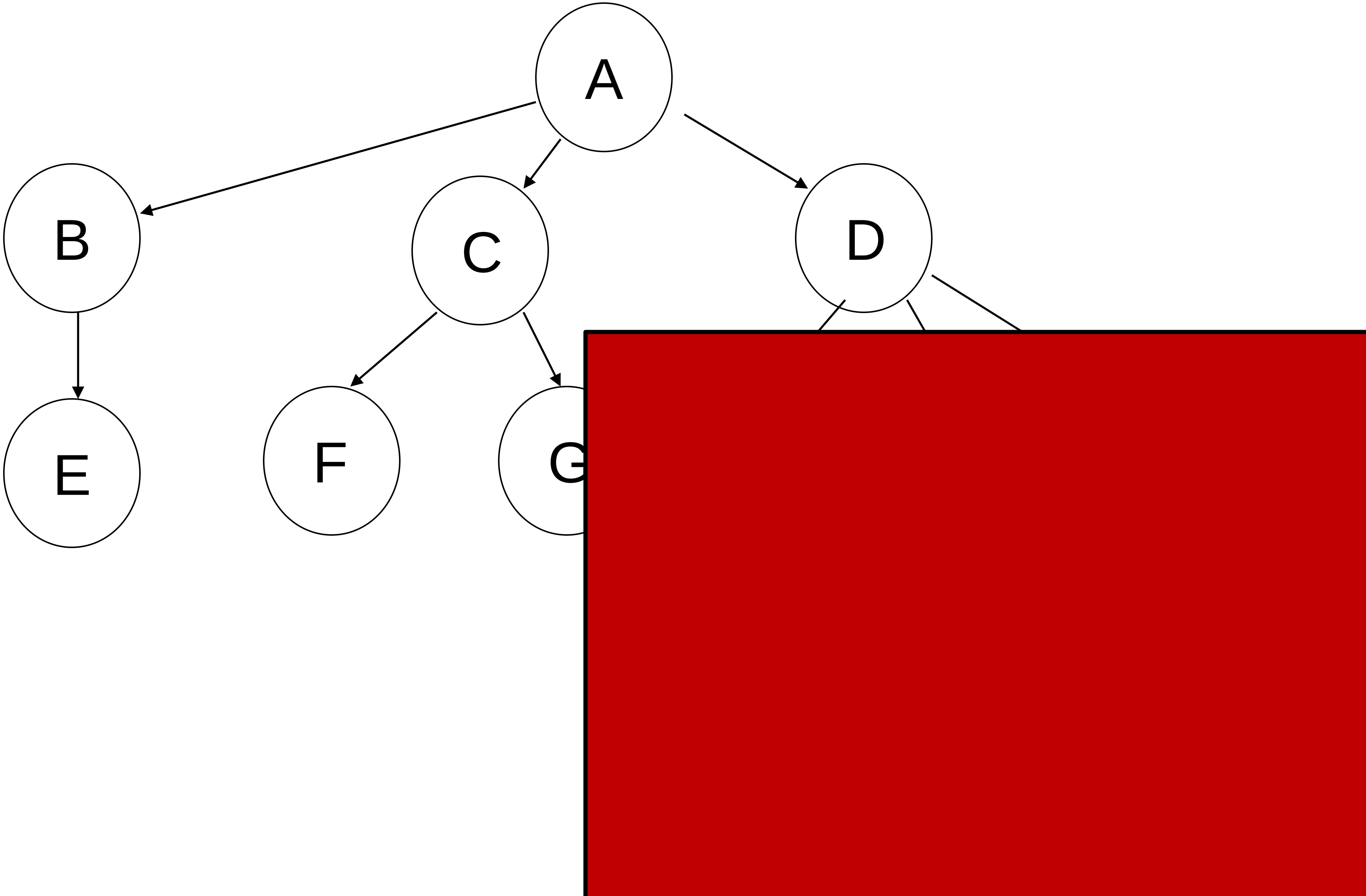
- *siblings:* two nodes that have the same parent
- *edge:* the link from one node to another
- *path length:* the number of edges that must be traversed to get from one node to another



More Properties of Trees

- ***depth**:* the path length from the root of the tree to this node
- ***height of a node**:* The maximum distance (path length) of any leaf from this node
  - a leaf has a height of 0
  - the height of a tree is the height of the root of that tree
  - what if only one node, the root?
  - what if empty? Discover OptionalInt
- *descendants*: any nodes that can be reached via 1 or more edges from this node
- *ancestors:* any nodes for which this node is a descendant

Tree Visualization



Clicker 1

- What is the depth of the node that contains M on the previous slide?

A. 0

B. 1

C. 2

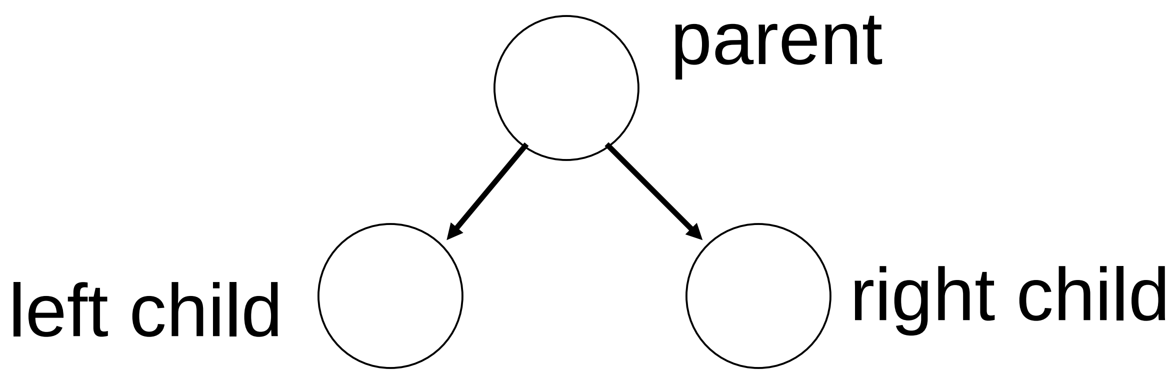
D. 3

E. 4

Clicker 2 - Same tree, same choices
What is the height of the node that contains D?

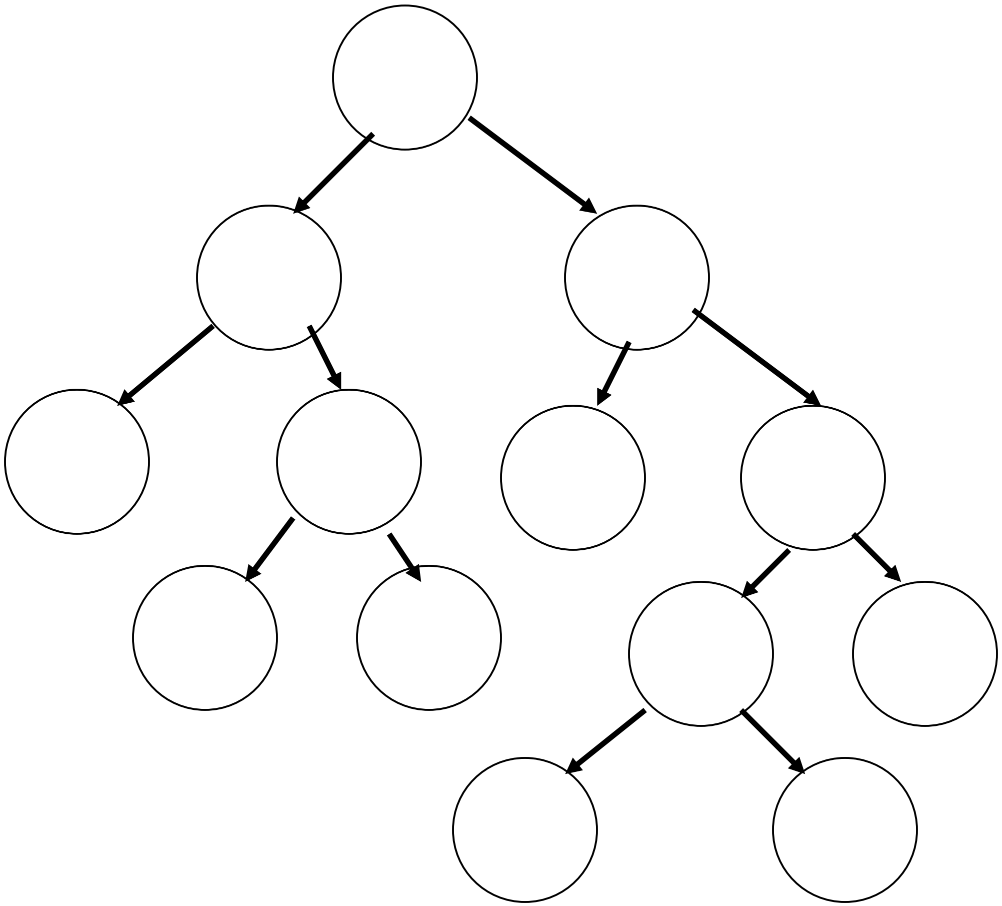
Binary Trees

- There are many variations on trees but we will start with *binary trees*
- *binary tree:* each node has at most two children
  - – the possible children are usually referred to as the left child and the right child



Full Binary Tree

- *full binary tree:* a binary tree in which each node has 2 or 0 children



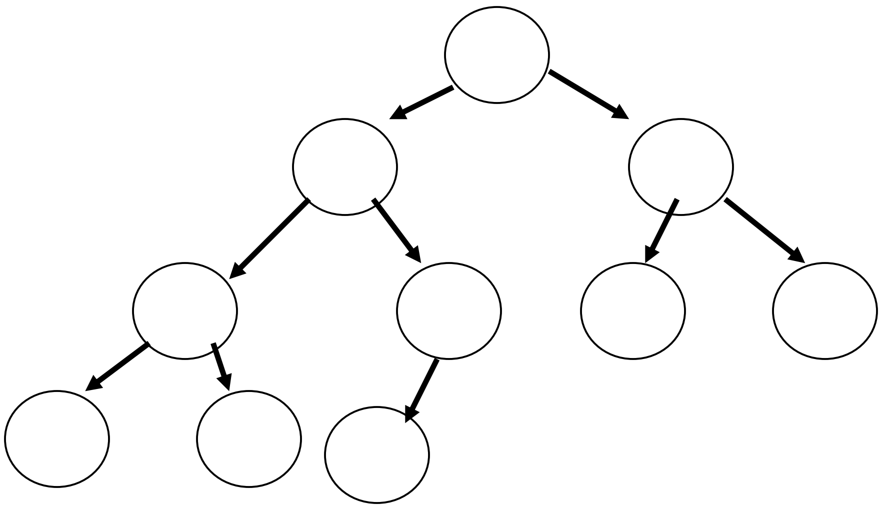
Clicker 3

- What is the maximum height of a full binary tree with 11 nodes?

A. 3
B. 5
C. 7
D. 10
E. Not possible to have full binary tree with 11 nodes.

Complete Binary Tree

- *complete binary tree:* a binary tree in which every level, except possibly the deepest is completely filled. At depth n, the height of the tree, all nodes are as far left as possible



Where would the next node go to maintain a complete tree?

Clicker 4

- What is the height of a complete binary tree that contains N nodes?

A. O(1)

B. O(logN)

C. $O(N^{1/2})$

D. O(N)

E. O(NlogN)

- Recall, order can be applied to any function. It doesn't just apply to running time.

Perfect Binary Tree

- *perfect binary tree:* a binary tree with all leaf nodes at the same depth. All internal nodes have exactly two children.
- a perfect binary tree has the maximum number of nodes for a given height
- a perfect binary tree has $(2^{(n+1)} - 1)$ nodes where n is the height of the tree
  - height = 0 -> 1 node
  - height = 1 -> 3 nodes
  - height = 2 -> 7 nodes
  - height = 3 -> 15 nodes

A Binary Node class

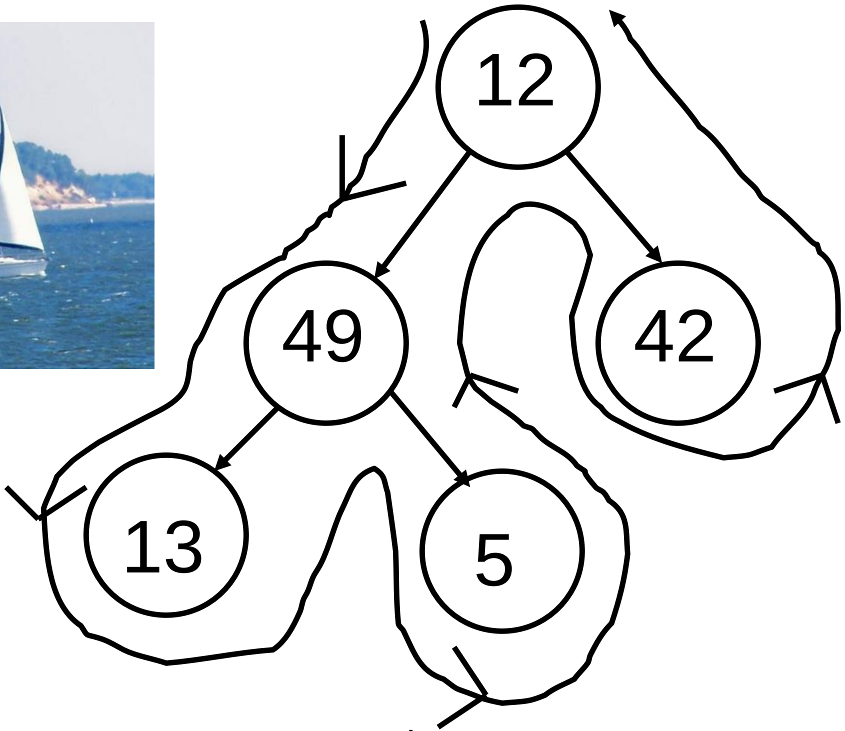
```
public class Bnode<E> {
     private E myData;
     private Bnode<E> myLeft;
     private Bnode<E> myRight;

     public BNode();
     public BNode(Bnode<E> left, E data,
          Bnode<E> right)
     public E getData()
     public Bnode<E> getLeft()
     public Bnode<E> getRight()

     public void setData(E data)
     public void setLeft(Bnode<E> left)
     public void setRight(Bnode<E> right)
}
```

Binary Tree Traversals

- Many algorithms require all nodes of a binary tree be visited and the contents of each node processed or examined.
- There are 4 traditional types of traversals
  - preorder traversal: process the root, then process all sub trees (left to right)
  - in order traversal: process the left sub tree, process the root, process the right sub tree
  - post order traversal: process the left sub tree, process the right sub tree, then process the root
  - level order traversal: starting from the root of a tree, process all nodes at the same depth from left to right, then proceed to the nodes at the next depth.

Results of Traversals

- To determine the results of a traversal on a given tree draw a path around the tree.
  - start on the left side of the root and trace around the tree. The path should stay close to the tree.

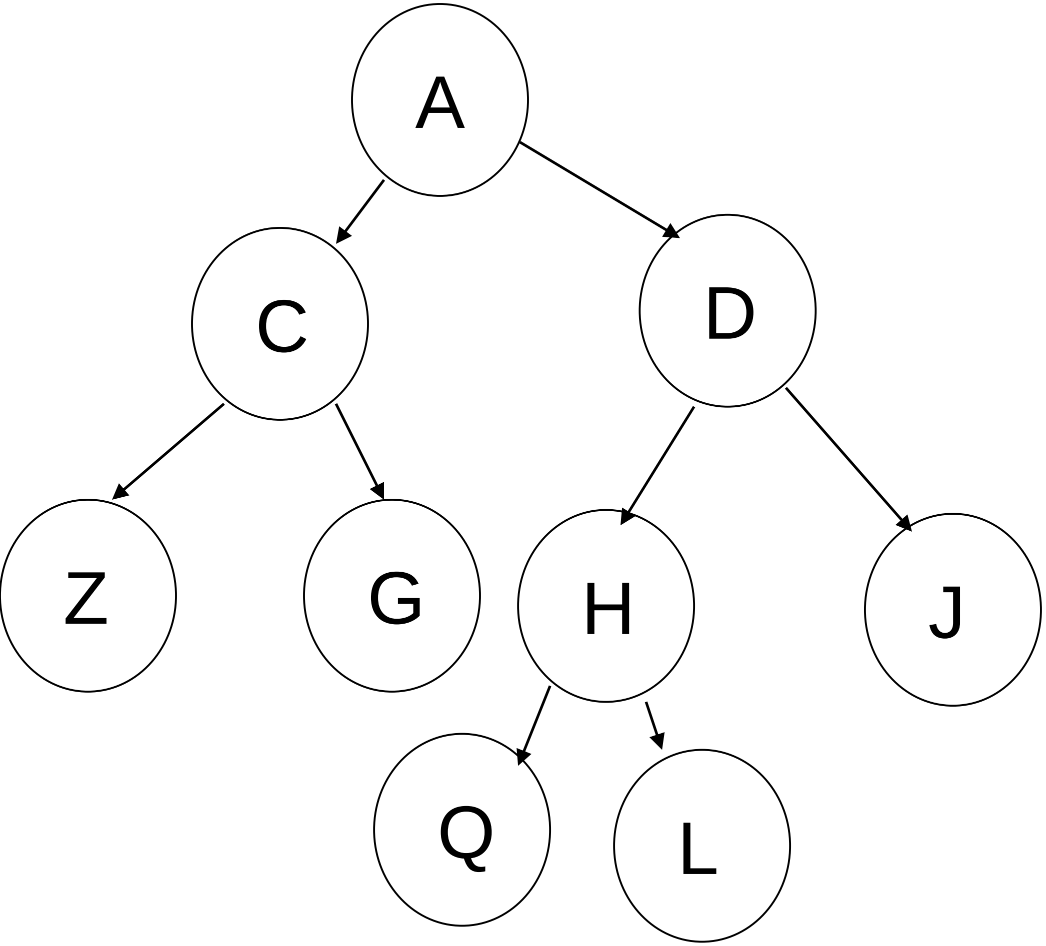


pre order: process when pass down left side of node
12 49 13 5 42

in order: process when pass underneath node
13 49 5 12 42

post order: process when pass up right side of node
13 5 49 42 12

Clicker 5 - Tree Traversals



What is a the result of a post order traversal of the tree to the left?

A. Z C G A Q H L D J
B. Z G C Q L H J D A
C. A C Z G D H Q L J
D. A C D Z G H J Q L
E. None of these

Implement Traversals

- Implement preorder, inorder, and post order traversal
  - Big O time and space?
- Implement a level order traversal using a queue
  - Big O time and space?
- Implement a level order traversal without a queue
  - target depth

Breadth First Search
Depth First Search

- from NIST - DADS
- **breadth first search:** Any search algorithm that considers neighbors of a *vertex* (node), that is, outgoing *edges* (links) of the vertex's predecessor in the search, before any outgoing edges of the vertex
- **depth first search:** Any search algorithm that considers outgoing *edges* (links of *children*) of a *vertex* (node) before any of the vertex's (node) *siblings*, that is, outgoing edges of the vertex's predecessor in the search. Extremes are searched first.

Clicker 6

- Which traversal of a tree is a breadth first search?

A. Level order traversal

B. Pre order traversal

C. In order traversal

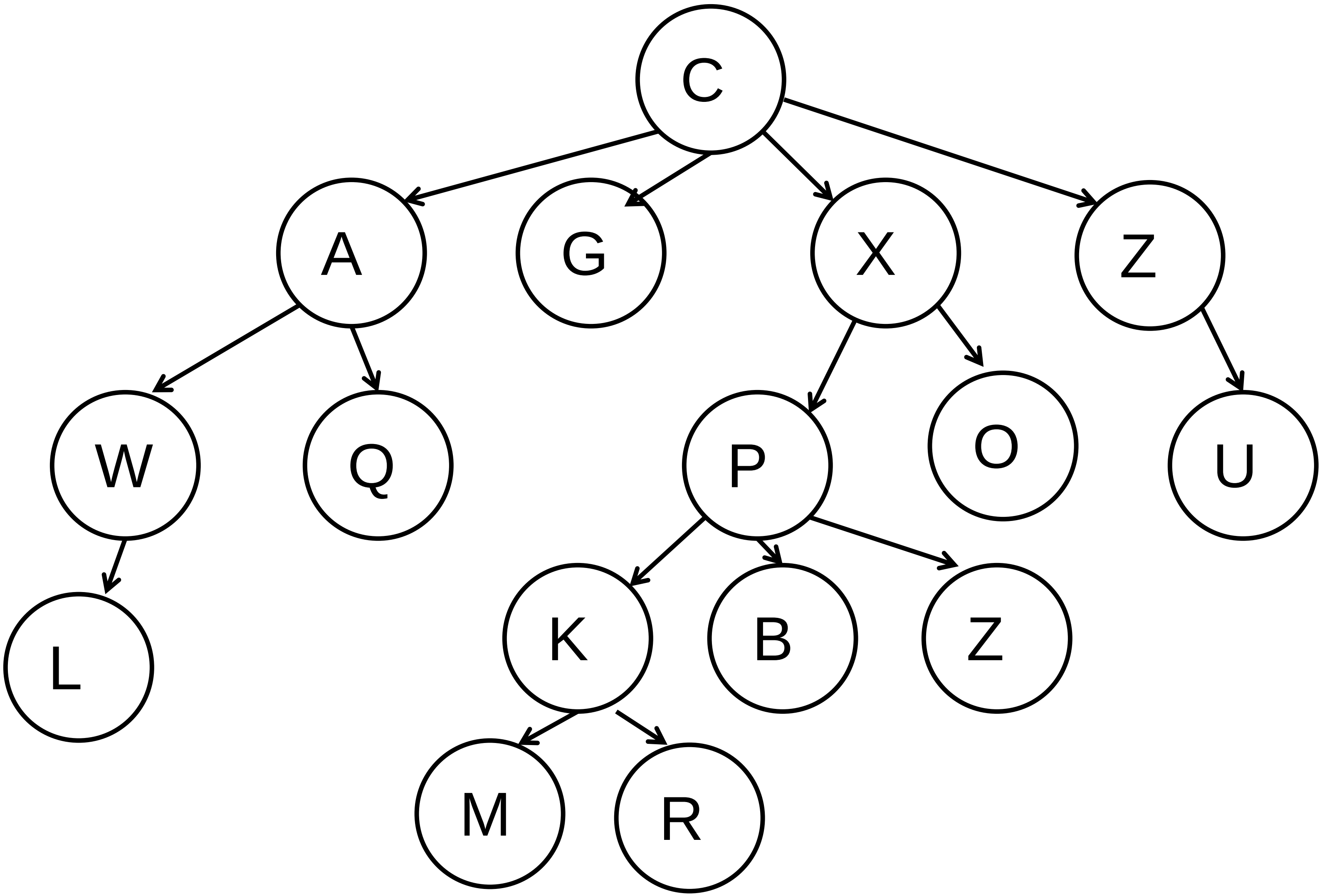
D. Post order traversal

E. More than one of these

Breadth First

- A level order traversal of a tree could be used as a breadth first search
- Search all nodes in a level before going down to the next level

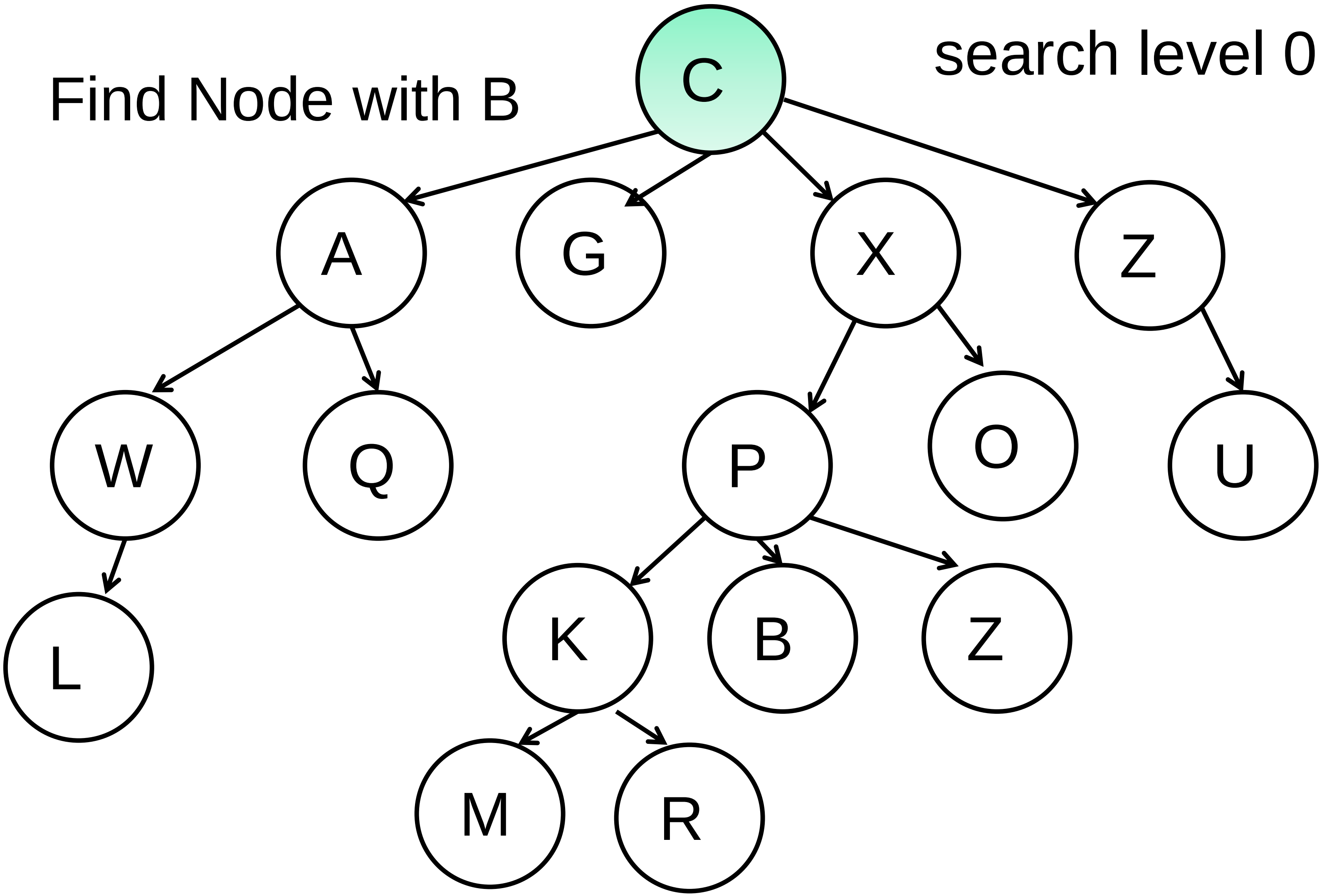
Breadth First Search of Tree



Breadth First Search

Find Node with B

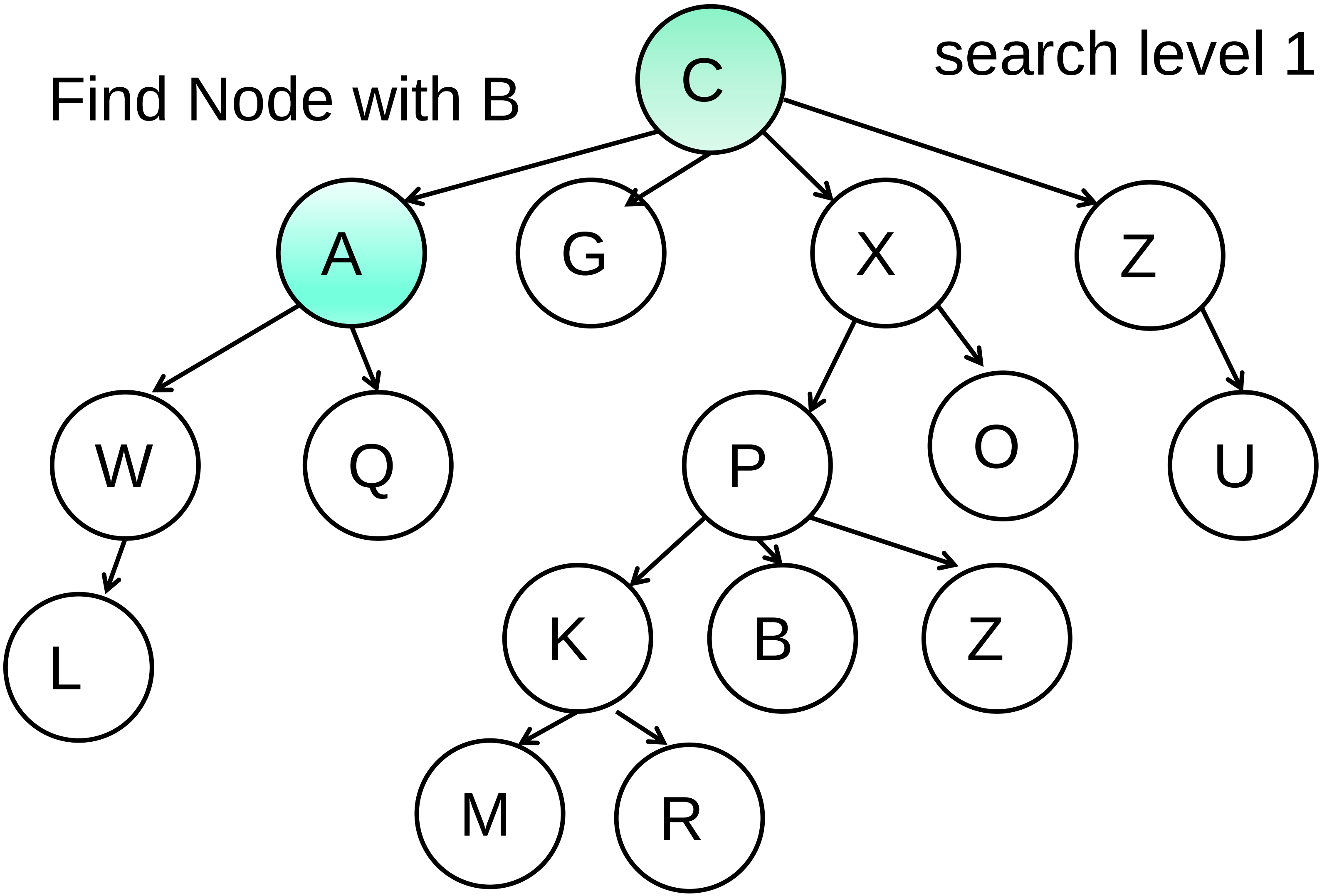
search level 0 first



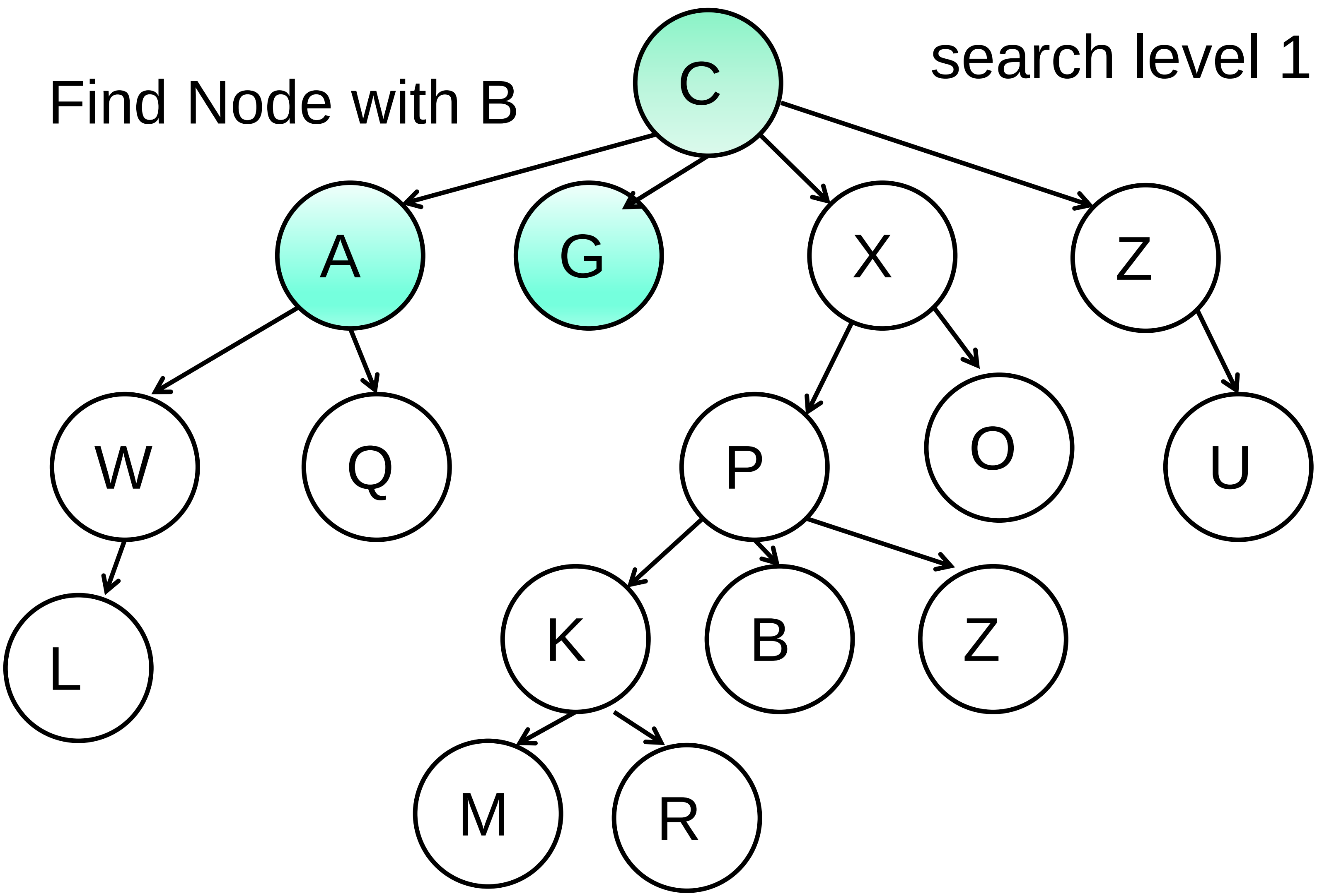
Breadth First Search

Find Node with B

search level 1



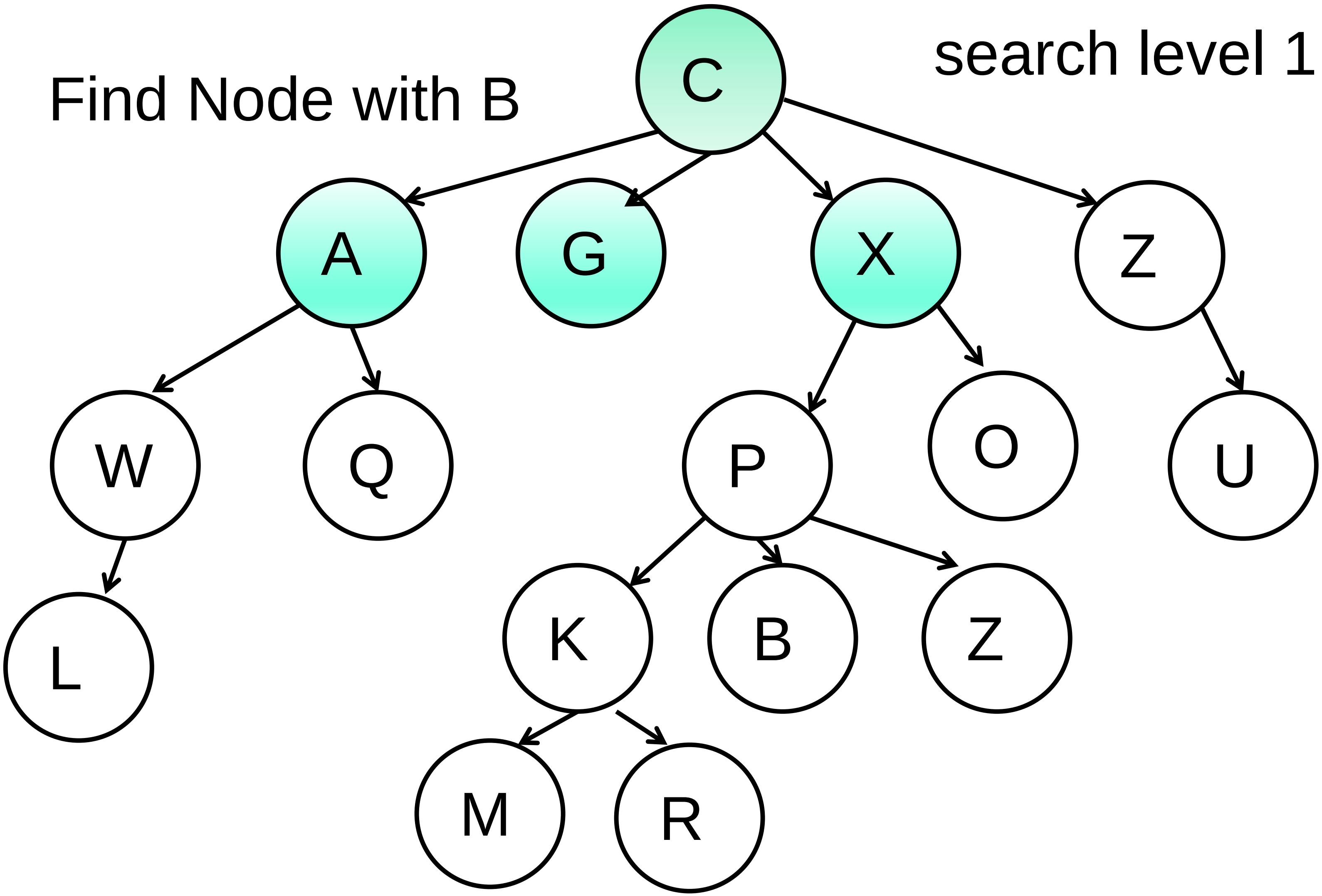
Breadth First Search



Breadth First Search

Find Node with B

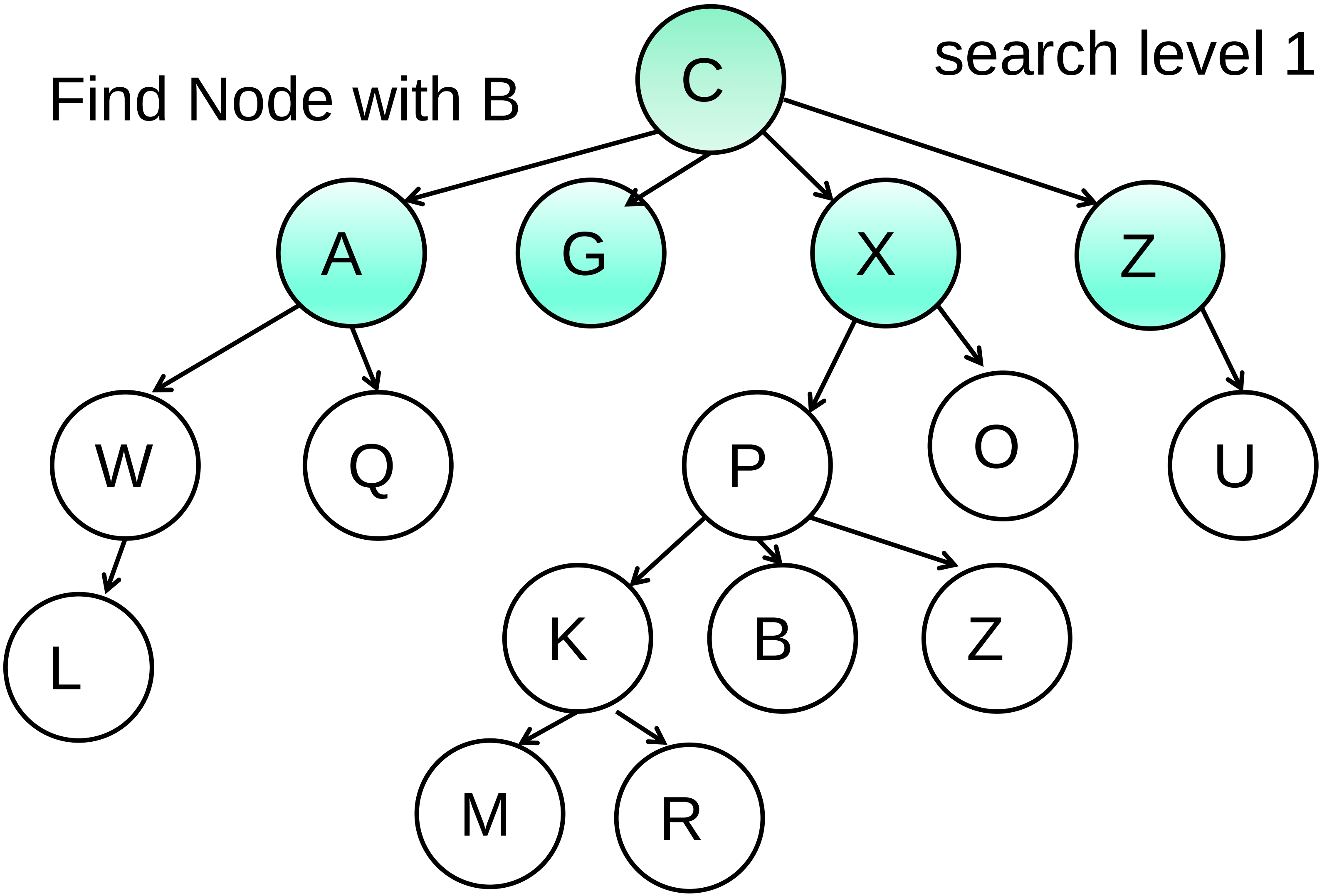
search level 1



Breadth First Search

Find Node with B

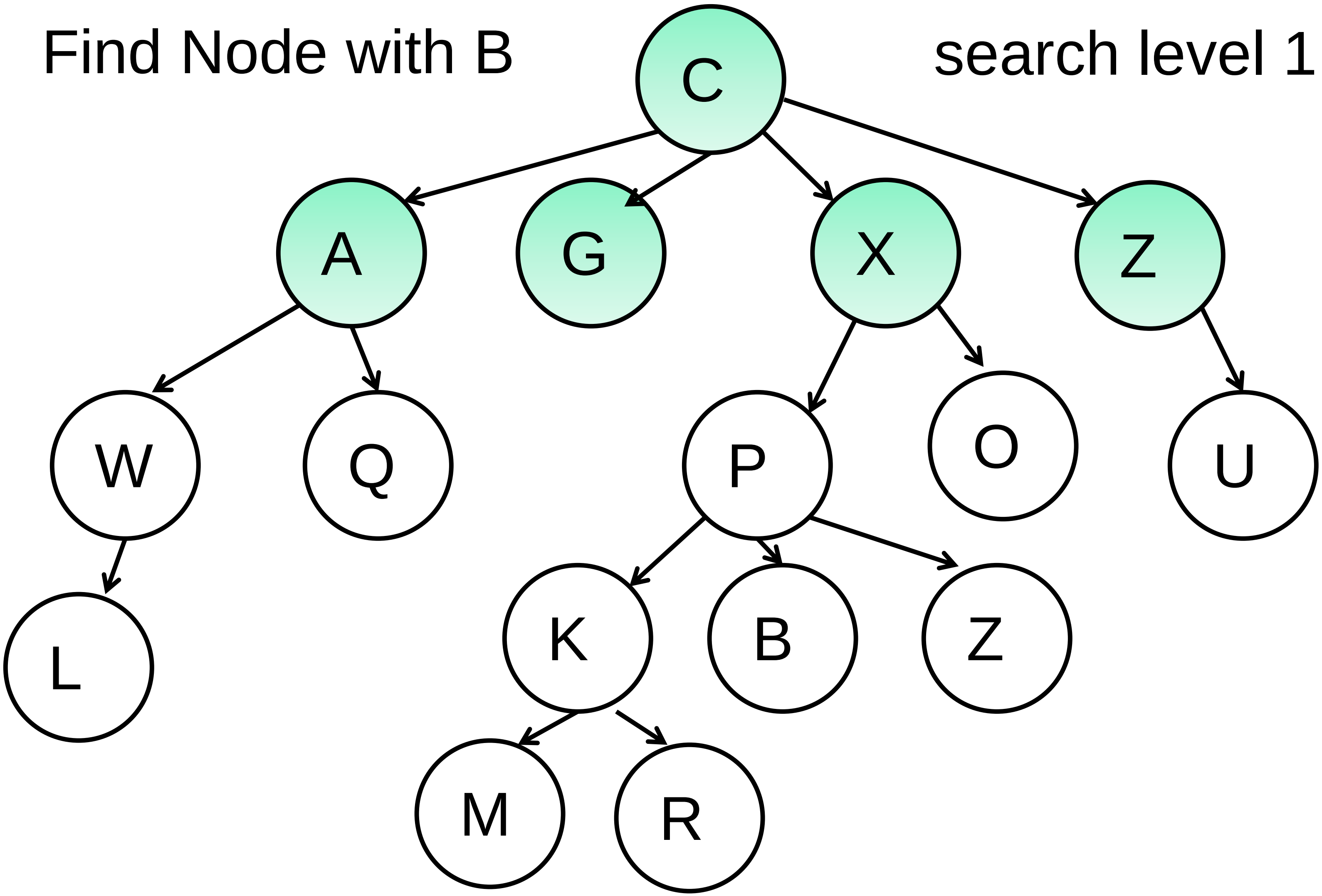
search level 1



Breadth First Search

Find Node with B

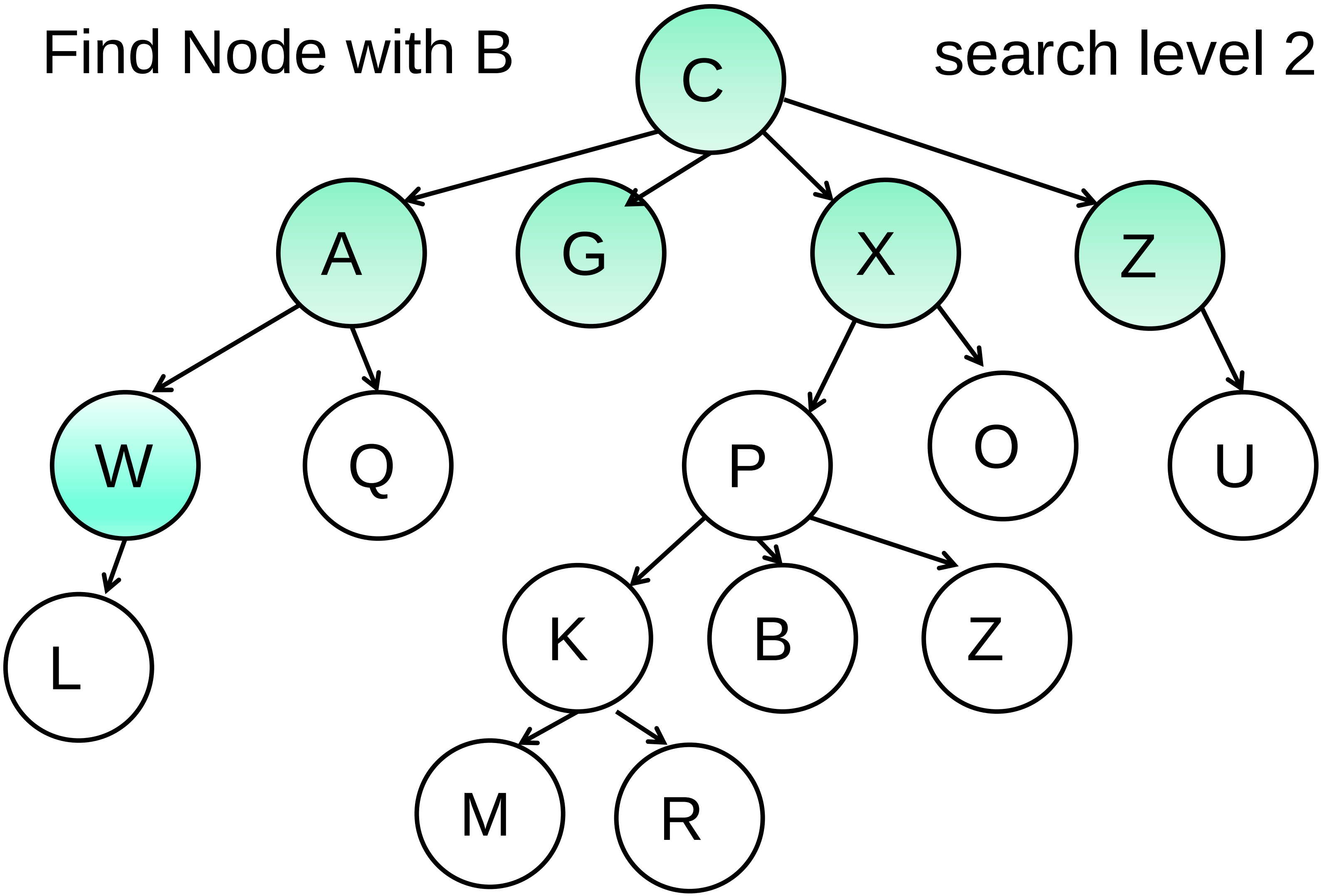
search level 1 next



Breadth First Search

Find Node with B

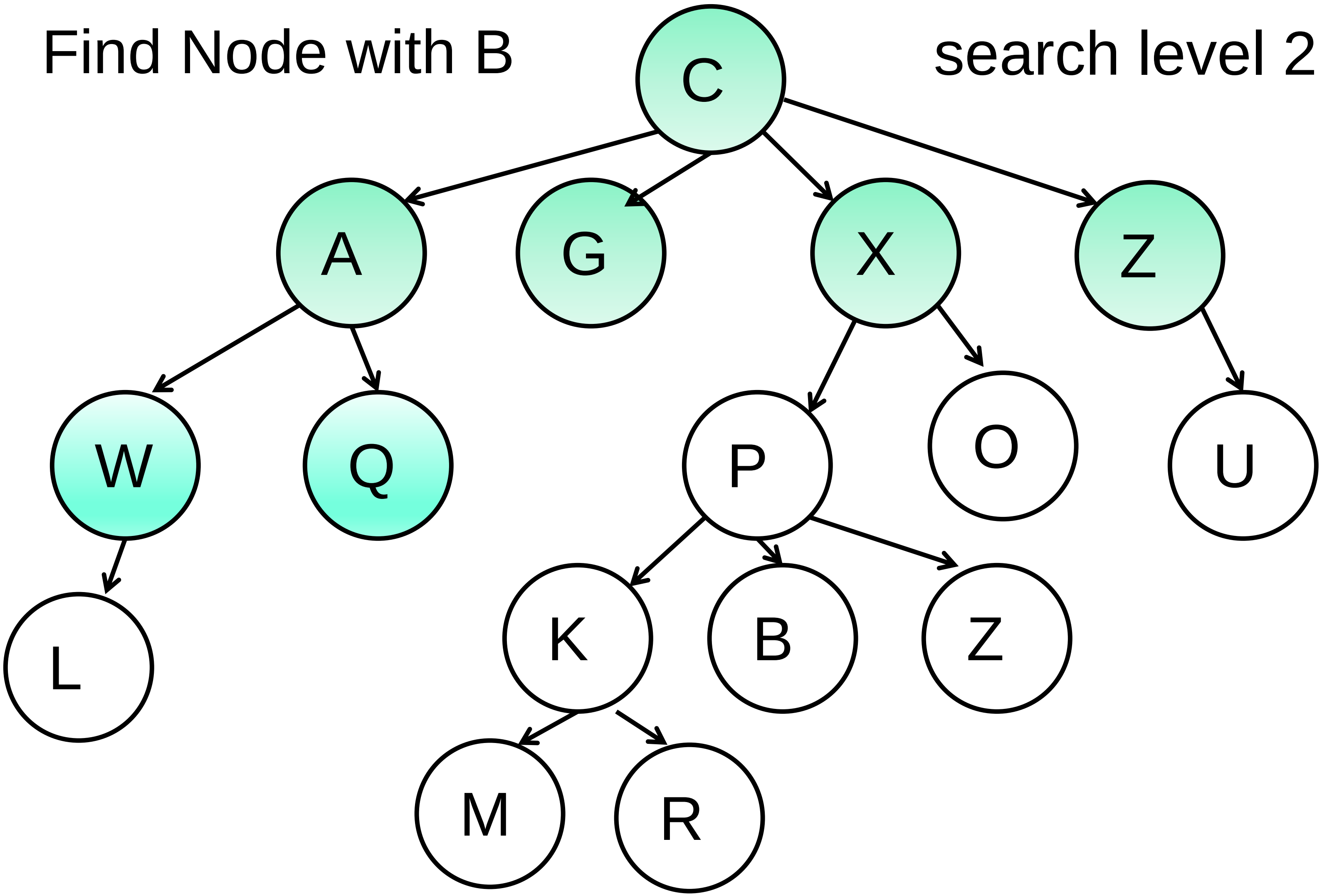
search level 2 next



Breadth First Search

Find Node with B

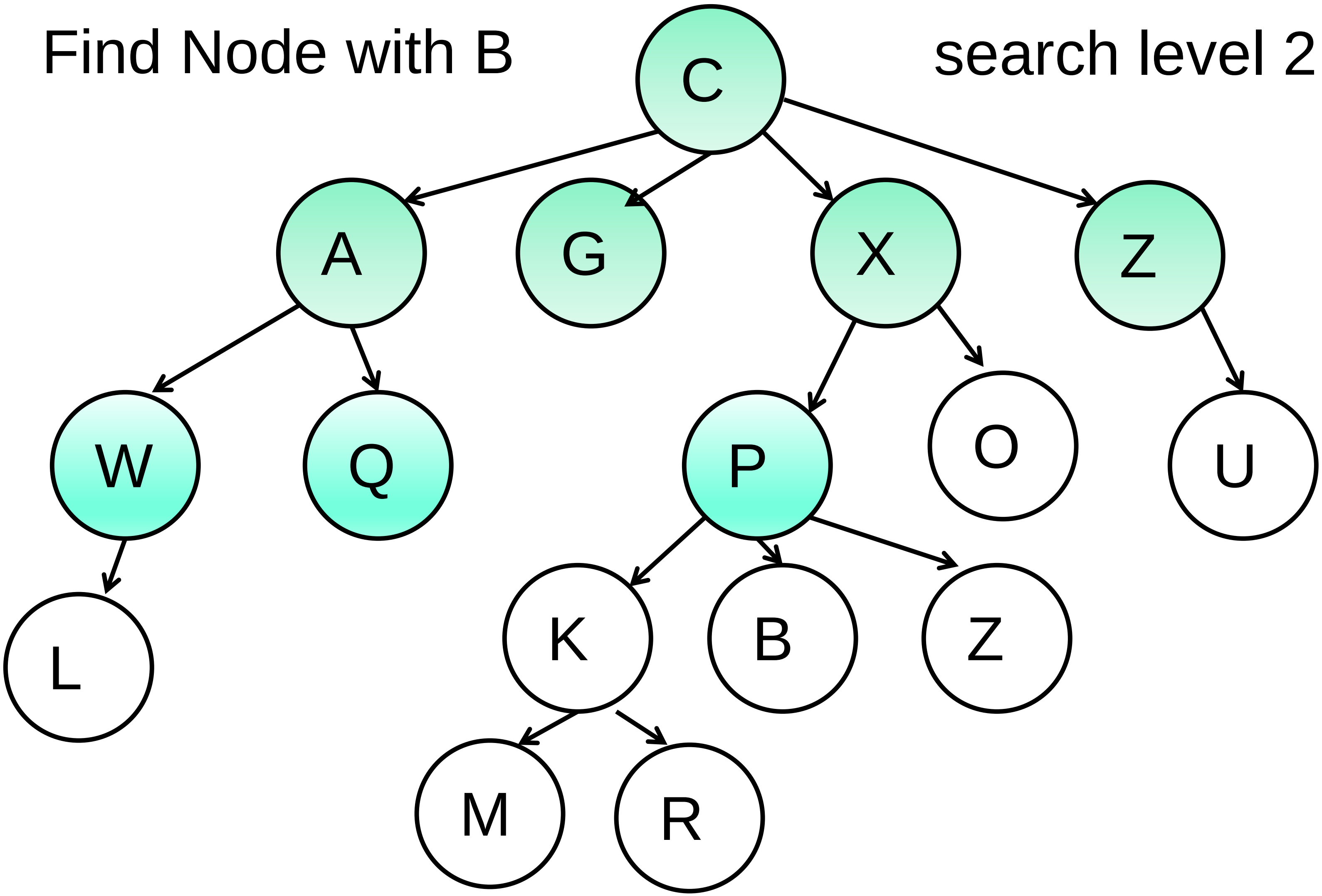
search level 2 next



Breadth First Search

Find Node with B

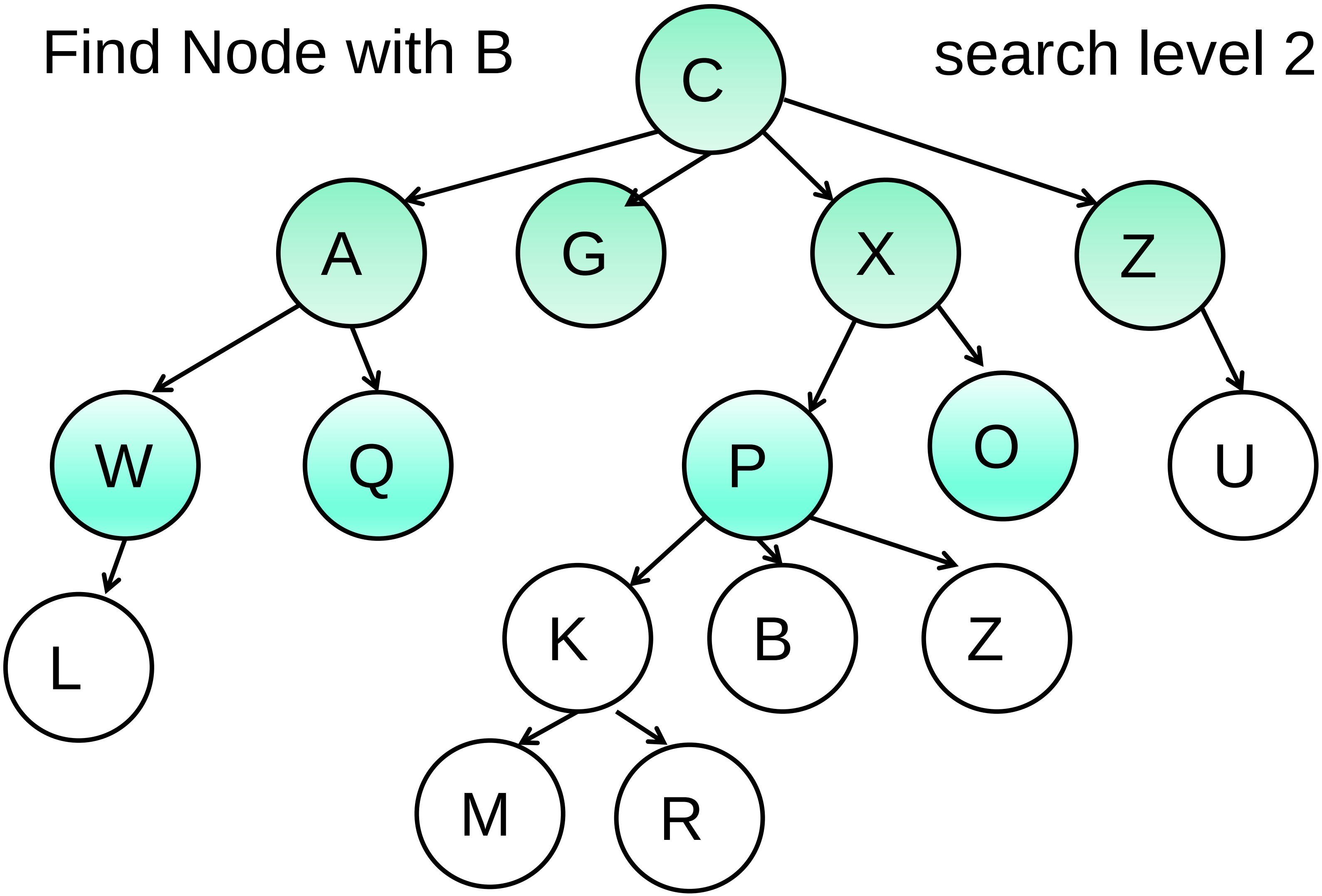
search level 2 next



Breadth First Search

Find Node with B

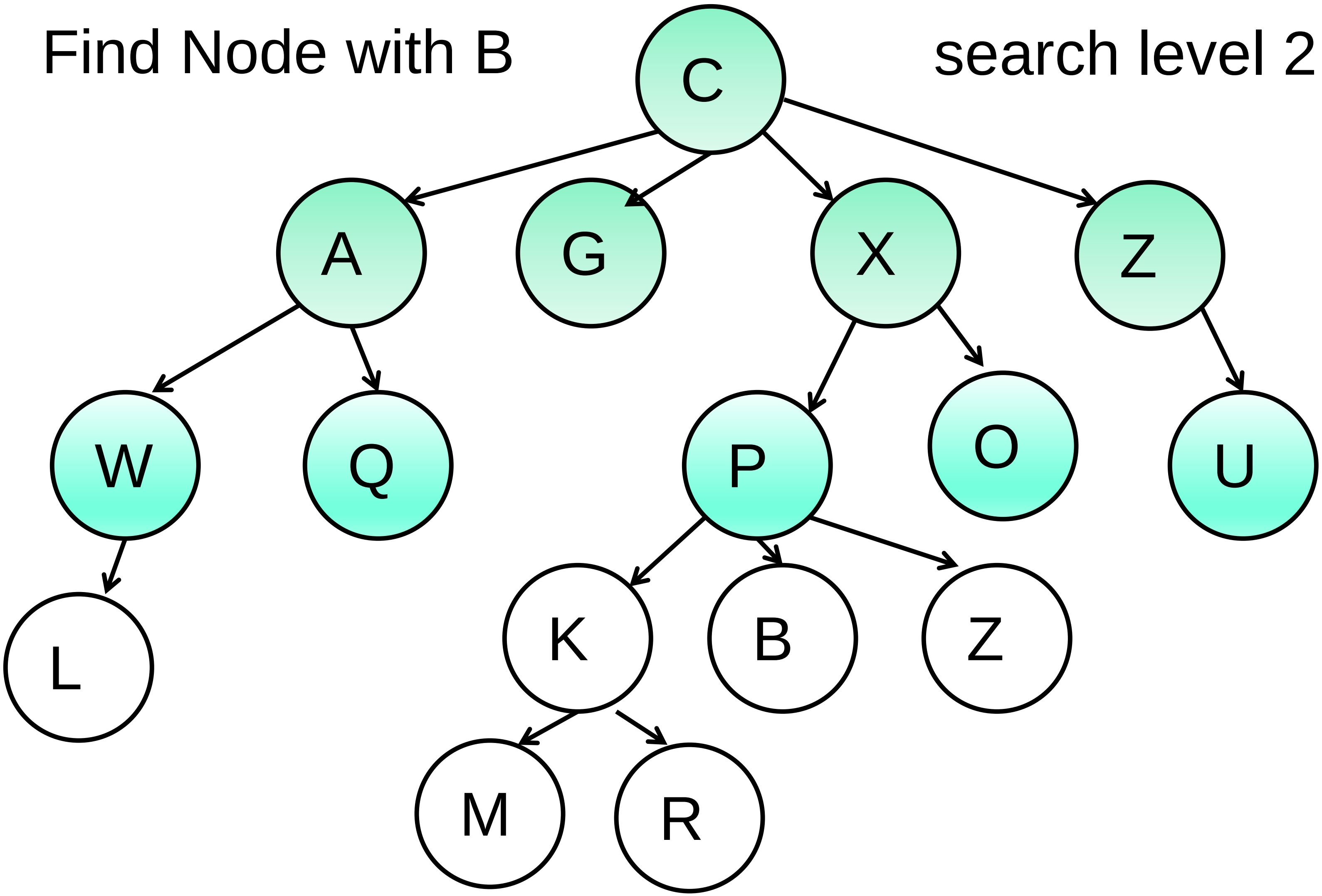
search level 2 next



Breadth First Search

Find Node with B

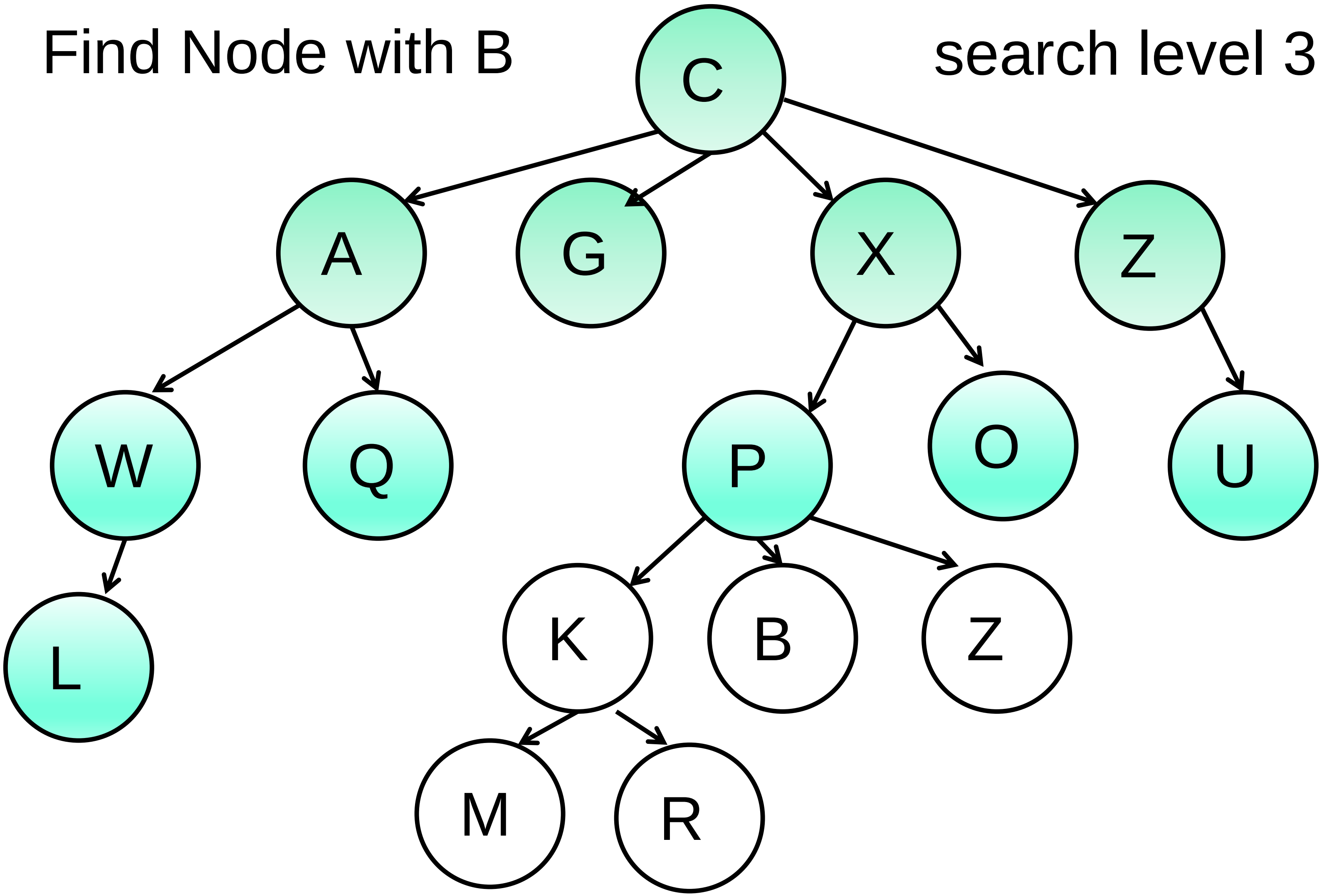
search level 2 next



Breadth First Search

Find Node with B

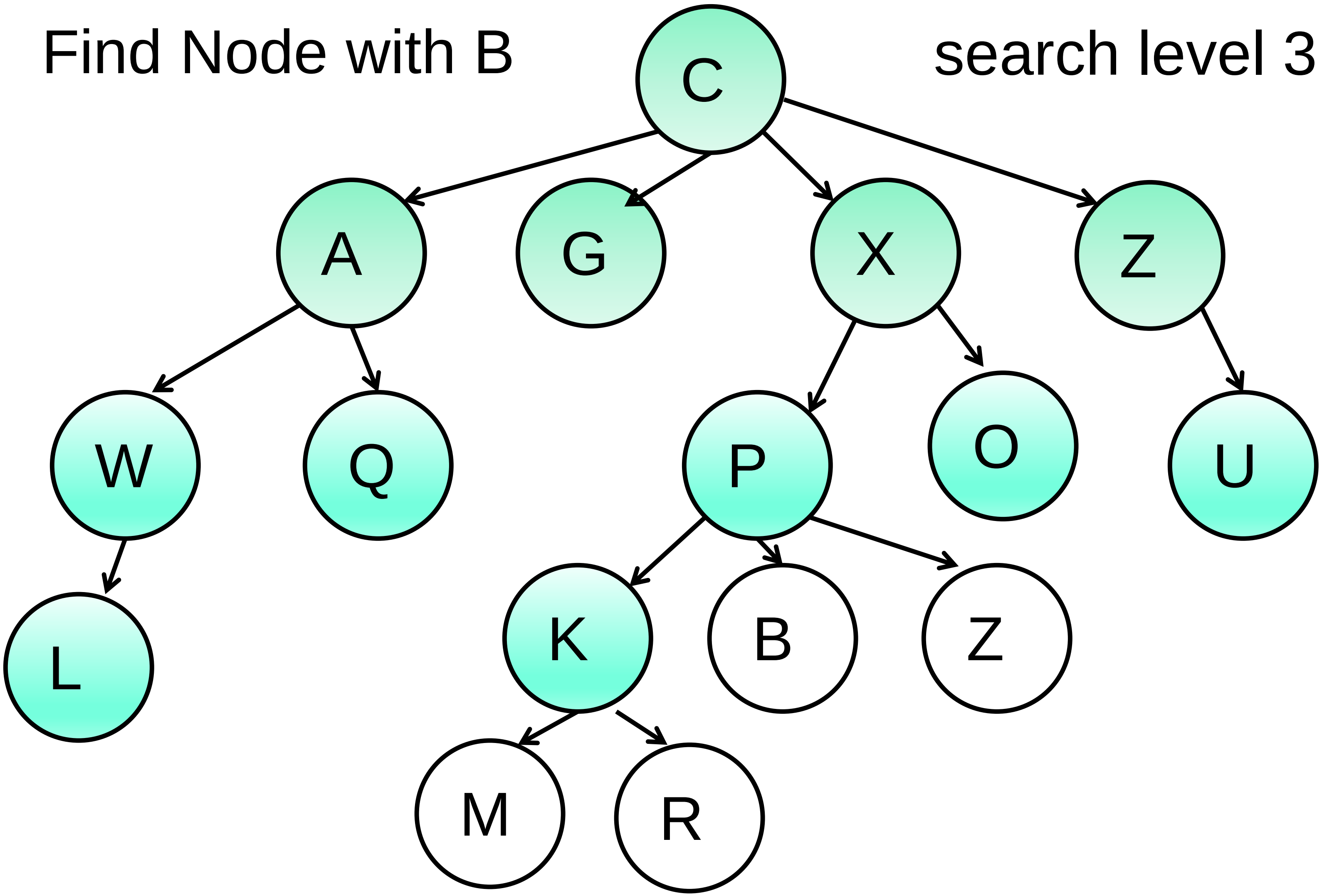
search level 3 next



Breadth First Search

Find Node with B

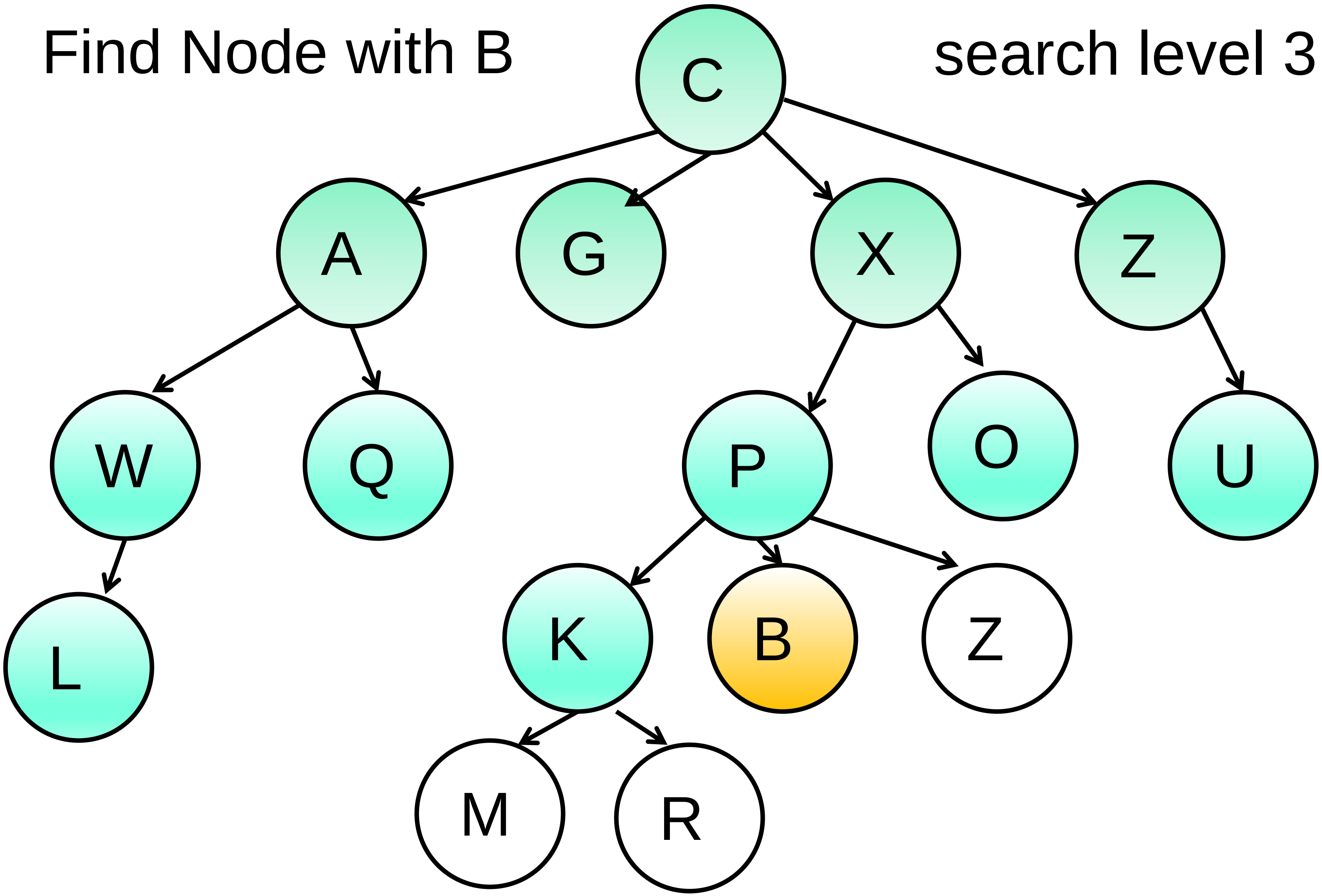
search level 3 next



Breadth First Search

Find Node with B

search level 3 next

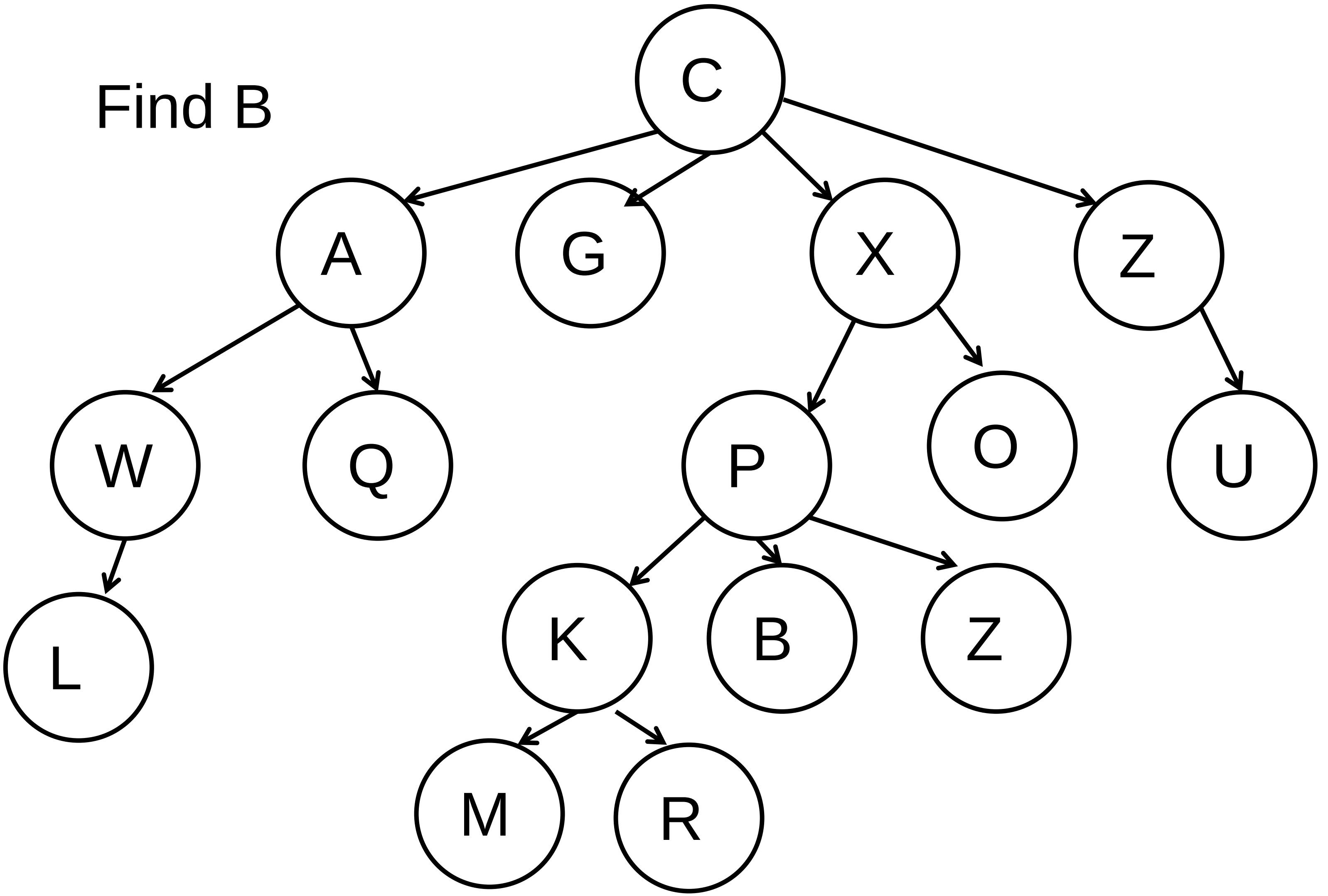


BFS - DFS

- Breadth first search typically implemented with a Queue
- Depth first search typically implemented with a stack, implicit with recursion or iteratively with an explicit stack
- which technique do I use?
  - depends on the problem

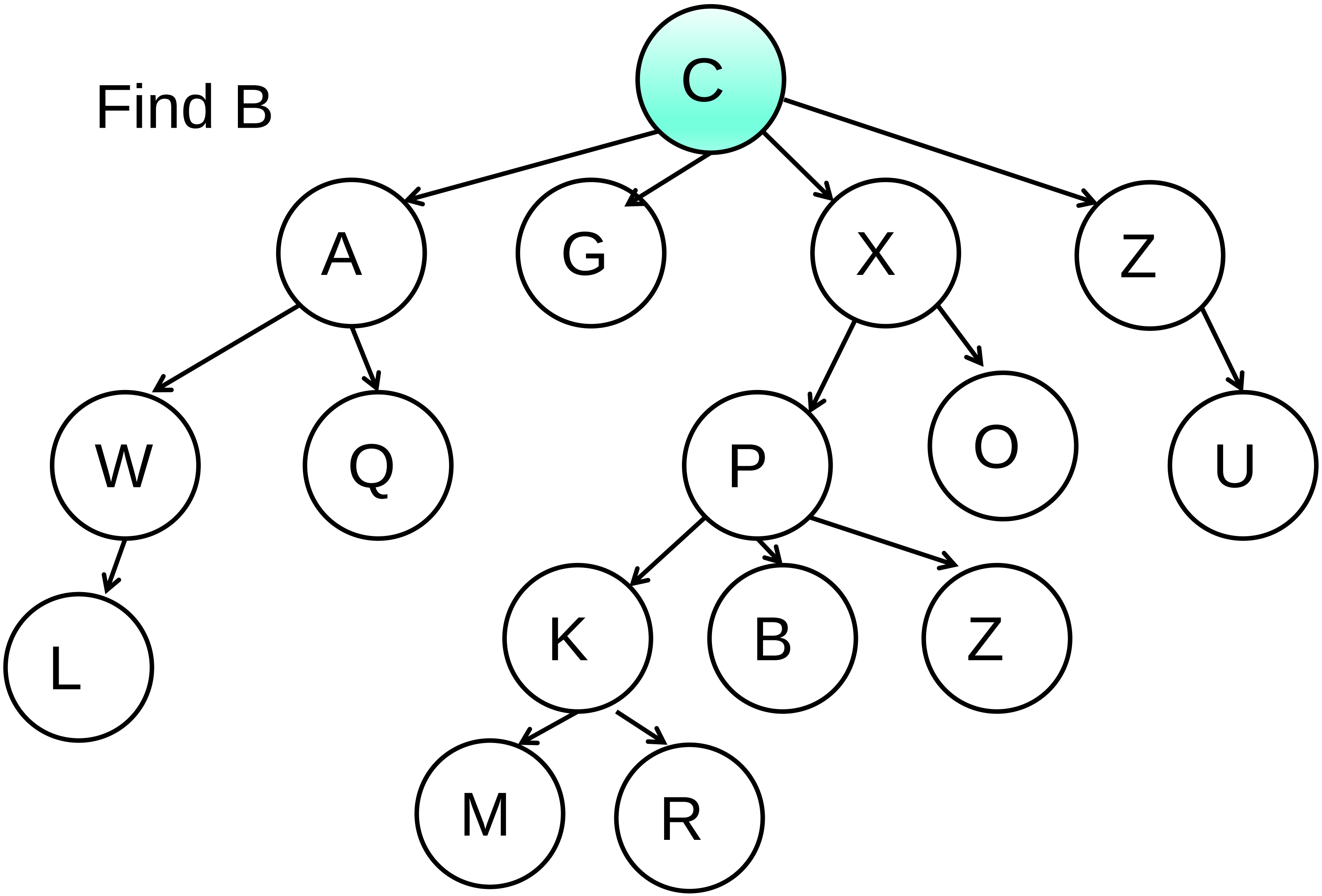
Depth First Search of Tree

Find B



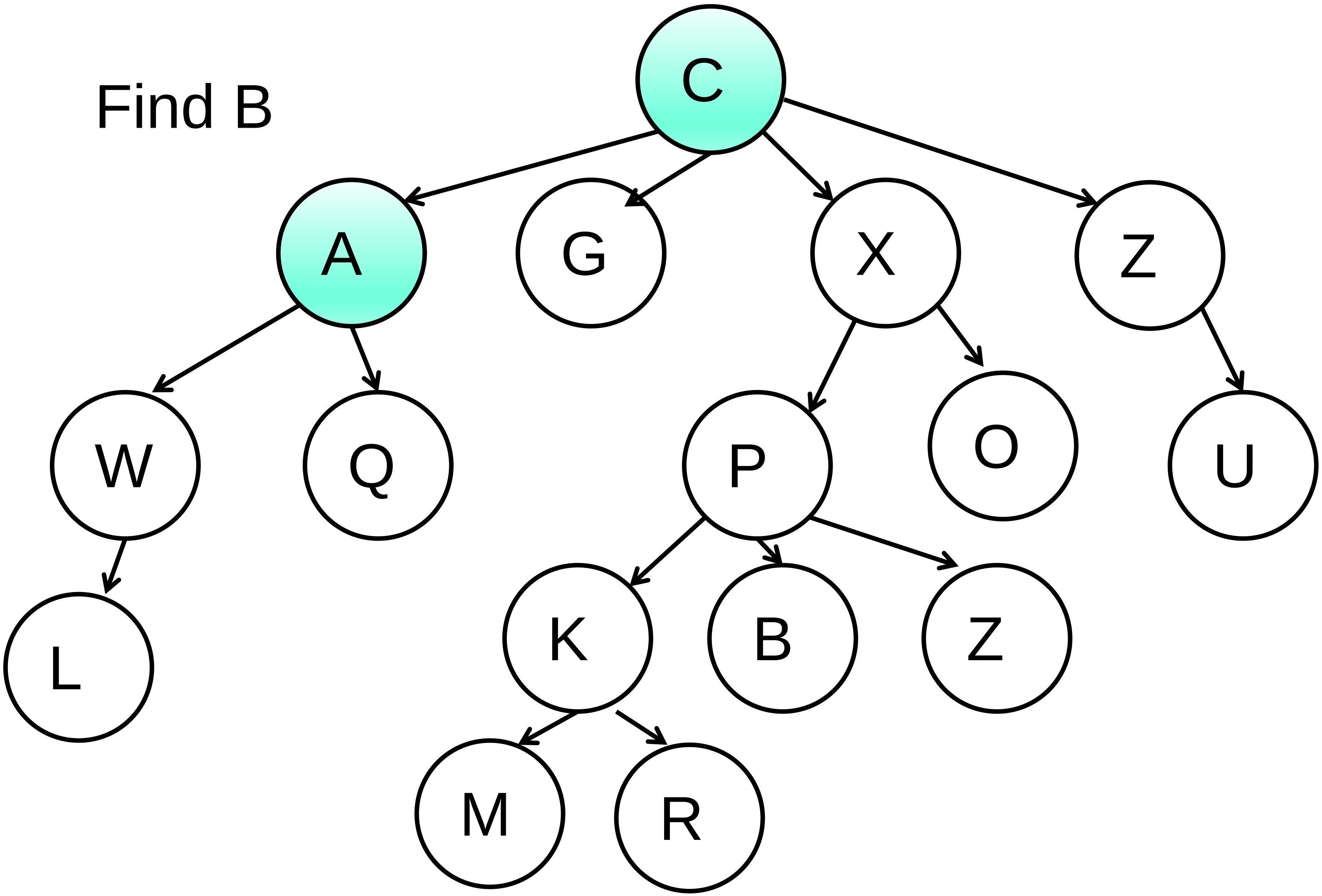
Depth First Search of Tree

Find B



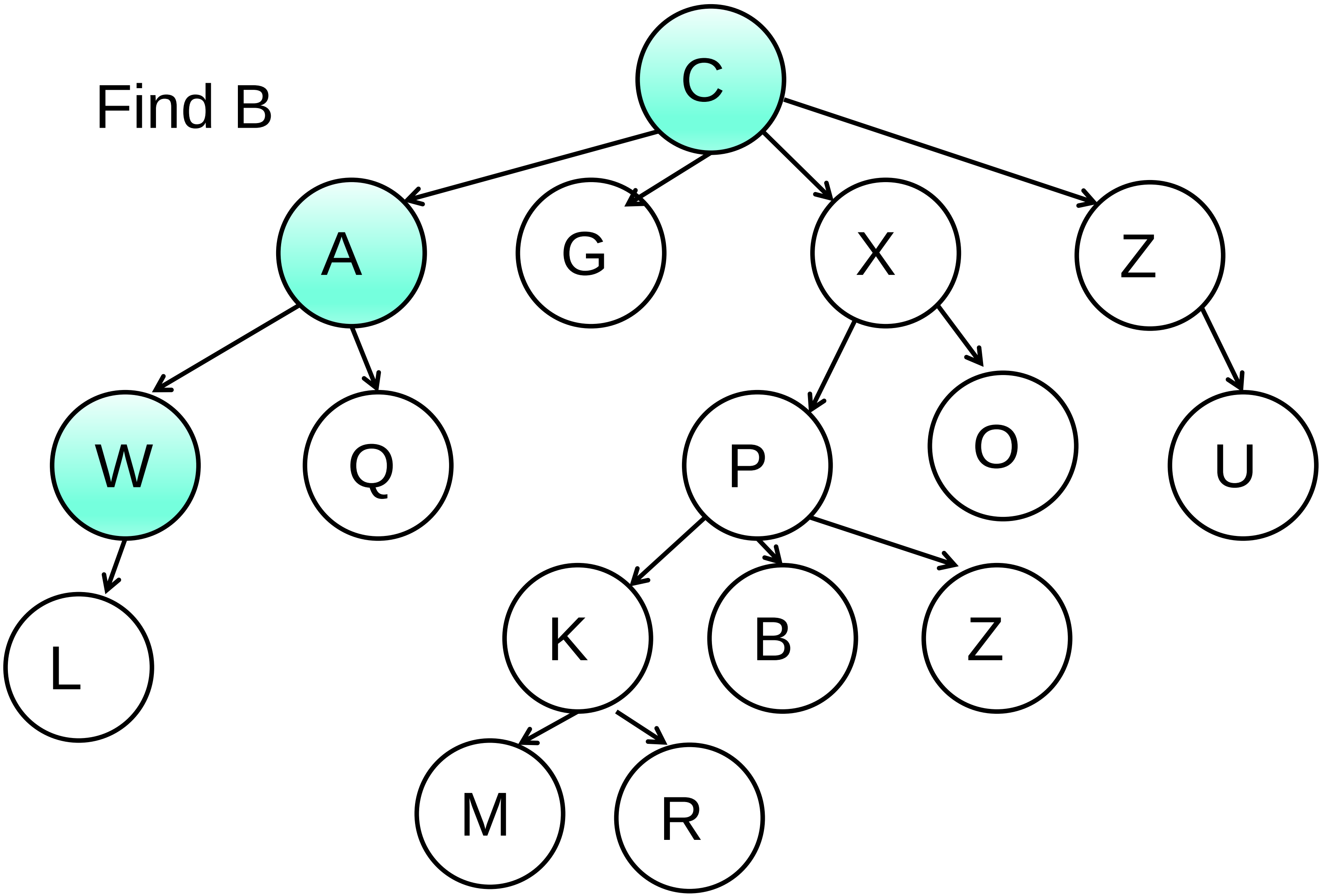
Depth First Search of Tree

Find B



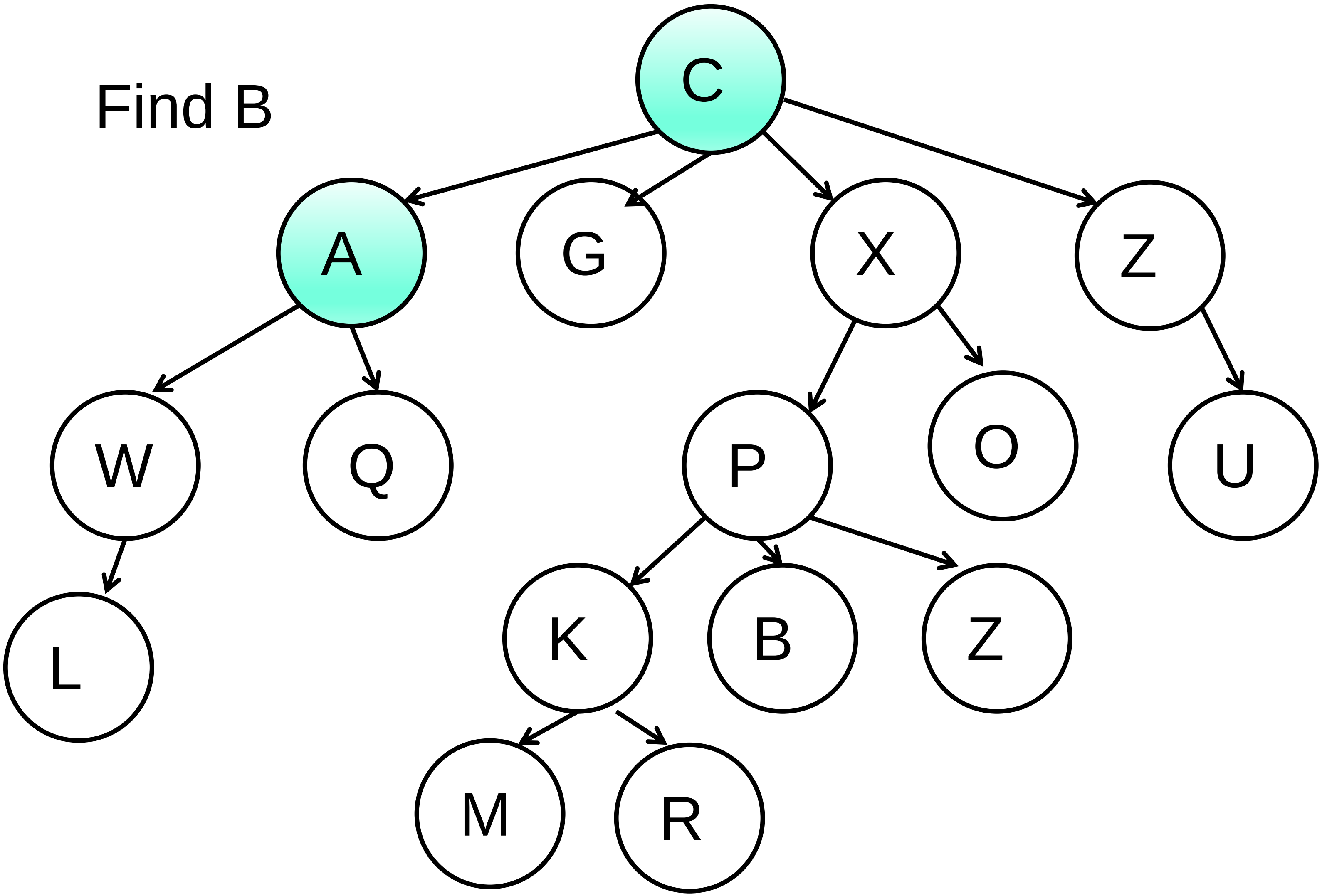
Depth First Search of Tree

Find B



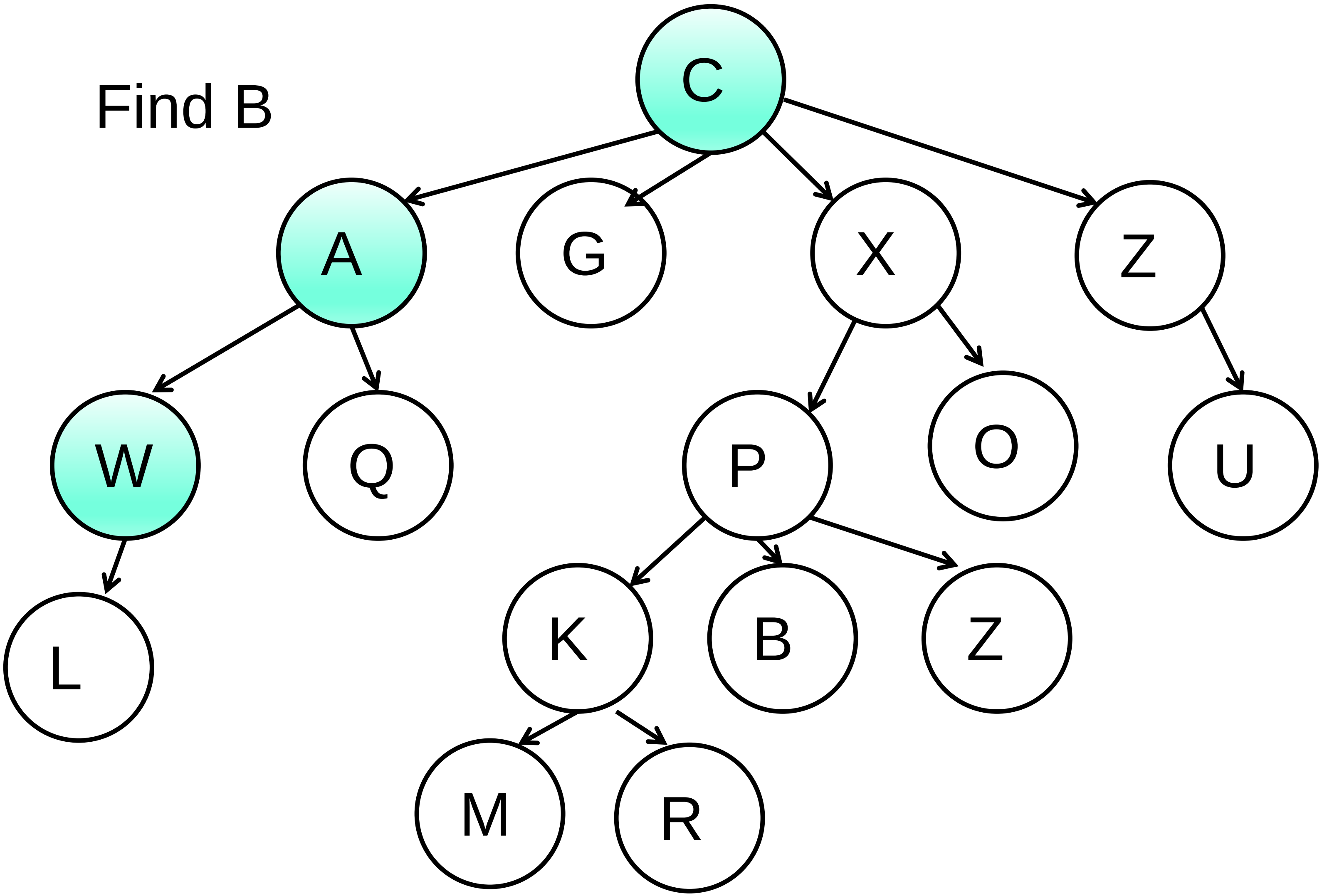
Depth First Search of Tree

Find B



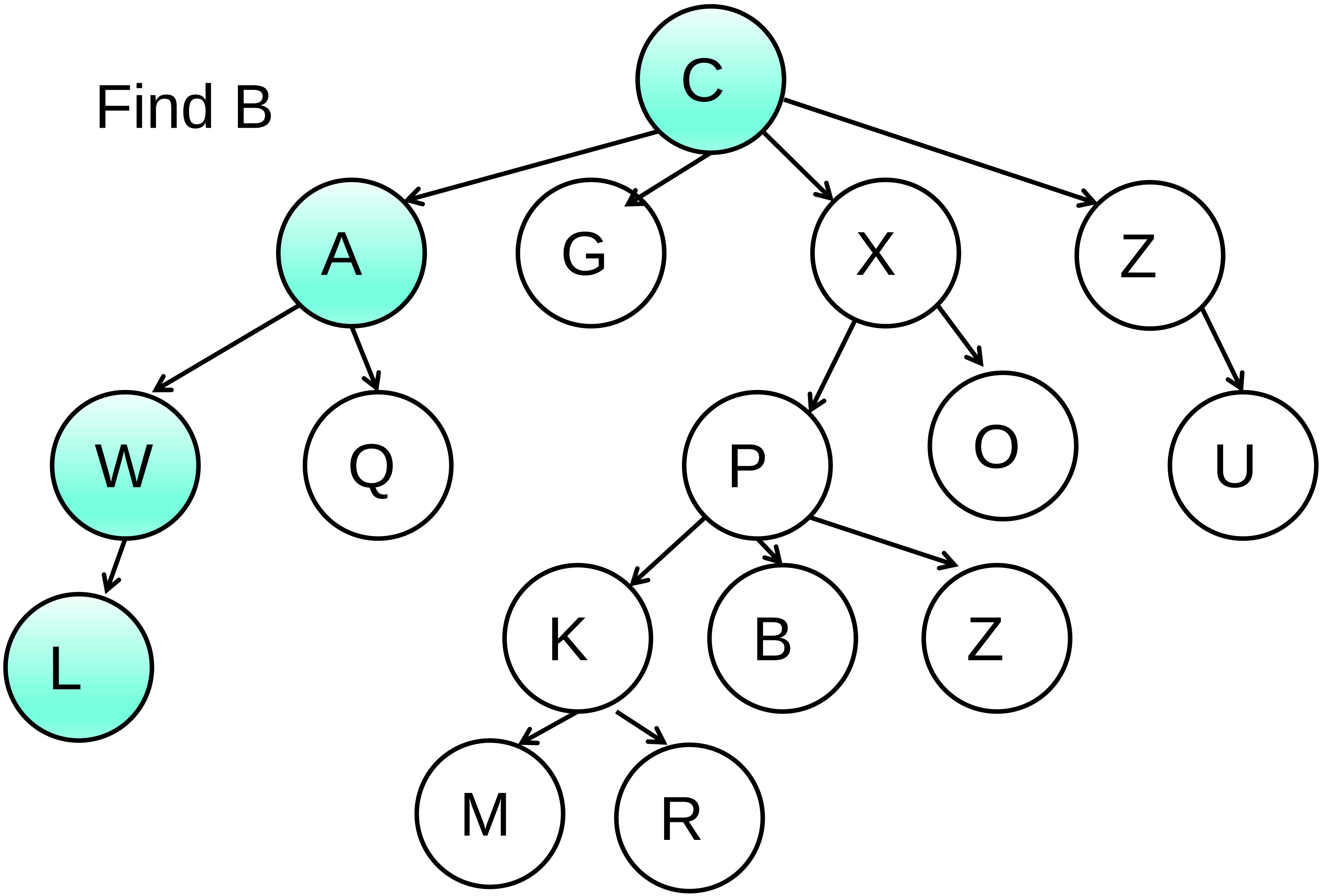
Depth First Search of Tree

Find B

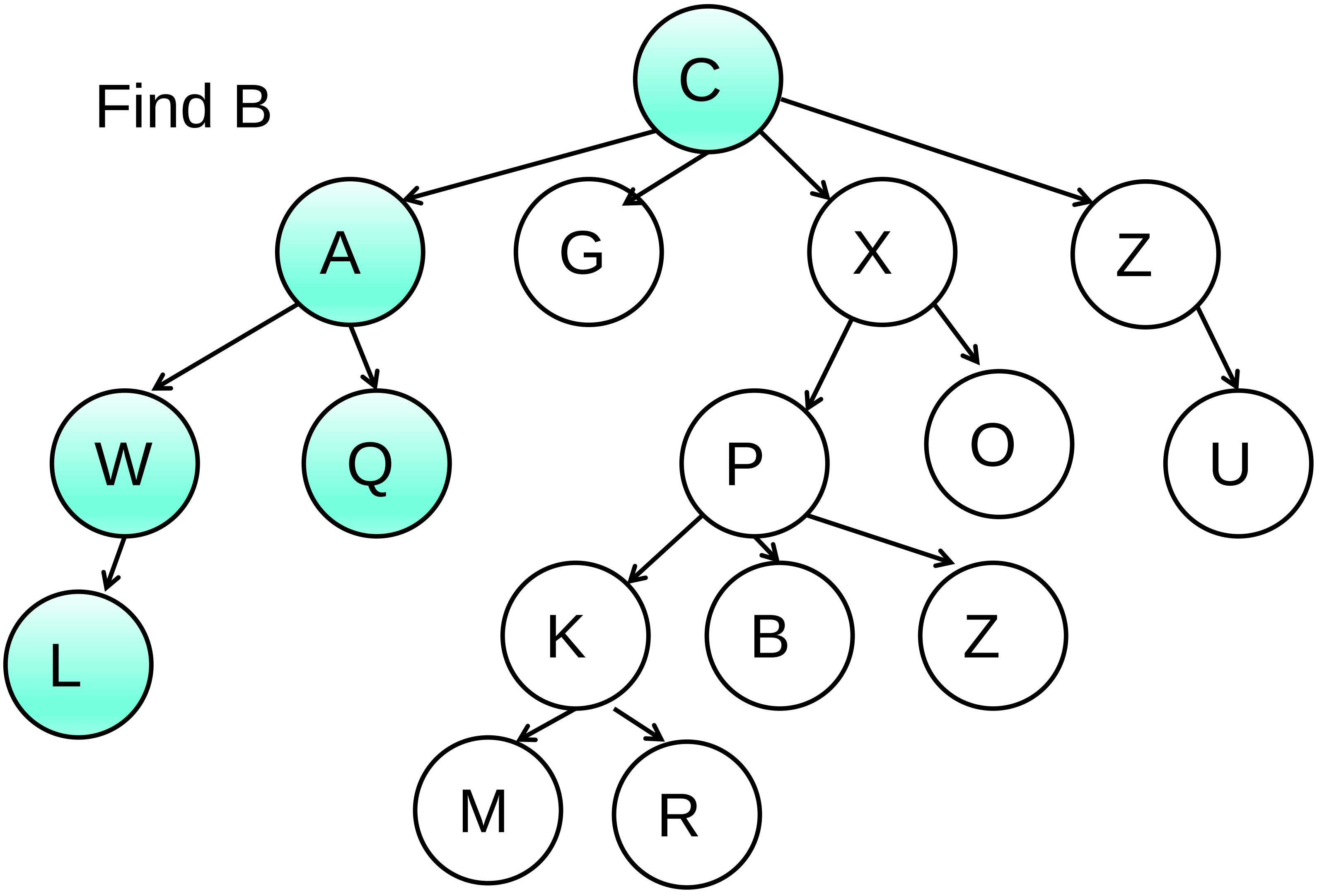


Depth First Search of Tree

Find B

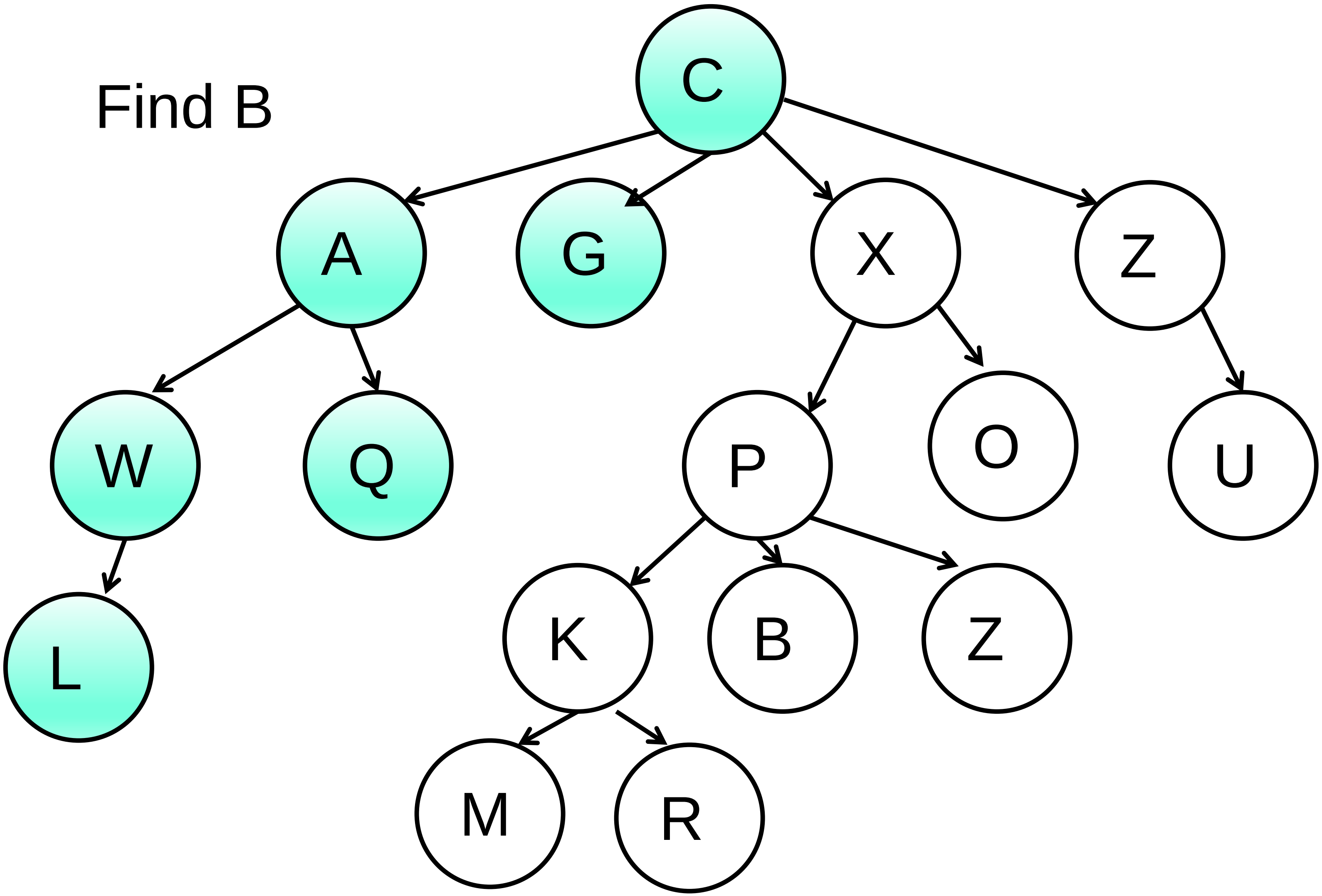


Depth First Search of Tree



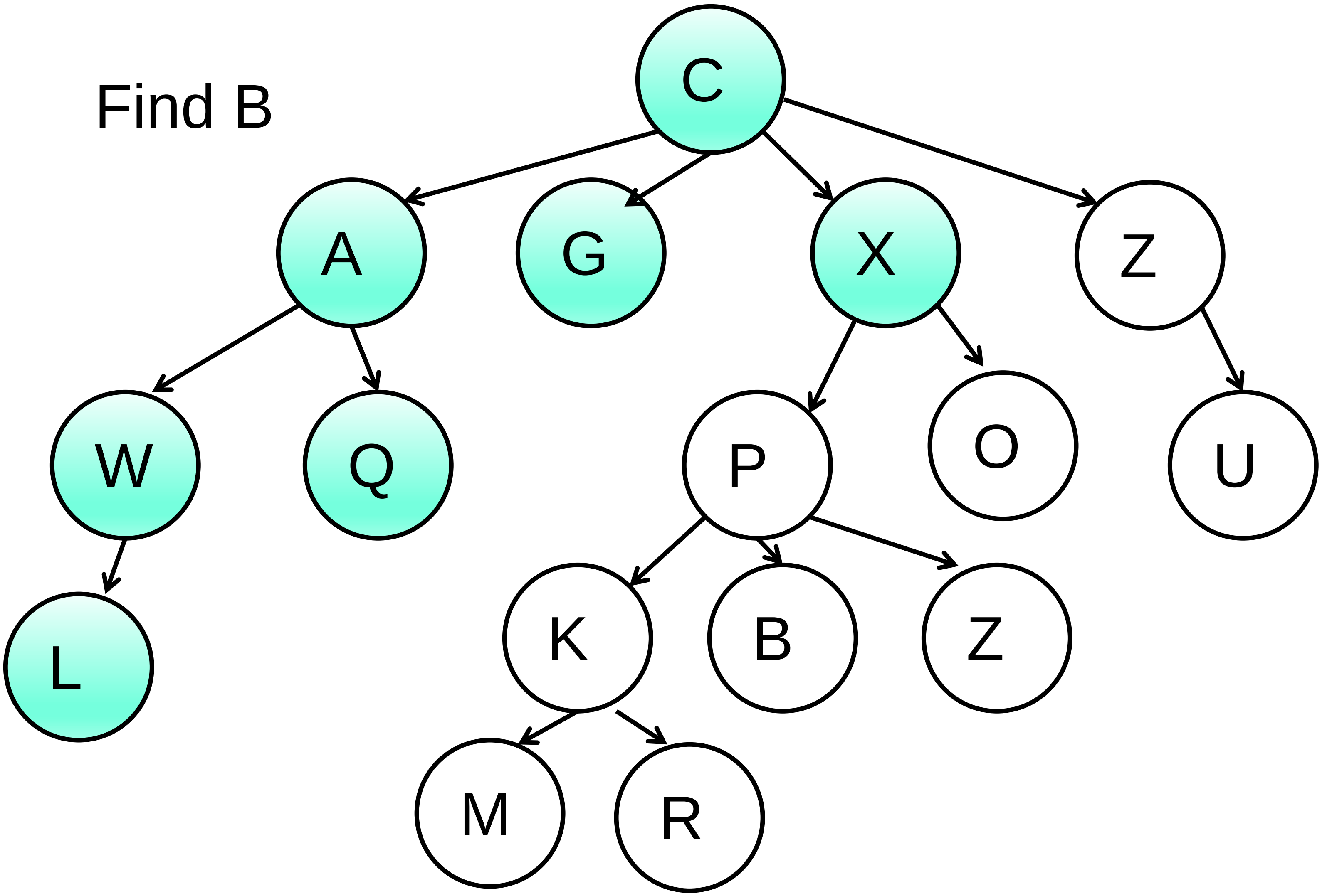
Depth First Search of Tree

Find B



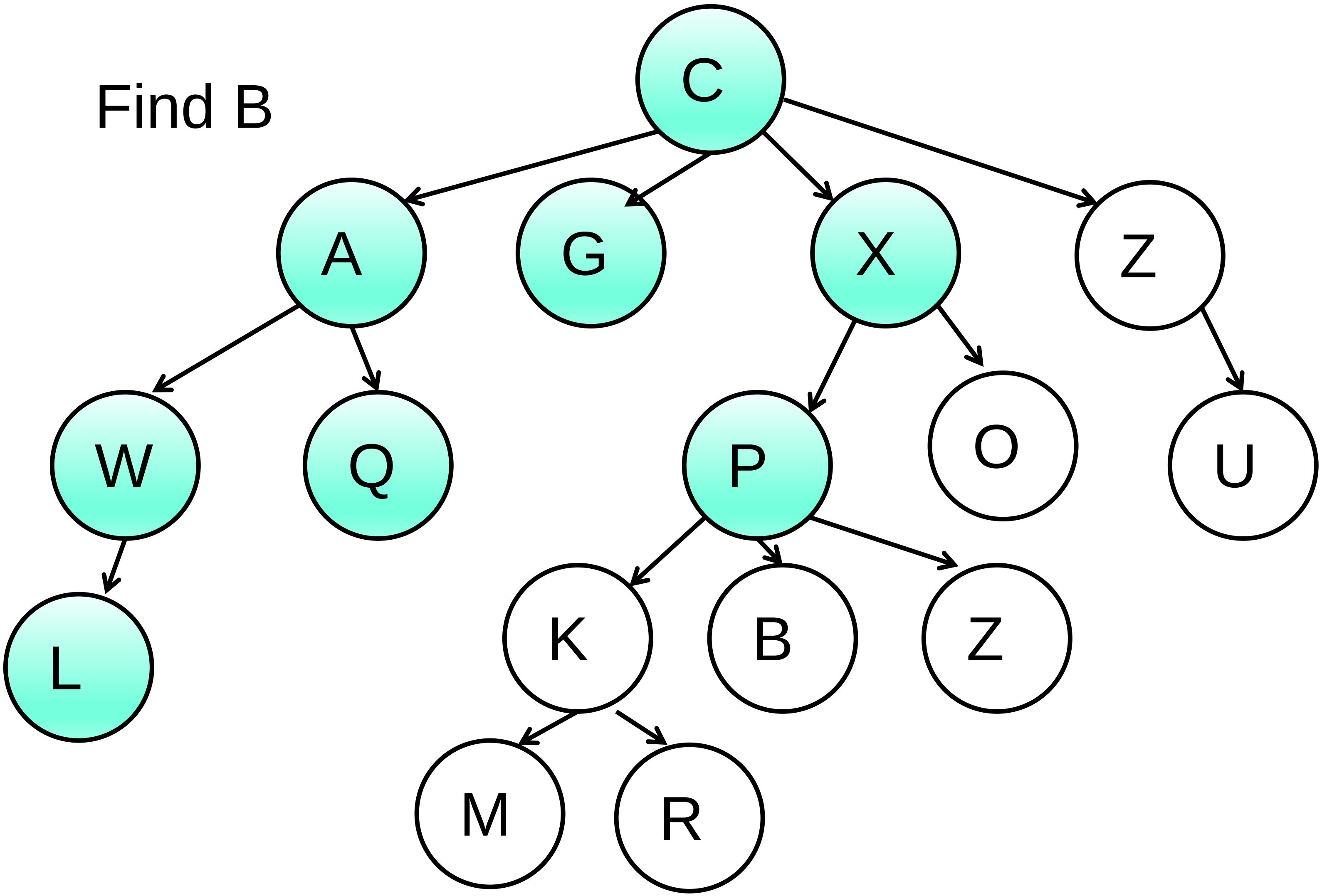
Depth First Search of Tree

Find B



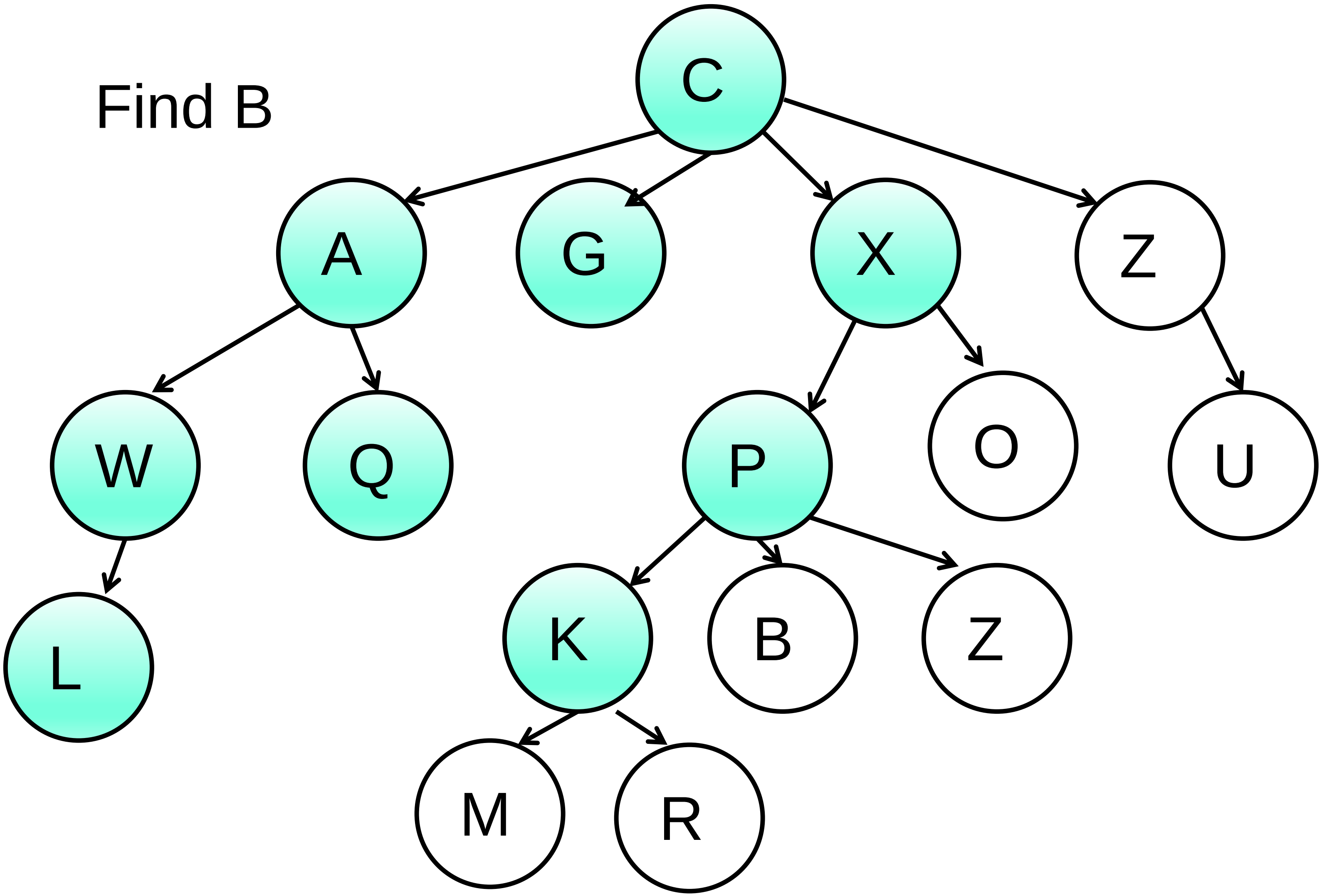
Depth First Search of Tree

Find B



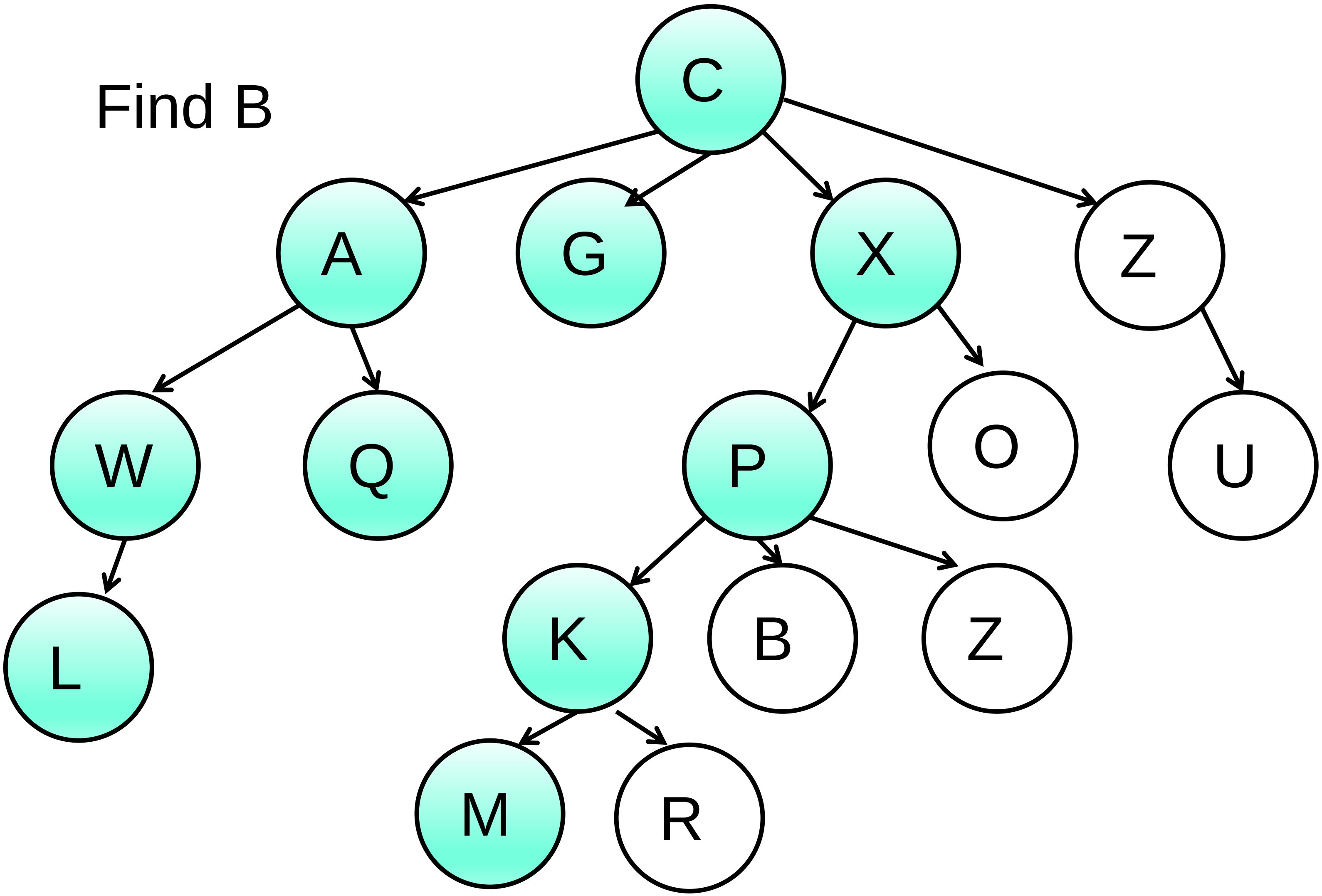
Depth First Search of Tree

Find B



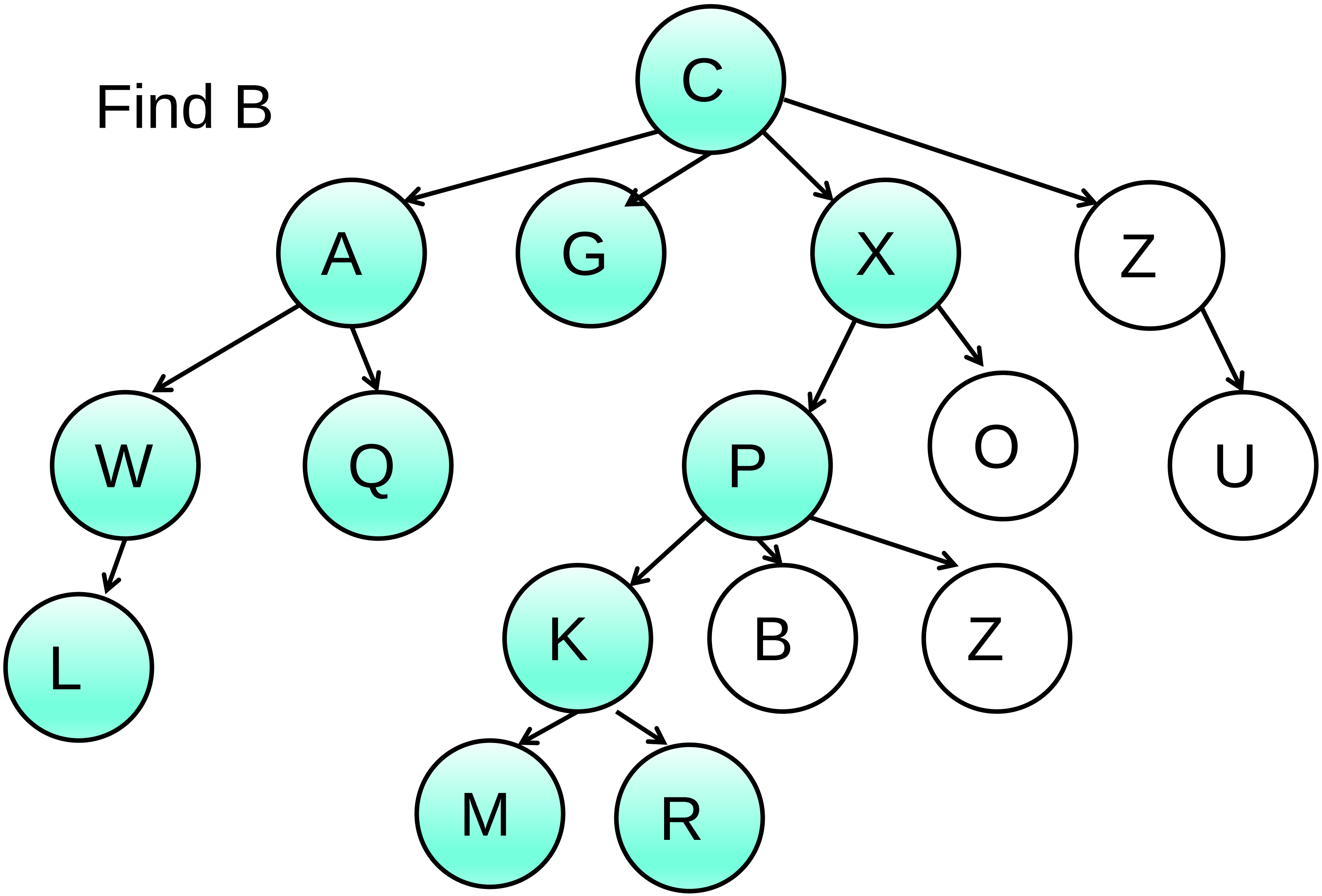
Depth First Search of Tree

Find B



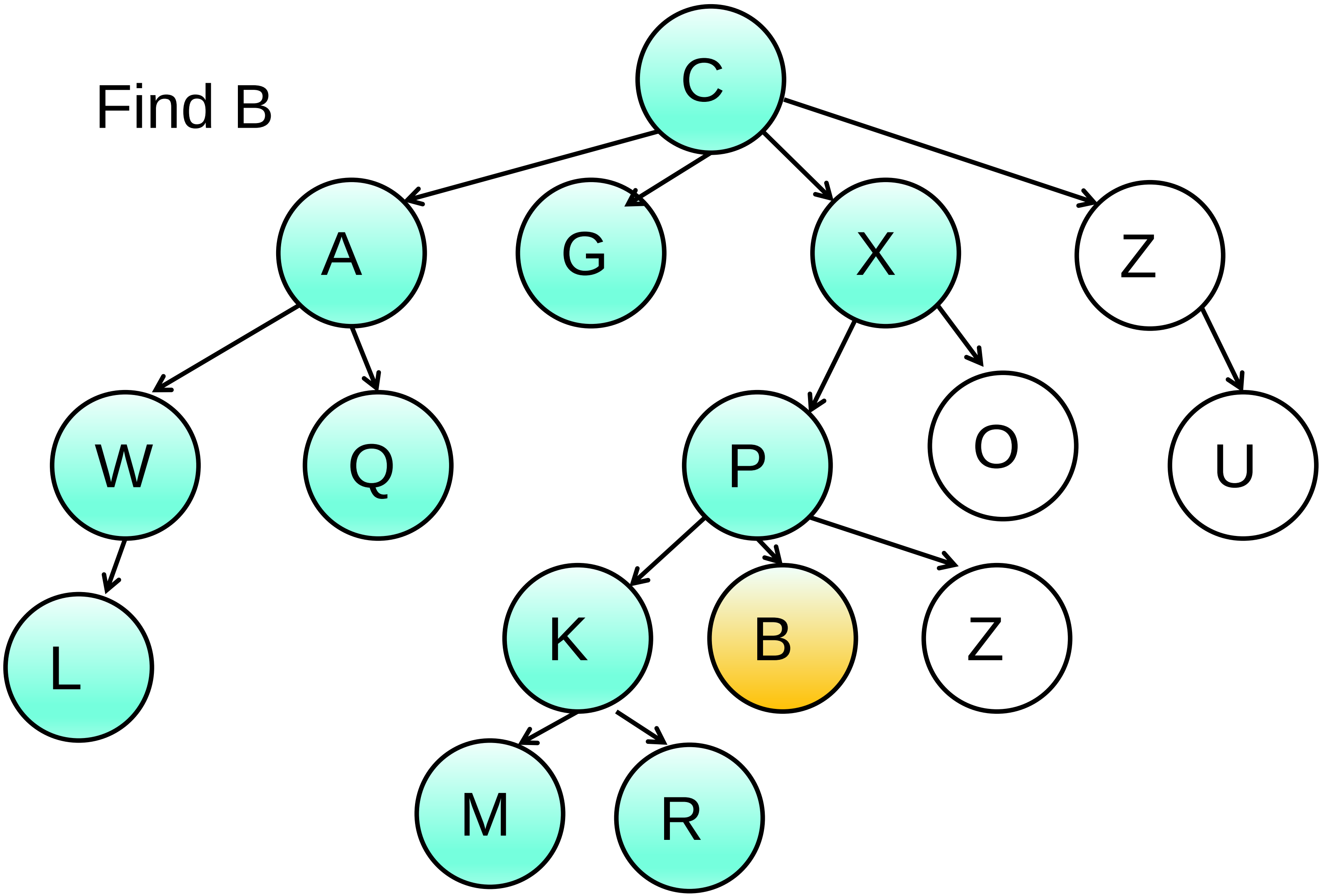
Depth First Search of Tree

Find B



Depth First Search of Tree

Find B



Topic 19
Binary Search Trees

"Yes. Shrubberies are my trade. I am a shrubber. My name is 'Roger the Shrubber'. I arrange, design, and sell shrubberies."

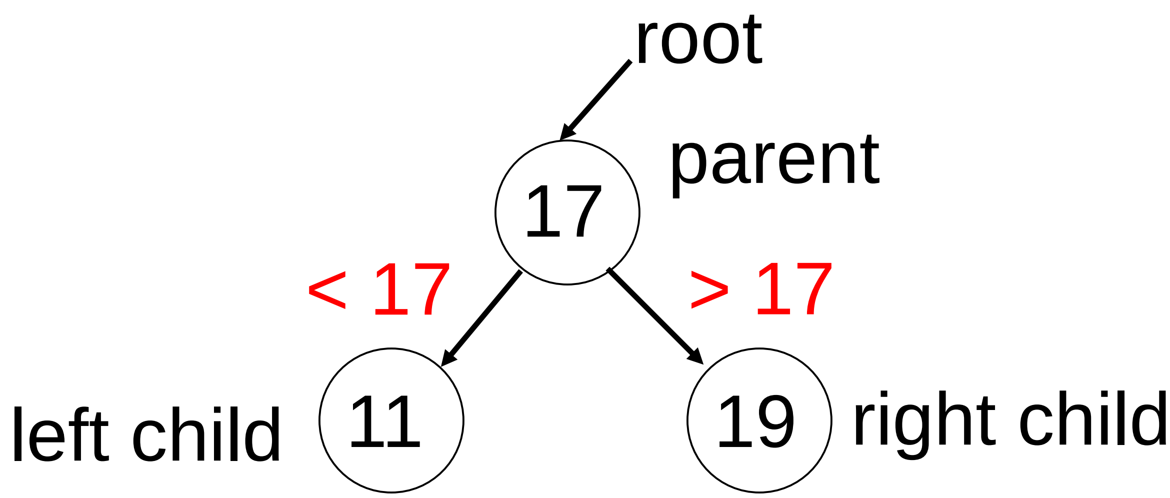
-Monty Python and The Holy Grail

The Problem with Linked Lists

- Accessing a item from a linked list takes O(N) time for an arbitrary element
- Binary trees can improve upon this and reduce access to O(log N) time for the average case
- Expands on the binary search technique and allows insertions and deletions
- Worst case degenerates to O(N) but this can be avoided by using balanced trees (AVL, Red-Black)

Binary Search Trees

- A binary search tree is a binary tree in which **every node's** left subtree holds values less than the node's value, and every right subtree holds values greater than the node's value.
- A new node is added as a leaf.



BST Insertion

- Add the following values one at a time to an initially empty binary search tree using the simple algorithm:

  50 90 20 78 10 20 28 -25

- What is the resulting tree?

Traversals

- What is the result of an inorder traversal of the resulting tree?
- How could a preorder traversal be useful?

Clicker 1

- After adding N distinct elements in random order to a Binary Search Tree what is the expected height of the tree? (using the simple insertion algorithm)

A. $O(\log N)$

B. $O(N^{1/2})$

C. $O(N)$

D. $O(N \log N)$

E. $O(N^2)$

Clicker 2

- After adding N distinct elements to a Binary Search Tree what is the **worst case** height of the tree? (using the simple insertion algorithm)

A. $O(\log N)$
B. $O(N^{1/2})$
C. $O(N)$
D. $O(N\log N)$
E. $O(N^2)$ `

Worst Case Performance

- Insert the following values into an initially empty binary search tree using the simple, naïve algorithm:

2 3 5 7 11 13 17

- What is the height of the tree?
- What is the worst case height of a BST?

Node for Binary Search Trees

```
public class BSTNode<E extends Comparable<E> {
      private Comparable<E> myData;
      private BSTNode<E> myLeft;
      private BSTNode<E> myRightC;

      public BinaryNode(E item)
      {     myData = item;     }

      public E getValue()
      {     return myData;     }

      public BinaryNode<E> getLeft()
      {     return myLeft;     }

      public BinaryNode<E> getRight()
      {     return myRight;    }

      public void setLeft(BSTNode<E> b)
      {     myLeft = b; }
      // setRight not shown
}
```

More on Implementation

- Many ways to implement BSTs
- Using nodes is just one and even then many options and choices

```
public class BinarySearchTree<E extends Comparable<E>> {

      private BSTNode<E> root;
      private int size;
```

Add an Element, Recursive

Add an Element, Iterative

Clicker 3

- What are the best case and worst case order to add N distinct elements, one at a time, to an initially empty binary search tree using the simple add algorithm?

	Best	Worst
A.	O(N)	O(N)
B.	O(NlogN)	O(NlogN)
C.	O(N)	O(NlogN)
D.	O(NlogN)	$O(N^2)$
E.	O(N)	$O(N^2)$

```
// given int[] data
// no duplicates in
// data
BST<Integer> b =
  new BST<Integer>();
for(int x : data)
  b.add(x);
```

Performance of Binary Trees

- For the three core operations (add, access, remove) a binary search tree (BST) has an average case performance of O(log N)
- Even when using the *naïve insertion / removal algorithms*
  - no checks to maintain balance
  - balance achieved based on the randomness of the data inserted

Remove an Element

- Five (four?) cases
  - not present
  - node is a leaf, 0 children (easy)
  - node has 1 child, left or right (easy)
  - node has 2 children ("interesting")

Properties of a BST

- The minimum value is in the left most node
- The maximum value is in the right most node
  - useful when removing an element from the BST

Alternate Implementation

- In class examples of dynamic data structures have relied on *null terminated ends.*
  – Use null to show end of list or no children
- Alternative form
  – use structural recursion and polymorphism

BST Interface

```
public interface BST<E extends
                       Comparable<? super E>> {

    public int size();
    public boolean contains(E obj);
    public BST<E> add(E obj);
}
```

EmptyBST

```
public class EmptyBST<E extends Comparable<? super E>>
      implements BST<E> {

   private static final EmptyBST theOne = new EmptyBST();

   private EmptyBST() {}

   public static EmptyBST getEmptyBST(){ return theOne; }

   public  BST<E> add(E obj) { return new NEBST(obj); }

   public boolean contains(E obj) { return false; }

   public int size() { return 0; }
}
```

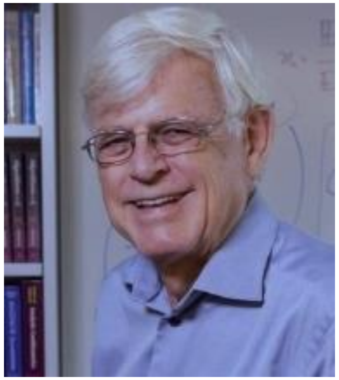
Non Empty BST – Part 1

```
public class NEBST <E extends Comparable<? super E>> implements BST<E> {

    private E data;
    private BST left;
    private BST right;

    public NEBST(E d) {
        data = d;
        right = EmptyBST.getEmptyBST();
        left = EmptyBST.getEmptyBST();
    }

    public BST add(E obj)  {
        int direction = obj.compareTo( data );
        if ( direction < 0 )
          left = left.add( obj );
        else if ( direction > 0 )
          right = right.add ( obj );
        return this;
    }
```

Non Empty BST – Part 2

```
public boolean contains(E obj){
    int dir = obj.compareTo(data);
    if( dir == 0 )
        return true;
    else if (dir < 0)
        return left.contains(obj);
    else
        return right.contains(obj);
  }

  public int size() {
    return 1 + left.size() + right.size();
  }
}
```

Topic 20
Red Black Trees



"Welcome to L.A.'s Automated Traffic Surveillance and Control Operations Center. See, they use video feeds from intersections and specifically designed algorithms to predict traffic conditions, and thereby control traffic lights. So all I did was come up with my own... **kick ass algorithm** to sneak in, and now we own the place."

-Lyle, the Napster, (Seth Green), *The Italian Job*

Red Black Trees were created by Leonidas J. Guibas and Robert Sedgewick in 1978

Clicker 1

- 2000 elements are inserted one at a time into an initially empty binary search tree using the simplenaive algorithm. What is the maximum possible height of the resulting tree?

A. 1

B. 11

C. 21

D. 500

E. 1999

Binary Search Trees

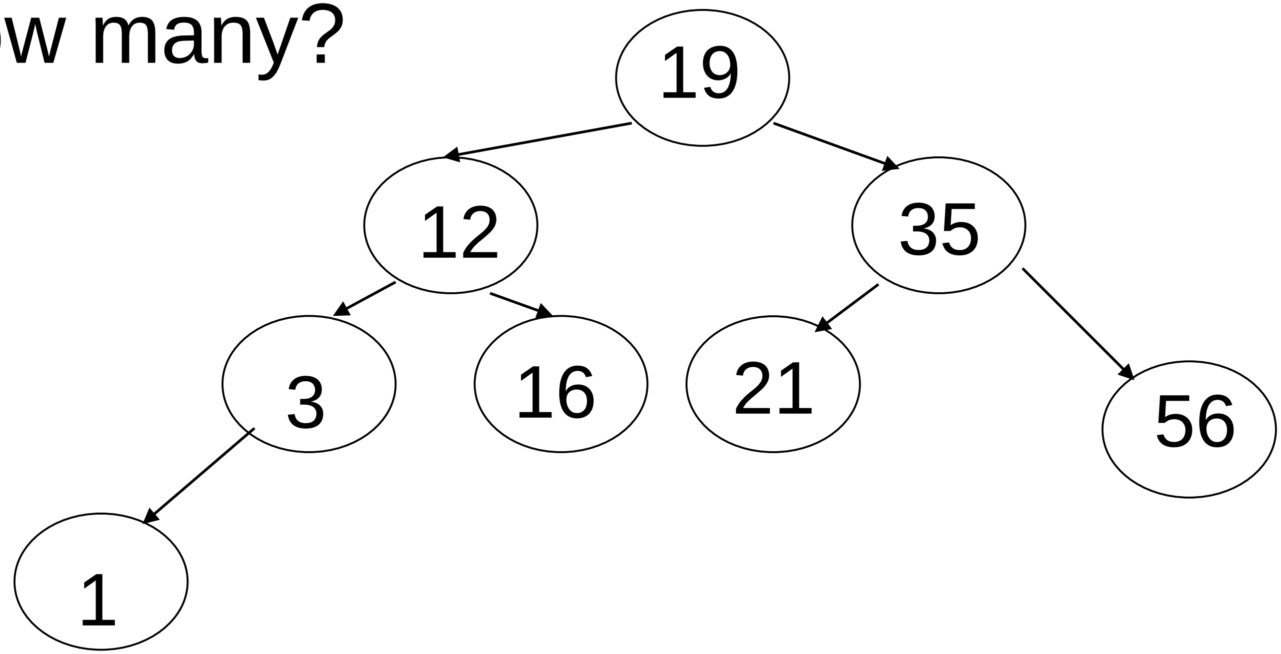
- Average case and worst case Big O for
  - insertion
  - deletion
  - access
- Balance is important. Unbalanced trees give worse than log N times for the basic tree operations
- Can balance be guaranteed?

Red Black Trees

- A BST with more complex algorithms to ensure balance
- Each node is labeled as Red or Black.
- Path: A unique series of links (edges) traverses from the root to each node.
  – The number of edges (links) that must be followed is the path length
- In Red Black trees paths from the root to elements with 0 or 1 child are of particular interest

Paths to Single or Zero Child Nodes

- How many?

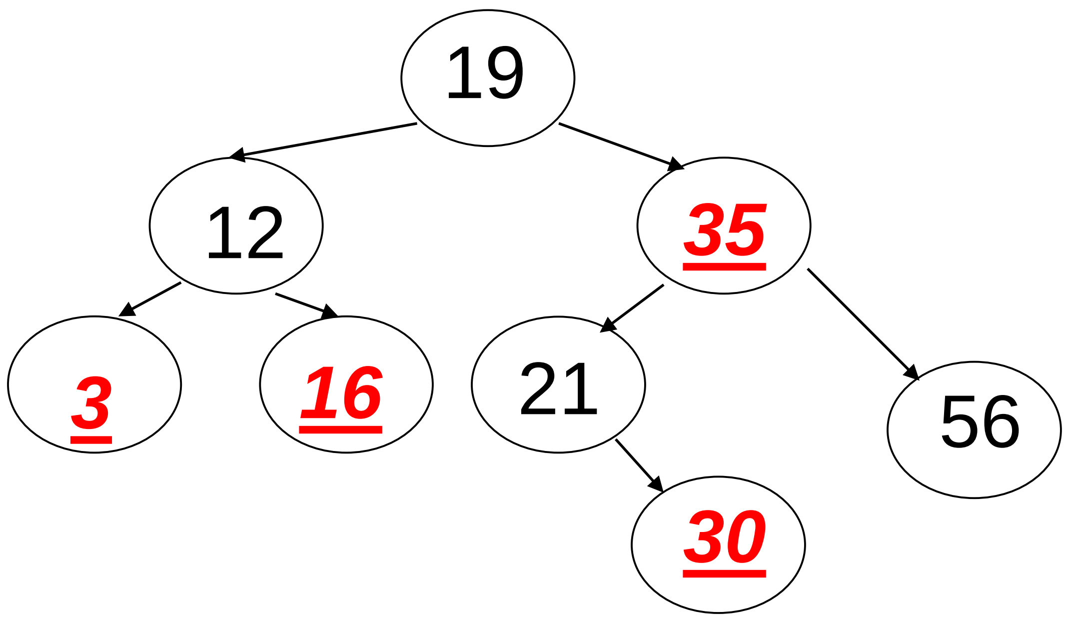


Red Black Tree Rules

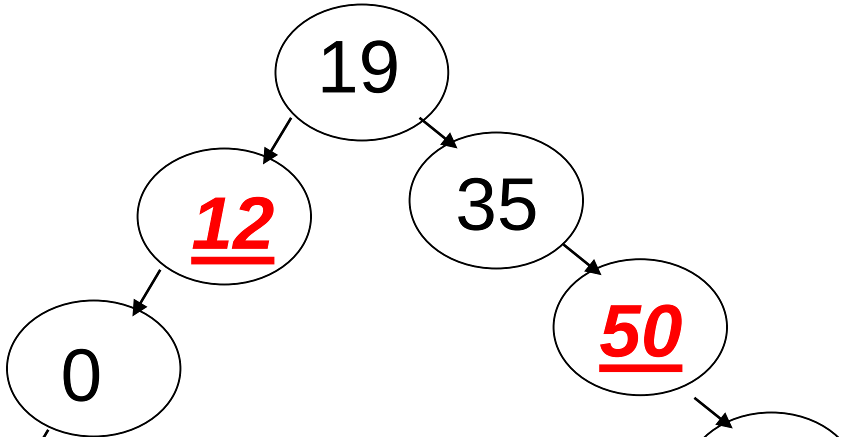
1. Is a binary search tree
2. Every node is colored either red or black
3. The root of the whole tree is black
4. If a node is red its children must be black. (a.k.a. the red rule)
5. Every path from a node to a null link must contain the same number of black nodes (a.k.a. the path rule)

Example of a Red Black Tree

- The root of a Red Black tree is black
- Every other node in the tree follows these rules:
  - Rule 3: If a node is Red, all of its children are Black
  - Rule 4: The number of Black nodes must be the same in all paths from the root node to null nodes



Red Black Tree?

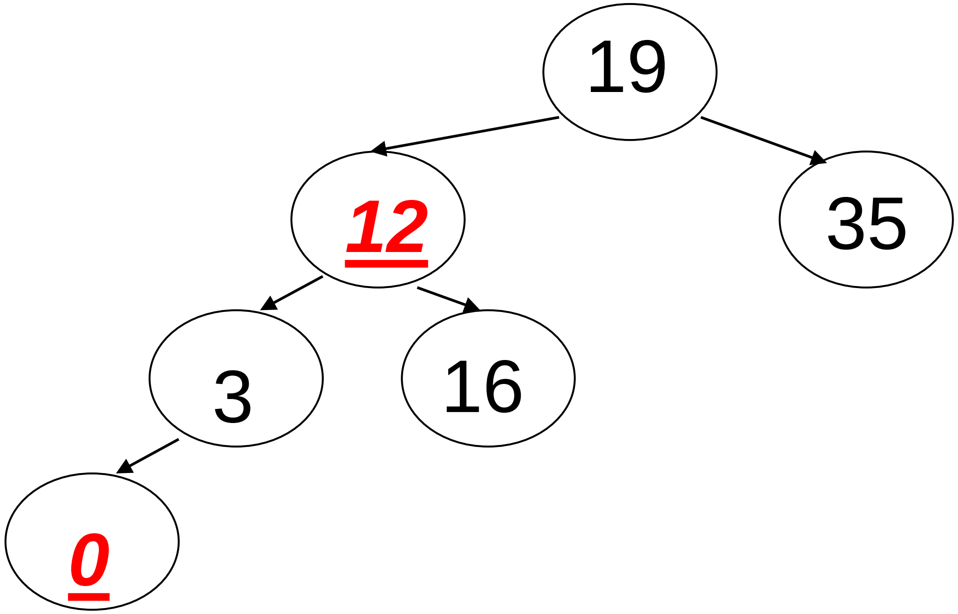


Clicker 2

- Is the tree on the previous slide a binary search tree? Is it a red black tree?

	BST?	Red-Black?
A.	No	No
B.	No	Yes
C.	Yes	No
D.	Yes	Yes

Red Black Tree?



Perfect?
Full?
Complete?

Clicker 3

- Is the tree on the previous slide a binary search tree? Is it a red black tree?

	BST?	Red-Black?
A.	No	No
B.	No	Yes
C.	Yes	No
D.	Yes	Yes

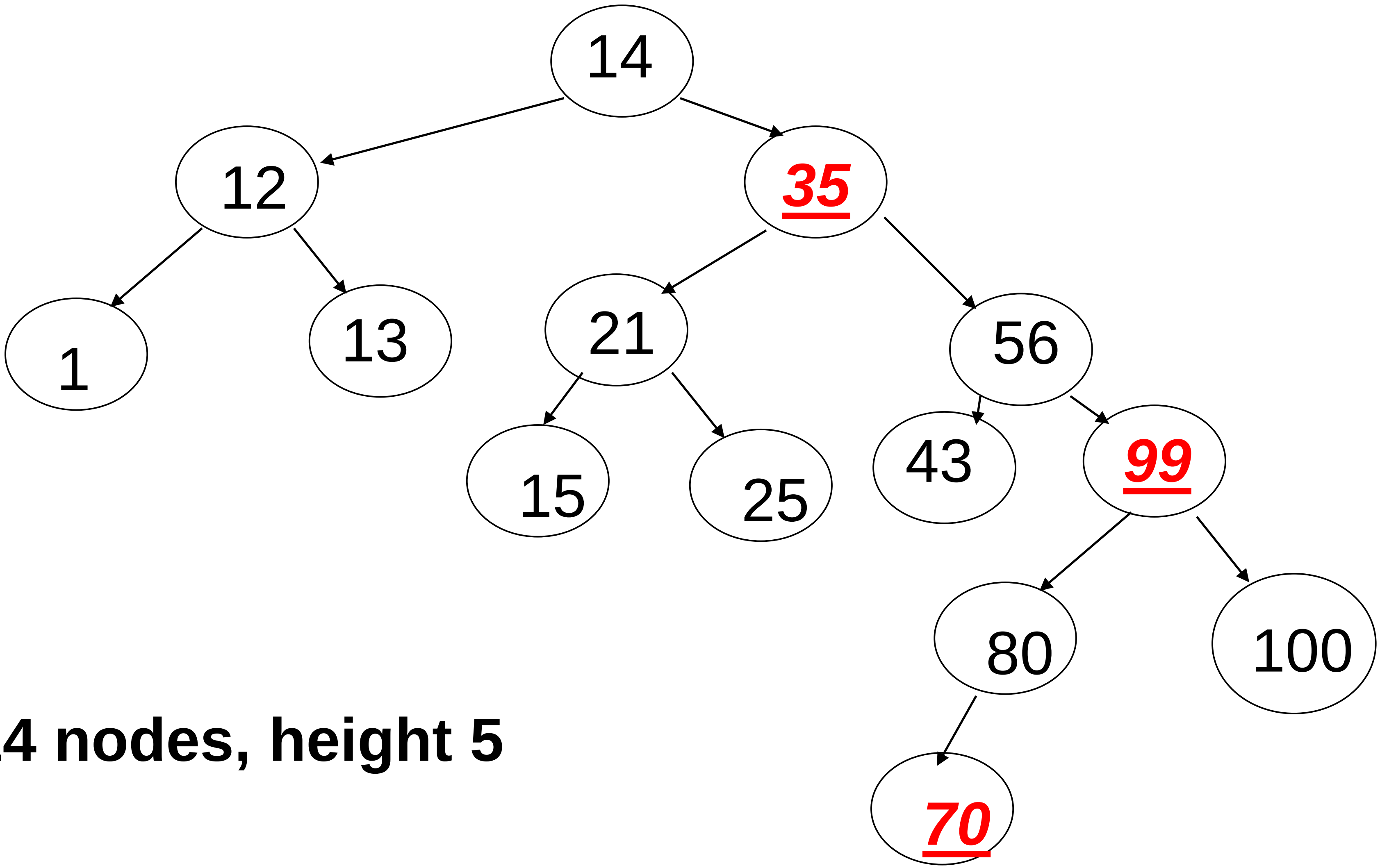
Implications of the Rules

- If a Red node has any children, it must have two children and they must be Black. (Why?)
- If a Black node has only one child that child must be a Red leaf. (Why?)
- Due to the rules there are limits on how unbalanced a Red Black tree may become.
  - on the previous example may we hang a new node off of the leaf node that contains 0?

Properties of Red Black Trees

- If a Red Black Tree is complete, with all Black nodes except for Red leaves at the lowest level the height will be minimal, ~log N
- To get the max height for N elements there should be as many Red nodes as possible down one path and all other nodes are Black
  - This means the max height would b **approximately** 2 * log N (don't use this as a formula)
  - typically less than this
  - see example on next slide
  - interesting exercise, draw max height tree with N nodes

Max Height Red Black Tree



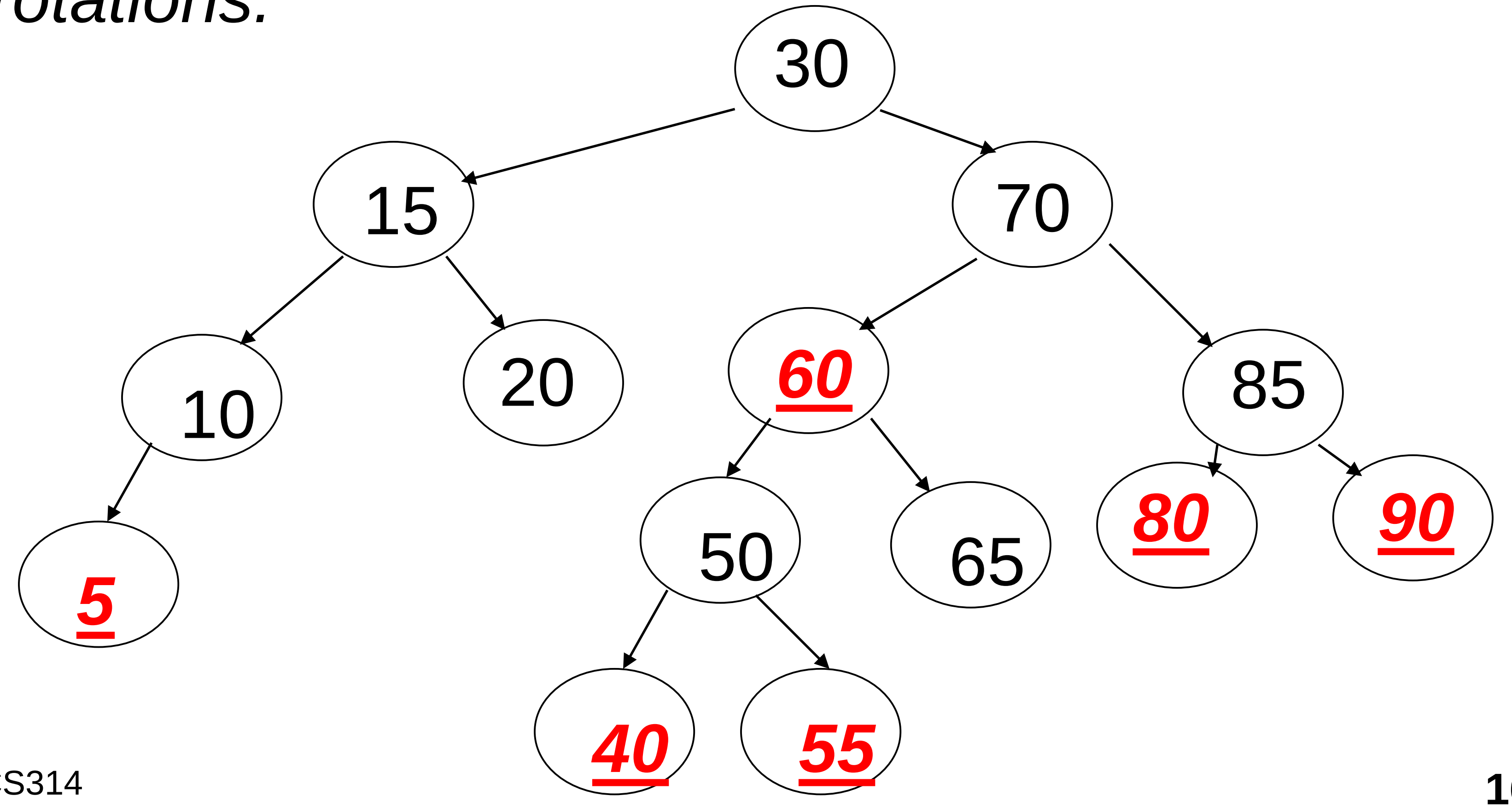
14 nodes, height 5

Maintaining the Red Black Properties in a Tree

- Insertions
- Must maintain rules of Red Black Tree.
- New Value always in a new leaf, to start
  - can't be black or we will violate rule 4
  - therefore the new leaf must be red
  - If parent is black, done (trivial case)
  - if parent red, things get interesting because a red leaf with a red parent violates rule 3

Insertions with Red Parent - Child

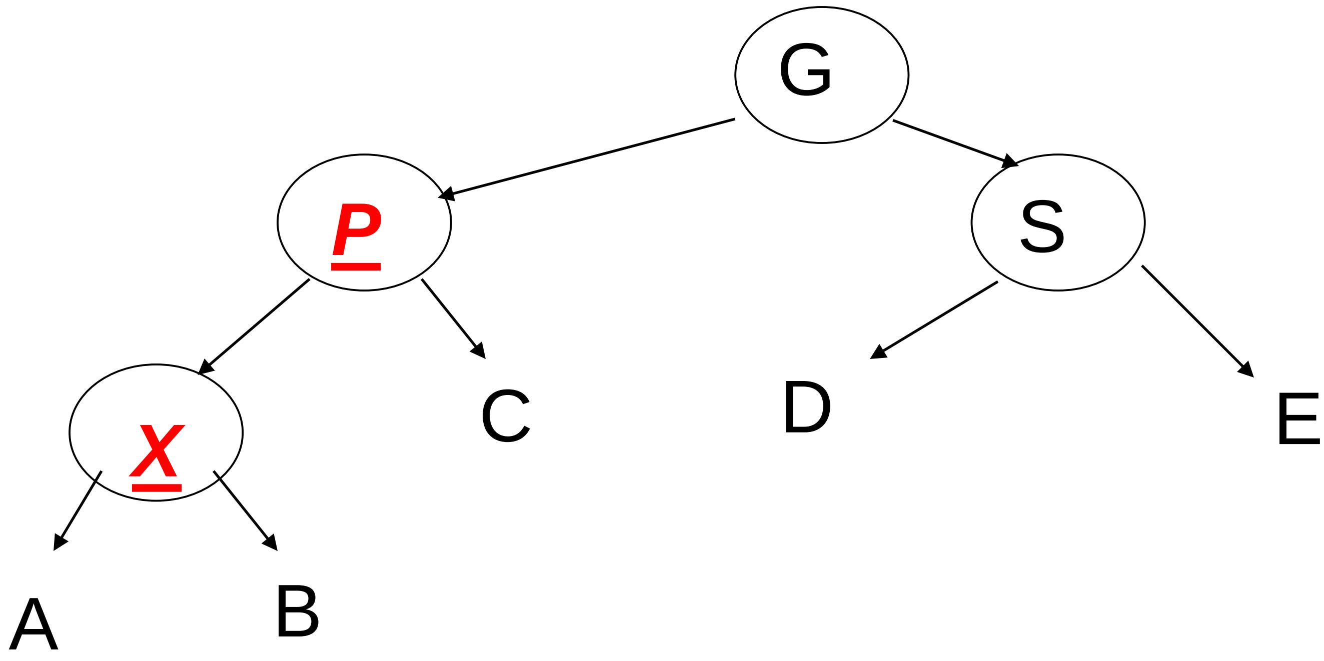
Must modify tree when insertion would result in Red Parent - Child pair using color changes and *rotations.*



Case 1

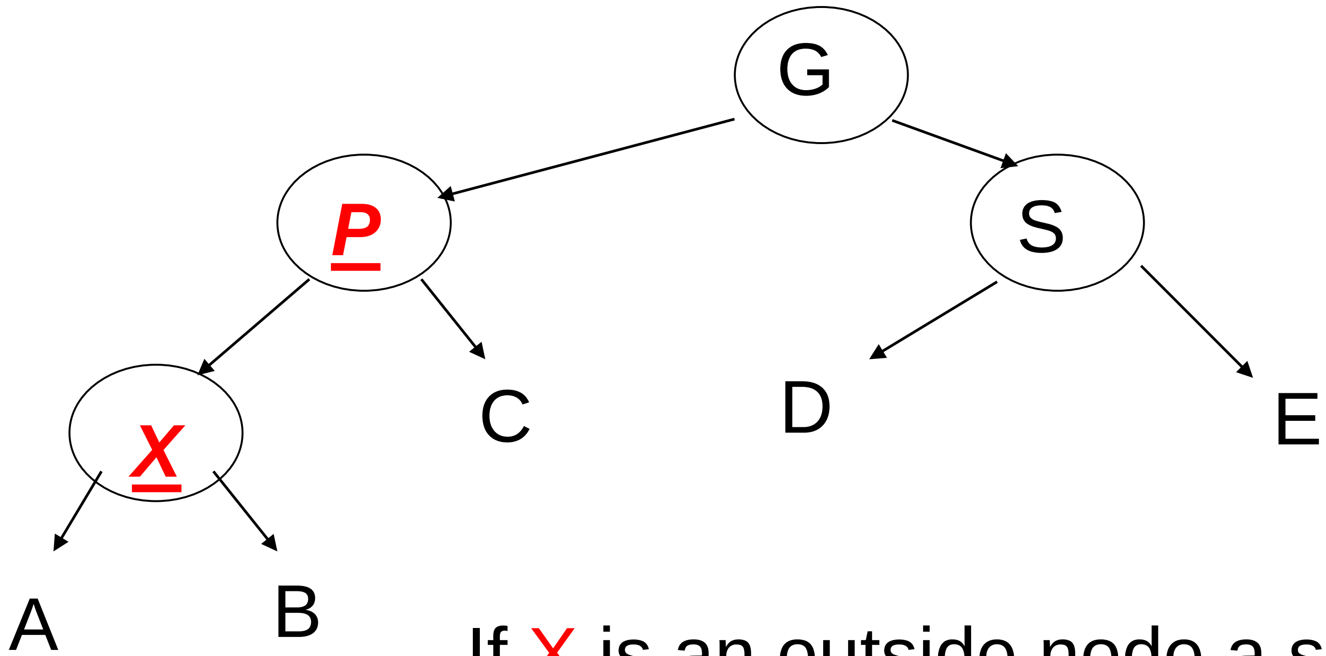
- Suppose sibling of parent is Black.
  - by convention null nodes are black
- In the previous tree, true if we are inserting a 3 or an 8.
  - What about inserting a 99? Same case?
- Let X be the new leaf Node, P be its Red Parent, S the Black sibling and G, P's and S's parent and X's grandparent
  - What color is G?

Case 1 - The Picture



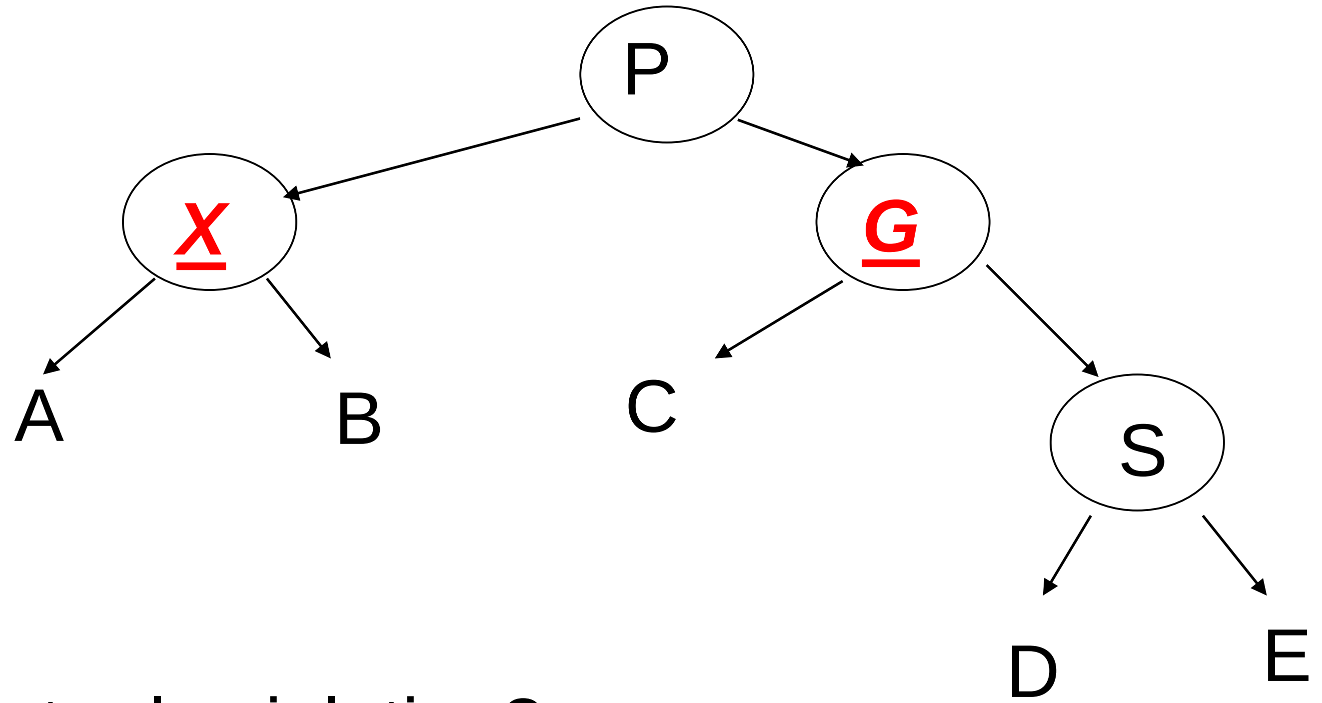
Relative to G, X could be an *inside* or *outside* node.
Outside -> left left or right right moves
Inside -> left right or right left moves

Fixing the Problem



If X is an outside node a single *rotation* between P and G fixes the problem.
A rotation is an exchange of roles between a parent and child node. So P becomes G's parent. Also must recolor P and G.

Single Rotation



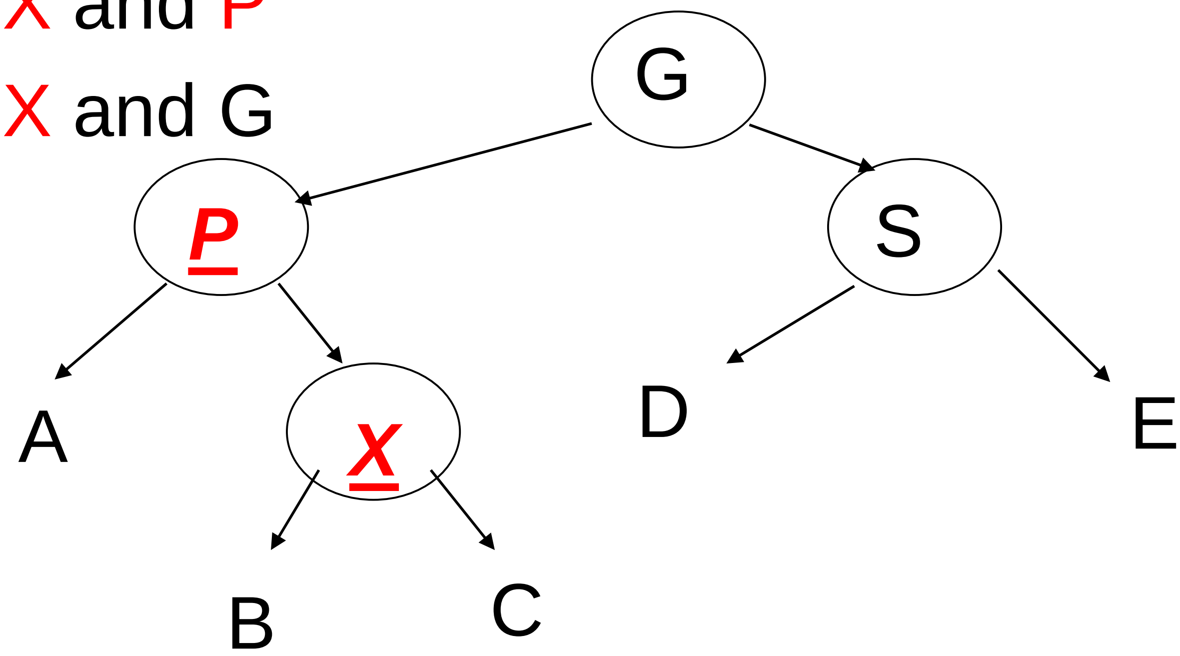
Apparent rule violation?

Recall, S is null if X is a leaf, so no problem

If this occurs higher in the tree (why?) subtrees A, B, and C will have one more black node than D and E.

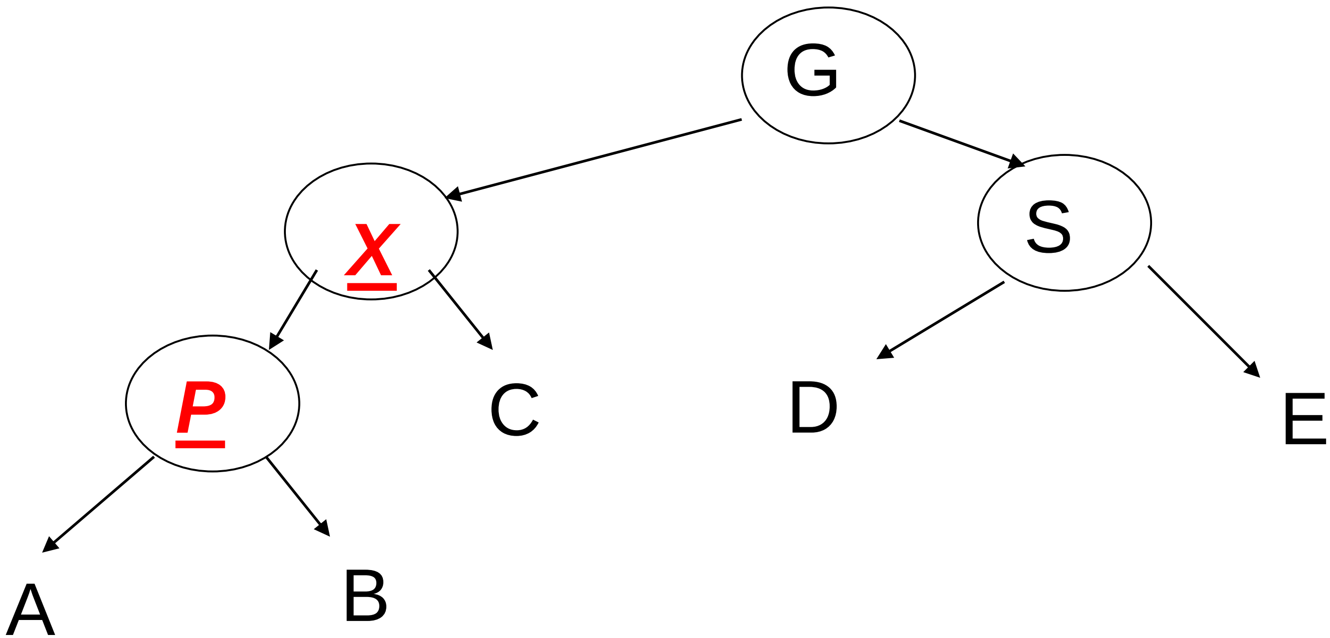
Case 2

- What if X is an inside node relative to G?
  - a single rotation will not work
- Must perform a double rotation
  - rotate X and P
  - rotate X and G



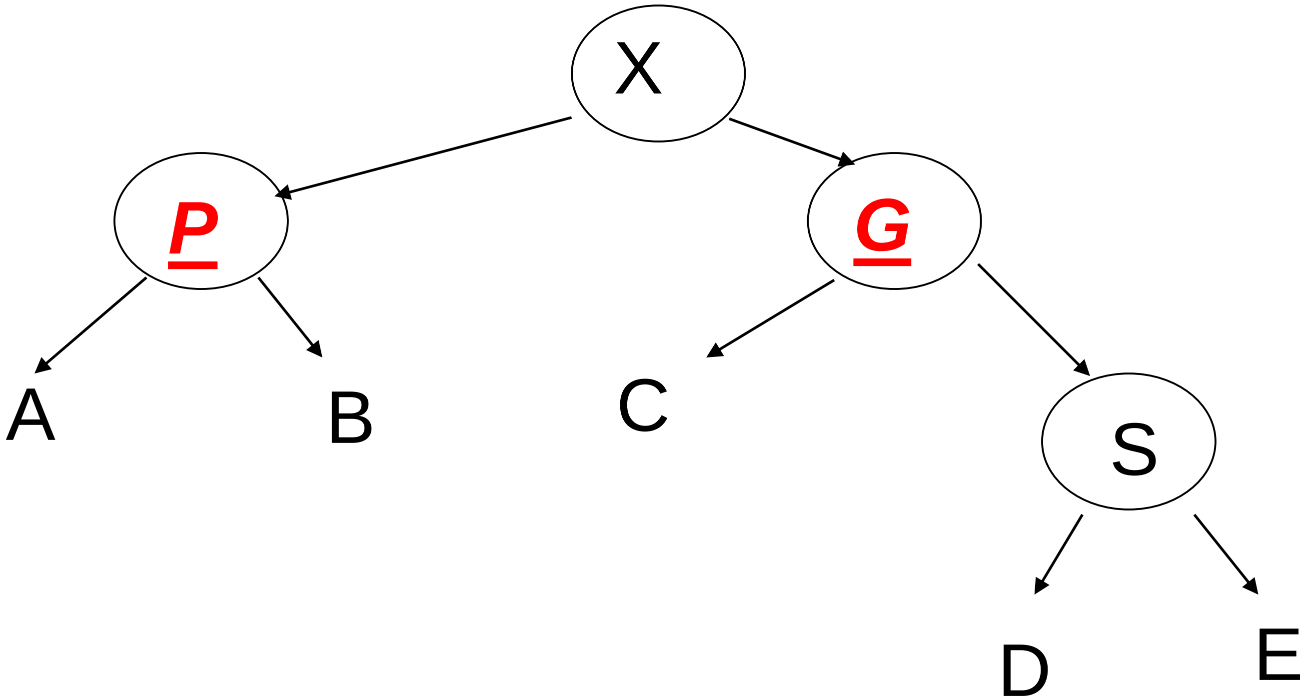
First Rotation

- Rotate P and X, no color change

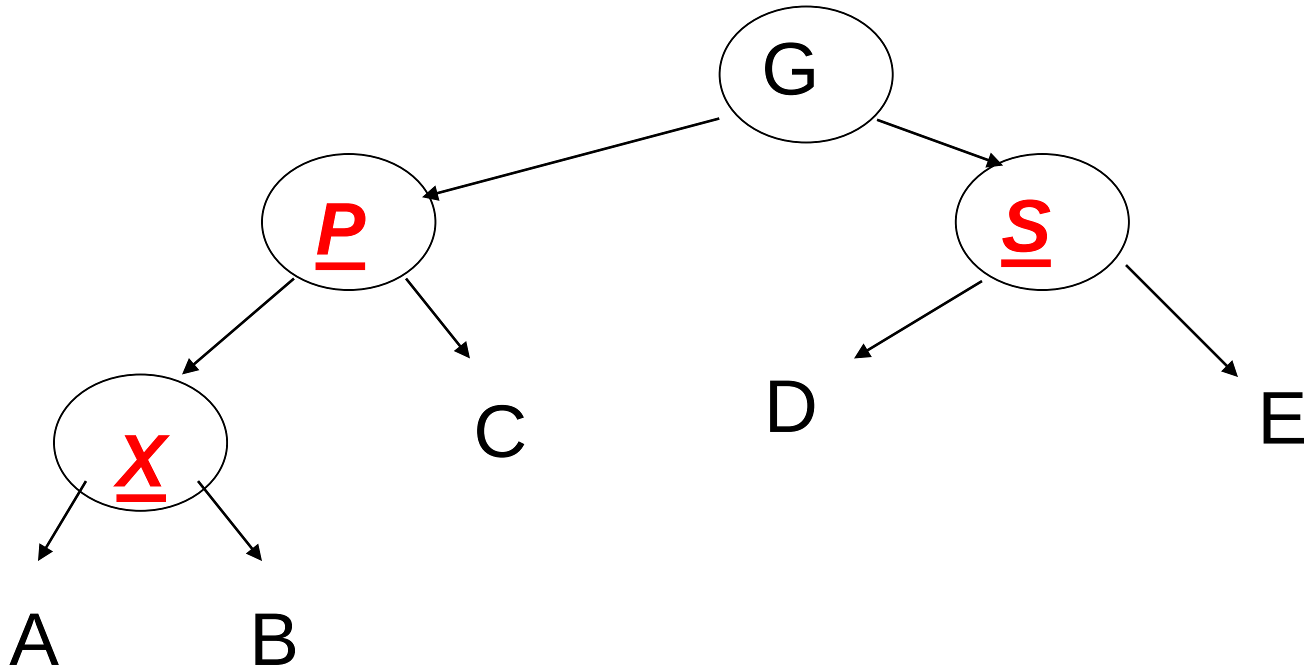


- What does this actually do?

After Double Rotation



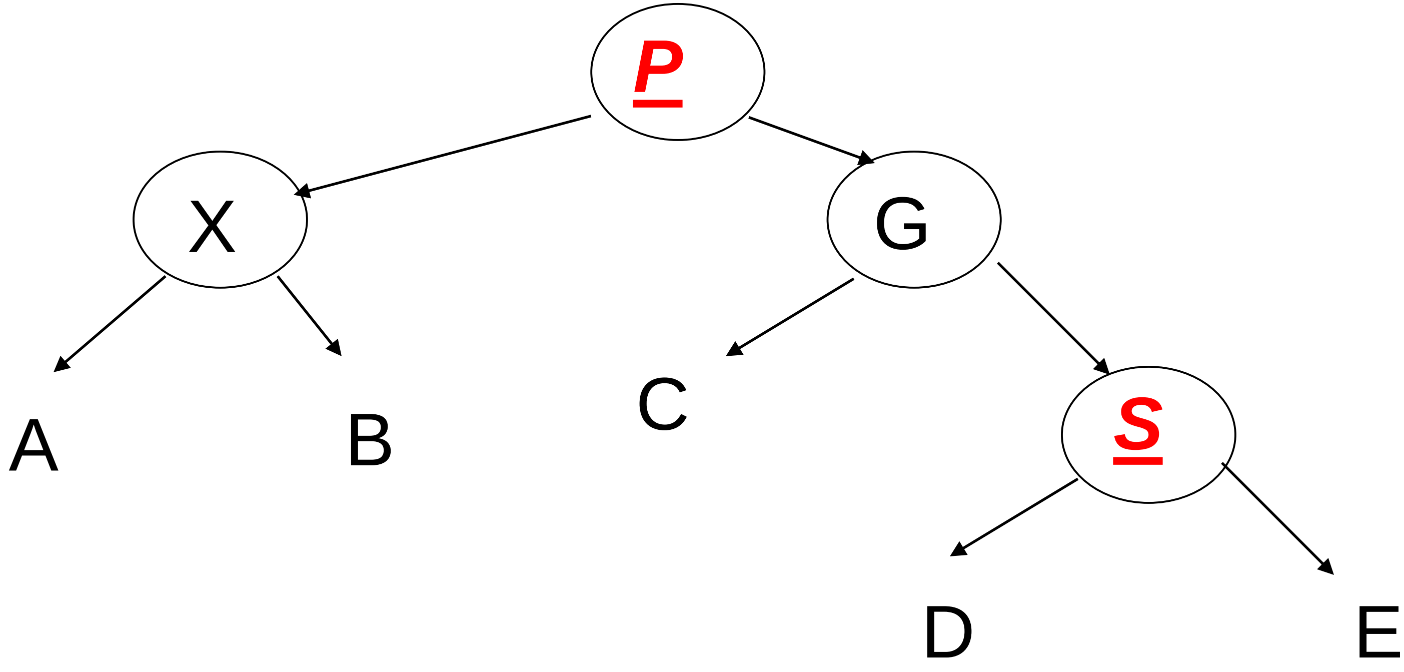
Case 3
Sibling is Red, not Black



Any problems?

Fixing Tree when S is Red

- Must perform single rotation between parent, P and grandparent, G, and then make appropriate color changes

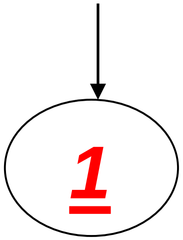


More on Insert

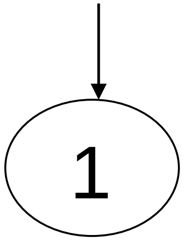
- Problem: What if on the previous example G's parent (GG!) had been red?
- Easier to never let Case 3 ever occur!
- On the way down the tree, if we see a node X that has 2 Red children, we make X Red and its two children black.
  - if recolor the root, recolor it to black
  - the number of black nodes on paths below X remains unchanged
  - If X's parent was Red then we have introduced 2 consecutive Red nodes.(violation of rule)
  - to fix, apply rotations to the tree, same as inserting node

Example of Inserting Sorted Numbers

- 1 2 3 4 5 6 7 8 9 10

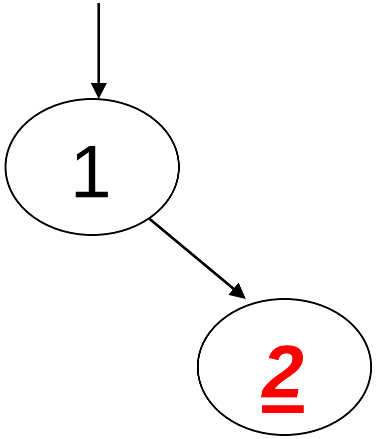


Insert 1. A leaf so red. Realize it is root so recolor to black.



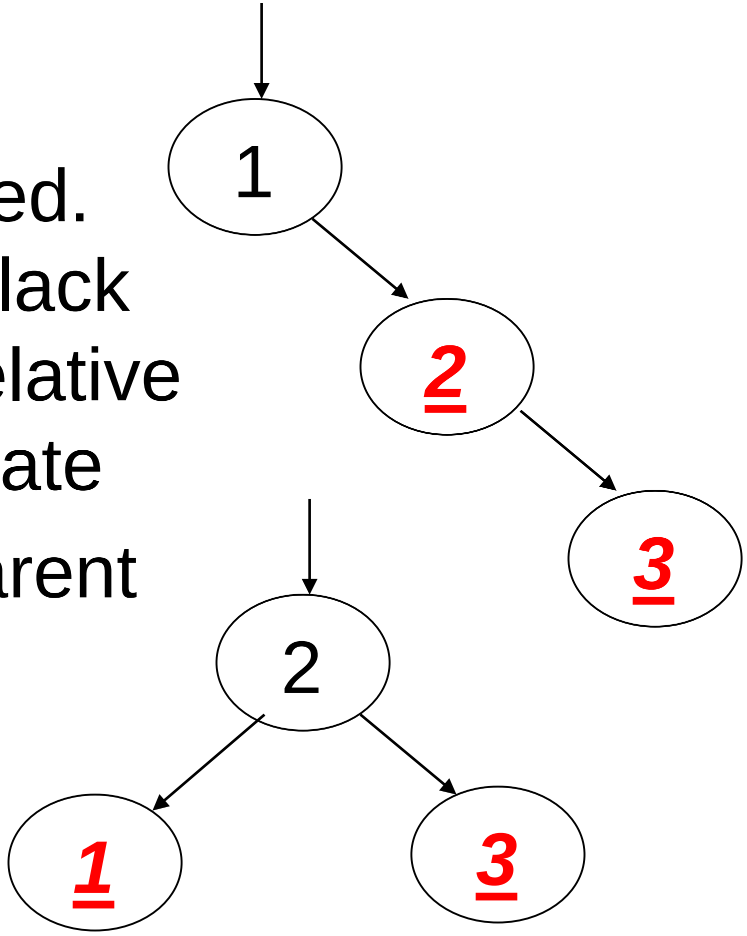
Insert 2

make 2 red. Parent is black so done.



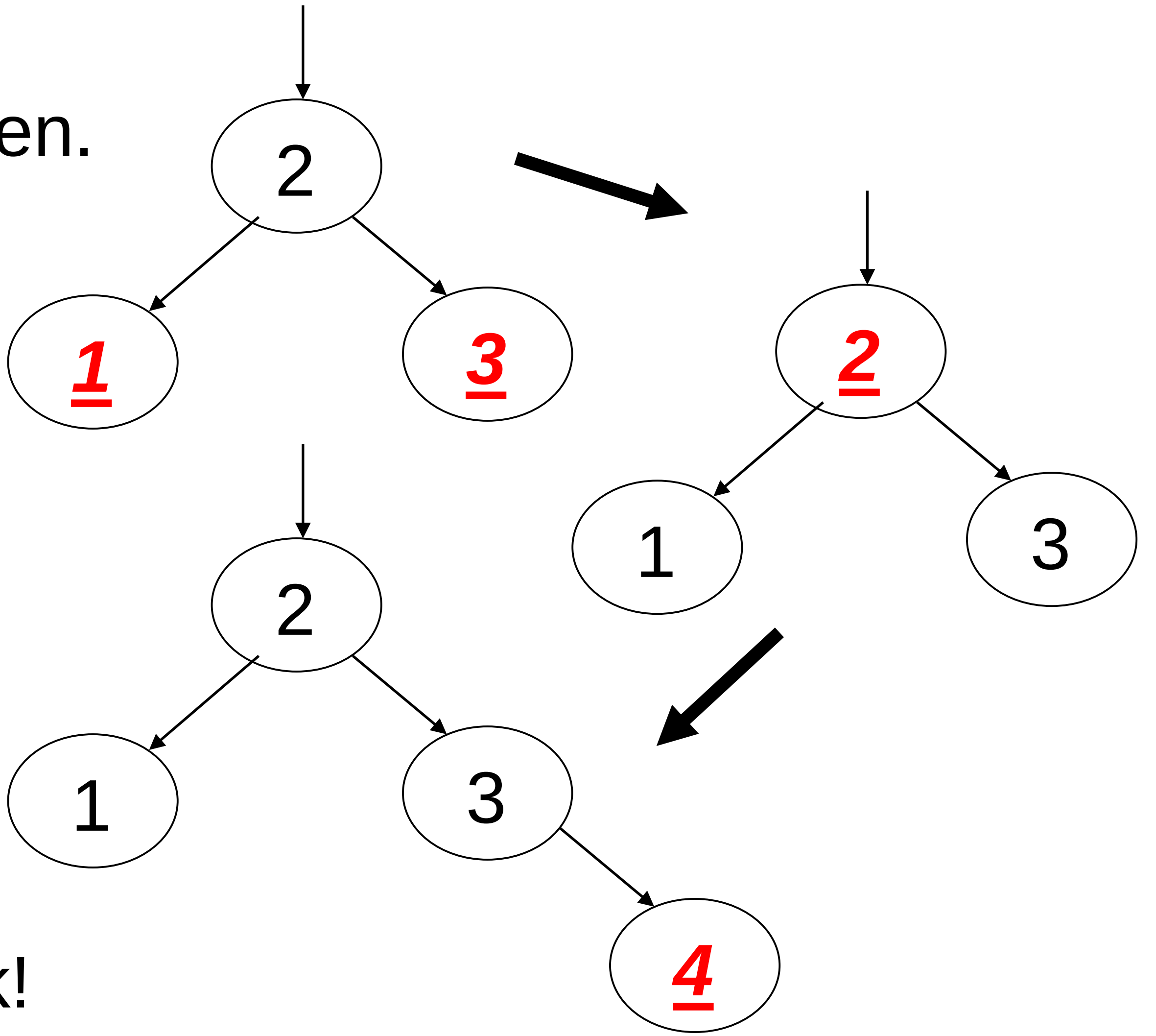
Insert 3

Insert 3. Parent is red. Parent's sibling is black (null) 3 is outside relative to grandparent. Rotate parent and grandparent



Insert 4

On way down see 2 with 2 red children. Recolor 2 red and children black.

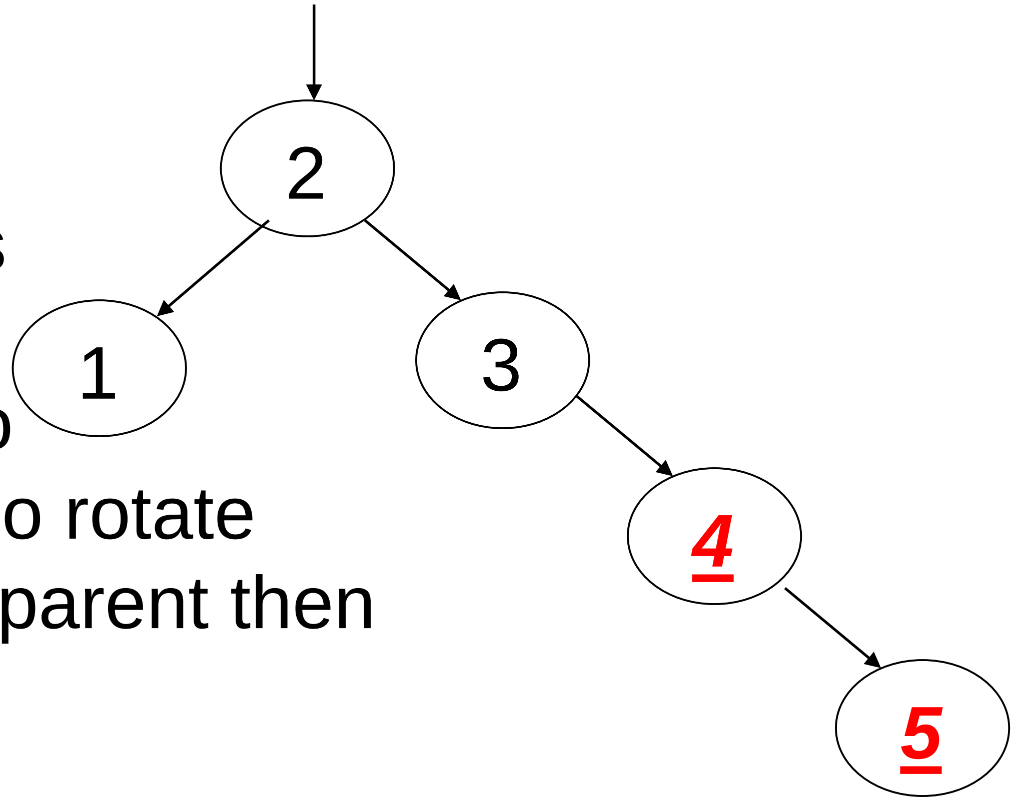


When adding 4 parent is black so done.

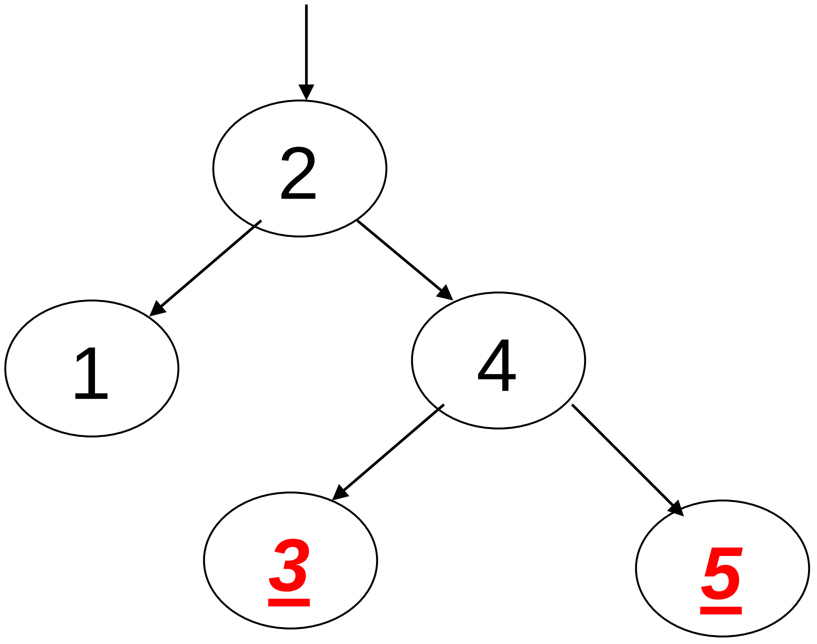
Set root to black!

Insert 5

5's parent is red. Parent's sibling is black (null). 5 is outside relative to grandparent (3) so rotate parent and grandparent then recolor

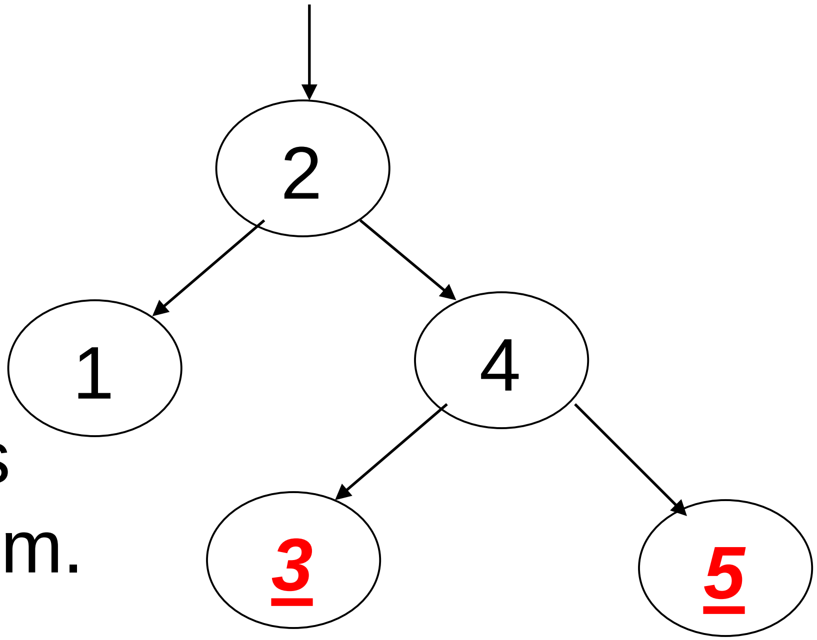


Finish insert of 5



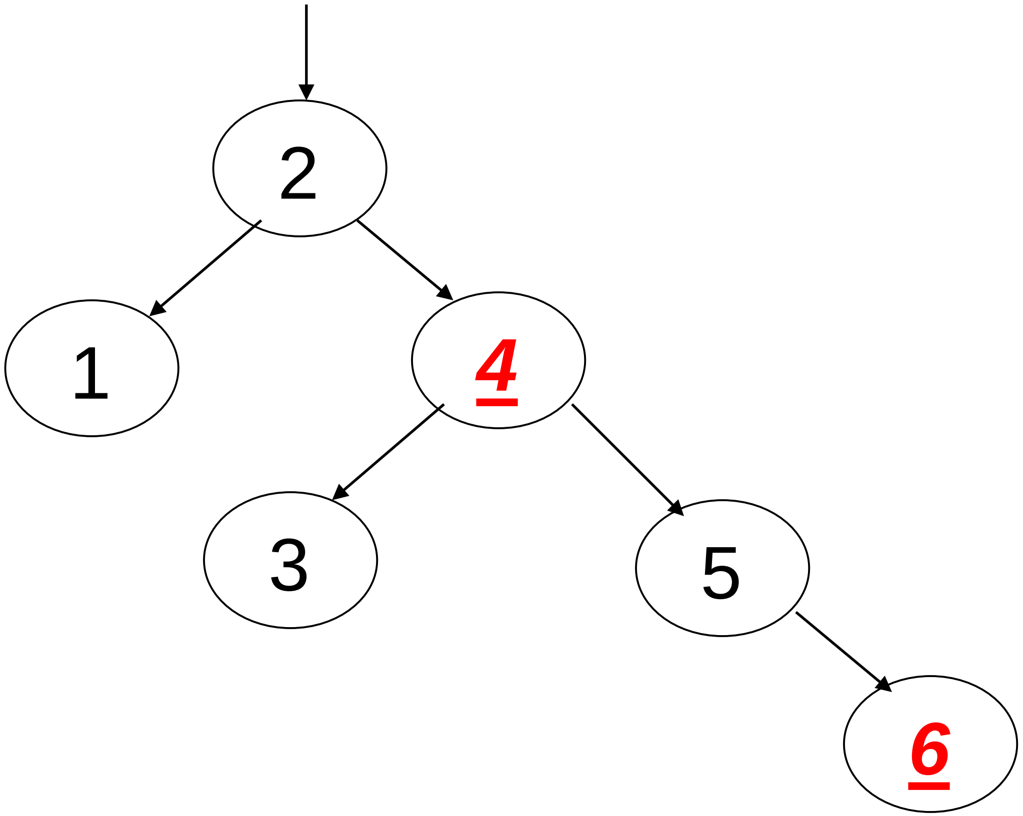
Insert 6

On way down see 4 with 2 red children. Make 4 red and children black. 4's parent is black so no problem.



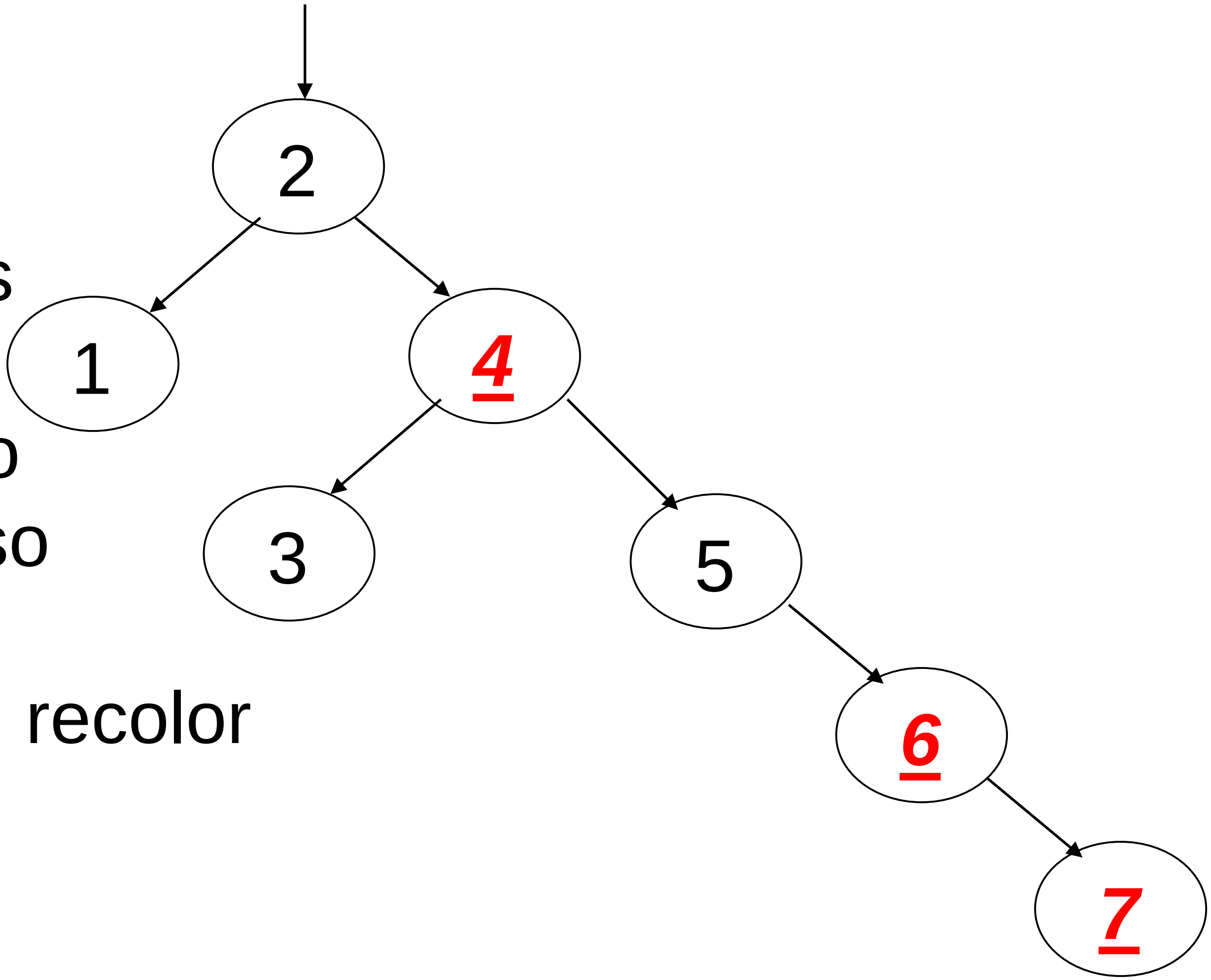
Finishing insert of 6

6's parent is black so done.

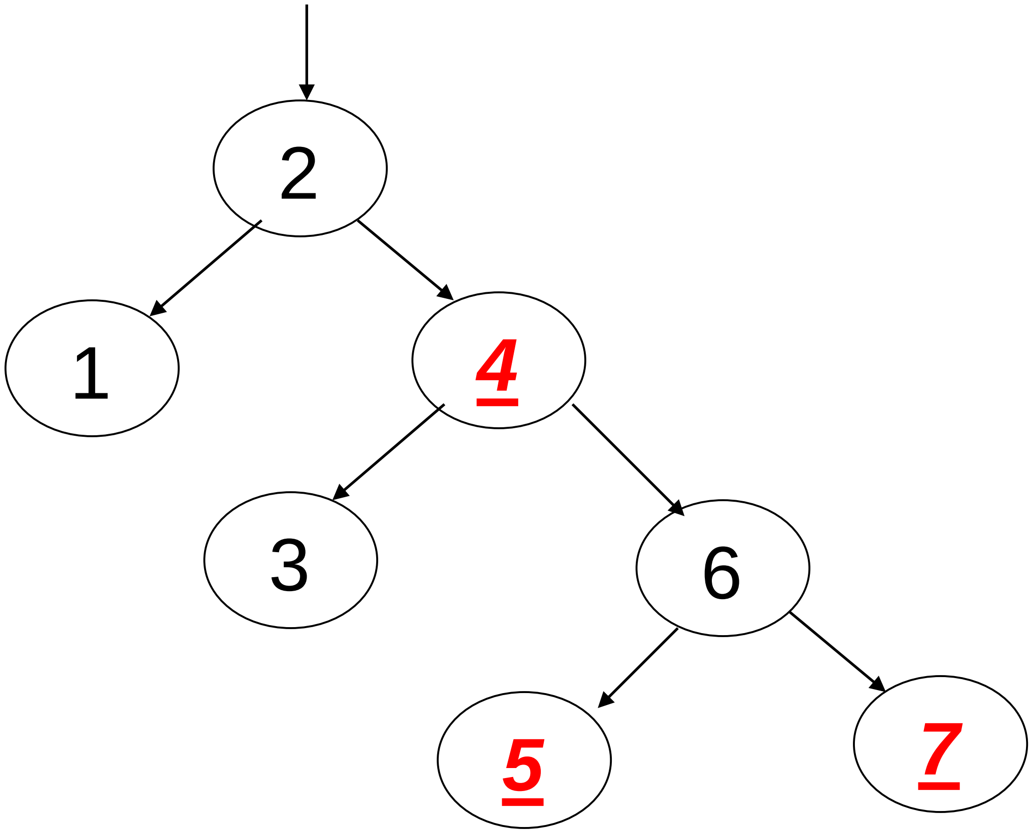


Insert 7

7's parent is red. Parent's sibling is black (null). 7 is outside relative to grandparent (5) so rotate parent and grandparent then recolor

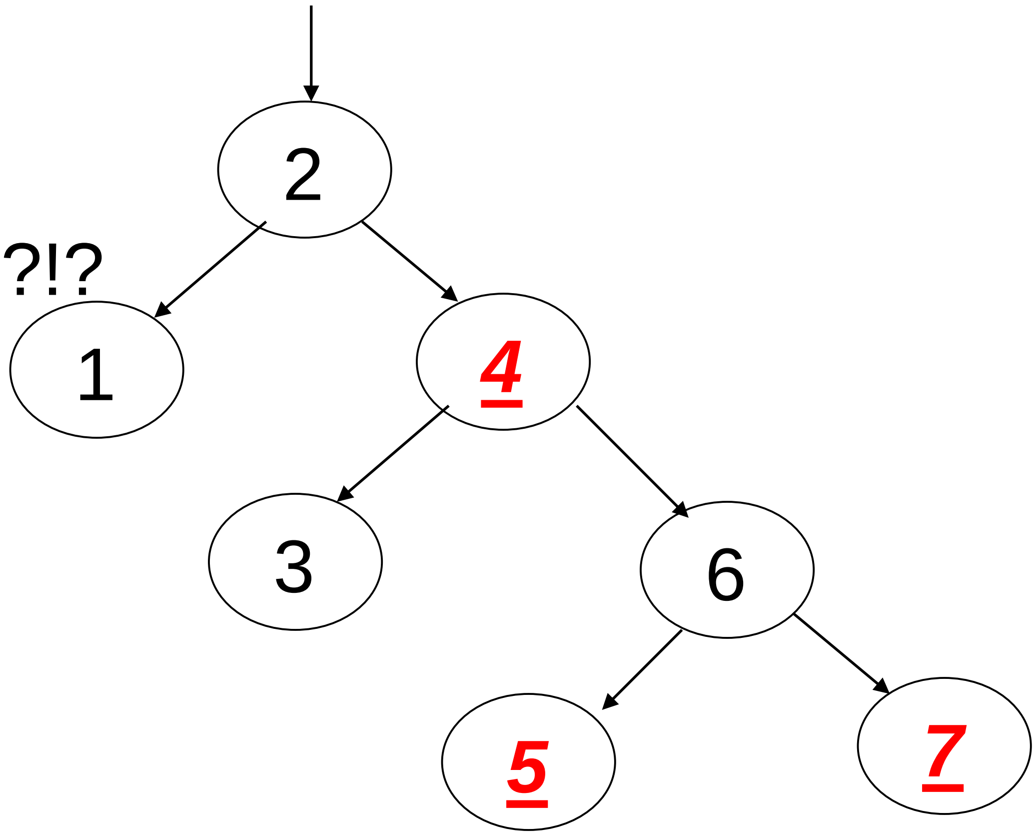


Finish insert of 7



Insert 8

The caveat!!!
Getting unbalanced on that right subtree?!?

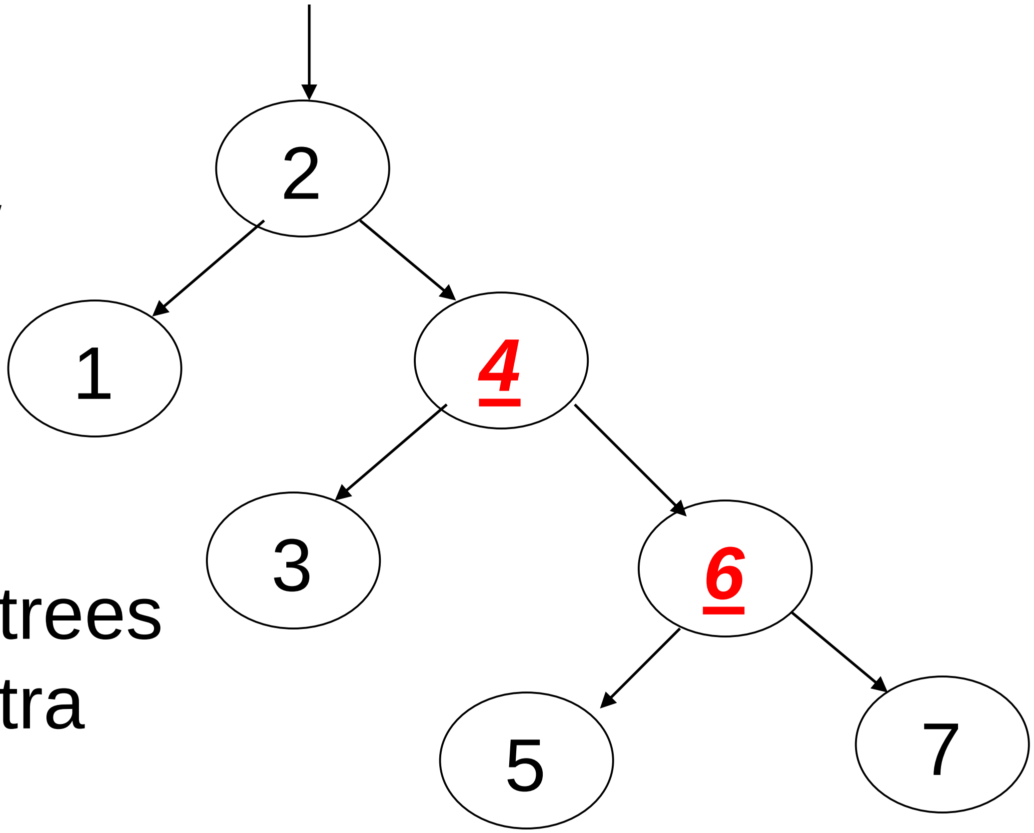


On way down see 6 with 2 red children. Make 6 red and children black. This creates a problem because 6's parent, 4, is also red. Must perform rotation.

Still Inserting 8

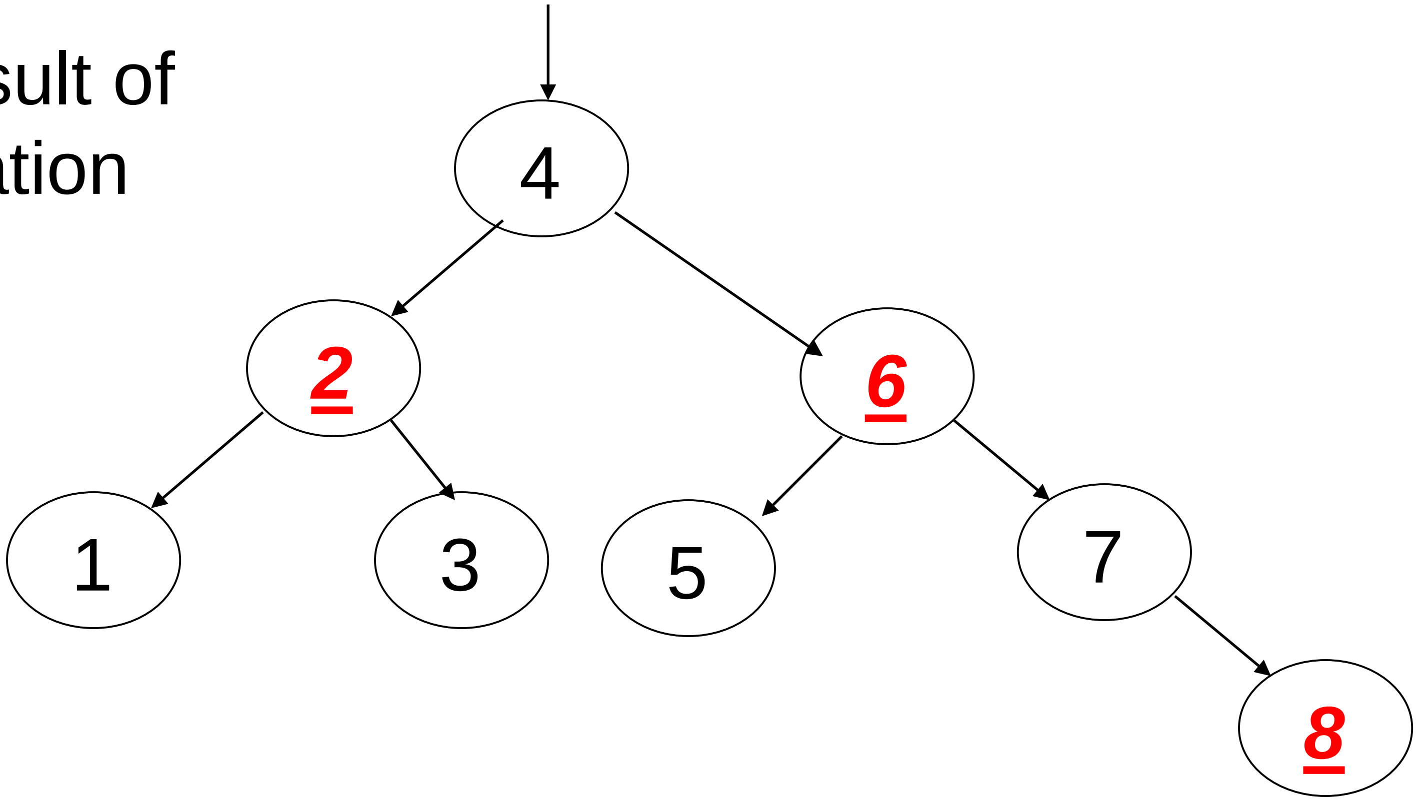
Recolored now need to rotate.

Recall, the subtrees and the one extra black node.

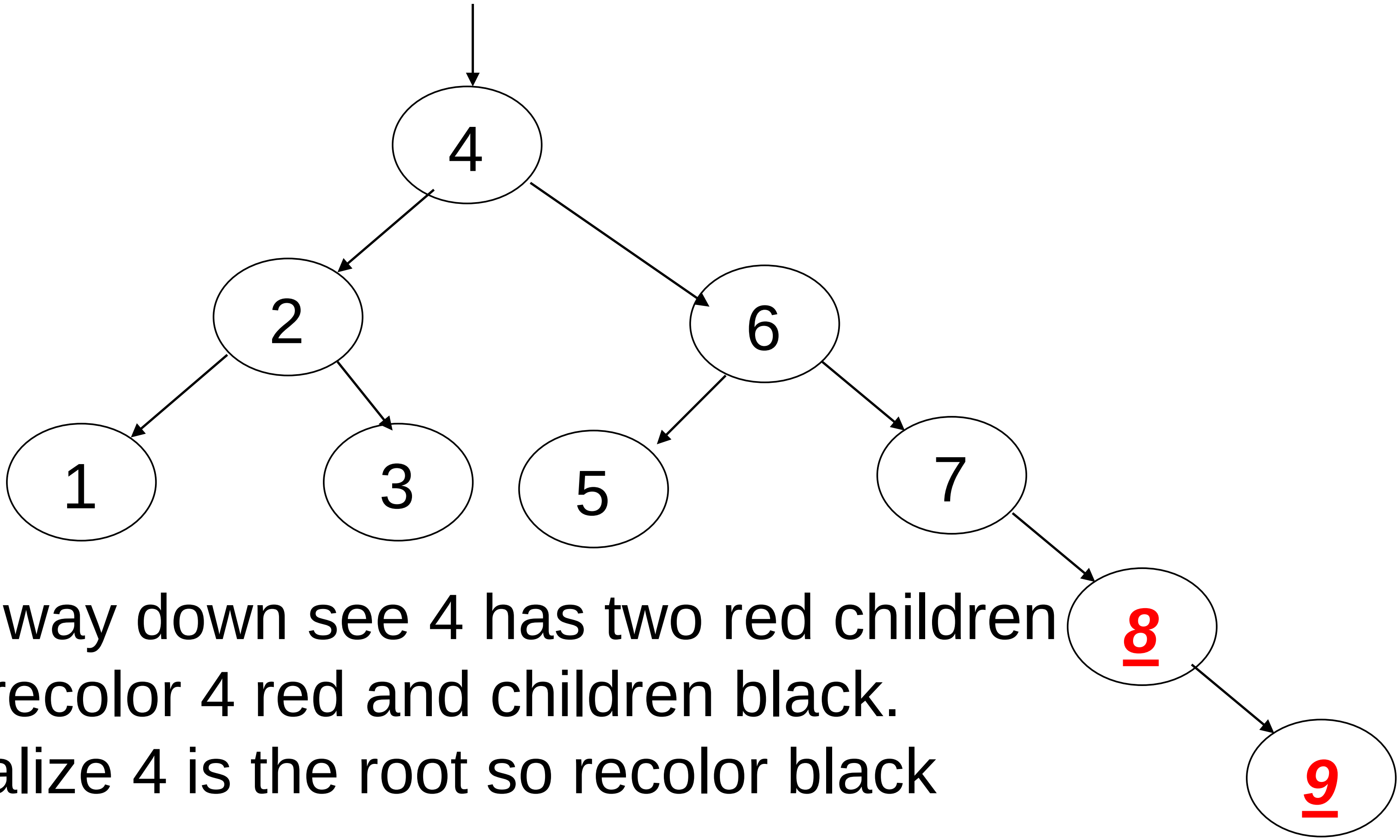


Finish inserting 8

Result of rotation

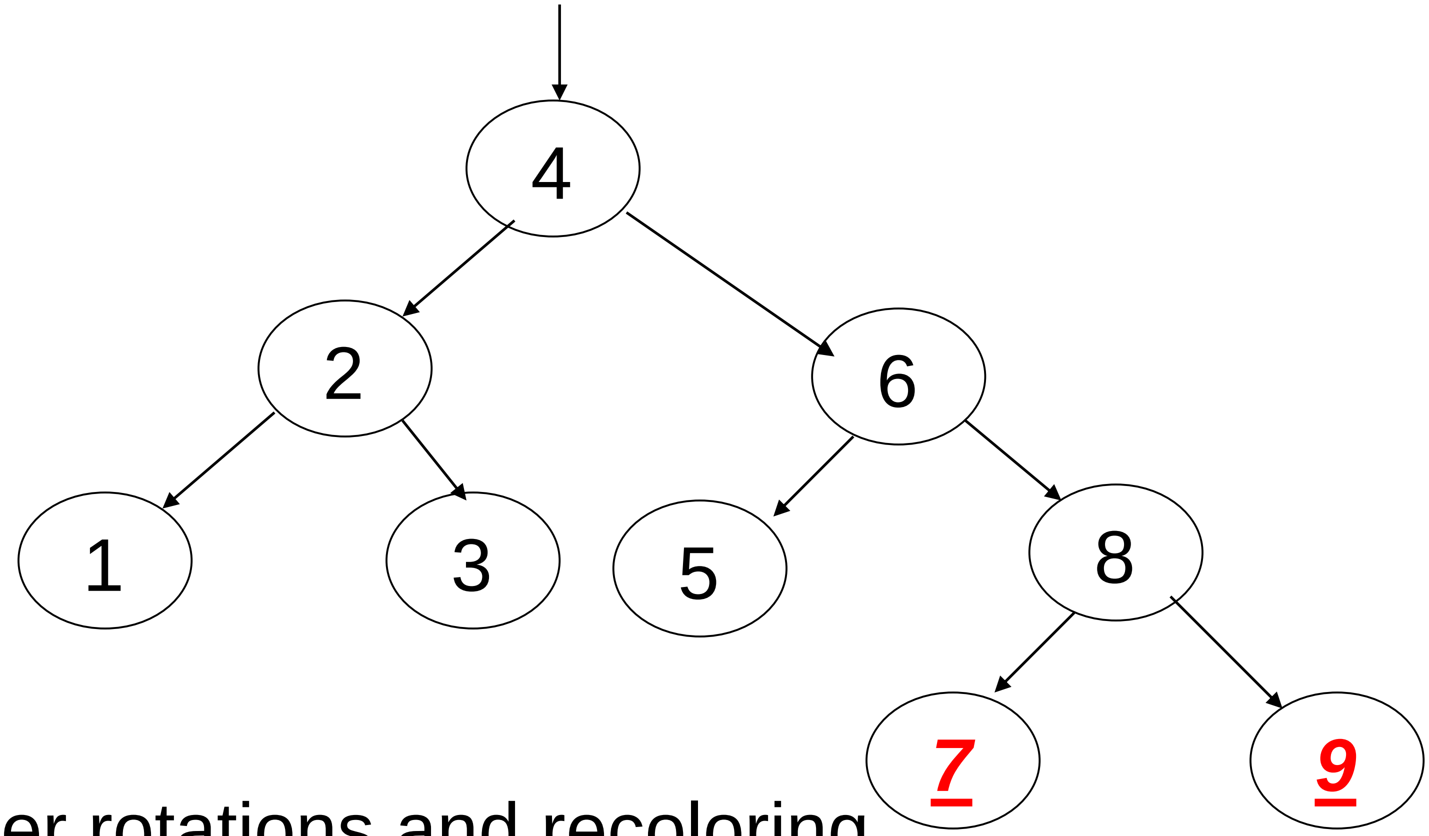


Insert 9



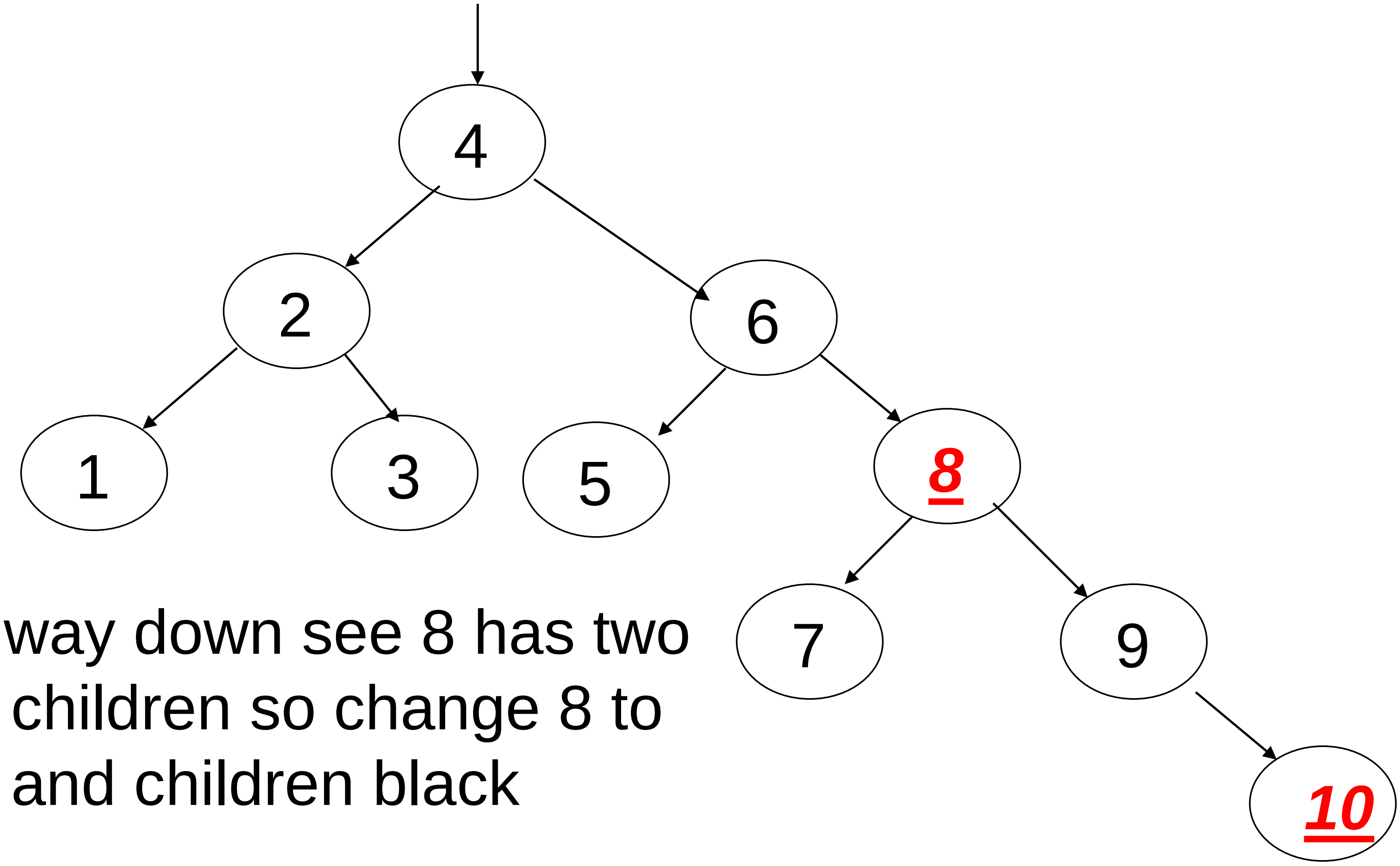
On way down see 4 has two red children so recolor 4 red and children black.
Realize 4 is the root so recolor black

Finish Inserting 9



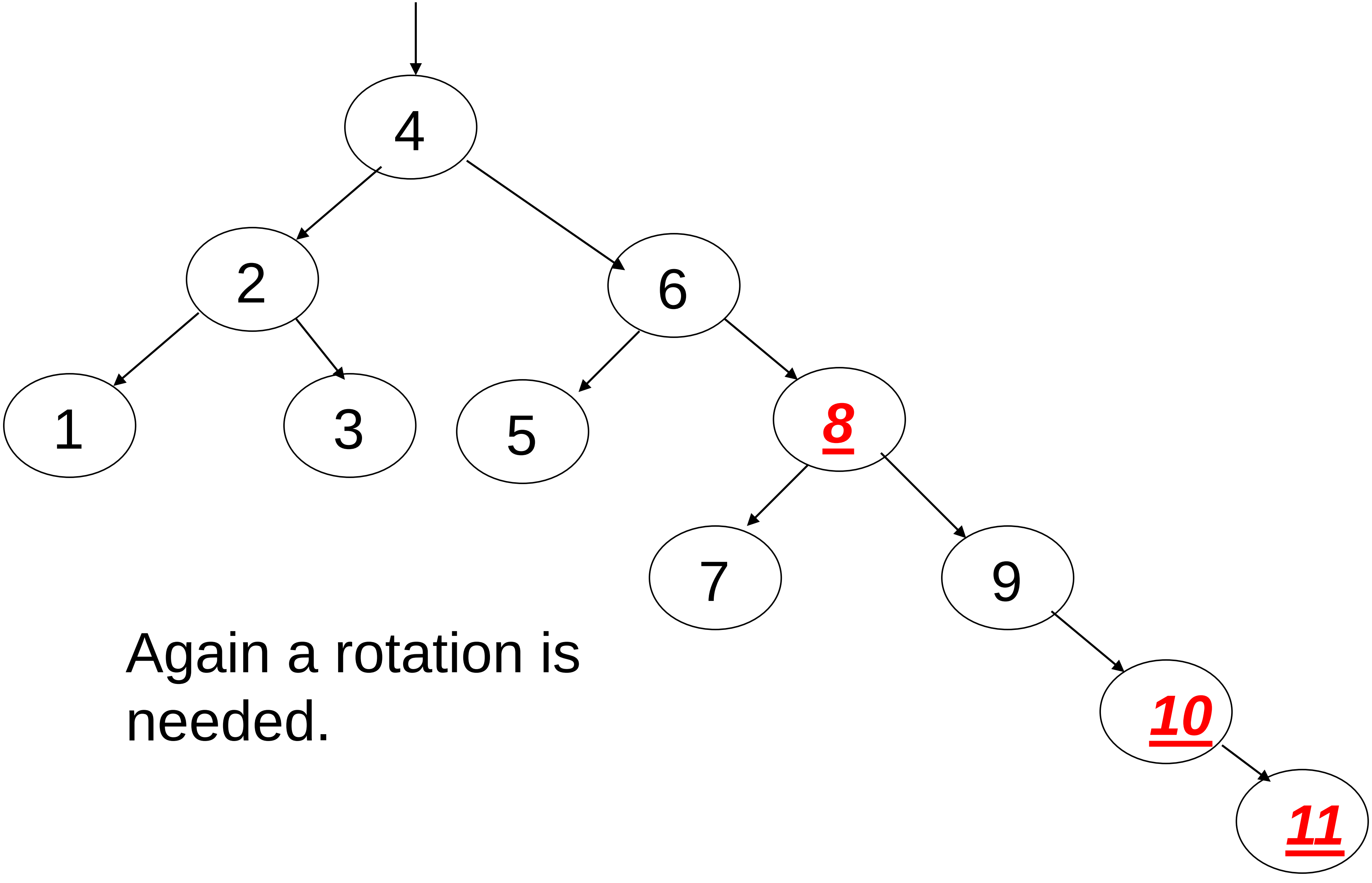
After rotations and recoloring

Insert 10



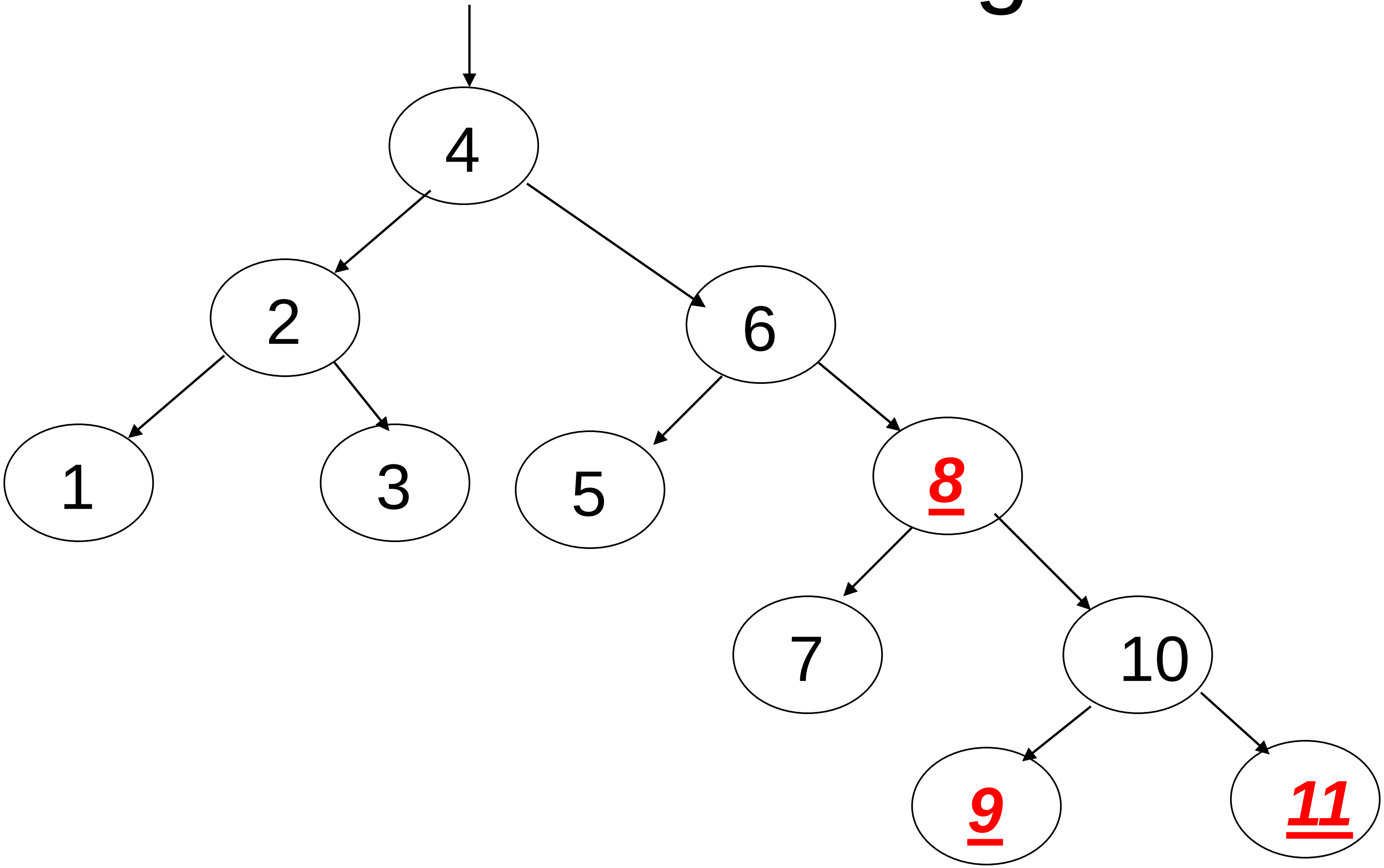
On way down see 8 has two red children so change 8 to red and children black

Insert 11



Again a rotation is needed.

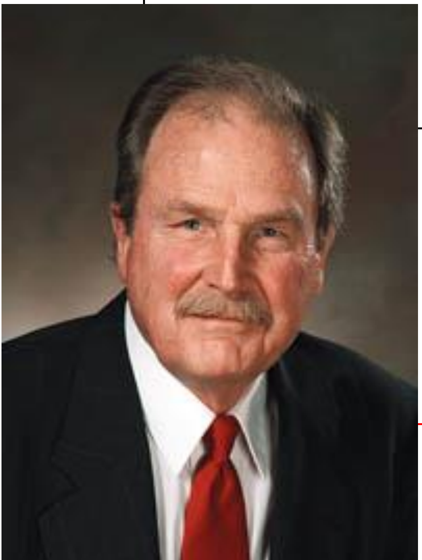
Finish inserting 11



Topic 21: Huffman Coding

The author should gaze at Noah, and ... learn, as they did in the Ark, to crowd a great deal of matter into a very small compass.

Sydney Smith, Edinburgh Review



Agenda

- **Encoding**
- Compression
- Huffman Coding

Encoding

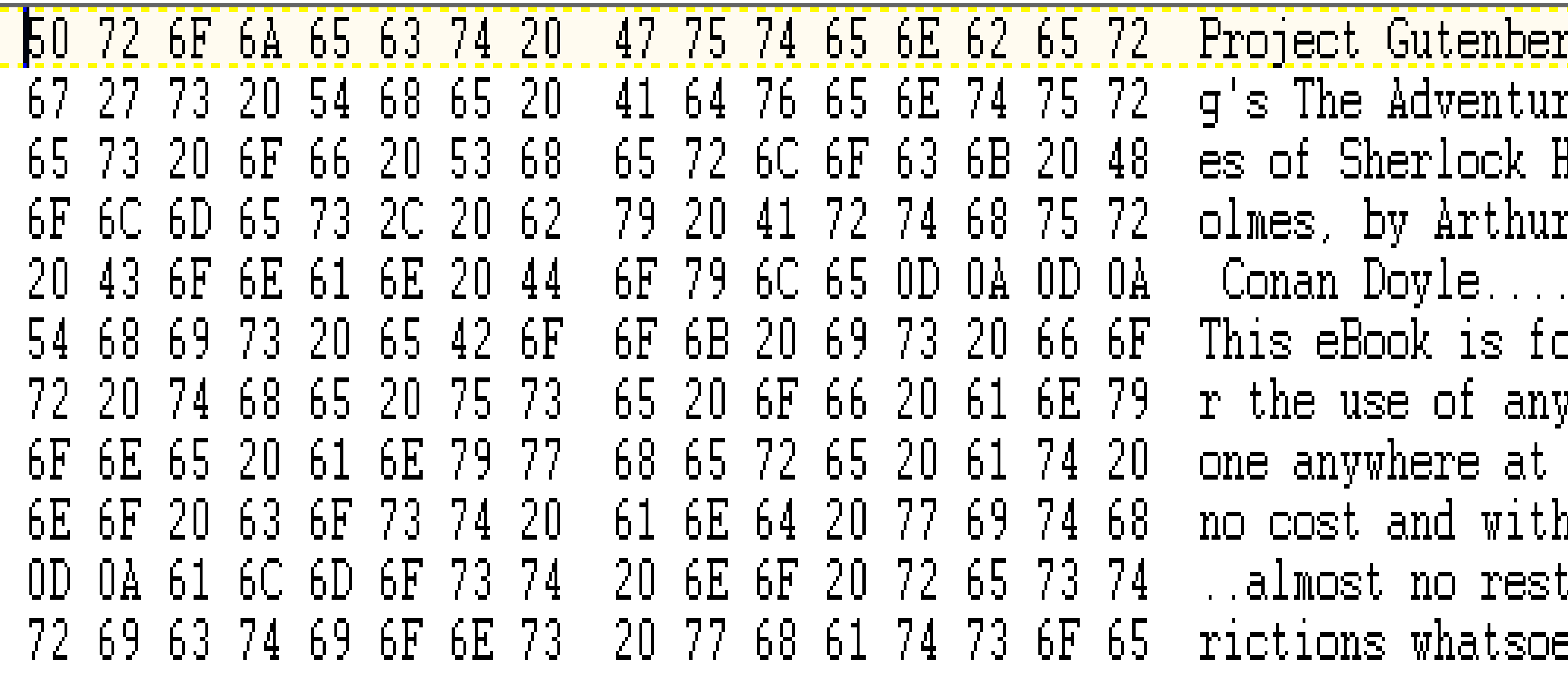
- UTCS
- 85 84 67 83
- 01010101 01010100 01000011 01010011
- What is stored in a jpg file? A text file? A Java file? A png file? A pdf file? An mp3 file? An mp4 file? An excel spreadsheet file? A zip file?
- open a bitmap in a text editor

Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	(space)	100 0000	100	64	40	@	110 0000	140	96	60	`
010 0001	041	33	21	!	100 0001	101	65	41	A	110 0001	141	97	61	a
010 0010	042	34	22	"	100 0010	102	66	42	B	110 0010	142	98	62	b
010 0011	043	35	23	#	100 0011	103	67	43	C	110 0011	143	99	63	c
010 0100	044	36	24	$	100 0100	104	68	44	D	110 0100	144	100	64	d
010 0101	045	37	25	%	100 0101	105	69	45	E	110 0101	145	101	65	e
010 0110	046	38	26	&	100 0110	106	70	46	F	110 0110	146	102	66	f
010 0111	047	39	27	'	100 0111	107	71	47	G	110 0111	147	103	67	g
010 1000	050	40	28	(	100 1000	110	72	48	H	110 1000	150	104	68	h
010 1001	051	41	29	)	100 1001	111	73	49	I	110 1001	151	105	69	i
010 1010	052	42	2A	*	100 1010	112	74	4A	J	110 1010	152	106	6A	j
010 1011	053	43	2B	+	100 1011	113	75	4B	K	110 1011	153	107	6B	k
010 1100	054	44	2C	,	100 1100	114	76	4C	L	110 1100	154	108	6C	l
010 1101	055	45	2D	-	100 1101	115	77	4D	M	110 1101	155	109	6D	m
010 1110	056	46	2E	.	100 1110	116	78	4E	N	110 1110	156	110	6E	n
010 1111	057	47	2F	/	100 1111	117	79	4F	O	110 1111	157	111	6F	o
011 0000	060	48	30	0	101 0000	120	80	50	P	111 0000	160	112	70	p
011 0001	061	49	31	1	101 0001	121	81	51	Q	111 0001	161	113	71	q
011 0010	062	50	32	2	101 0010	122	82	52	R	111 0010	162	114	72	r
011 0011	063	51	33	3	101 0011	123	83	53	S	111 0011	163	115	73	s
011 0100	064	52	34	4	101 0100	124	84	54	T	111 0100	164	116	74	t
011 0101	065	53	35	5	101 0101	125	85	55	U	111 0101	165	117	75	u
011 0110	066	54	36	6	101 0110	126	86	56	V	111 0110	166	118	76	v
011 0111	067	55	37	7	101 0111	127	87	57	W	111 0111	167	119	77	w

Text File

```
Project Gutenberg's The Adventures of Sherlock

This eBook is for the use of anyone anywhere at
almost no restrictions whatsoever.  You may cop
re-use it under the terms of the Project Gutenb
with this eBook or online at www.gutenberg.net


Title: The Adventures of Sherlock Holmes

Author: Arthur Conan Doyle

Posting Date: April 18, 2011 [EBook #1661]
First Posted: November 29, 2002
```

Text File???


50 72 6F 6A 65 63 74 20 47 75 74 65 6E 62 65 72 Project Gutenber
67 27 73 20 54 68 65 20 41 64 76 65 6E 74 75 72 g's The Adventur
65 73 20 6F 66 20 53 68 65 72 6C 6F 63 6B 20 48 es of Sherlock H
6F 6C 6D 65 73 2C 20 62 79 20 41 72 74 68 75 72 olmes, by Arthur
20 43 6F 6E 61 6E 20 44 6F 79 6C 65 0D 0A 0D 0A Conan Doyle....
54 68 69 73 20 65 42 6F 6F 6B 20 69 73 20 66 6F This eBook is fo
72 20 74 68 65 20 75 73 65 20 6F 66 20 61 6E 79 r the use of any
6F 6E 65 20 61 6E 79 77 68 65 72 65 20 61 74 20 one anywhere at
6E 6F 20 63 6F 73 74 20 61 6E 64 20 77 69 74 68 no cost and with
0D 0A 61 6C 6D 6F 73 74 20 6E 6F 20 72 65 73 74 ..almost no rest
72 69 63 74 69 6F 6E 73 20 77 68 61 74 73 6F 65 rictions whatsoe

Bitmap and JPEG File

Bitmap File????

```
07 01 06 05 00 04 03 01  04 02 02 05 03 03 07 02   ................
05 06 02 04 05 01 00 01  00 00 02 00 00 02 00 00   ................
03 01 04 07 05 09 0C 0A  07 0A 08 02 05 03 00 04   ................
05 00 04 03 00 04 03 01  03 03 02 05 03 04 05 03   ................
05 06 02 05 06 02 08 09  07 05 06 04 01 02 00 04   ................
04 04 0E 0E 0E 0E 0E 0E  0C 0B 0D 11 10 12 06 00   ................
0D 00 15 00 8F AD 3C FF  F8 A1 FB F9 BF F1 F8 B5   ....—<ÿø¡ûù¿ñøµ
FF FF C7 F3 FD A0 D2 EB  6B EE F9 7F FB FF 8F F7   ÿÿÇóý Òëkîù▮ûÿ÷
FF 94 FD FF A5 AA 9F 6B  02 03 00 00 00 16 42 81   ÿ▮ýÿ¥ª▮k......B
25 55 8D 3A 3D 68 25 2F  4C 1B 31 41 22 38 3F 30   %U:=h%/L.1A"8?0
0F 11 0B 0D 0B 0A 09 0E  11 00 04 07 00 01 02 06   ................
0C 0B 0C 13 10 06 0D 08  00 07 00 00 04 00 00 01   ................
00 02 07 06 04 09 08 00  04 03 00 03 02 05 0A 09   ................
06 0B 0A 01 06 05 12 06  12 03 05 0D 00 0E 00 28   ...............(
6C 17 E5 FA 7B EE FF AA  FB FB DD F5 F8 D8 FF FD   l.åú{îÿªûûÝõøØÿý
C5 FA F4 B3 F7 FA BC F9  FD D9 FF FF FA FC F5 F8   Åúô³÷ú¼ùýÙÿÿúüõø
FF FB E9 FF F8 CC C9 FC  82 F1 FA C8 FF F4 F5 FB   ÿûéÿøÌÉü▮ñúÈÿôõû
F5 E8 F9 FD DA FF FF E8  FF FF FB BC D7 CE 31 50   õèùýÚÿÿèÿÿû¼×Î1P
2B 01 16 00 00 02 00 01  03 04 09 08 04 0E 0D 03   +...............
03 04 00 09 08 04 0B 0C  0A 0A 0B 09 06 07 05 00   ................
01 00 00 01 00 06 07 05  0B 0C 0A 0D 0E 0C 19 11   ................
```

JPEG File

JPEG VS BITMAP

Tower-number1-1024x768.bmp	Bitmap image	2,305 KB
Tower-number1-1024x768.jpg	JPEG Image	283 KB

- JPEG File

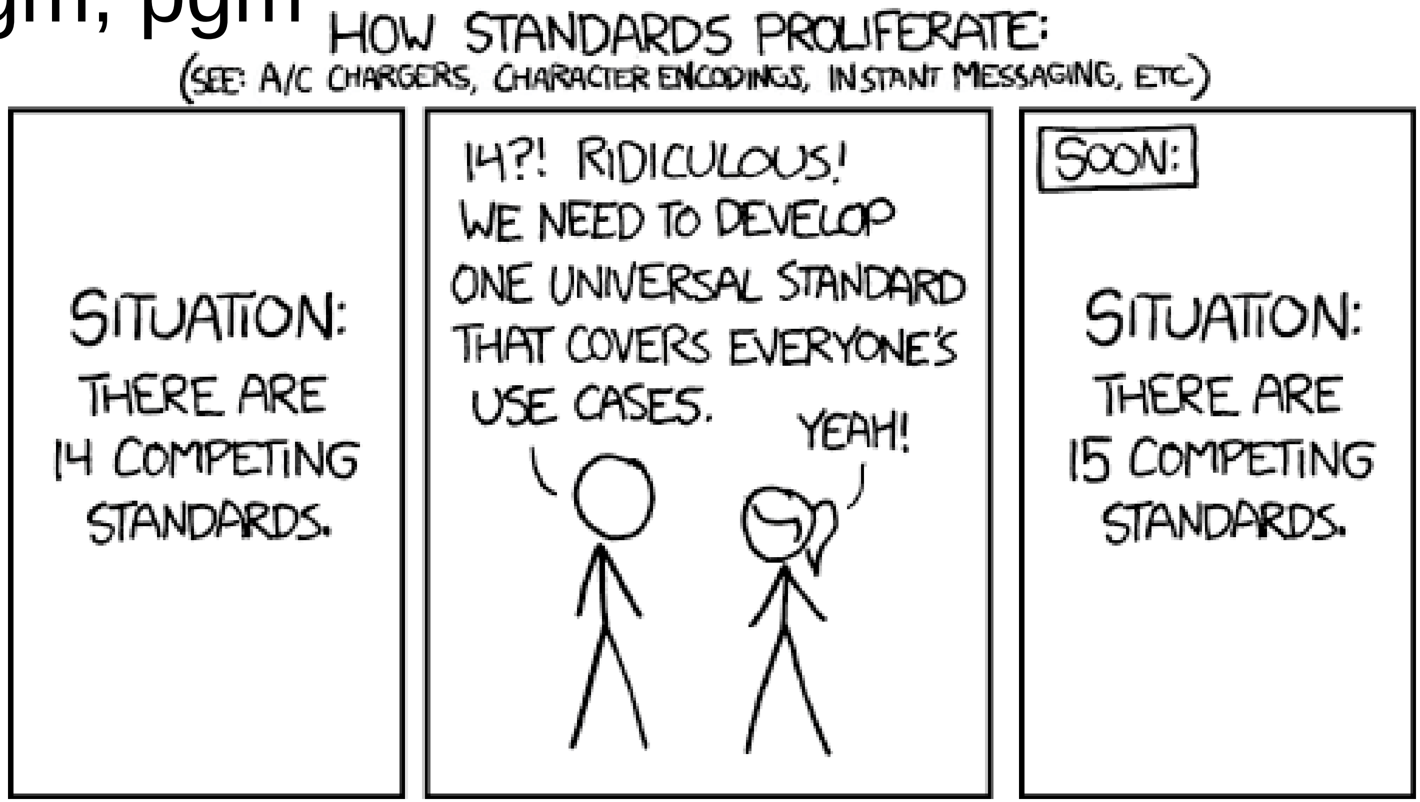
```
F D8 FF E0 00 10 4A 46  49 46 00 01 01 01 00 48  ÿØÿà..JFIF.....H
0 48 00 00 FF DB 00 43  00 02 01 01 02 01 01 02  .H..ÿÛ.C........
2 02 02 02 02 02 02 03  05 03 03 03 03 03 06 04  ................
4 03 05 07 06 07 07 07  06 07 07 08 09 0B 09 08  ................
8 0A 08 07 07 0A 0D 0A  0A 0B 0C 0C 0C 0C 07 09  ................
E 0F 0D 0C 0E 0B 0C 0C  0C FF DB 00 43 01 02 02  ..........ÿÛ.C...
2 03 03 03 06 03 03 06  0C 08 07 08 0C 0C 0C 0C  ................
C 0C 0C 0C 0C 0C 0C 0C  0C 0C 0C 0C 0C 0C 0C 0C  ................
C 0C 0C 0C 0C 0C 0C 0C  0C 0C 0C 0C 0C 0C 0C 0C  ................
C 0C 0C 0C 0C 0C 0C 0C  0C 0C 0C 0C 0C 0C FF C0  ..............ÿÀ
0 11 08 03 00 04 00 03  01 22 00 02 11 01 03 11  ..........".....
1 FF C4 00 1F 00 00 01  05 01 01 01 01 01 01 00  .ÿÄ.............
0 00 00 00 00 00 00 01  02 03 04 05 06 07 08 09  ................
A 0B FF C4 00 B5 10 00  02 01 03 03 02 04 03 05  ..ÿÄ.µ..........
5 04 04 00 00 01 7D 01  02 03 00 04 11 05 12 21  ......}........!
1 41 06 13 51 61 07 22  71 14 32 81 91 A1 08 23  1A..Qa."q.2´i.#
2 B1 C1 15 52 D1 F0 24  33 62 72 82 09 0A 16 17  B±Á.RÑð$3br▮....
8 19 1A 25 26 27 28 29  2A 34 35 36 37 38 39 3A  ...%&'()*456789:
```

Encoding Schemes

- "It's all 1s and 0s"
- What do the 1s and 0s mean?
- 50 121 109
- ASCII -> 2ym
- Red Green Blue-> dark teal?

Why So Many Encoding / Decoding Schemes?

- Image file formats: bmp, png, jpg, gif, tiff, svg, cgm, pgm



- XKCD, Standards: https://xkcd.com/927/

Agenda

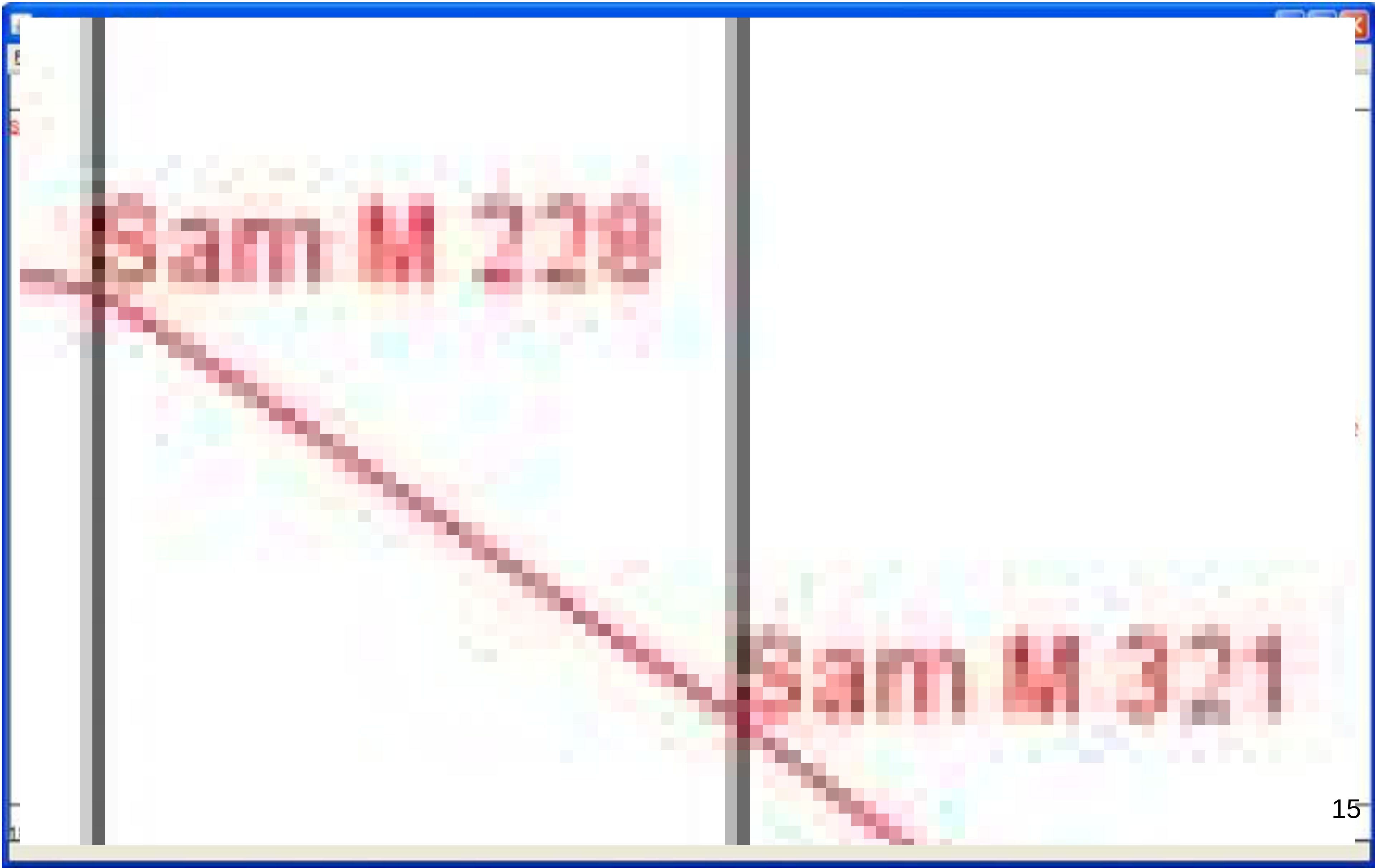
- Encoding
- **Compression**
- Huffman Coding

Compression

- Compression: Storing the same information but in a form that takes less memory
- lossless and lossy compression
- Recall:

Tower-number1-1024x768.bmp	Bitmap image	2,305 KB
Tower-number1-1024x768.jpg	JPEG Image	283 KB

Lossy Artifacts



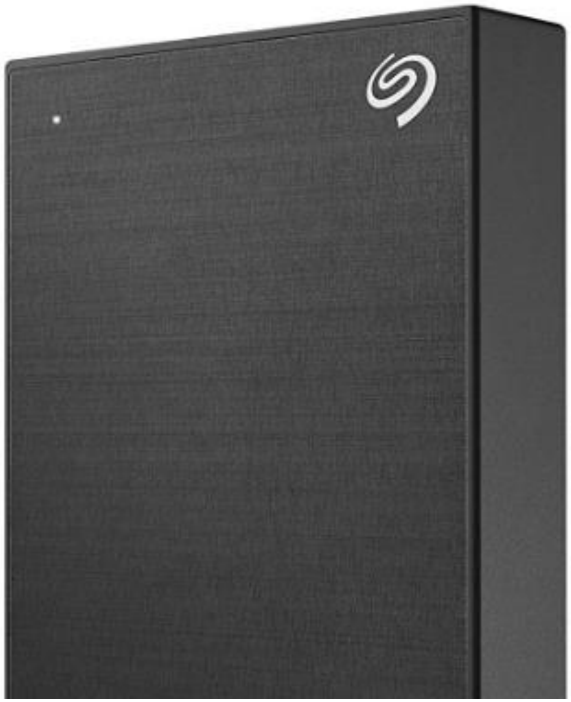
Compression

- 00000000000000000000000000000000
  111111111111111111111111111111

- 0 00100000 1 00011110

Why Bother?

- Is compression really necessary?



5 Terabytes.

~5,000,000,000,0000 bytes

Clicker 1

- With computer storage so cheap, is compression really necessary?

A. No

B. Yes

C. It Depends

Little Pipes and Big Pumps

Home Internet Access

- 400 Mbps roughly $65 per month
- 12 months * 3 years * $115 =
- 400,000,000 *bits* /second = 5 * 10^7 bytes / sec

CPU Capability

- $2,000 for a good laptop or desktop
- **Intel® Core™ i9-7900X**
- Assume it lasts 3 years.
- Memory bandwidth 40 GB / sec = 4.0 * 10^{10} bytes / sec
- on the order of 6.4 * 10^{11} instructions / second

Mobile Devices?

Cellular Network

- Your mileage may vary …
- Mega bits per second
- AT&T

  17 mbps download, 7 mbps upload
- T-Mobile & Verizon

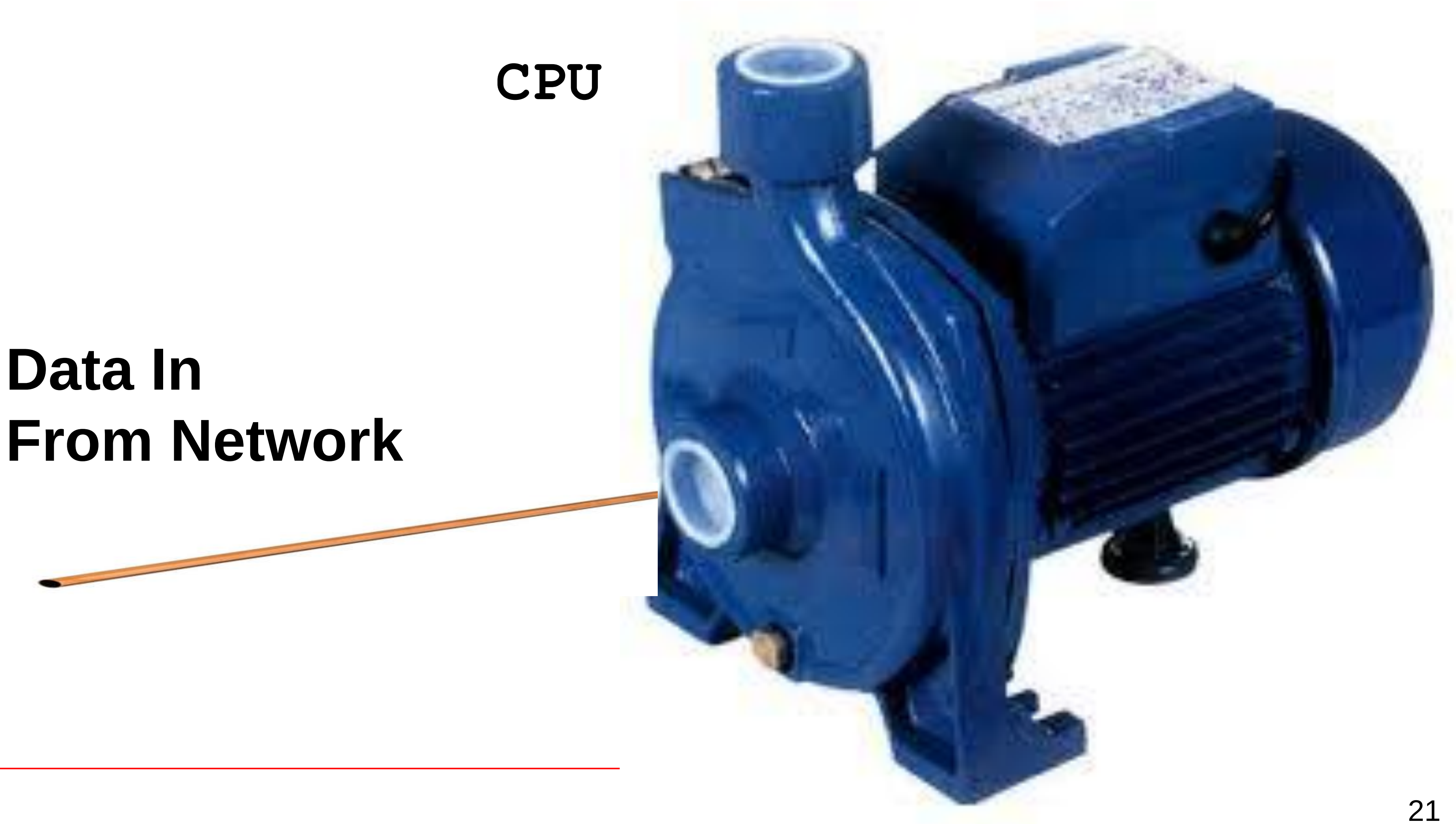
  12 mbps download, 7 mbps upload
- 17,000,000 bits per second = 2.125 x 10^6 bytes per second

http://tinyurl.com/q6o7wan

iPhone CPU

- Apple A6 System on a Chip
- Coy about IPS
- 2 cores
- Rough estimates: 1 x 10^{10} instructions per second

Little Pipes and Big Pumps



Agenda

- Encoding
- Compression
- **Huffman Coding**

Huffman Coding

- Proposed by Dr. David A. Huffman
  - Graduate class in 1951 at MIT with Robert Fano
  - term paper or final
  - term paper: prove min bits needed for binary coding of data
  - *A Method for the Construction of Minimum Redundancy Codes*
- Applicable to many forms of data transmission
  - Our example: text files
  - still used in fax machines, mp3 encoding, others

The Basic Algorithm

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency, in typical text files. (can be true when reading raw bytes as well)
- Yet in ASCII all characters are allocated the same amount of space
  – 1 char = 1 byte, be it e or x
  – fixed width encoding

The Basic Algorithm

- Any savings in tailoring codes to frequency of character?
- Code word lengths are no longer fixed like ASCII or Unicode
- Code word lengths vary and will be shorter for the more frequently used characters
- Examples use characters for clarity, but in reality just read raw bytes from file.

The Basic Algorithm

1. Scan file to be compressed and determine frequency of all values.
2. Sort or prioritize values based on frequency in file.
3. Build Huffman code tree based on prioritized values.
4. Perform a traversal of tree to determine new codes for values.
5. Scan file again to create new file using the new Huffman codes

Building a Tree
Scan the original text

- Consider the following short text

  Eerie eyes seen near lake.

- Determine frequency of all numbers (values or in this case characters) in the text

Building a Tree
Scan the original text

Eerie eyes seen near lake.

- What characters are present?

E e r i space

y s n a r l k .

Building a Tree
Scan the original text

Eerie eyes seen near lake.

- What is the frequency of each character in the text?

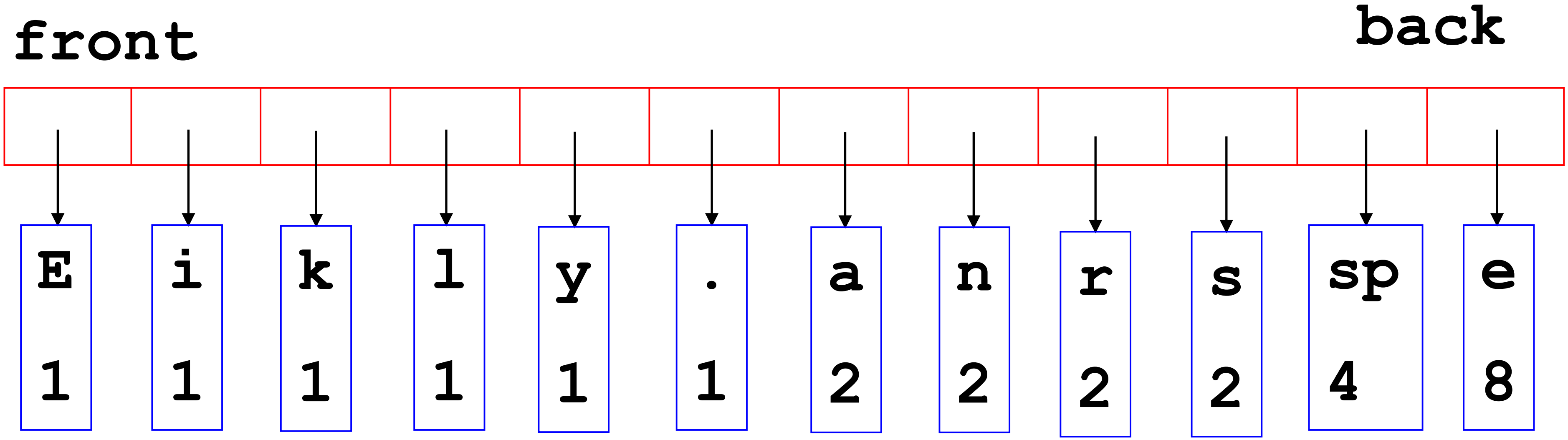
Char	Freq.	Char	Freq.	Char	Freq.
E	1	y	1	k	1
e	8	s	2	.	1
r	2	n	2		
i	1	a	2		
space	4	l	1		

Building a Tree
Prioritize values from file

- Create binary tree nodes with a value and the frequency for each value
- Place nodes in a priority queue
  - The ***lower*** the frequency, the ***higher*** the priority in the queue

Building a Tree

- The queue after enqueueing all nodes

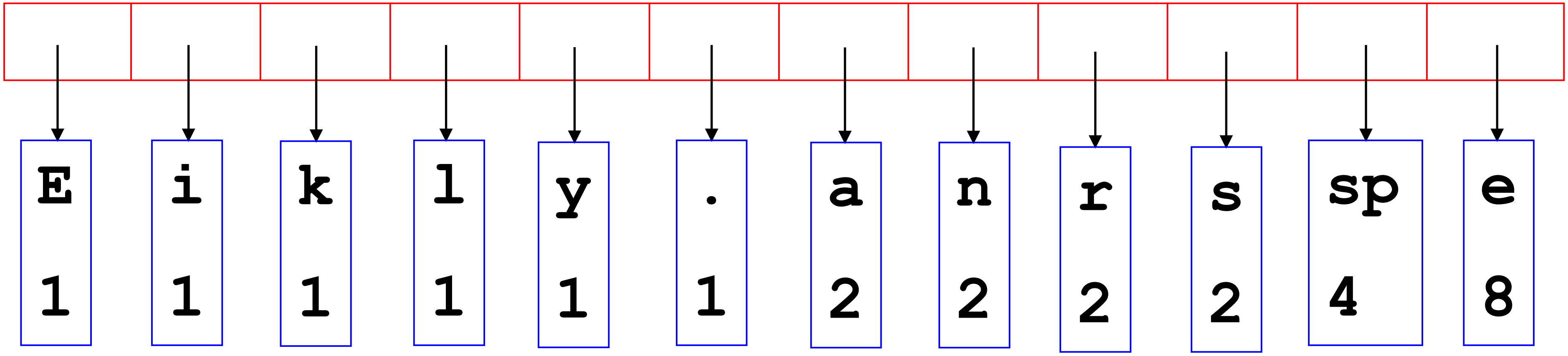


- Null Pointers are not shown
- sp = space
- See slide 67 for actual PQ and tree formed by following assignment specification

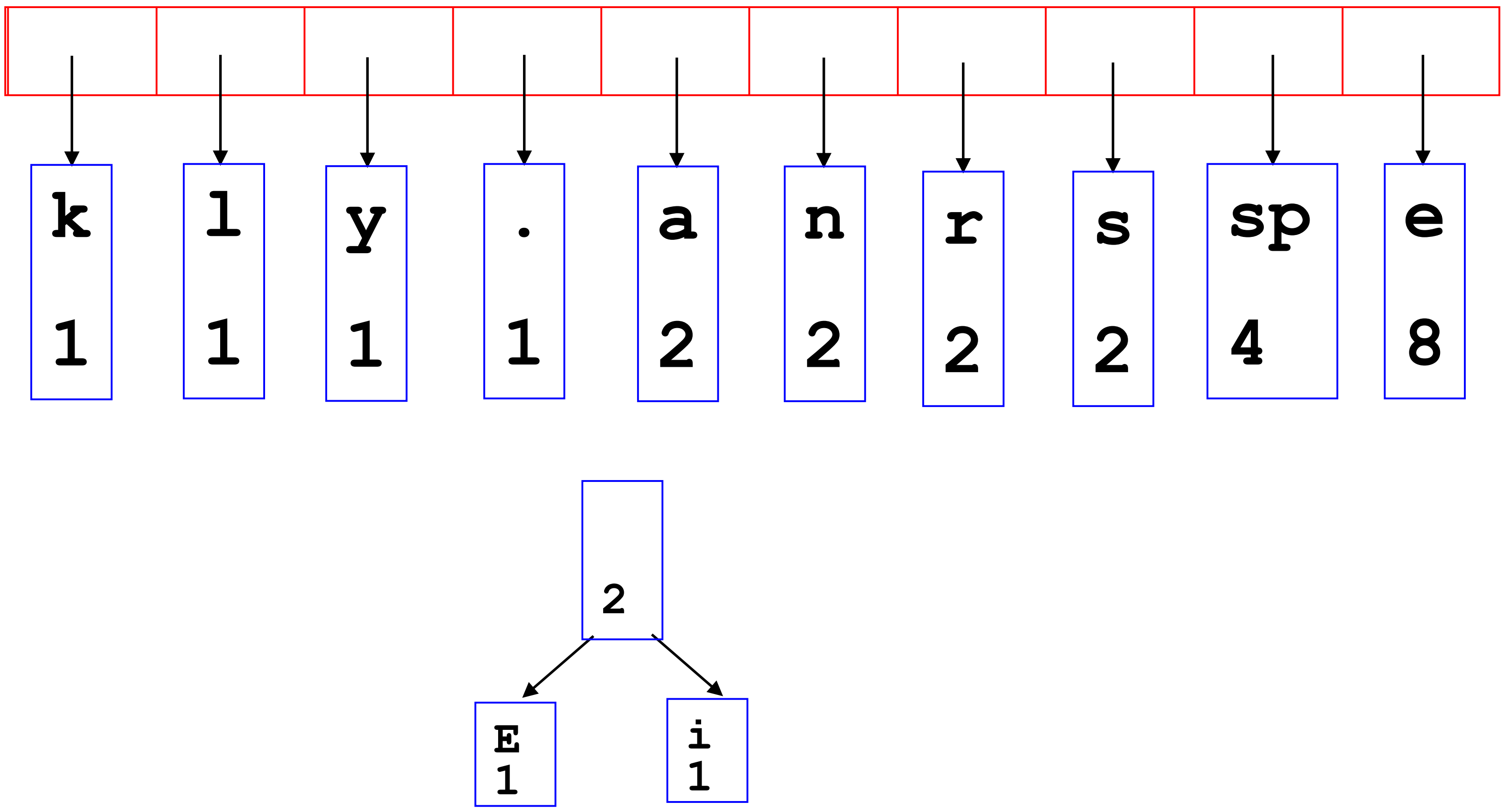
Building a Tree

- While priority queue contains two or more nodes
  - Create new node
  - Dequeue node and make it left child
  - Dequeue next node and make it right child
  - Frequency of new node equals sum of frequency of left and right children
    - New node does not contain value
  - Enqueue new node back into the priority queue

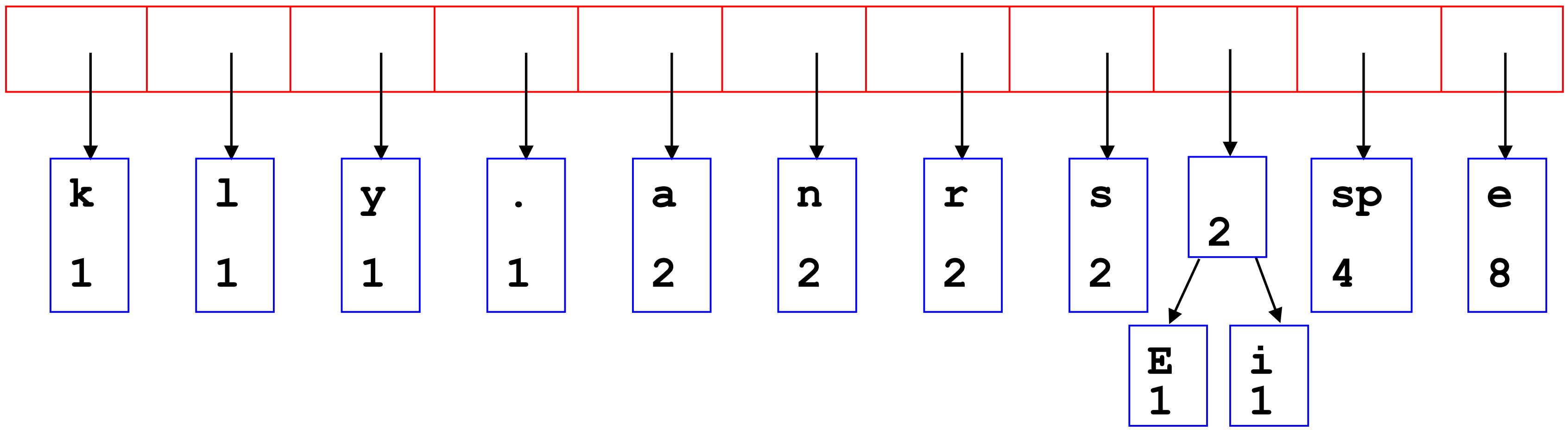
Building a Tree



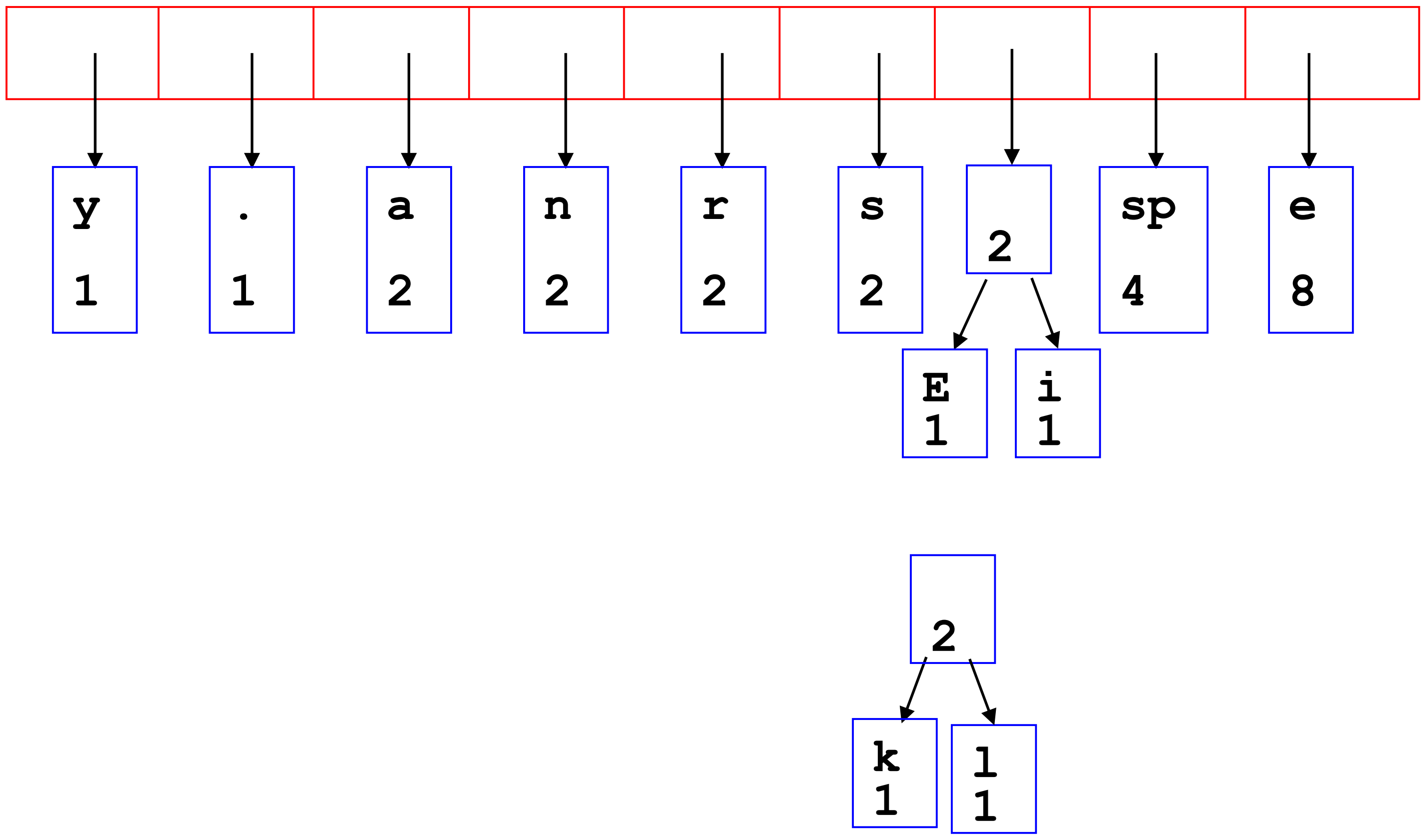
Building a Tree



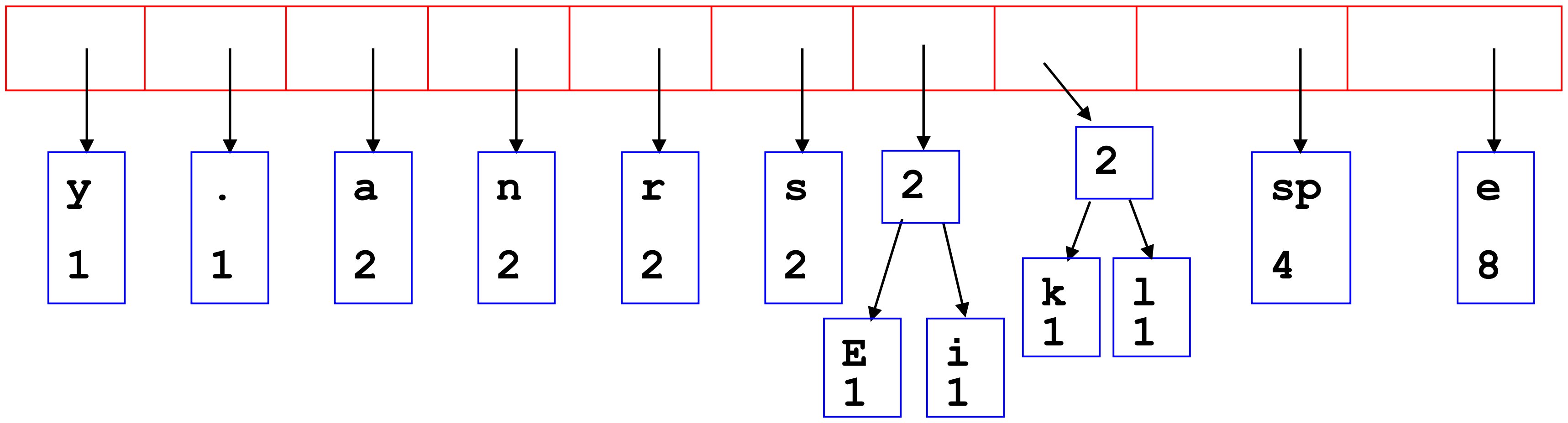
Building a Tree



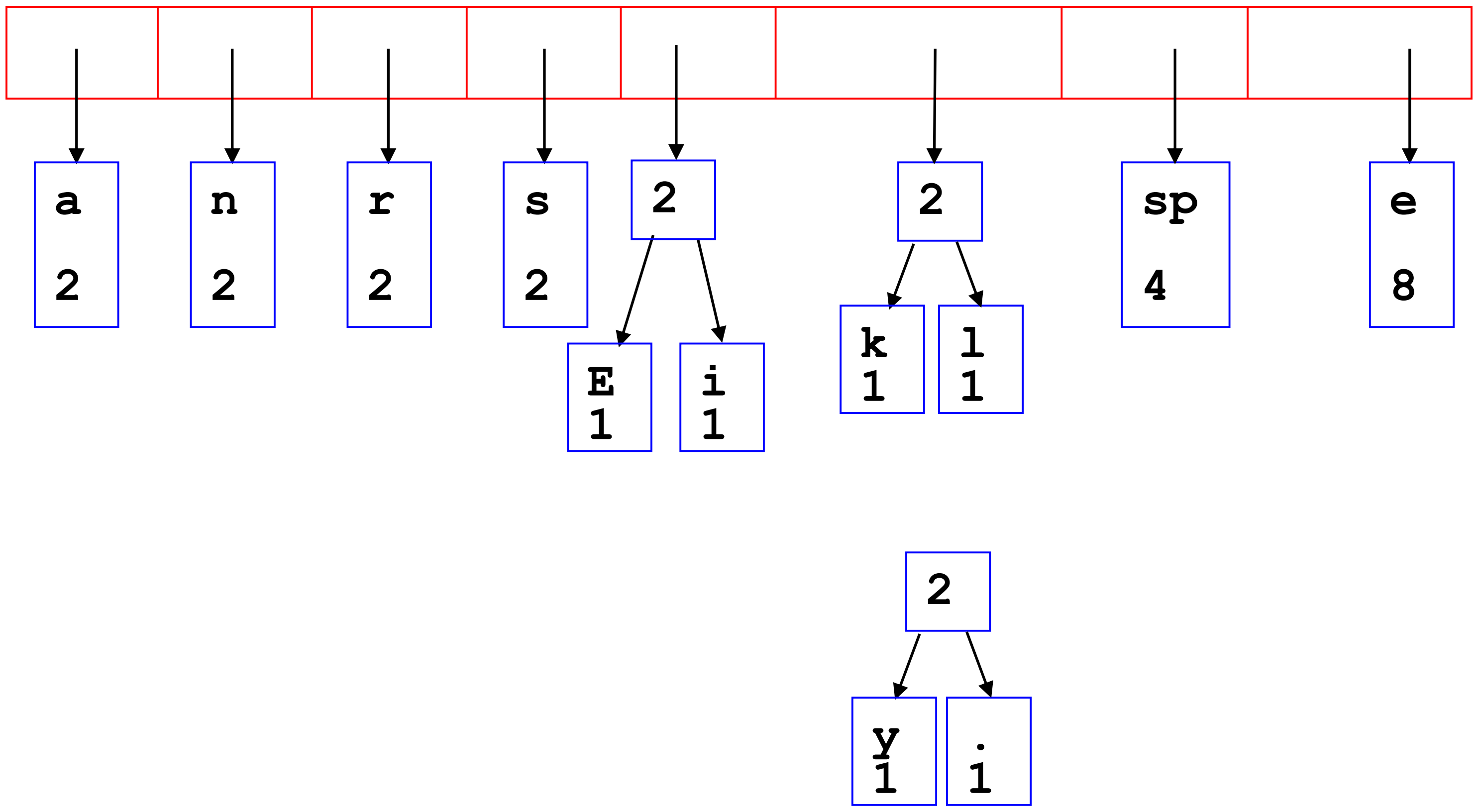
Building a Tree



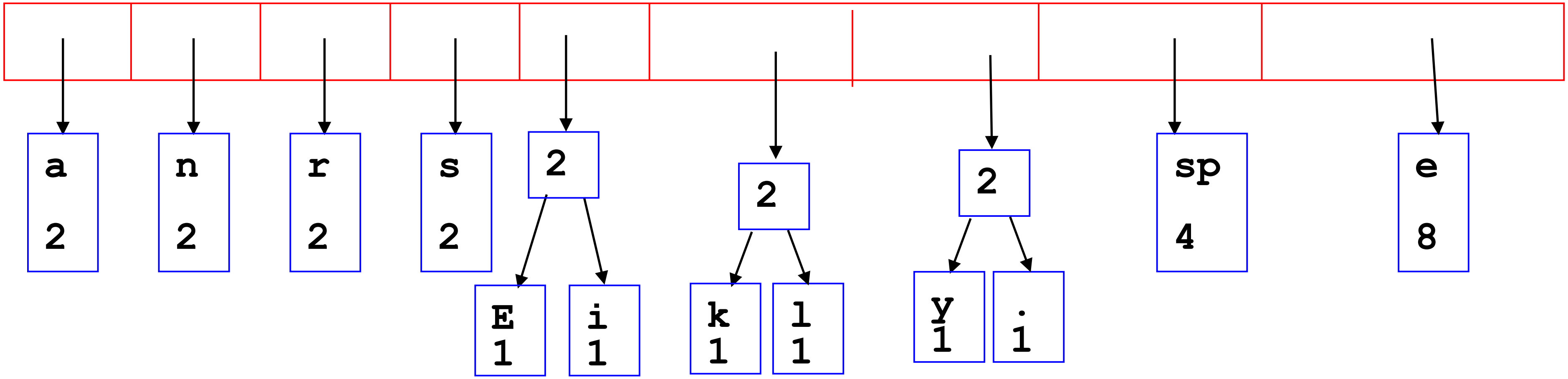
Building a Tree



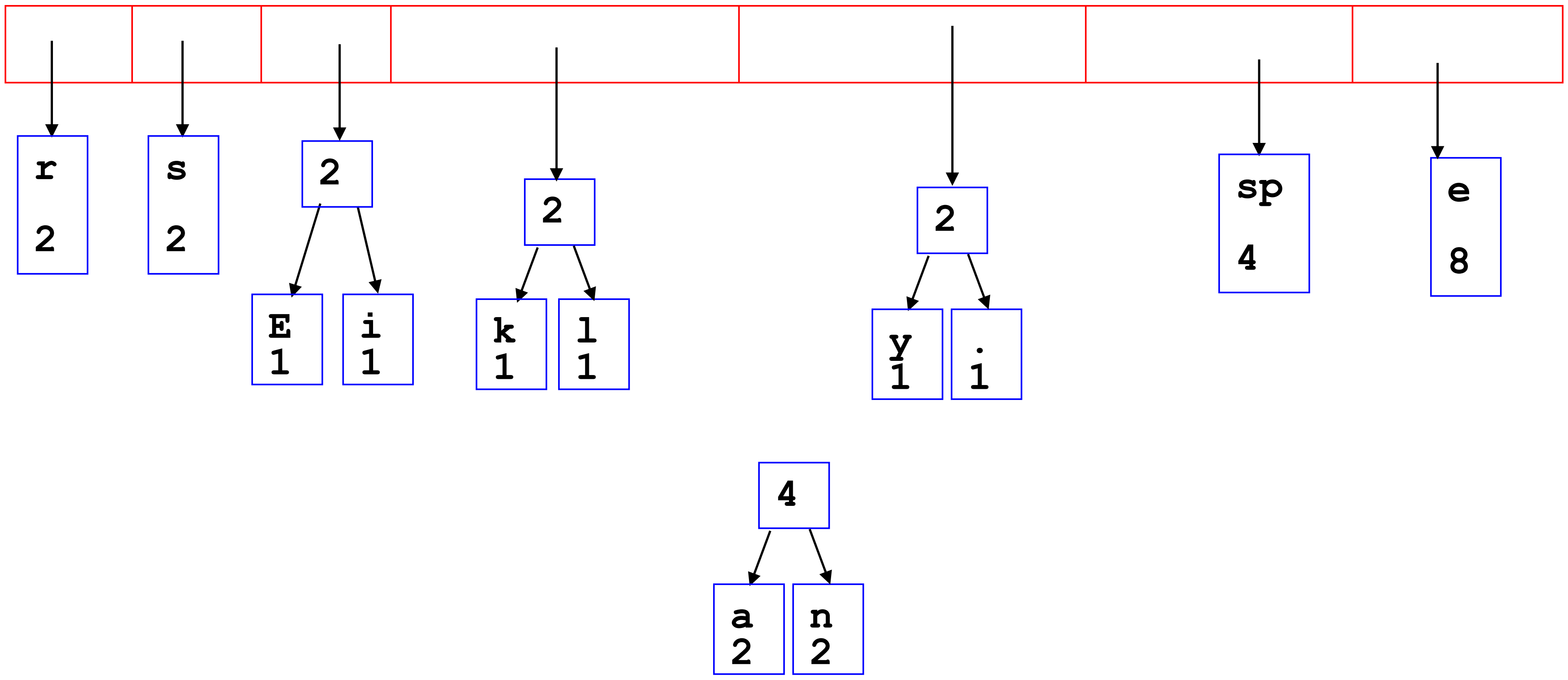
Building a Tree



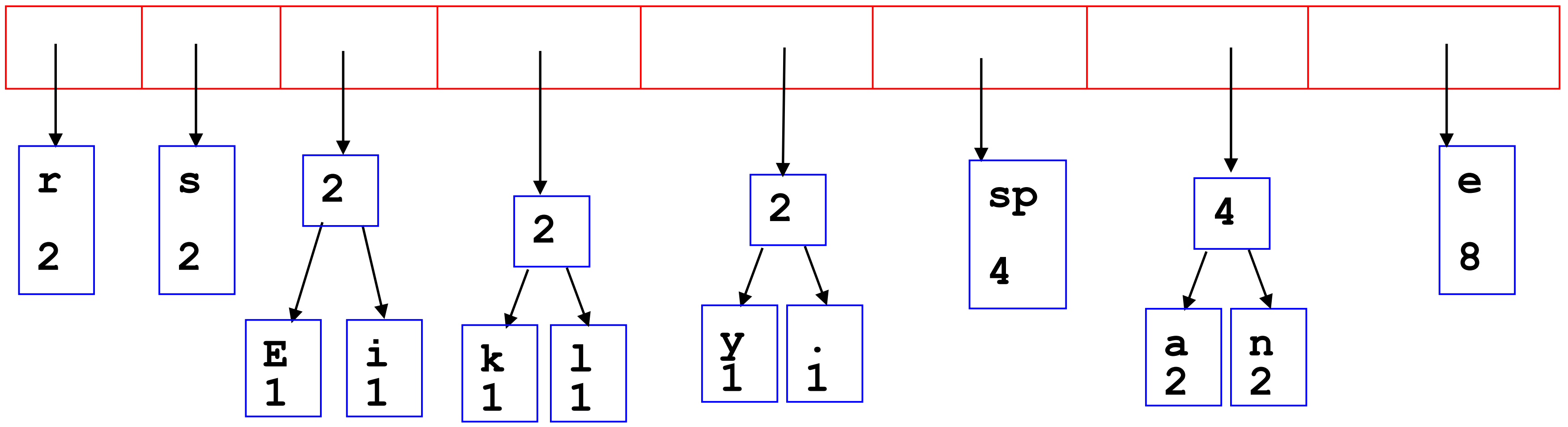
Building a Tree



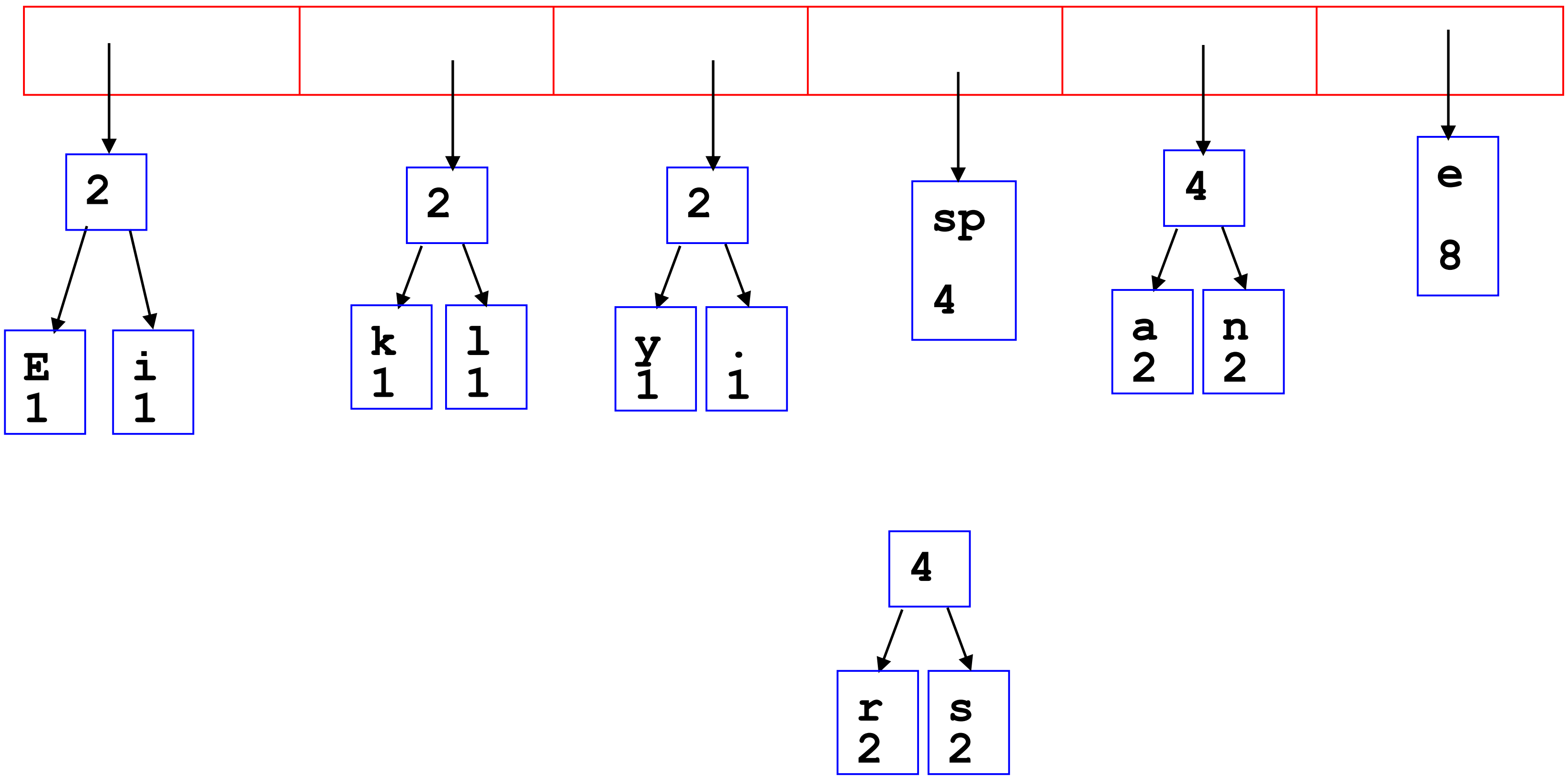
Building a Tree



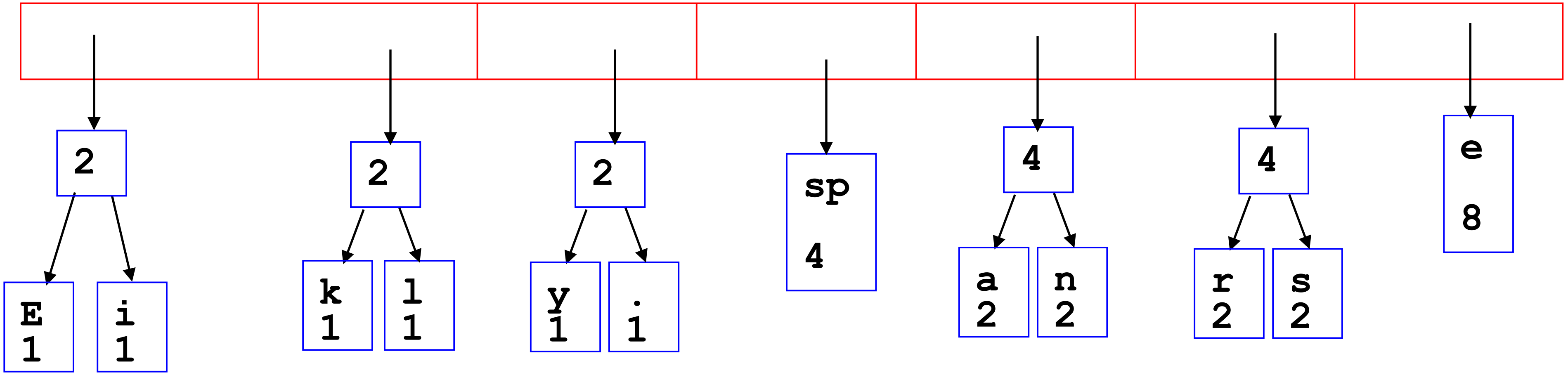
Building a Tree



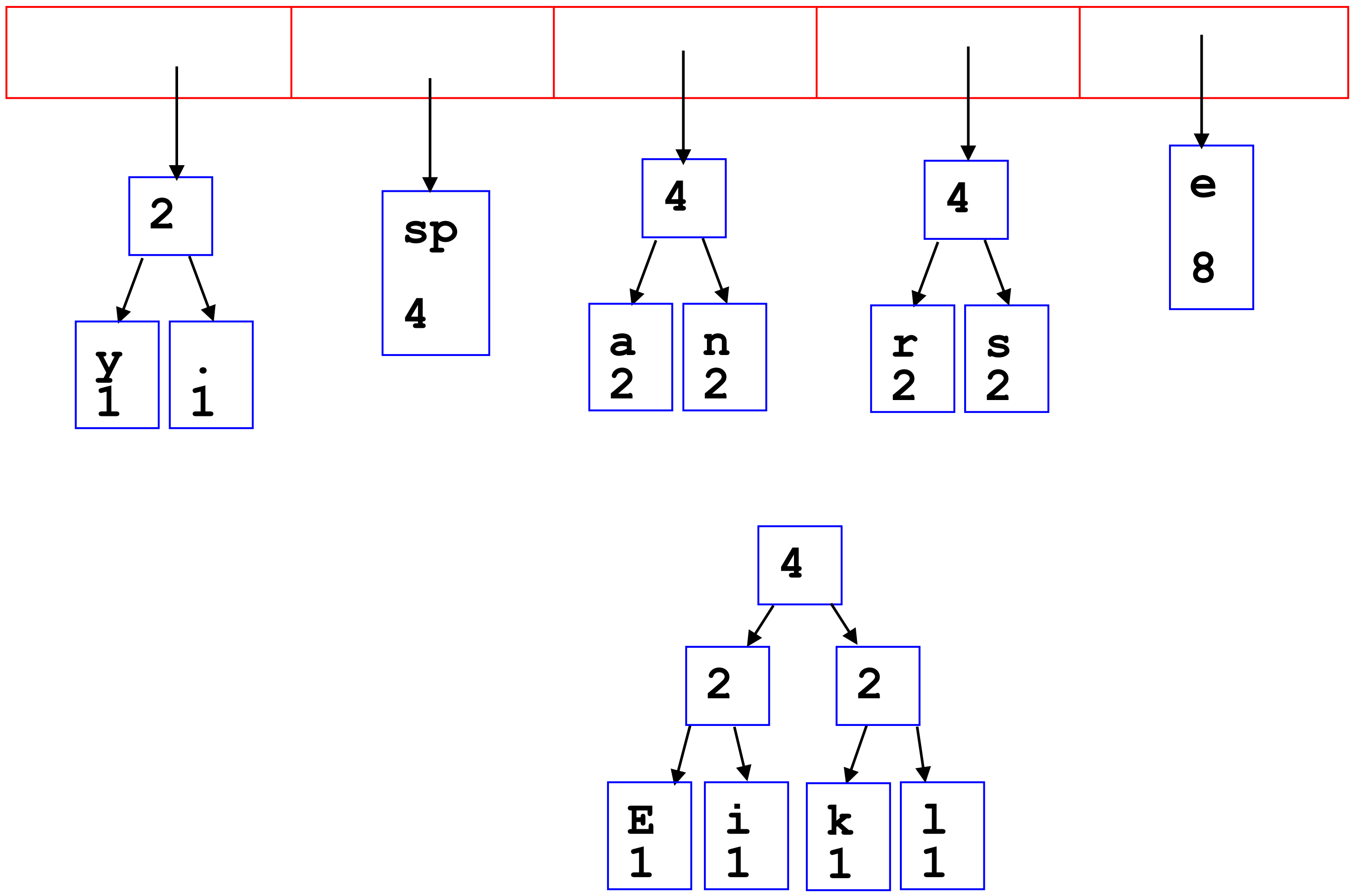
Building a Tree



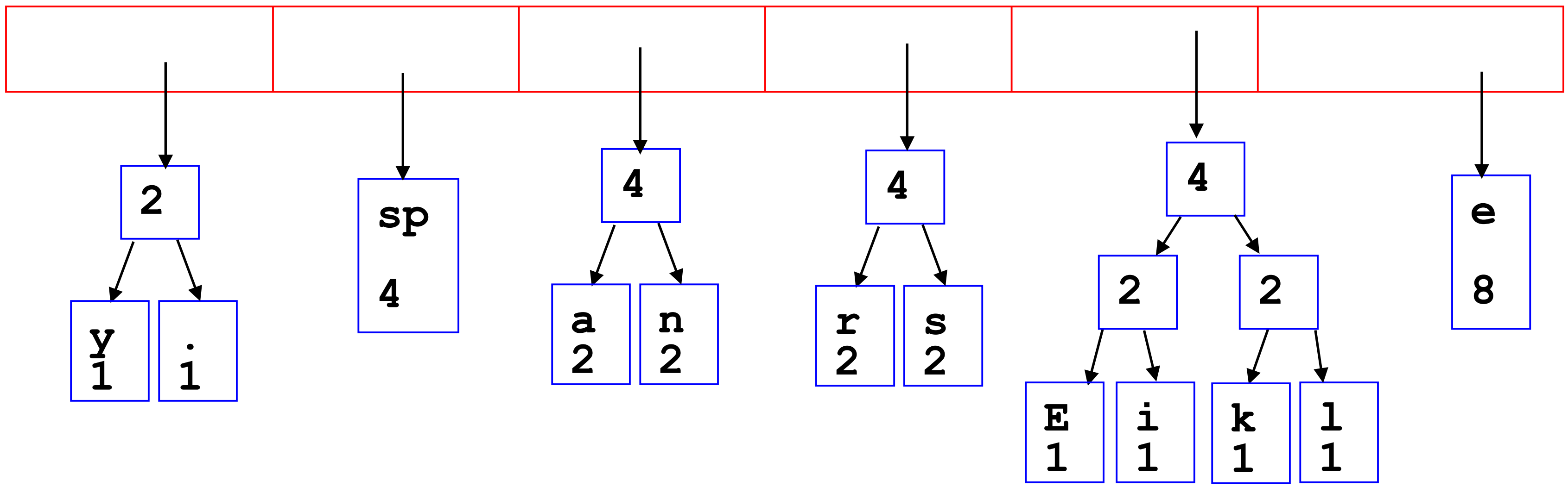
Building a Tree



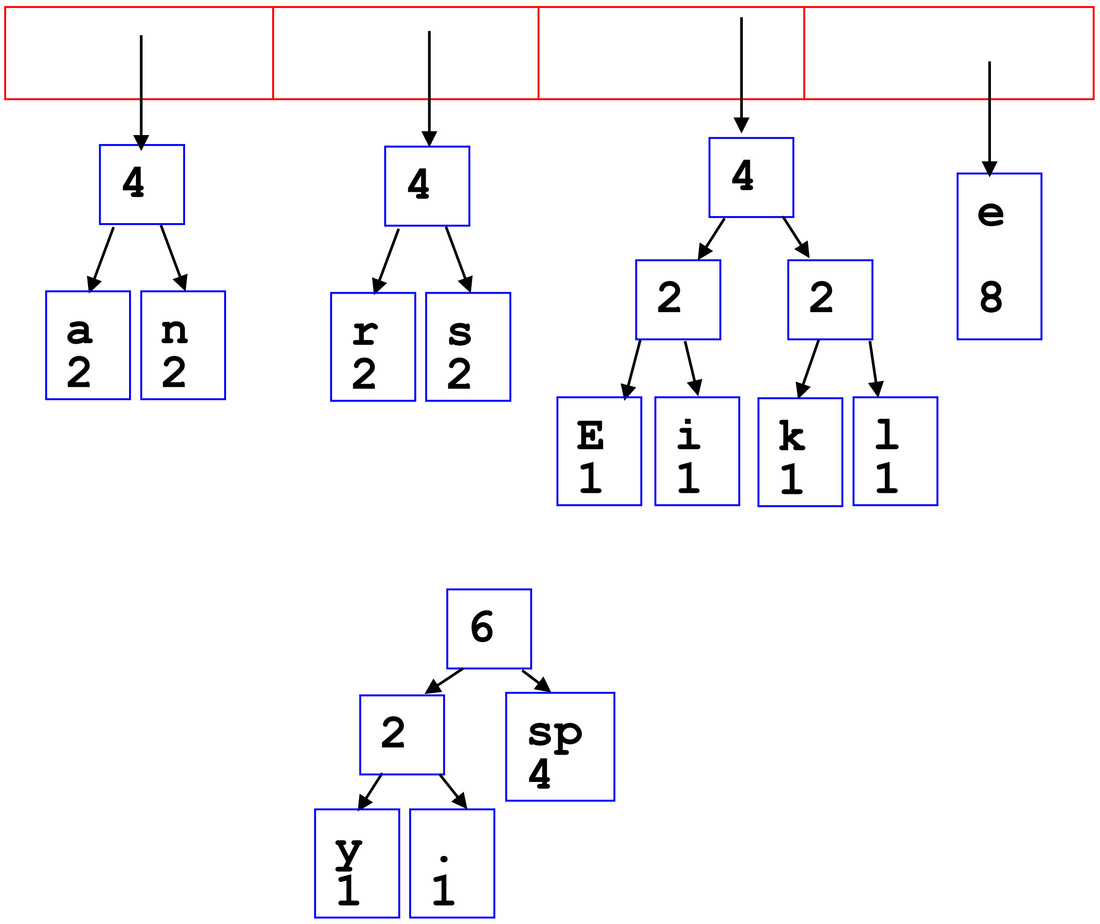
Building a Tree



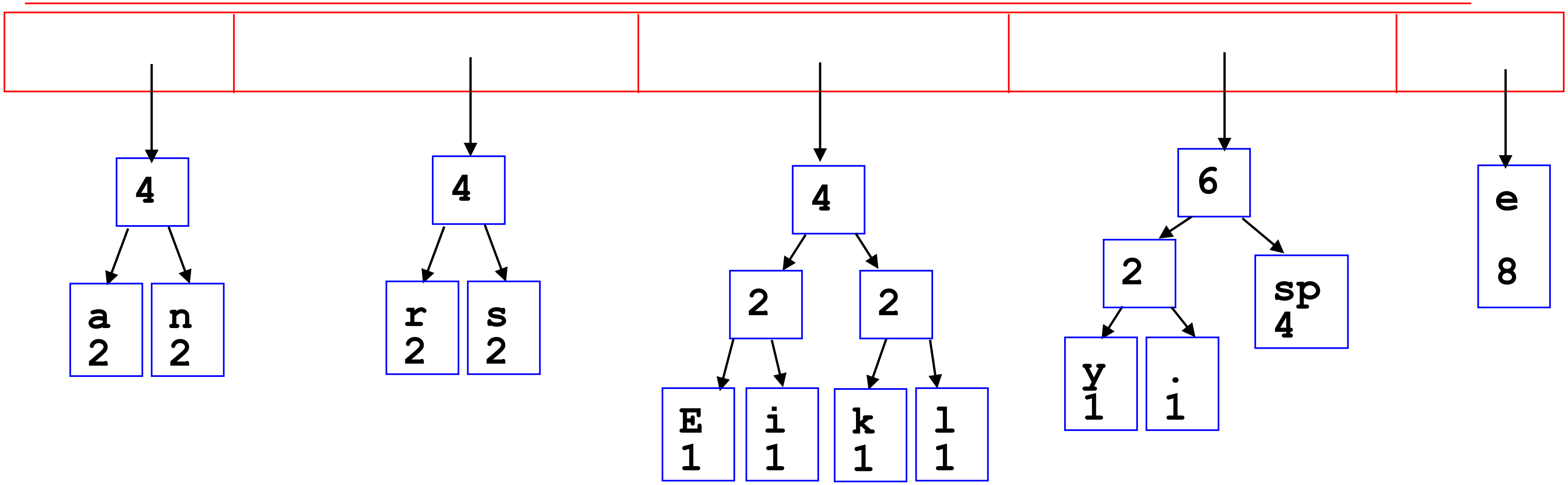
Building a Tree



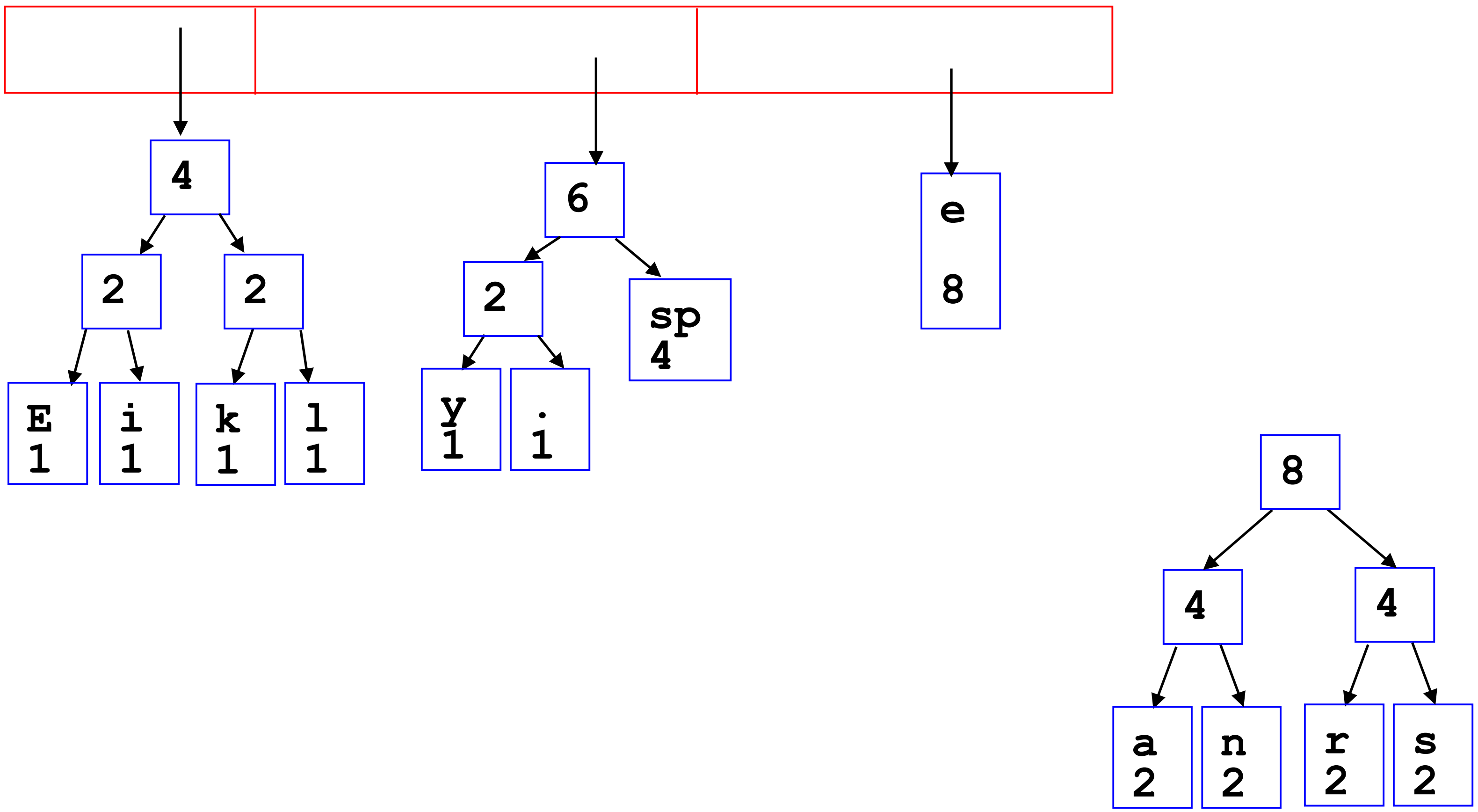
Building a Tree



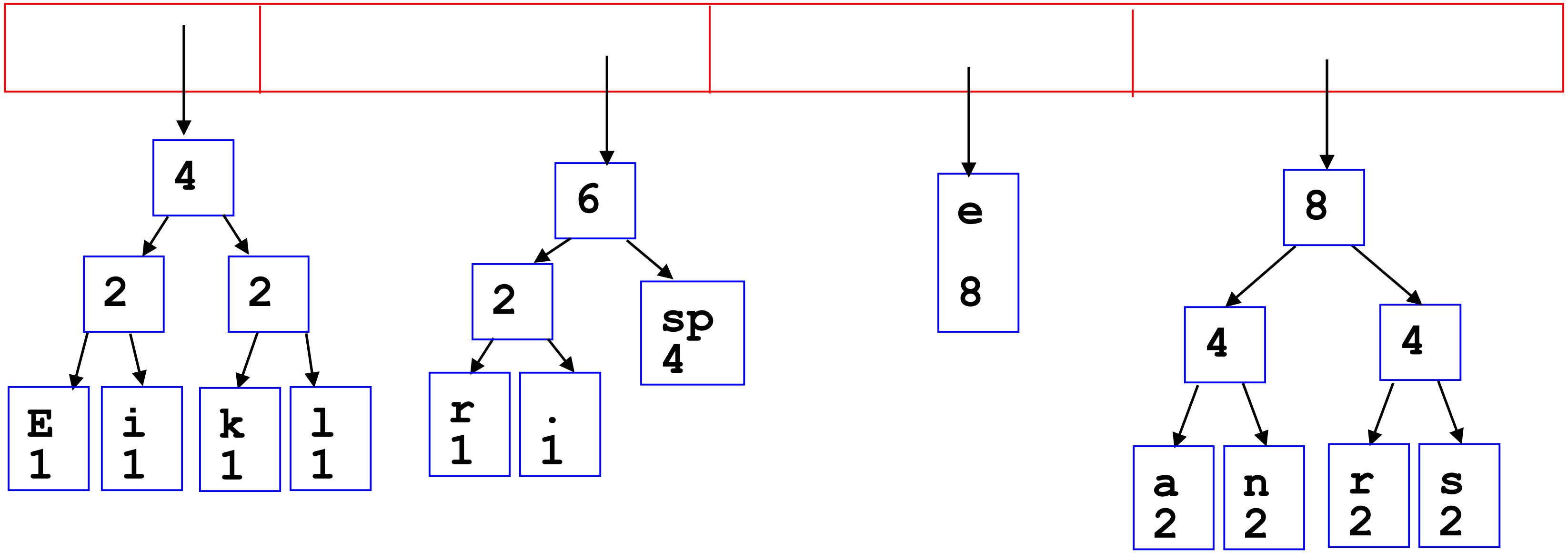
Building a Tree



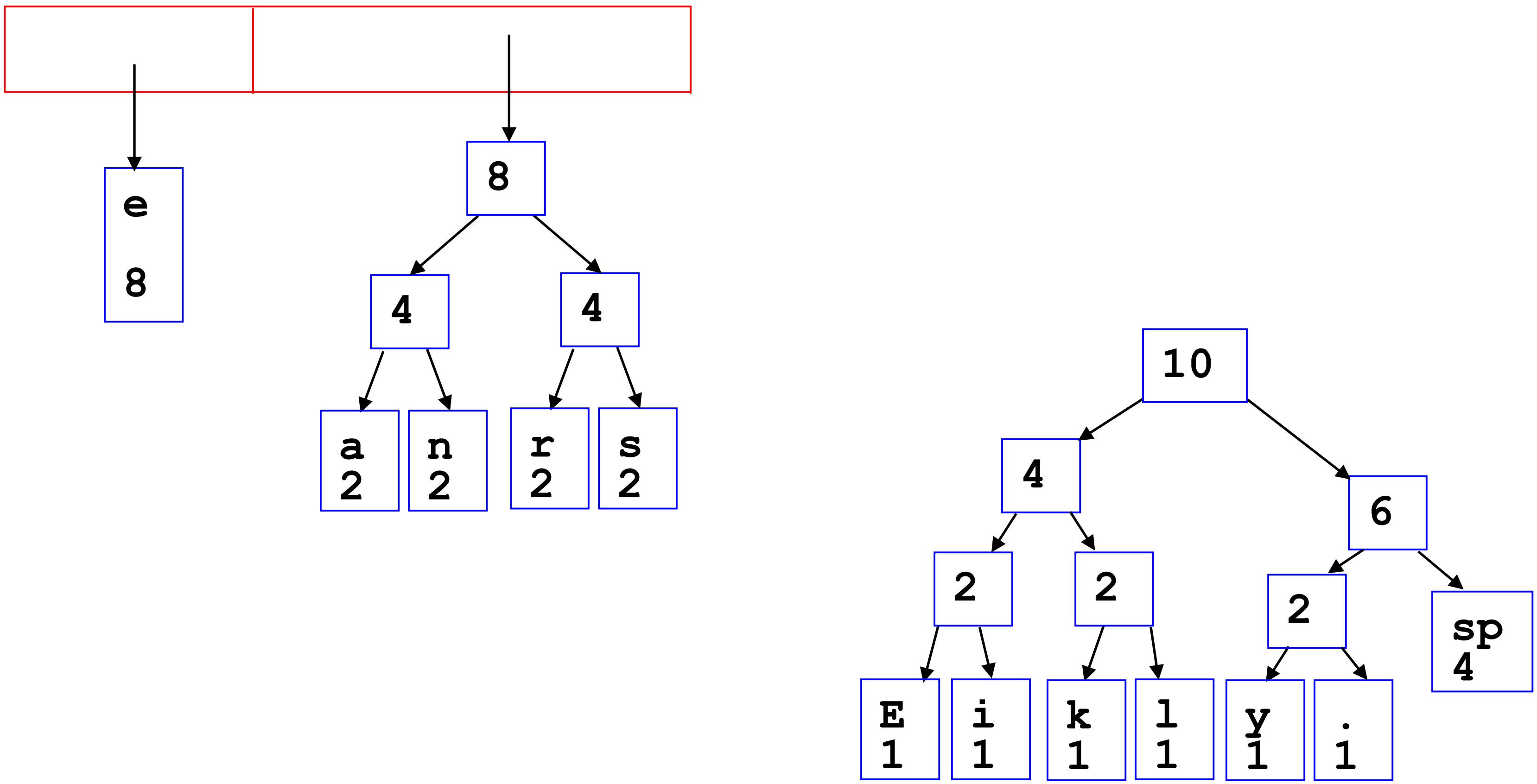
Building a Tree



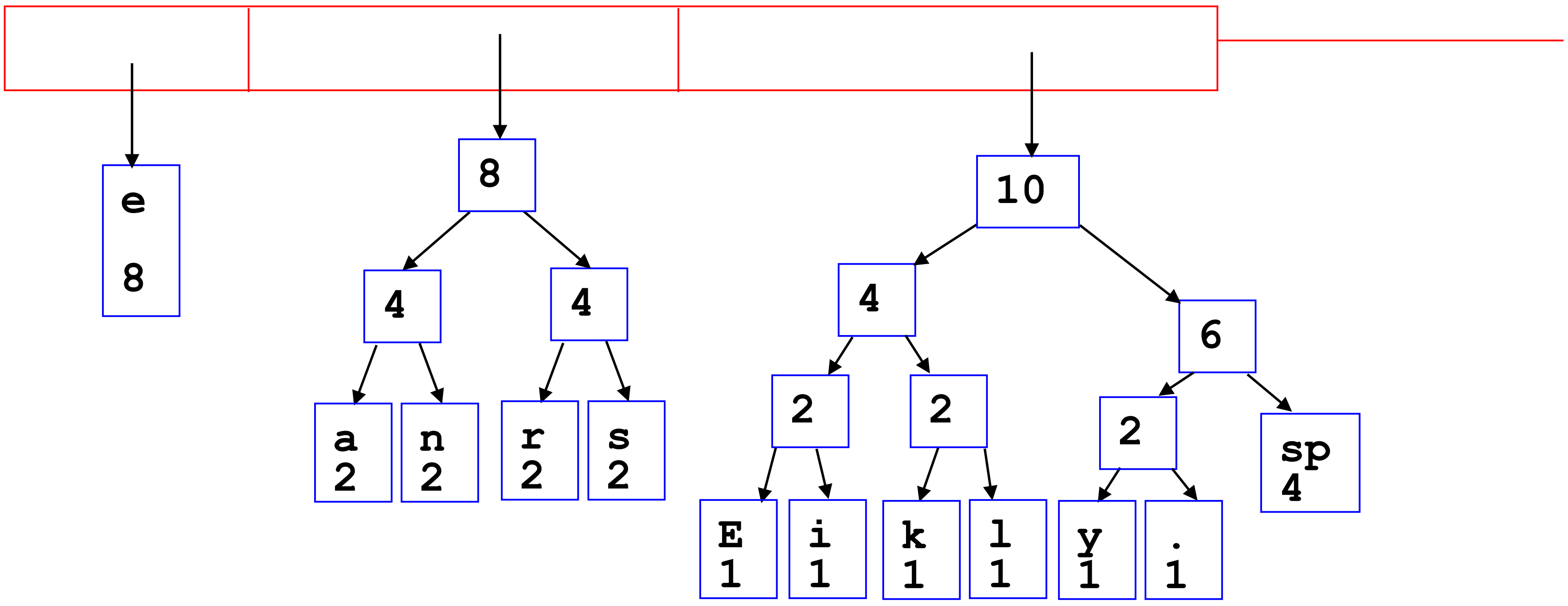
Building a Tree



Building a Tree



Building a Tree

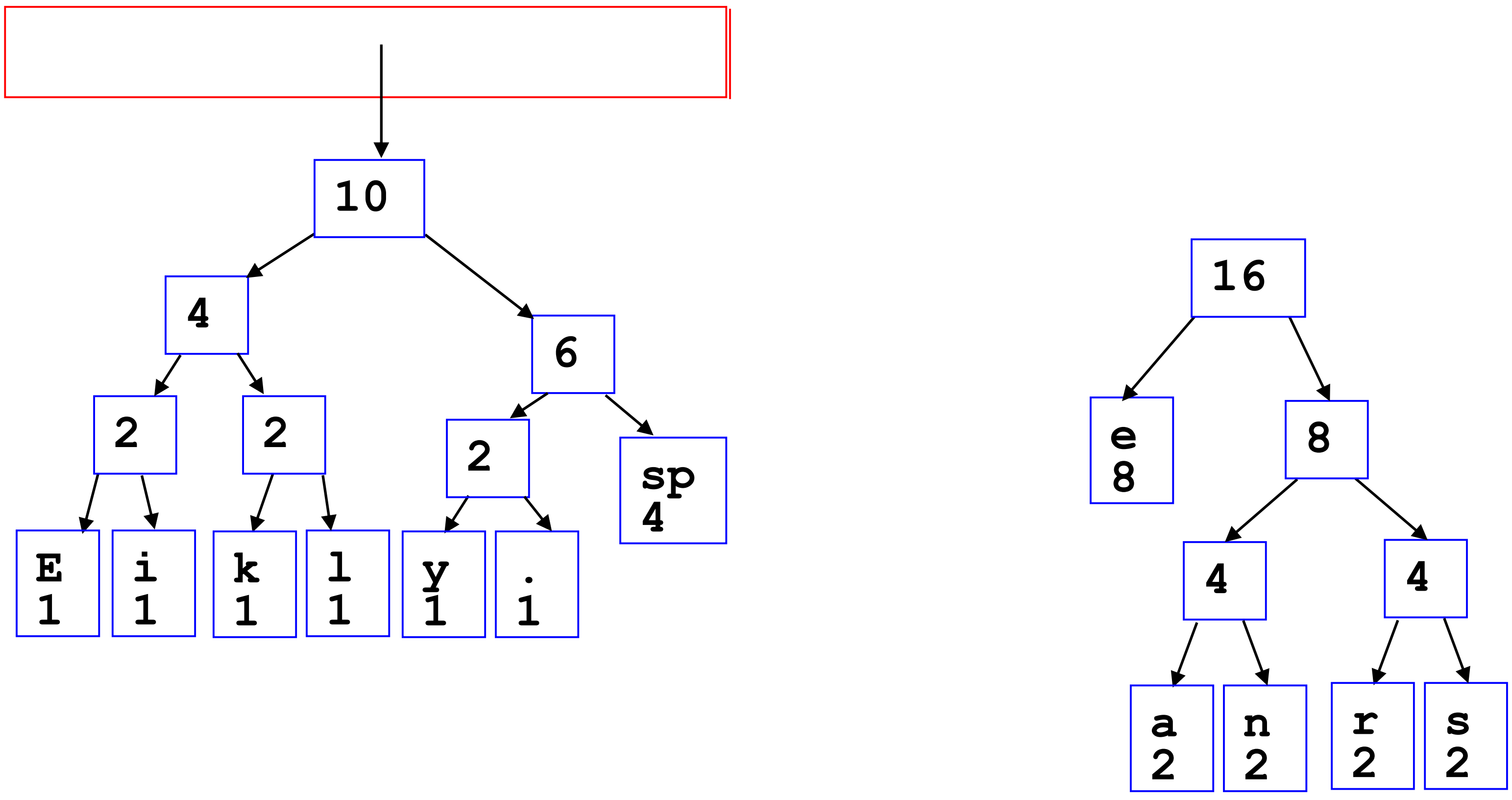


Clicker 2 - What is happening to the values with a low frequency compared to values with a high freq.?

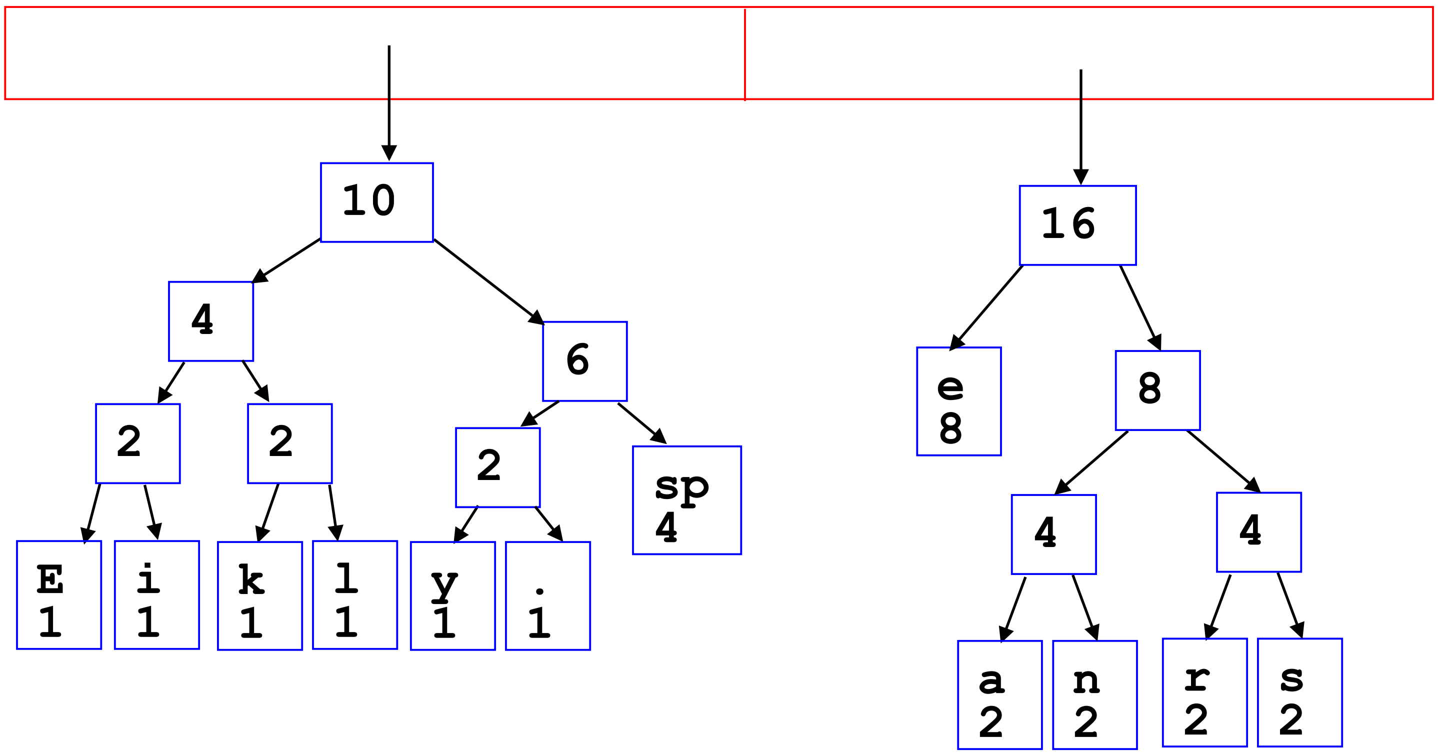
A. Smaller Depth B. Larger Depth

C. Something else

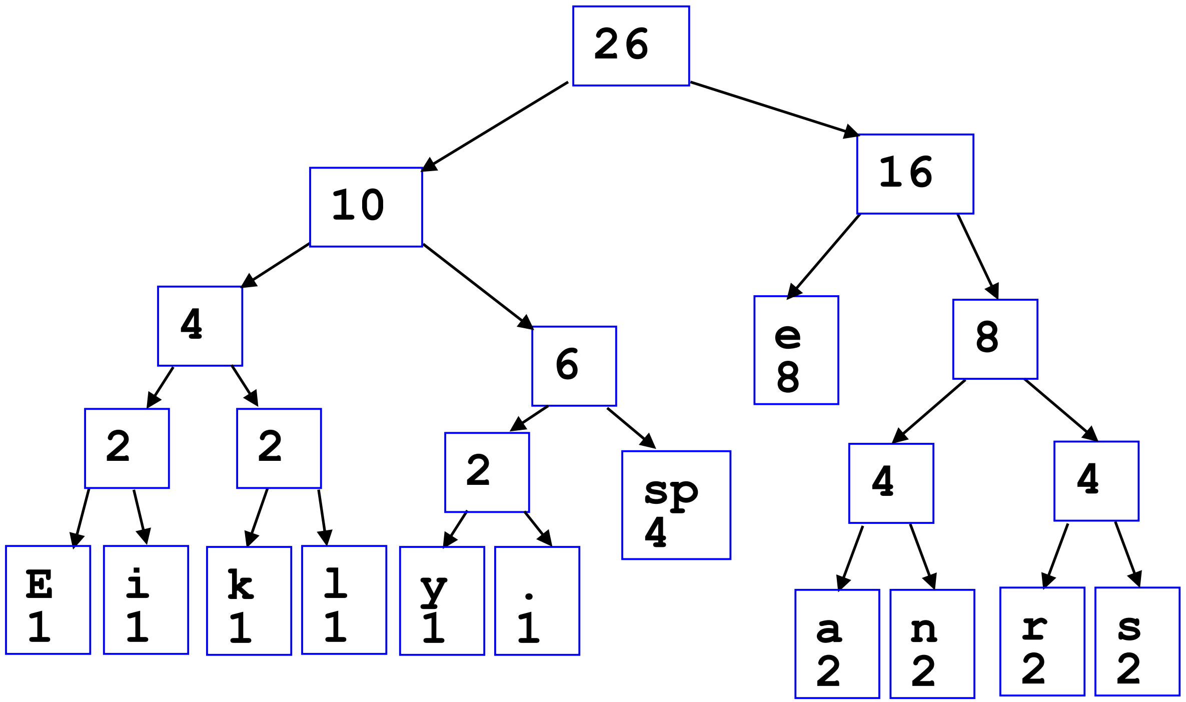
Building a Tree



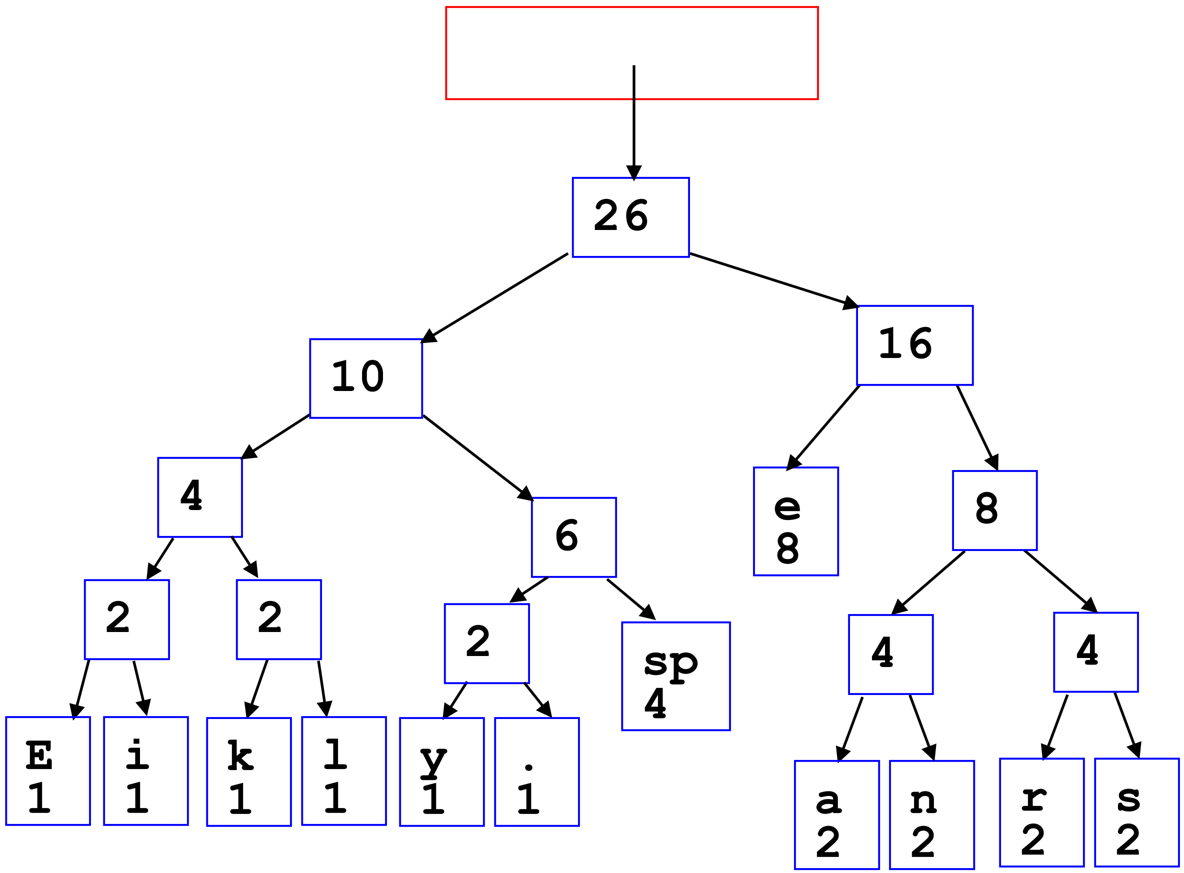
Building a Tree



Building a Tree



Building a Tree



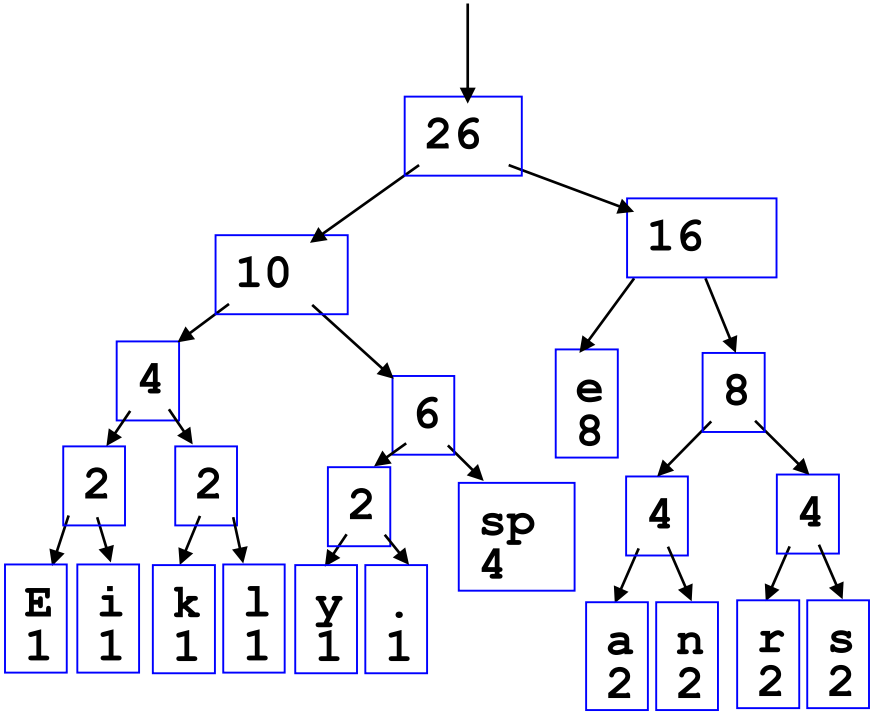
- **After enqueueing this node there is only one node left in priority queue.**

Building a Tree

Dequeue the single node left in the queue.

This tree contains the new code words for each character.

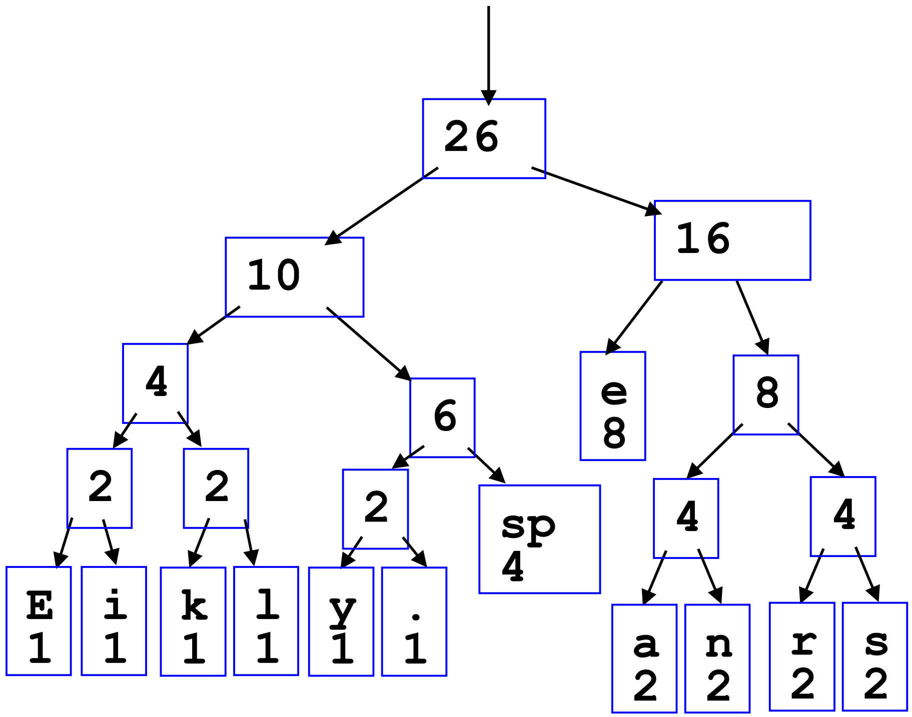
Frequency of root node should equal number of characters in text.



Eerie eyes seen near lake.   4 spaces,
26 characters total

Encoding the File
Traverse Tree for Codes

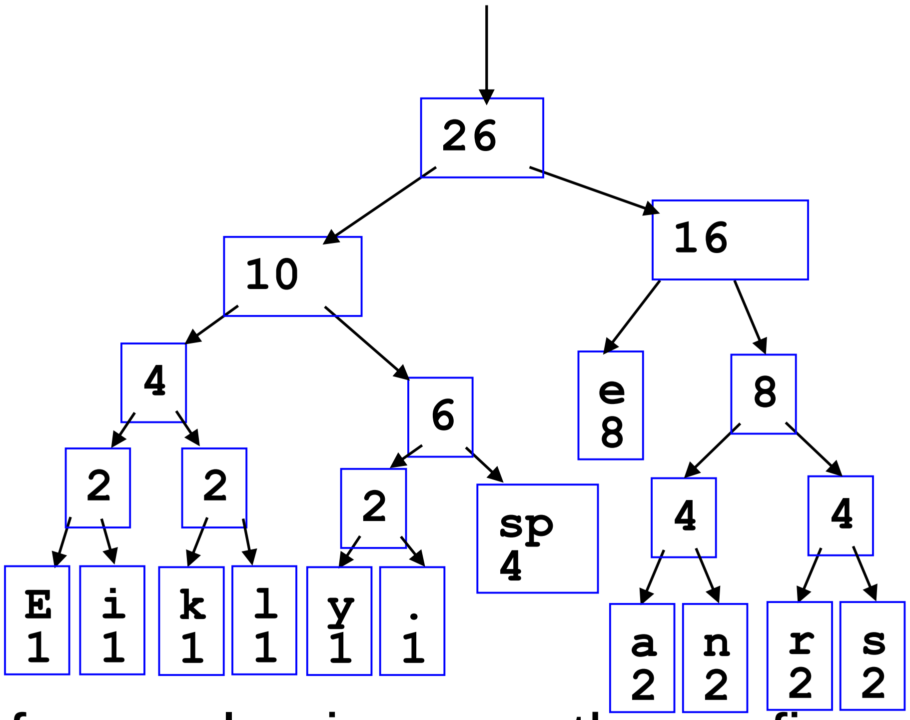
- Perform a traversal of the tree to obtain new code words (sequence of 0's and 1's)
- left, append a 0 to code word
- right append a 1 to code word
- code word is only complete when a leaf node is reached



Encoding the File

Traverse Tree for Codes

Original Value	New Code
E (0100 0101)	0000
i (0110 1001)	0001
k (0110 1011)	0010
l (0110 1100)	0011
y (0111 1001)	0100
. (0010 1110)	0101
space (0010 0000)	011
e (0110 0101)	10
a (0110 0001)	1100
n (0110 1110)	1101
r (0111 0010)	1110
s (0111 0011)	1111



Prefix free codes. The code for a value in never the prefix of another code.

Encoding the File

- Rescan original file and encode file using new code words

Eerie eyes seen near lake.

```
0000101111000001100111001001011110111111101011010111101101100111001100111100000100101
```

Char	New Code
E	0000
i	0001
k	0010
l	0011
y	0100
.	0101
space	011
e	10
a	1100
n	1101
r	1110
s	1111

Encoding the File
Results

- Have we made things any better?
- 84 bits to encode the file
- ASCII would take 8 * 26 = 208 bits

```
000010111000011001110
010010111101111111010
110101111011011001110
011001111000010100101
```

- **`If modified code used 4 bits per character are needed.  Total bits 4 * 26 = 104.  Savings not as great.`**

Decoding the File

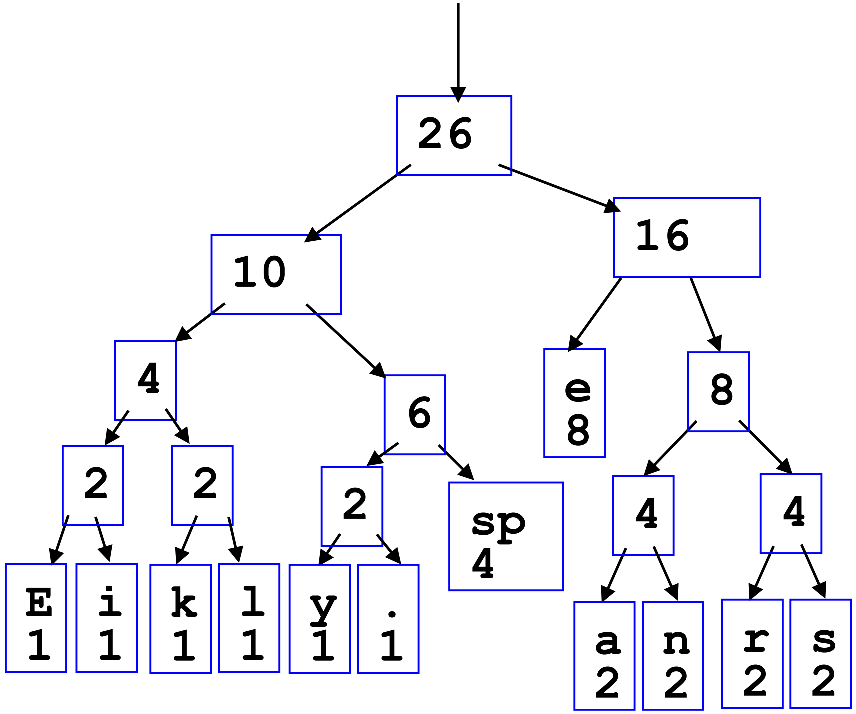
- How does receiver know what the codes are?
- Tree constructed for each file.
  - Considers frequency for each file
  - Big hit on compression, especially for smaller files
- Tree predetermined
  - based on statistical analysis of text files
    or other file types

Clicker 3 - Decoding the File

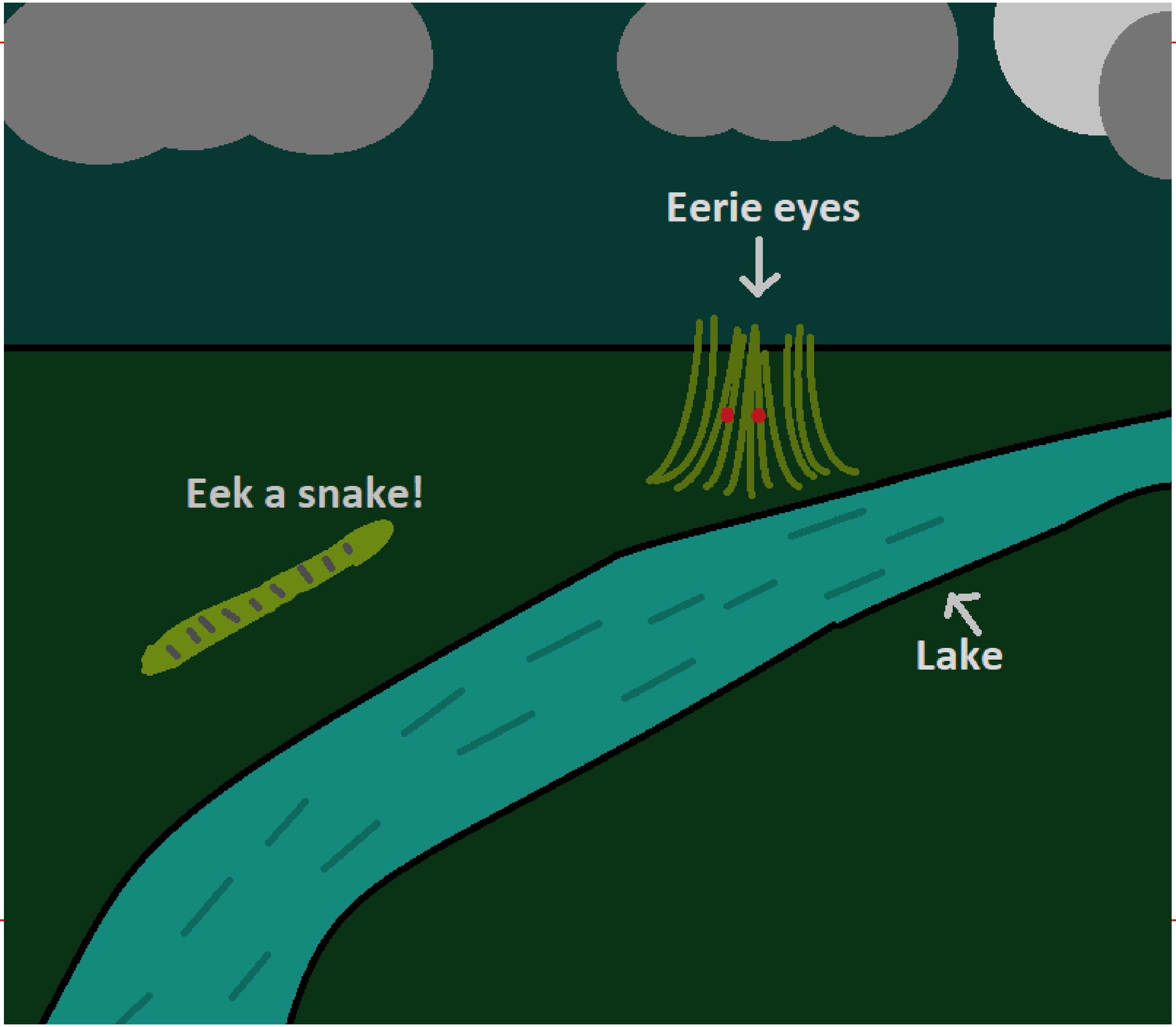
- Once receiver has tree it scans incoming bit stream
- 0 $\Rightarrow$ go left
- 1 $\Rightarrow$ go right

10100010011110001111111011100001010

A. elk nay sir
B. eek a snake
C. eek kin sly
D. eek snarl nil
E. eel a snarl



Alex Fall 2022



Assignment Hints

- reading chunks not chars
- header format
- the pseudo eof value
- the GUI

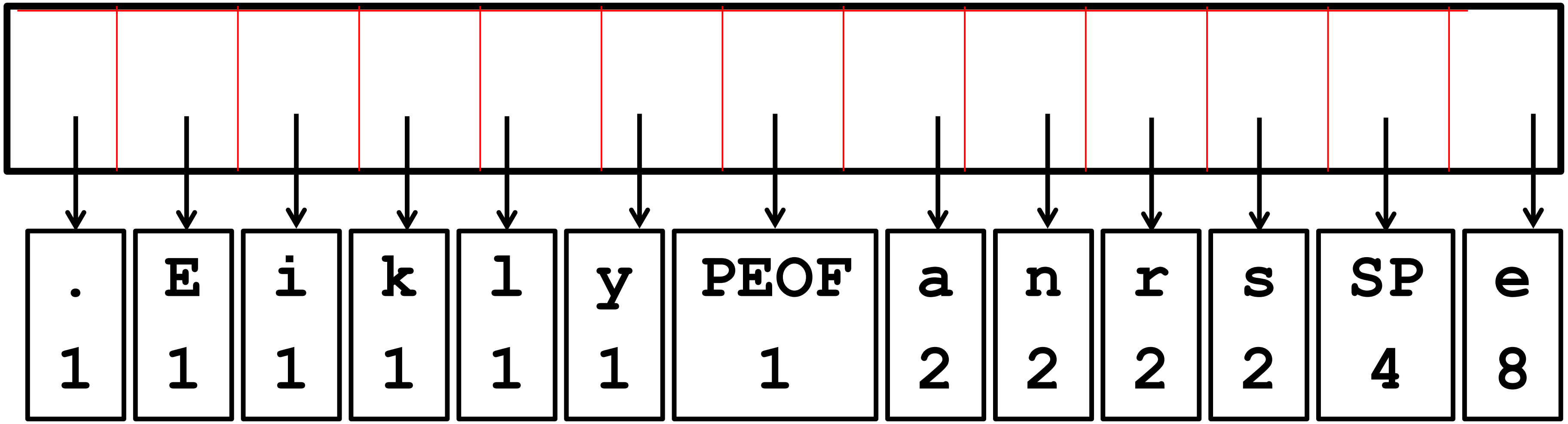
Assignment Example

- "Eerie eyes seen near lake." will result in different codes than those shown in slides due to:
  - adding elements in order to PriorityQueue
  - required pseudo eof value (PEOF)

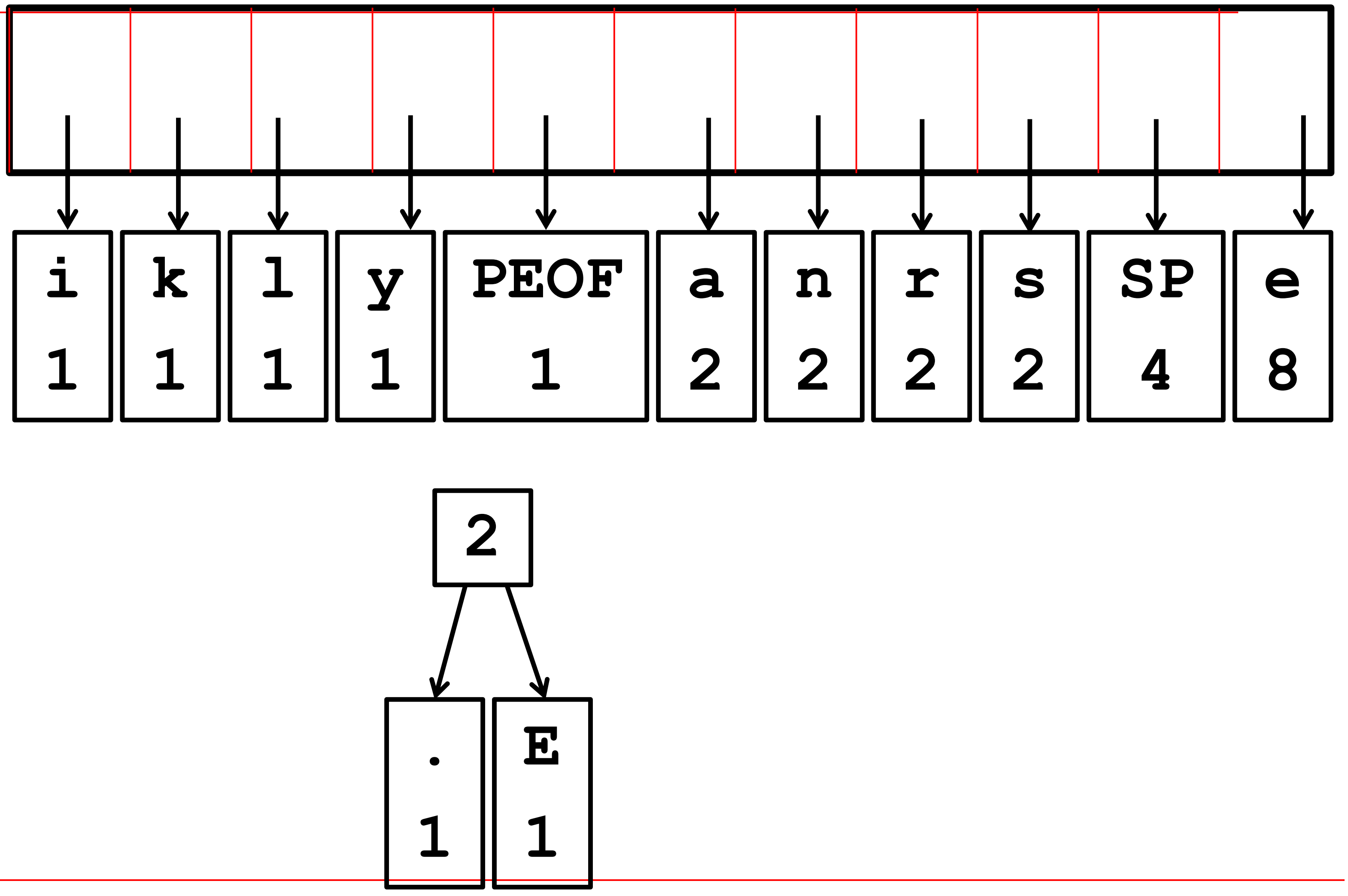
Assignment Example

Char	Freq.	Char	Freq.	Char	Freq.
E	1	y	1	k	1
e	8	s	2	.	1
r	2	n	2	PEOF	1
i	1	a	2		
space	4	l	1		

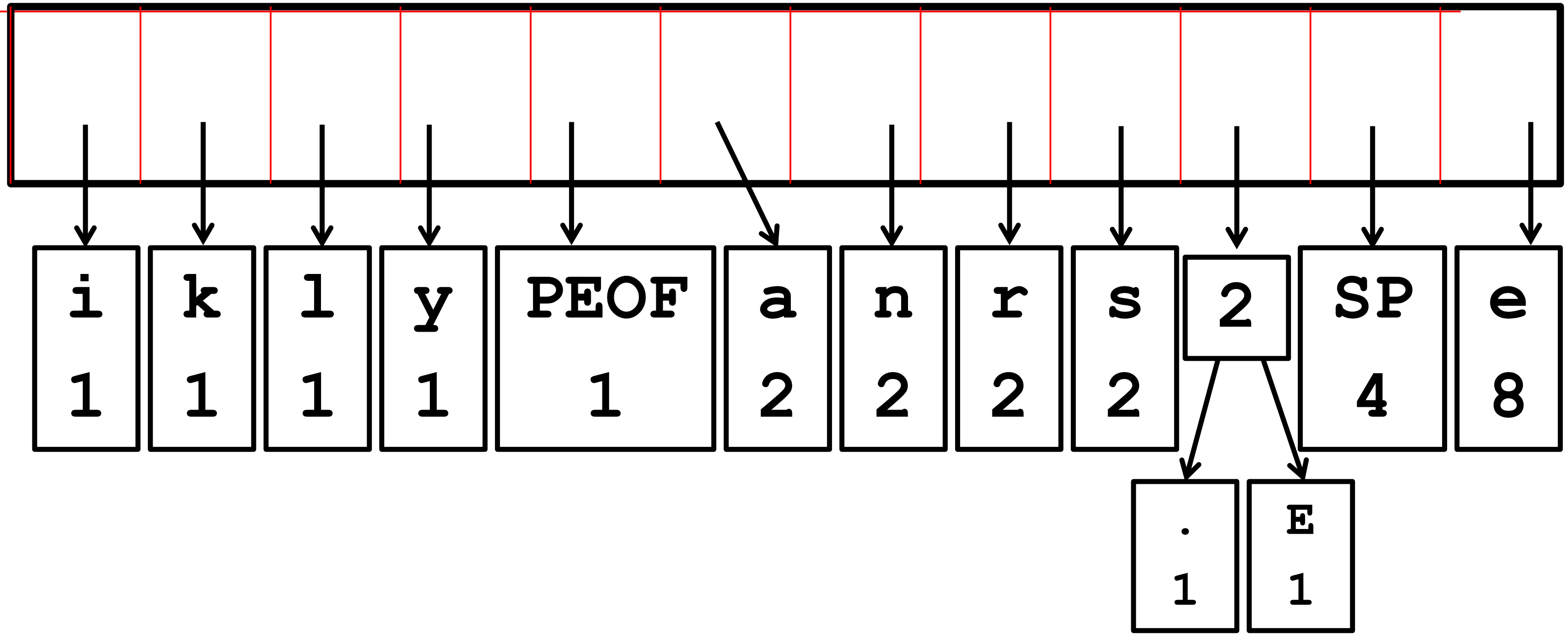
Assignment Example



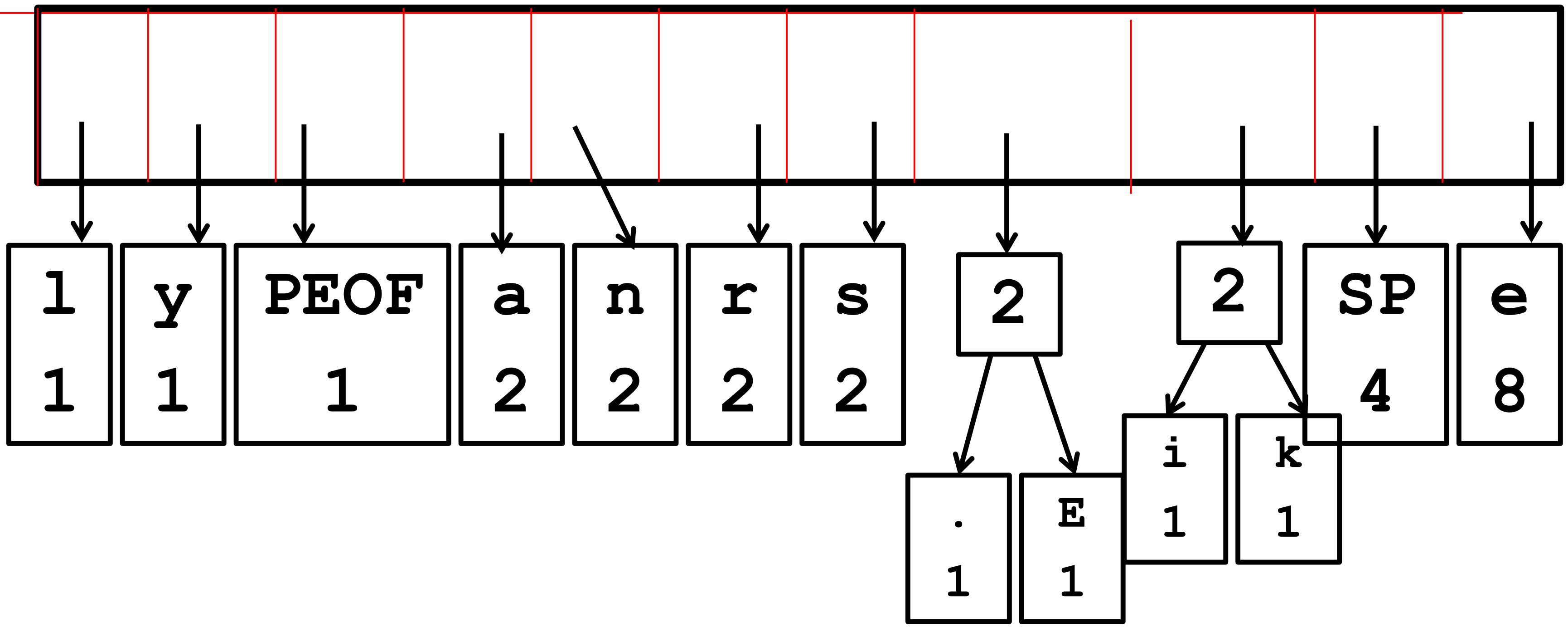
Assignment Example



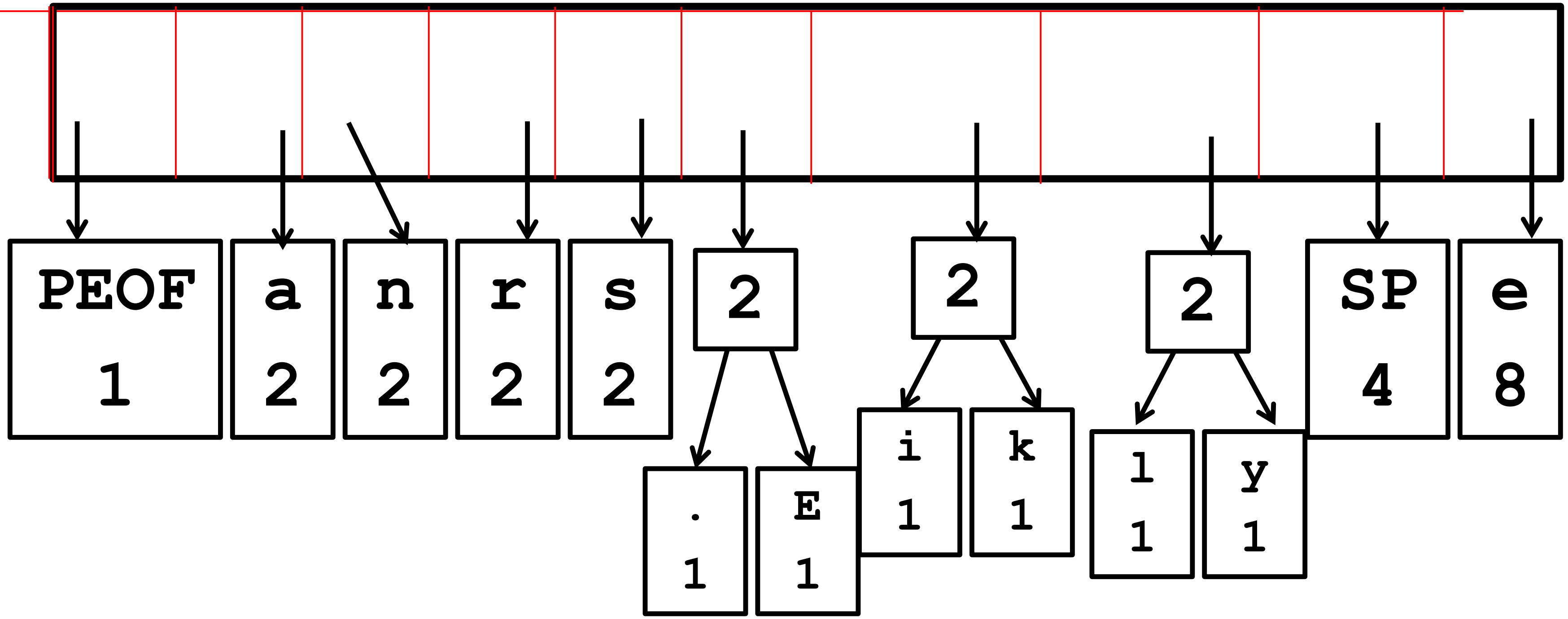
Assignment Example



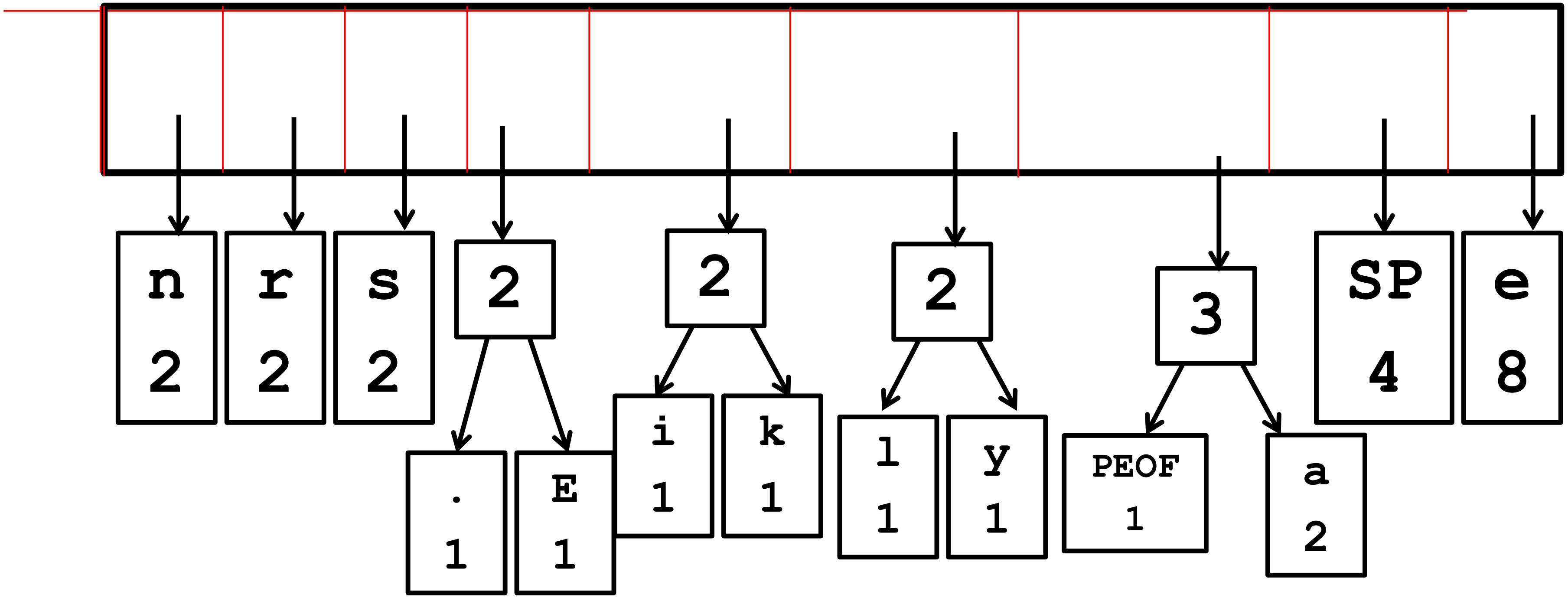
Assignment Example



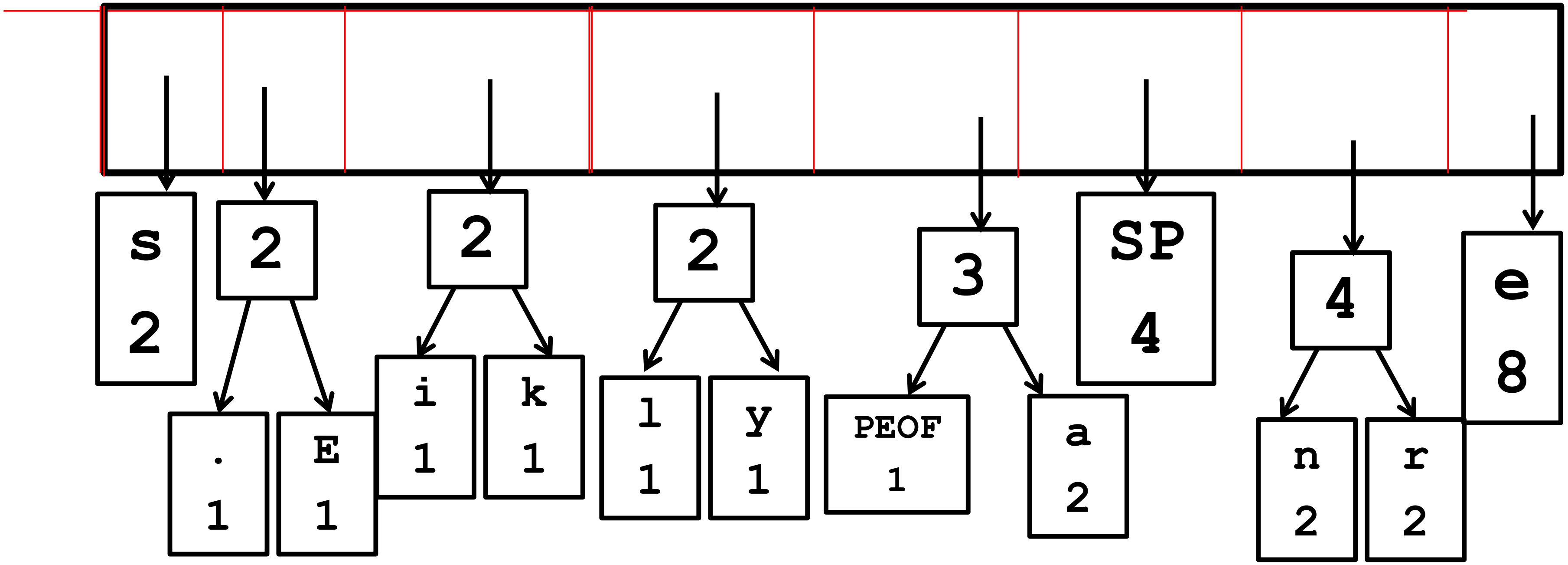
Assignment Example



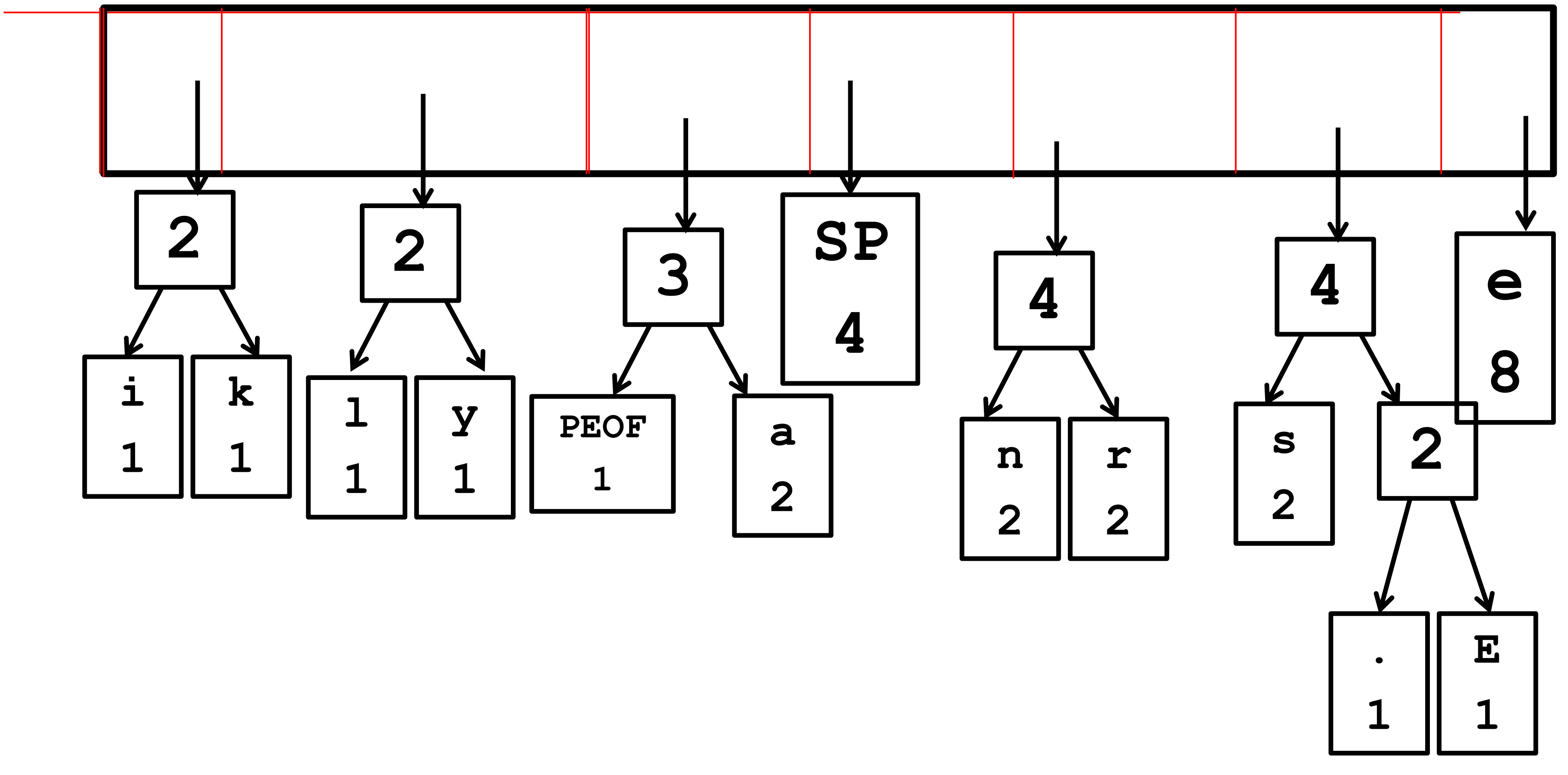
Assignment Example

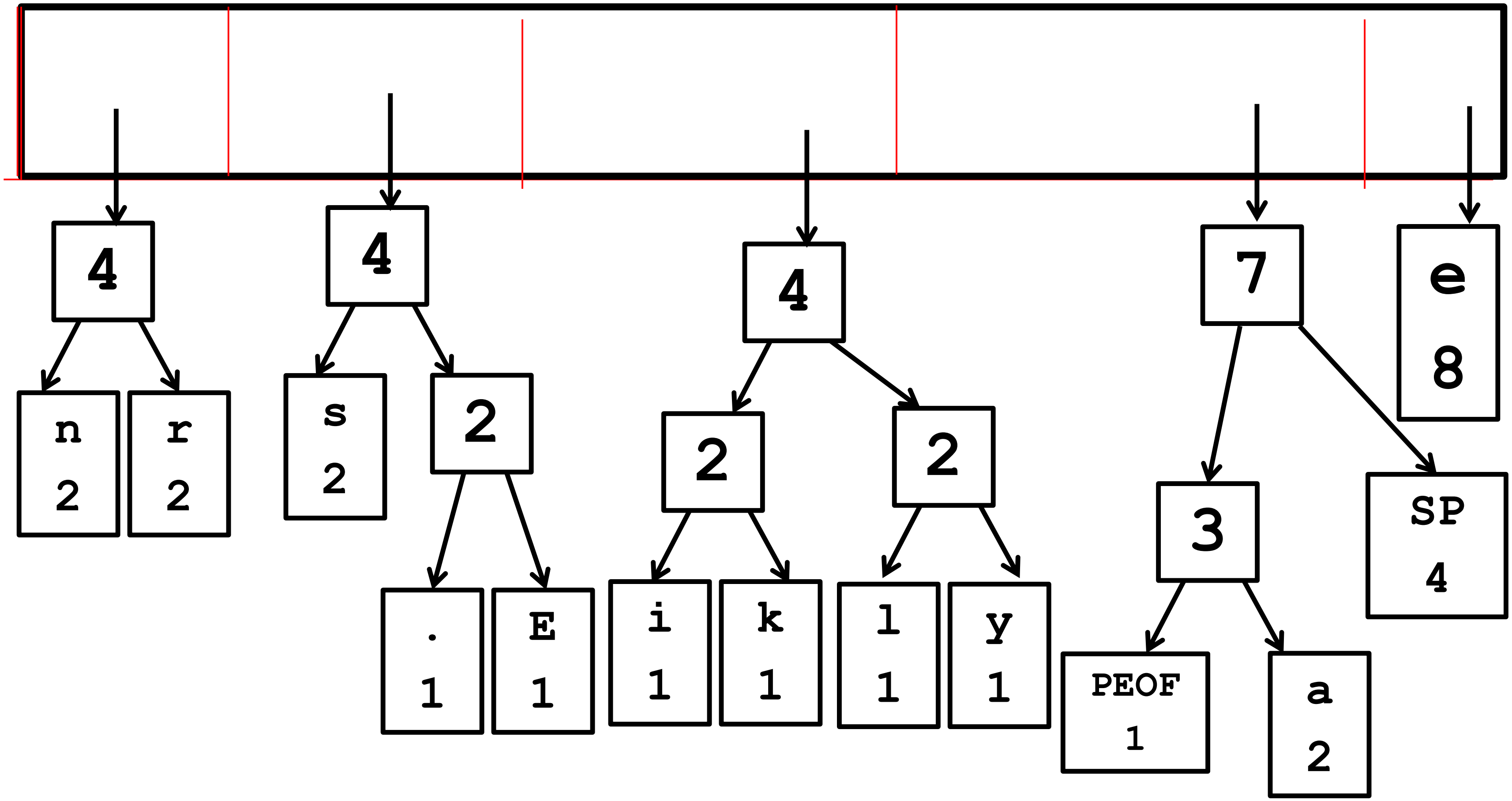


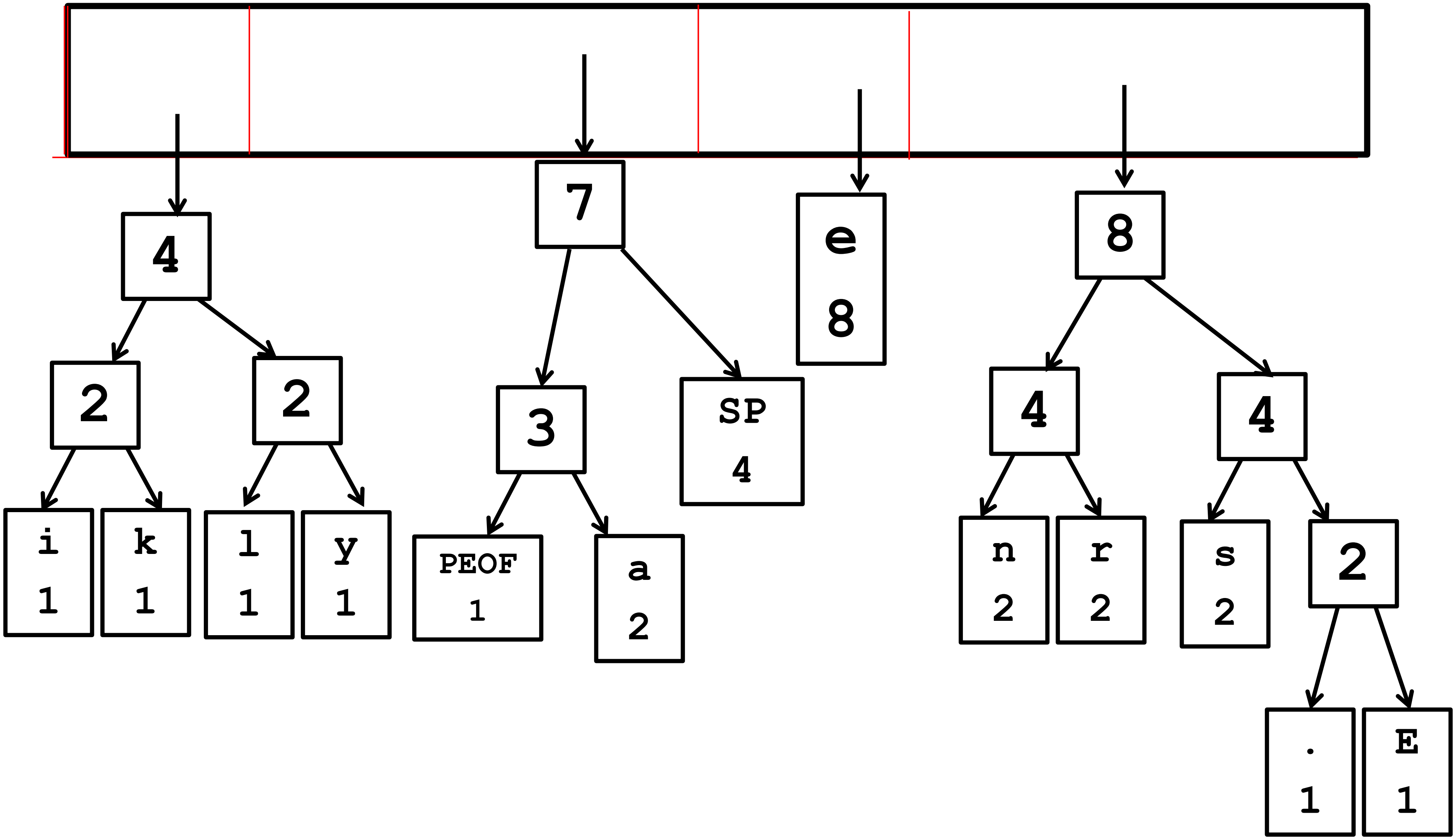
Assignment Example

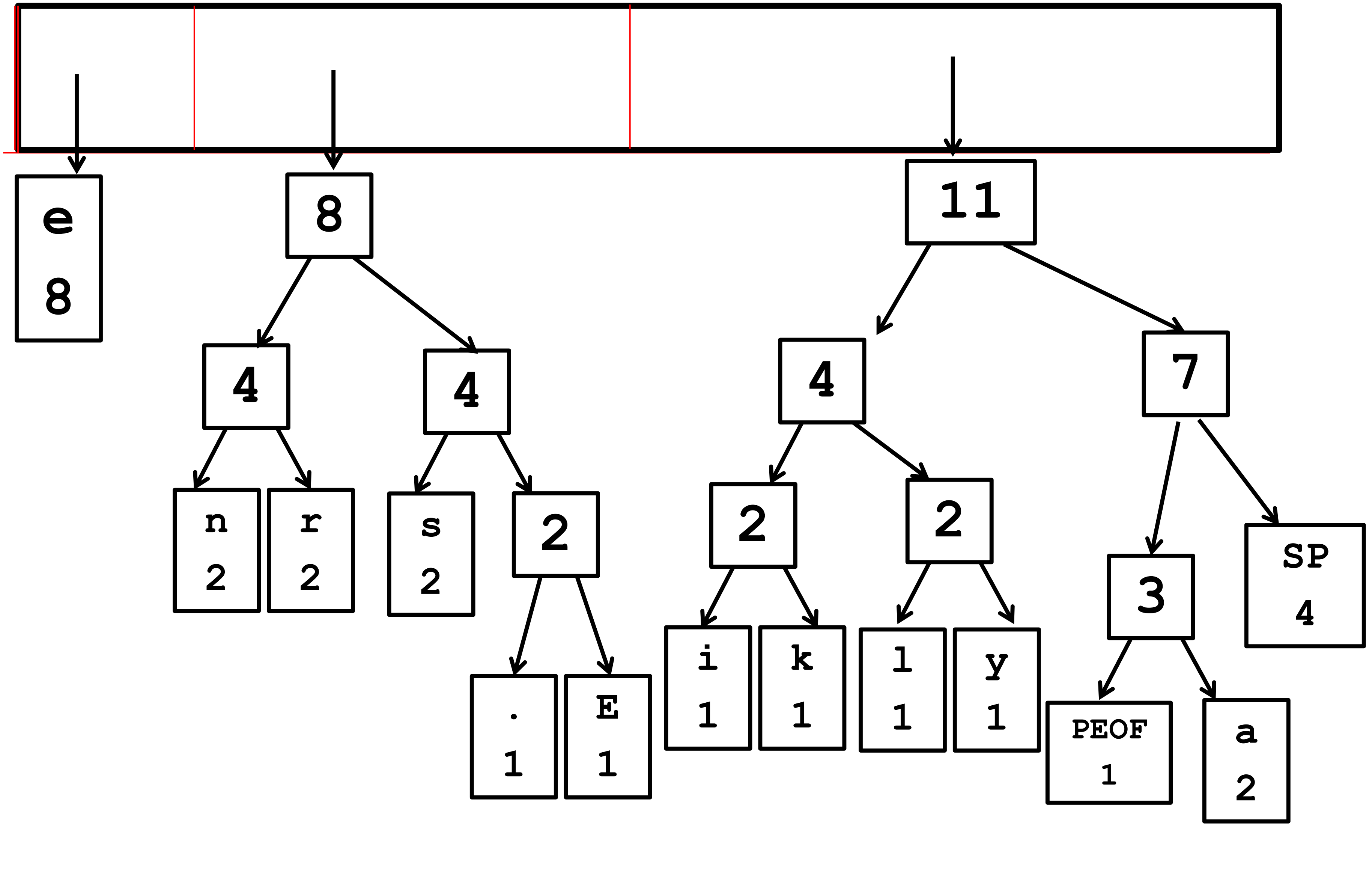


Assignment Example

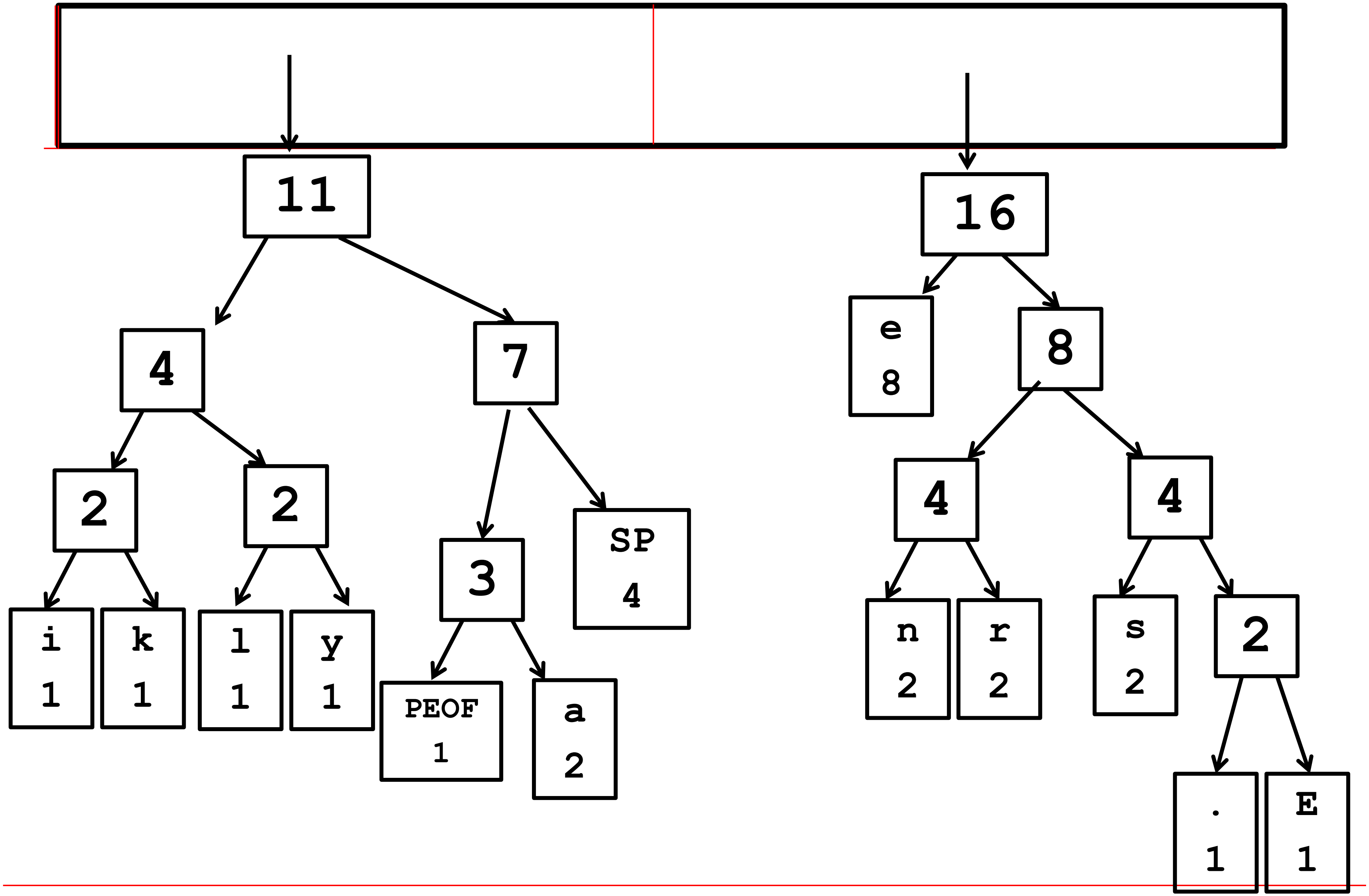


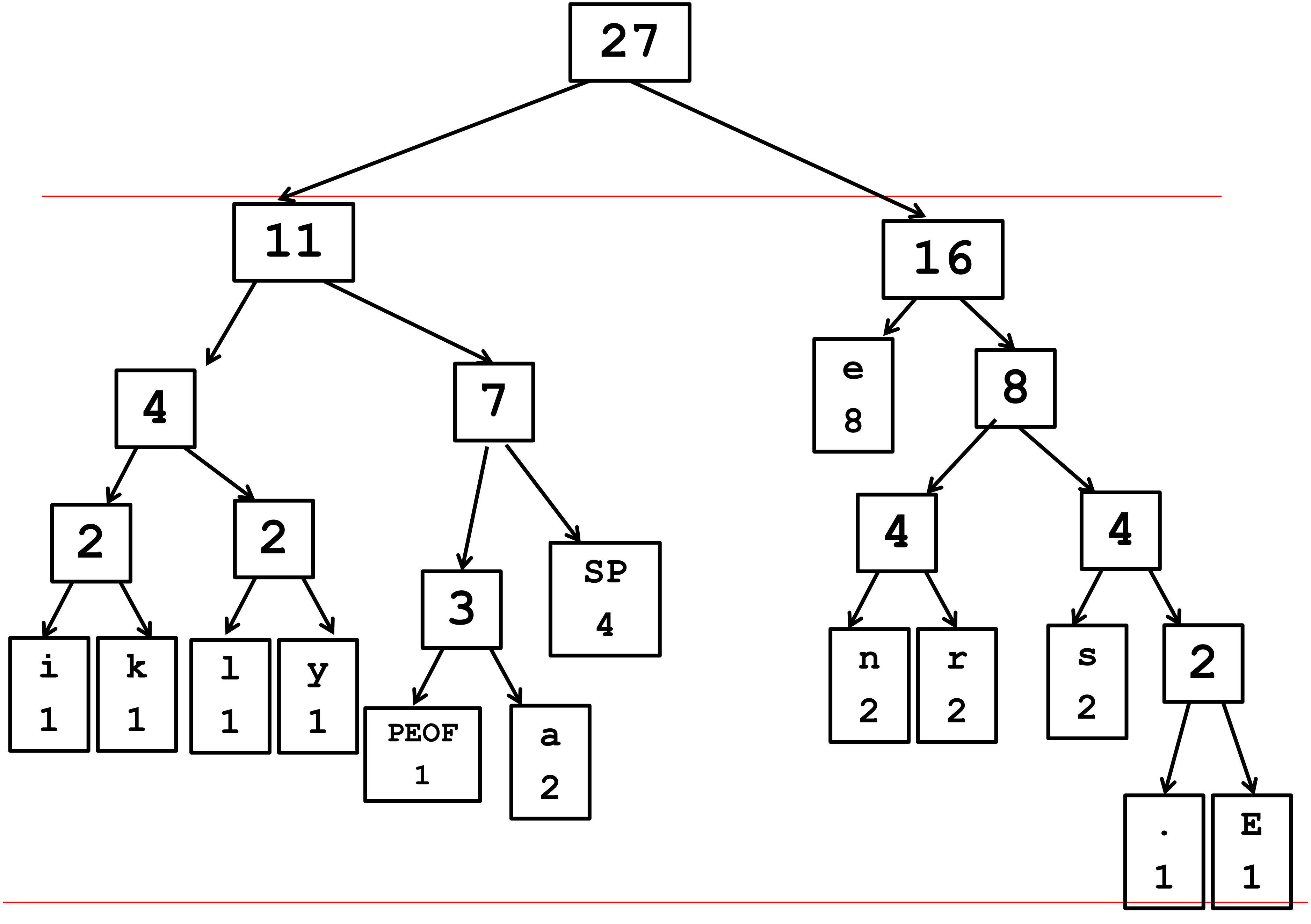

4
4
4
7
e
8
n
2
r
2
s
2
2
2
2
3
SP
4
.
1
E
1
i
1
k
1
l
1
y
1
PEOF
1
a
2


4
2
2
i
1
k
1
l
1
y
1
7
3
SP
4
PEOF
1
a
2
e
8
8
4
4
n
2
r
2
s
2
2
.
1
E
1



e
8
8
11
4
4
4
7
n
2
r
2
s
2
2
2
2
3
SP
4
.
1
E
1
i
1
k
1
l
1
y
1
PEOF
1
a
2


11
16
4
7
e
8
8
2
2
3
SP
4
4
4
i
1
k
1
l
1
y
1
PEOF
1
a
2
n
2
r
2
s
2
2
.
1
E
1


27
11
16
4
7
e
8
8
2
2
3
SP
4
4
4
i
1
k
1
l
1
y
1
PEOF
1
a
2
n
2
r
2
s
2
2
.
1
E
1

Codes

```
value: 32, equivalent char:  , frequency: 4, new code 011
value: 46, equivalent char: ., frequency: 1, new code 11110
value: 69, equivalent char: E, frequency: 1, new code 11111
value: 97, equivalent char: a, frequency: 2, new code 0101
value: 101, equivalent char: e, frequency: 8, new code 10
value: 105, equivalent char: i, frequency: 1, new code 0000
value: 107, equivalent char: k, frequency: 1, new code 0001
value: 108, equivalent char: l, frequency: 1, new code 0010
value: 110, equivalent char: n, frequency: 2, new code 1100
value: 114, equivalent char: r, frequency: 2, new code 1101
value: 115, equivalent char: s, frequency: 2, new code 1110
value: 121, equivalent char: y, frequency: 1, new code 0011
value: 256, equivalent char: ?, frequency: 1, new code 0100
```

Altering files

- Tower bit map (Eclipse/Huffman/Data). Alter the first 300 characters of line 16765 to this

```
~00~00~00~00~00~00~00~00~00~00~00~00~00
~00~00~00~00~00~00~00~00~00~00~00~00~00
~00~00~00~00~00~00~00~00~00~00~00~00~00
~00~00~00~00~00~00~00~00~00~00~00~00~00
~00~00~00~00~00~00~00~00~00~00~00~00~00
~00~00~00~00~00~00~00~00~00~00~00~00~00
~00~00~00~00~00~00~00~00~00~00~00~00~00
~00~00~00~00~00~00~00~00~00 xxx
```

Compression - Why Bother?

- Apostolos "Toli" Lerios
- Facebook Engineer
- Heads image storage group
- jpeg images already compressed
- look for ways to compress even more
- 1% less space = millions of dollars in savings

Graphs
Topic 22

" Hopefully, you've played around a bit with The Oracle of Bacon at Virginia and discovered how few steps are necessary to link just about anybody who has ever been in a movie to Kevin Bacon, but could there be some actor or actress who is even closer to the center of the Hollywood universe?.

By processing all of the almost half of a million people in the Internet Movie Database I discovered that there are currently 1160 people who are *better* centers than Kevin Bacon! … By computing the average of these numbers we see that the average (Sean) Connery Number is about 2.682 making Connery a better center than Bacon"

-**Who is the Center of the Hollywood Universe?, University of Virginia**

That was in 2001.

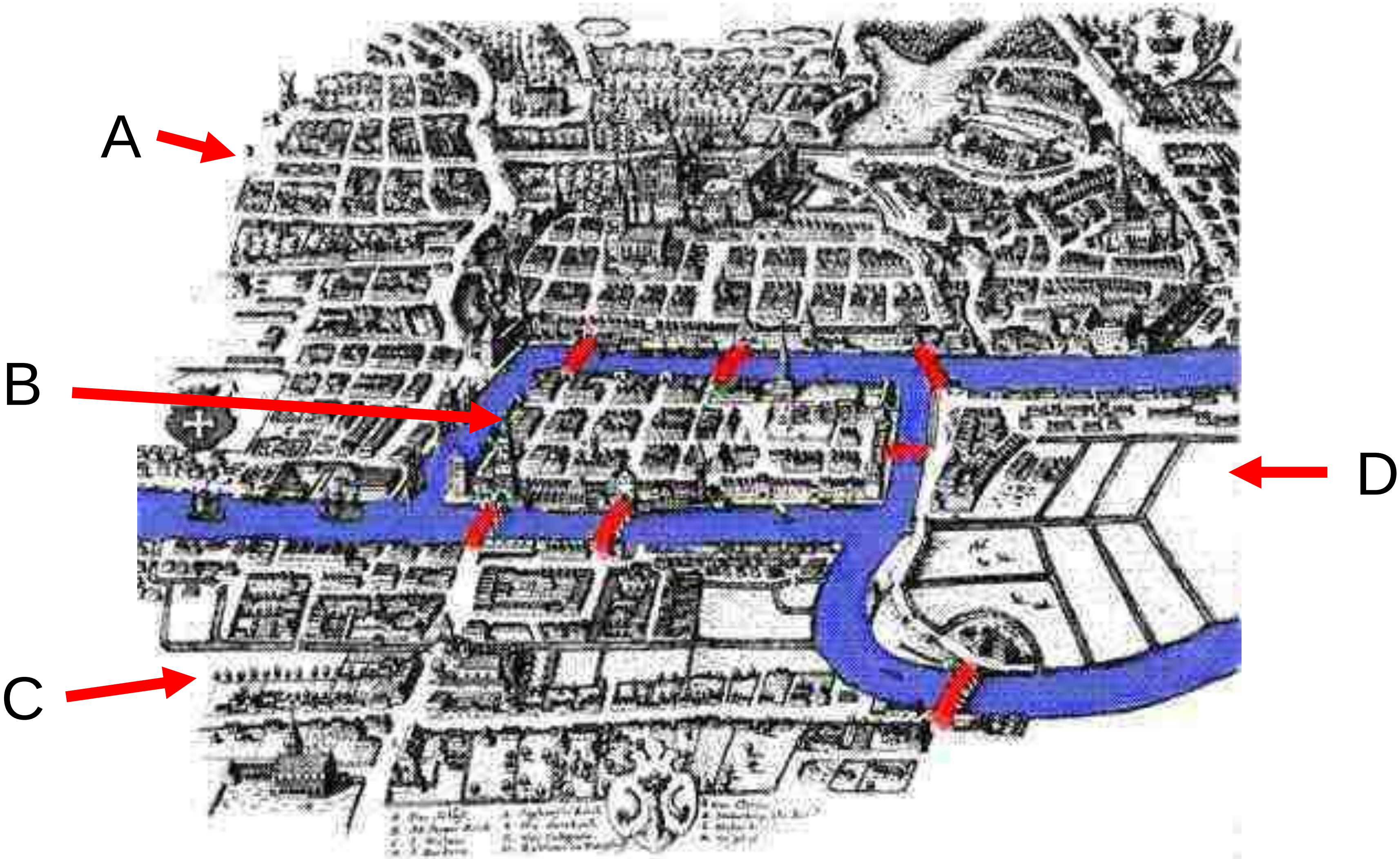
In 2013 Harvey Keitel has become the center of the Hollywood Universe. Connery is 136th.

Bacon has moved up to 370$^{th.}$

An Early Problem in Graph Theory

- Leonhard Euler (1707 - 1783)
  - One of the first mathematicians to study graphs
- The Seven Bridges of Konigsberg Problem
  - Konigsberg is now called Kaliningrad
- A puzzle for the residents of the city
- The river Pregel flows through the city
- 7 bridges crossed the river
- Can you cross all bridges while crossing each bridge only once? *An Eulerian Circuit*

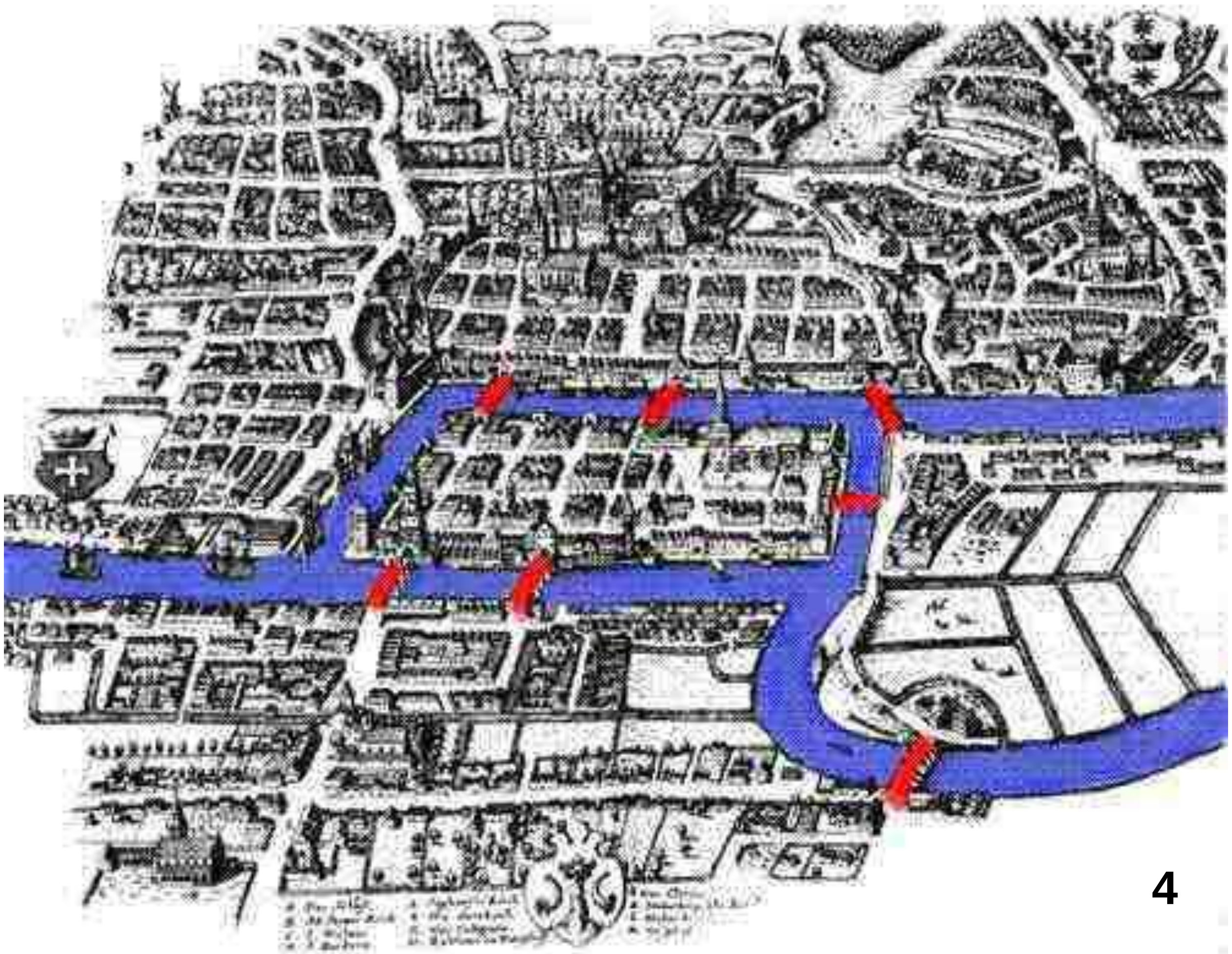
Konigsberg and the River Pregel



Clicker 1

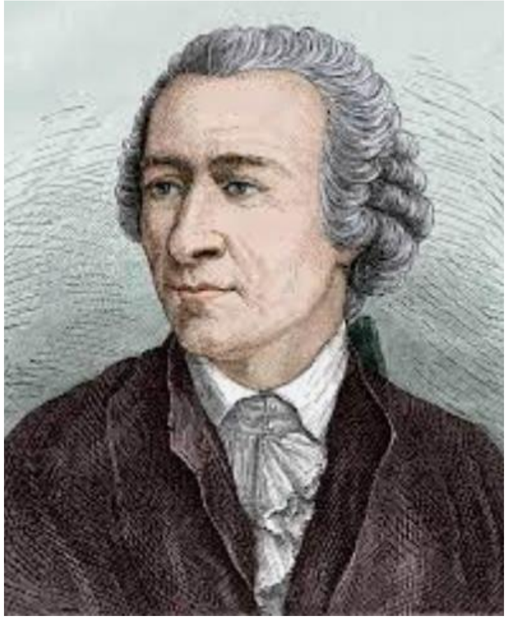
- How many solutions does the Seven Bridges of Konigsberg Problem have?

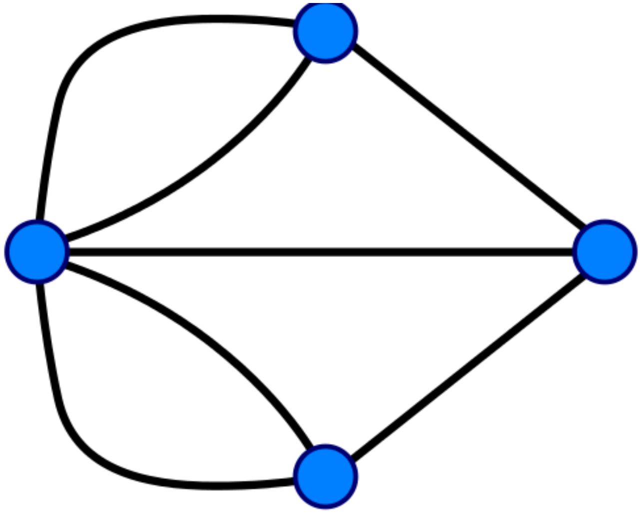
A. 0
B. 1
C. 2
D. 3
E. >= 4

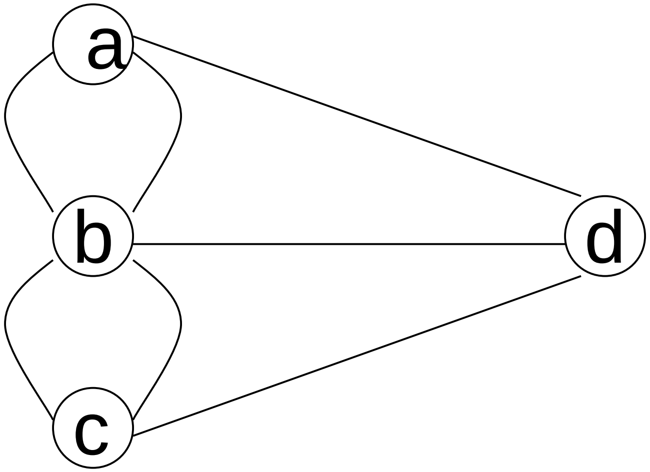


How to Solve

- Brute Force?
- Euler's Solution
  - Redraw the map as a graph (really a *multigraph as opposed to a simple graph, 1 or 0 edges per pair of vertices*)







Euler's Proposal

- A connected graph has an Euler tour (cross every edge exactly one time and end up at starting node) if and only if every vertex has an even number of edges
  - *Eulerian Circuit*
- Clicker 2 - What if we reduce the problem to only crossing each edge (bridge) exactly once?
  - Doesn't matter if we end up where we started
  - *Eulerian Trail*

  A. 0 B. 1 C. 2 D. 3 E. >= 4

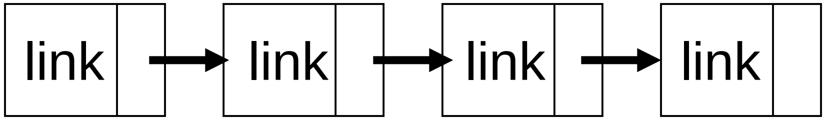
Graph Definitions

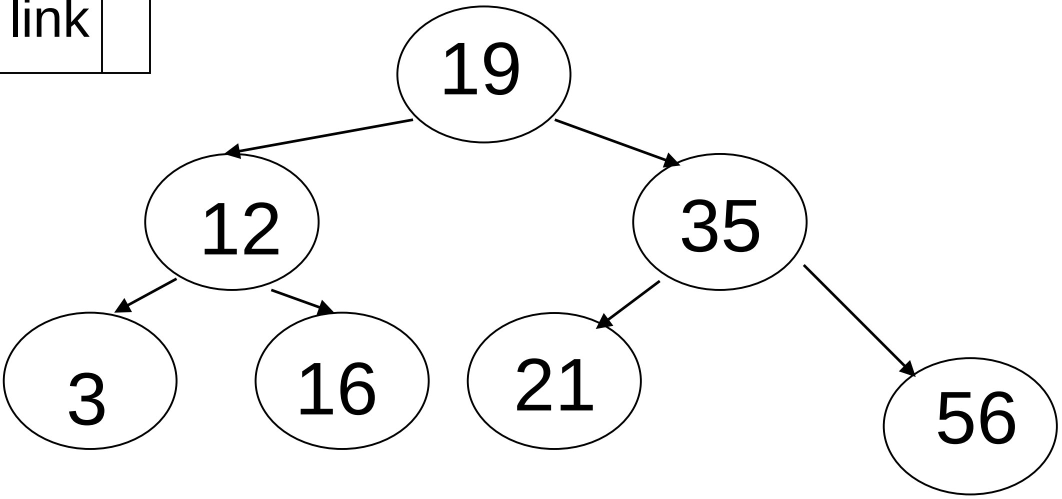
- A graph is comprised of a set of *vertices* (nodes) and a set of *edges* (links, arcs) connecting the vertices
  – An edge connects 2 vertices
- in a *directed* graph edges are one-way
  – movement allowed from first node to second, but not second to first
  – directed graphs also called *digraphs*
- in an *undirected* graph edges are two-way
  – movement allowed in either direction

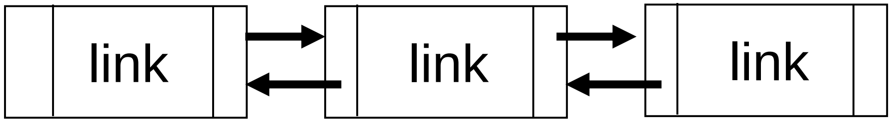
Definitions

- In a *weighted* graph the edge has cost or weight that measures the cost of traveling along the edge
- A *path* is a sequence of vertices connected by edges
  - The *path length* is the number of edges
  - The *weighted* path length is the sum of the cost of the edges in a path
- A *cycle* is a path of length 1 or more that starts and ends at the same vertex without repeating any other vertices
  - a *directed acyclic graph* is a directed graph with no cycles

Graphs We've Seen

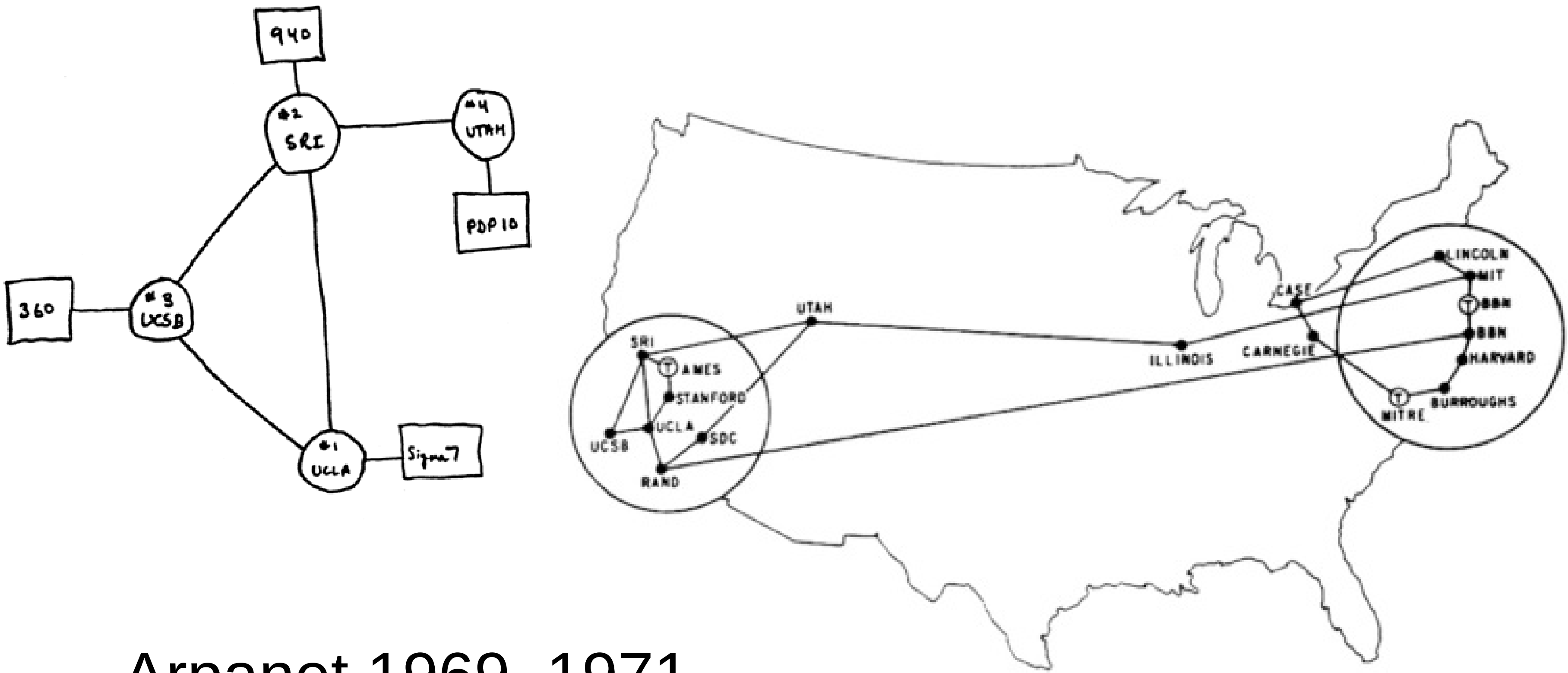






Example Graph

- Scientists (and academics of ALL kinds) use graphs to model all kinds of things.

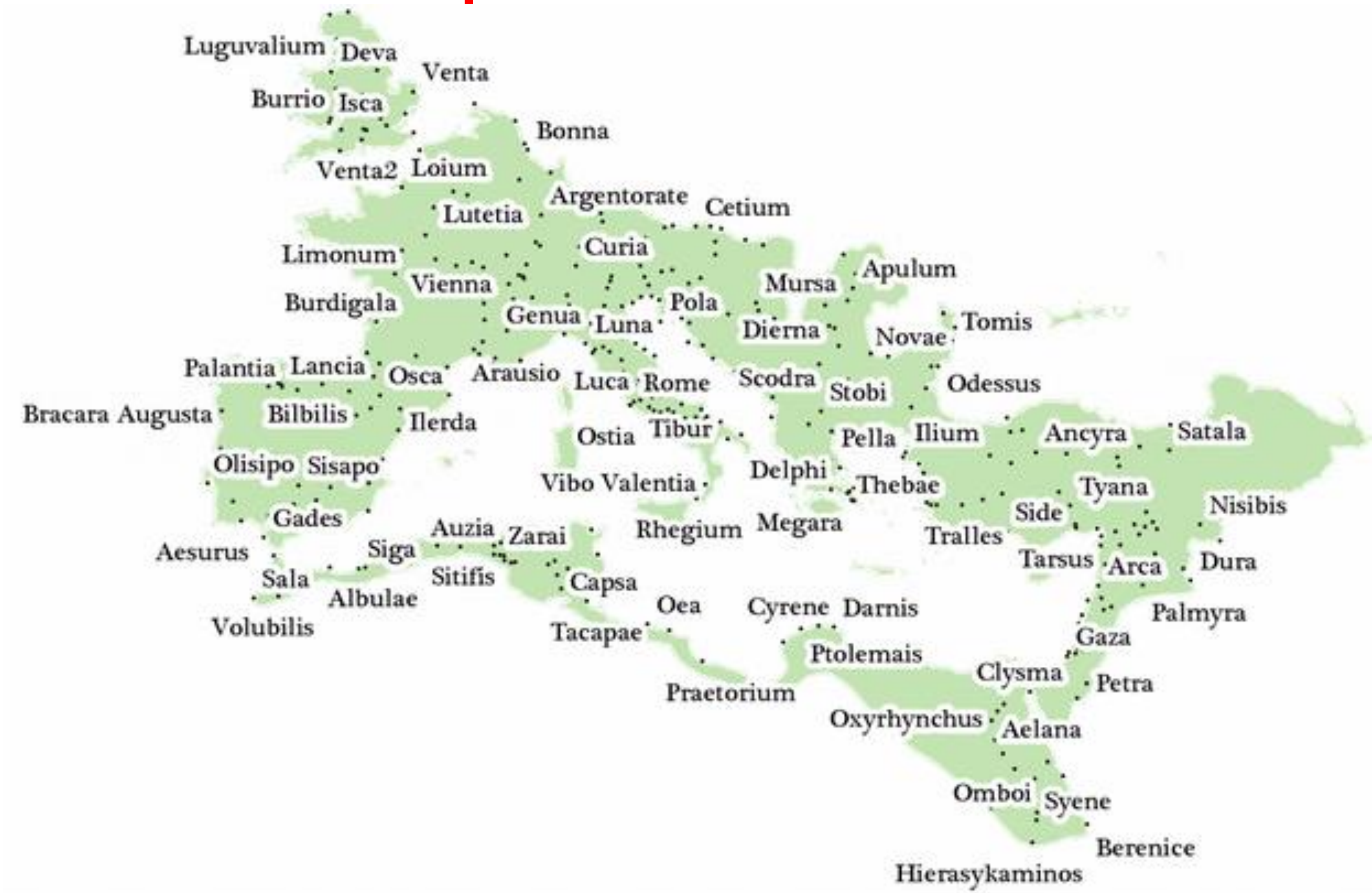


Arpanet 1969, 1971

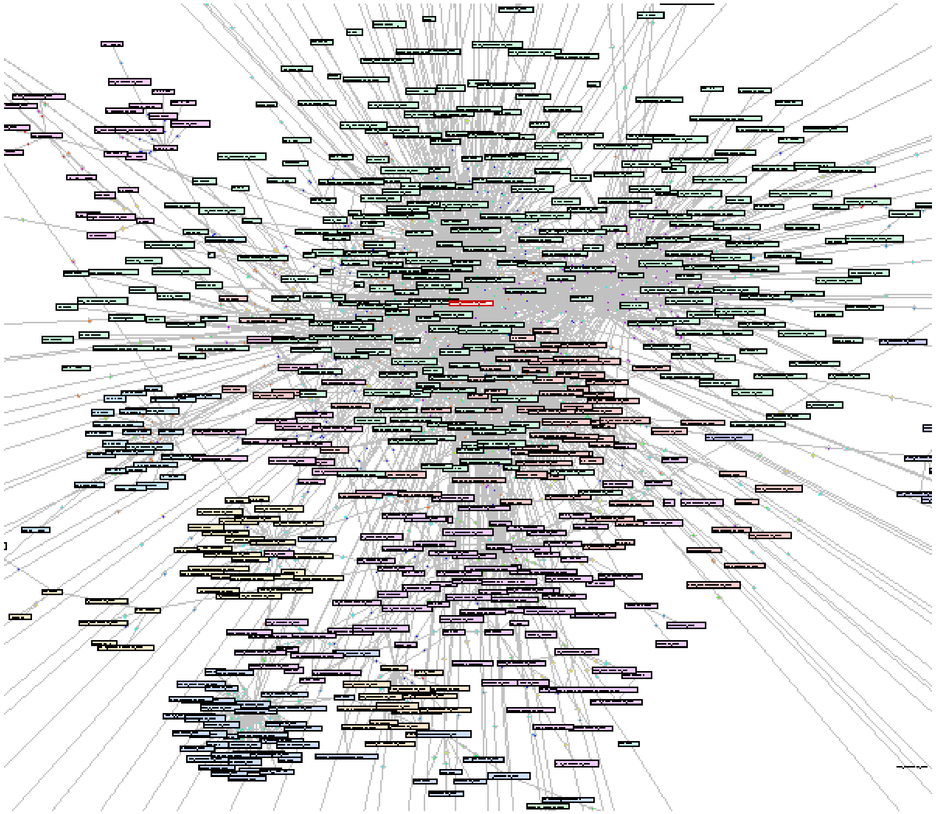
Example Graph

Roman Transportation Network

Roman Transportation Network

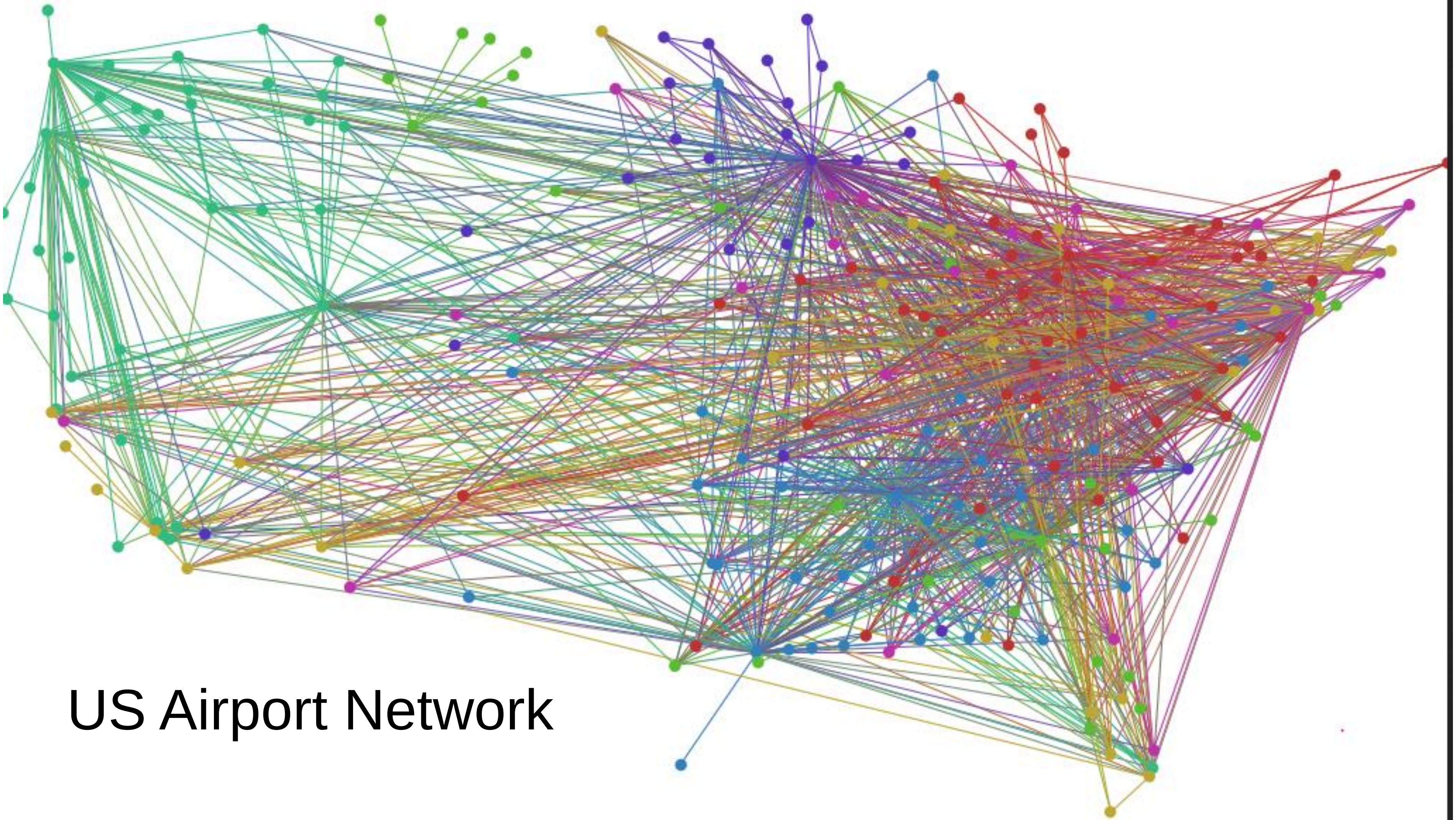


Example Graph



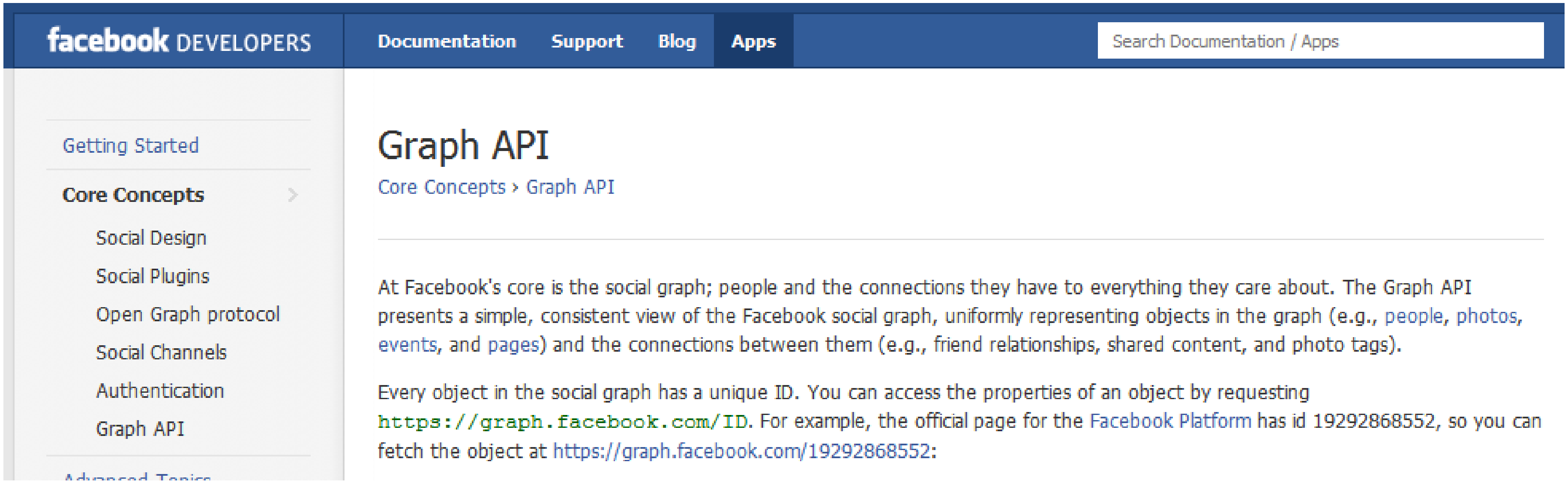
[Enron emails 2001]

Example Graph



US Airport Network

Example Graph



facebook DEVELOPERS | Documentation | Support | Blog | Apps | Search Documentation / Apps

Getting Started

Core Concepts

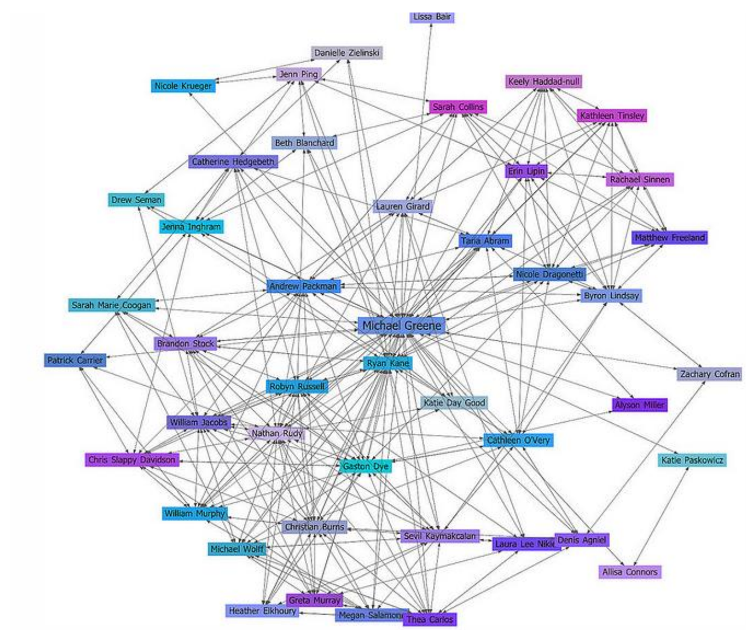
- Social Design
- Social Plugins
- Open Graph protocol
- Social Channels
- Authentication
- Graph API

Graph API

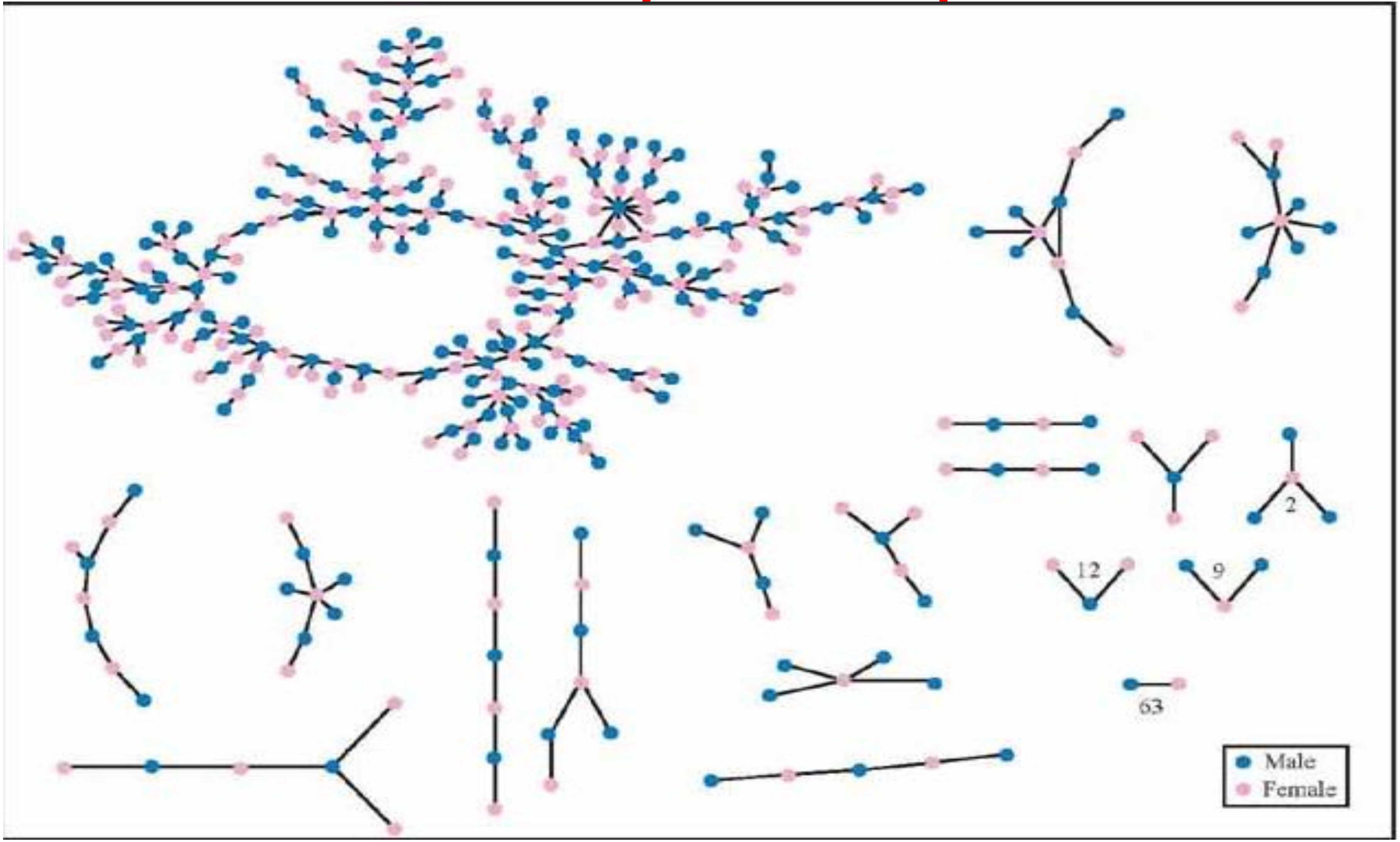
Core Concepts › Graph API

At Facebook's core is the social graph; people and the connections they have to everything they care about. The Graph API presents a simple, consistent view of the Facebook social graph, uniformly representing objects in the graph (e.g., people, photos, events, and pages) and the connections between them (e.g., friend relationships, shared content, and photo tags).

Every object in the social graph has a unique ID. You can access the properties of an object by requesting `https://graph.facebook.com/ID`. For example, the official page for the Facebook Platform has id 19292868552, so you can fetch the object at https://graph.facebook.com/19292868552:



Example Graph



"Jefferson" High School, Ohio Chains of Affection: The Structure of Adolescent Romantic and Sexual Networks, 2005,

Representing Graphs

- How to store a graph as a data structure?
-

Adjacency Matrix Representation

	A	Br	Bl	Ch	Co	E	FG	G	Pa	Pe	S	U	V
A	0	1	1	1	0	0	0	0	1	0	0	1	0
Br	1	0	1	0	1	0	1	1	1	1	1	1	1
Bl	1	1	0	1	0	0	0	0	1	1	0	0	0
Ch	1	0	1	0	0	0	0	0	0	1	0	0	0
Co	0	1	0	0	0	1	0	0	0	1	0	0	1
E	0	0	0	0	1	0	0	0	0	1	0	0	0
FG	0	1	0	0	0	0	0	0	0	0	1	0	0
G	0	1	0	0	0	0	0	0	0	0	1	0	1
Pa	1	1	1	0	0	0	0	0	0	0	0	0	0
Pe	0	1	1	1	1	1	0	0	0	0	0	0	0
S	0	1	0	0	0	0	1	1	0	0	0	0	0
U	1	1	0	0	0	0	0	0	0	0	0	0	0
V	0	1	0	0	1	0	0	1	0	0	0	0	0

Country	Code
Argentina	A
Brazil	Br
Bolivia	Bl
Chile	Ch
Colombia	Co
Ecuador	E
French Guiana	FG
Guyana	G
Paraguay	Pa
Peru	Pe
Suriname	S
Uruguay	U
Venezuela	V

Undirected Graph?

- Use a ragged 2d array to save space

NORTH ISLAND DISTANCE CHART

	AUCKLAND	CAPE REINGA	DARGAVILLE	GISBORNE	HAMILTON	HICKS BAY	KAITAIA	MASTERTON	NAPIER	NEW PLYMOUTH	PAIHIA	PALMERSTON NORTH	ROTORUA	TAUPO	TAUMARUNUI	TAURANGA	THAMES	WAIKAREMOANA	WAITOMO	WHANGANUI	WELLINGTON	WHAKAPAPA VILLAGE	WHAKATANE
CAPE REINGA	436																						
DARGAVILLE	180	285																					
GISBORNE	499	935	684																				
HAMILTON	126	562	306	394																			
HICKS BAY	503	939	683	180	398																		
KAITAIA	320	116	169	819	446	823																	
MASTERTON	646	1082	826	448	520	628	966																
NAPIER	421	857	601	215	295	395	744	233															
NEW PLYMOUTH	357	797	537	585	231	739	677	343	412														
PAIHIA	236	223	129	735	362	739	107	882	657	593													
PALMERSTON NORTH	537	973	717	393	411	573	857	109	178	234	773												
ROTORUA	234	670	414	286	108	290	554	448	223	299	470	339											
TAUPO	278	714	458	332	152	370	598	368	143	296	514	259	80										
TAUMARUNUI	286	722	466	449	160	487	606	348	260	183	522	239	172	117									
TAURANGA	205	641	385	298	106	302	525	524	299	308	441	415	86	156	235								
THAMES	114	550	294	410	108	414	434	574	349	339	350	465	164	206	268	116							
WAIKAREMOANA	390	826	570	162	264	342	710	416	183	482	626	361	156	186	303	242	320						
WAITOMO	200	636	380	445	74	449	520	450	306	173	436	341	166	163	102	151	182	322					
WHANGANUI	457	894	637	467	331	643	777	183	252	160	693	74	309	229	171	439	479	435	273				
WELLINGTON	658	1094	838	538	532	718	978	102	323	355	894	145	460	380	371	546	586	506	473	195			
WHAKAPAPA VILLAGE	345	780	525	429	219	467	665	335	240	242	581	223	177	97	59	236	303	283	159	141	344		
WHAKATANE	298	737	478	201	193	205	618	533	308	384	534	424	85	165	257	97	209	241	235	358	545	262	
WHANGAREI	165	271	58	664	291	668	155	811	586	522	71	702	400	443	451	370	279	555	365	622	823	510	463

At a Glance

Auckland to Paihia [236 km; 3 hr, 20 min]

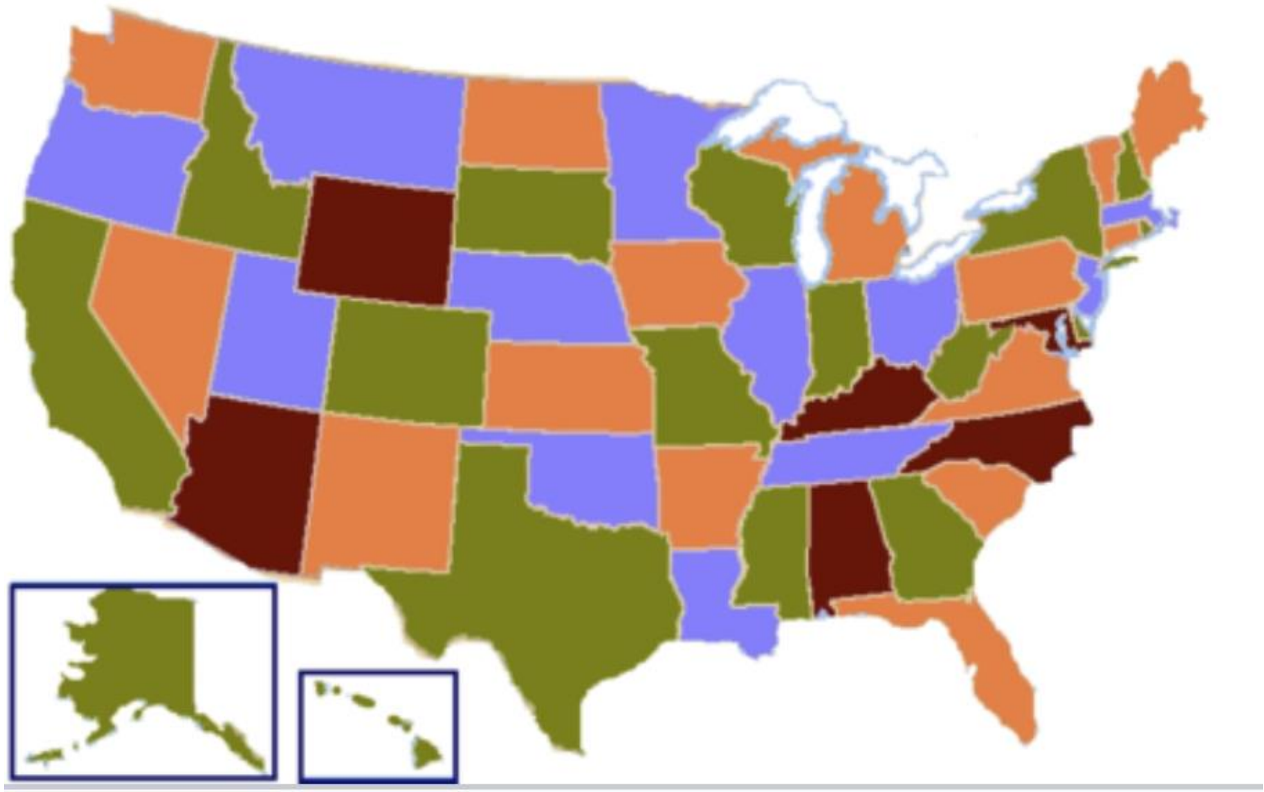
Auckland to Rotorua [234 km; 3 hr, 20 min]

Auckland to Wellington [658 km; 9 hr, 25 min]

The Map Coloring Problem

- How many colors do you need to color a map, so that no 2 countries that have a common border (not a point) are colored the same?
- How to solve using Brute Force?

Example



A four-coloring of a map of the states of the United States (ignoring lakes).

Source: https://en.wikipedia.org/wiki/Four_color_theorem

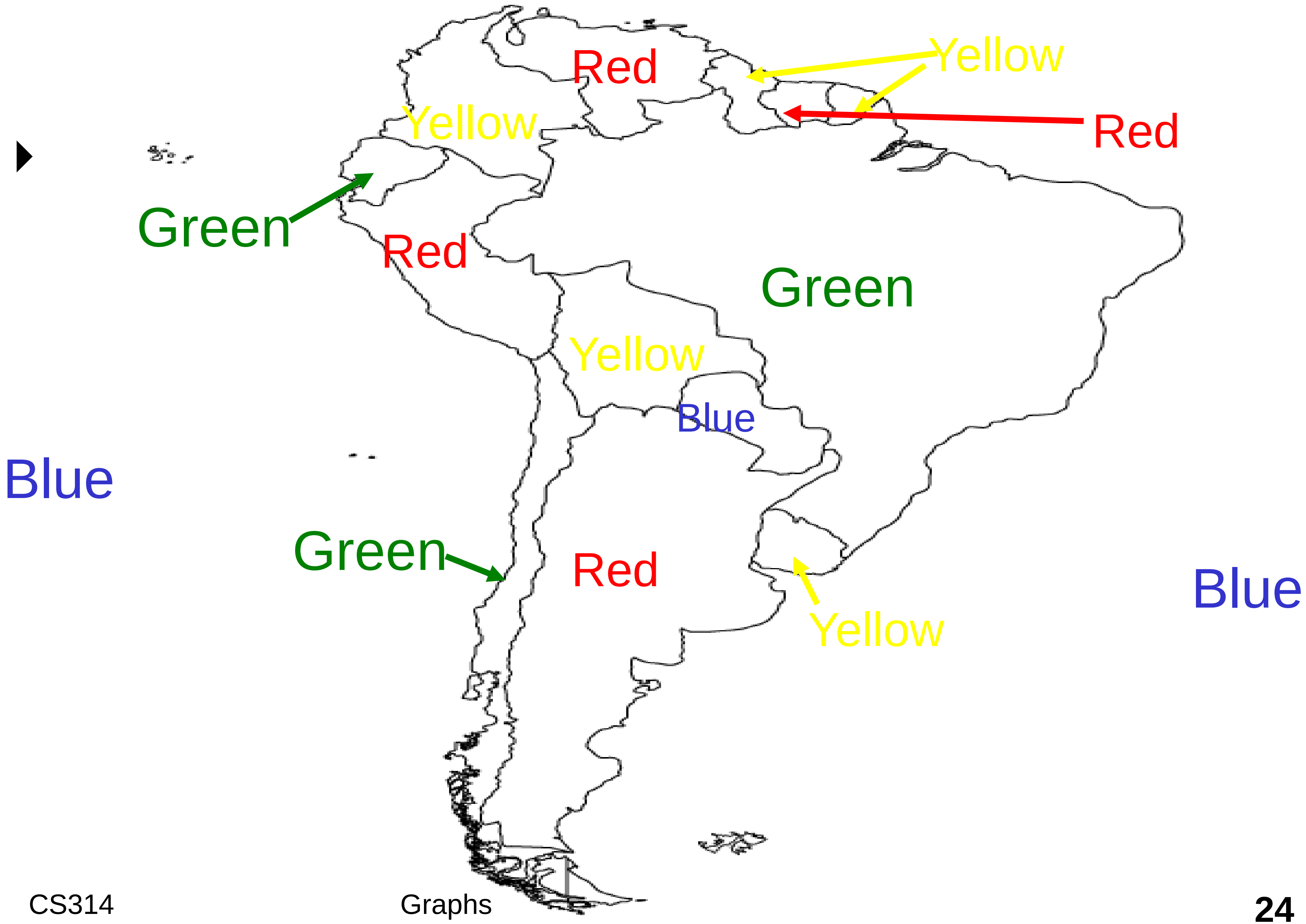
A Solution


Blue
Yellow
Blue
Yellow
Green
Blue
Green
Yellow
Red
Green
Blue
Yellow

What About the Ocean?

	A	Br	Bl	Ch	Co	E	FG	G	Pa	Pe	S	U	V	Oc
A	0	1	1	1	0	0	0	0	1	0	0	1	0	1
Br	1	0	1	0	1	0	1	1	1	1	1	1	1	1
Bl	1	1	0	1	0	0	0	0	1	1	0	0	0	0
Ch	1	0	1	0	0	0	0	0	0	1	0	0	0	1
Co	0	1	0	0	0	1	0	0	0	1	0	0	1	1
E	0	0	0	0	1	0	0	0	0	1	0	0	0	1
FG	0	1	0	0	0	0	0	0	0	0	1	0	0	1
G	0	1	0	0	0	0	0	0	0	0	1	0	1	1
Pa	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Pe	0	1	1	1	1	1	0	0	0	0	0	0	0	1
S	0	1	0	0	0	0	1	1	0	0	0	0	0	1
U	1	1	0	0	0	0	0	0	0	0	0	0	0	1
V	0	1	0	0	1	0	0	1	0	0	0	0	0	1
Oc	1	1	0	1	1	1	1	1	0	1	1	1	1	0

Make the Ocean Blue



More Definitions

- A *dense* graph is one with a "large" number of edges
  – maximum number of edges?
- A "*sparse"* graph is one in which the number of edges is "much less" than the maximum possible number of edges
  – No standard cutoff for dense and sparse graphs
  – Although a common one is more than half the possible edges

Graph Representation

- For dense graphs the adjacency matrix is a reasonable choice
  - For weighted graphs change booleans to double or int
  - Can the adjacency matrix handle directed graphs?
- Most graphs are sparse, not dense
- For sparse graphs an *adjacency list* is an alternative that uses less space
- Each vertex keeps a list of edges to the vertices it is connected to.

Graph Implementation

```
public class Graph
      private static final double INFINITY
                                = Double.MAX_VALUE;
      private Map<String, Vertex> vertices;

      public Graph() // create empty Graph

      public void addEdge(String source,
                  String dest, double cost)

      // find all paths from given vertex
      public void findUnweightedShortestPaths
                       (String startName)

      // called after findUnweightedShortestPath
      public void printPath(String destName)
```

Graph Class

- This Graph class stores vertices
- Each vertex has an adjacency list
  - what vertices does it connect to?
- shortest path method finds all paths from start vertex to every other vertex in graph
- after shortest path method called queries can be made for path length from start node to destination node

Vertex Class (nested in Graph)

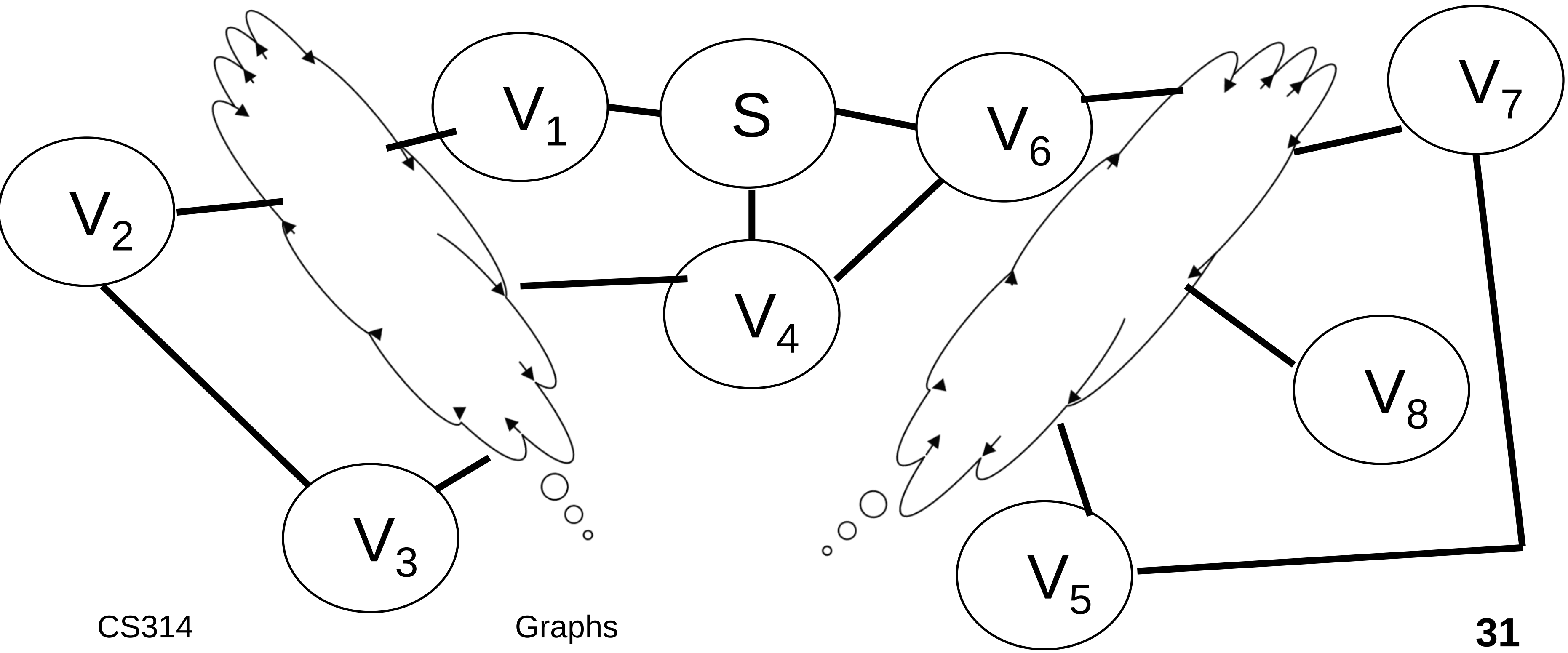
```
private static class Vertex
    private String name;
    private List<Edge> adjacent;

    public Vertex(String n)

    // for shortest path algorithms
    private double distance;
    private Vertex prev;
    private int scratch;

    // call before finding new paths
    public void reset()
```

Edge Class (nested in Graph)

```
private static class Edge
    private Vertex dest;
    private double cost;

    private Edge(Vertex d, double c)
```

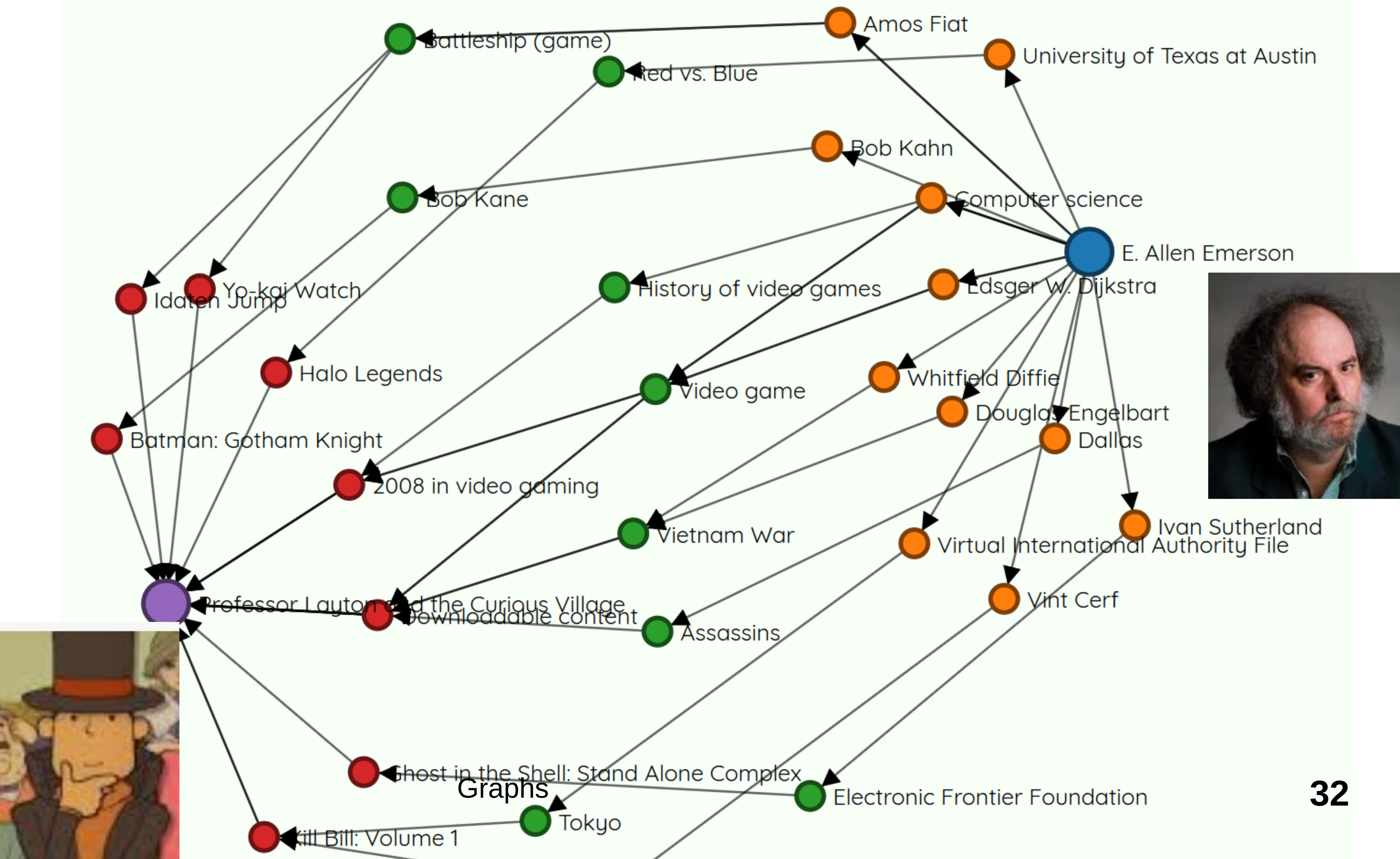
Unweighted Shortest Path

- Given a vertex, S (for start) find the shortest path from S to all other vertices in the graph
- Graph is unweighted (set all edge costs to 1)



6 Degrees of Wikipedia

- https://www.sixdegreesofwikipedia.com/



Word Ladders

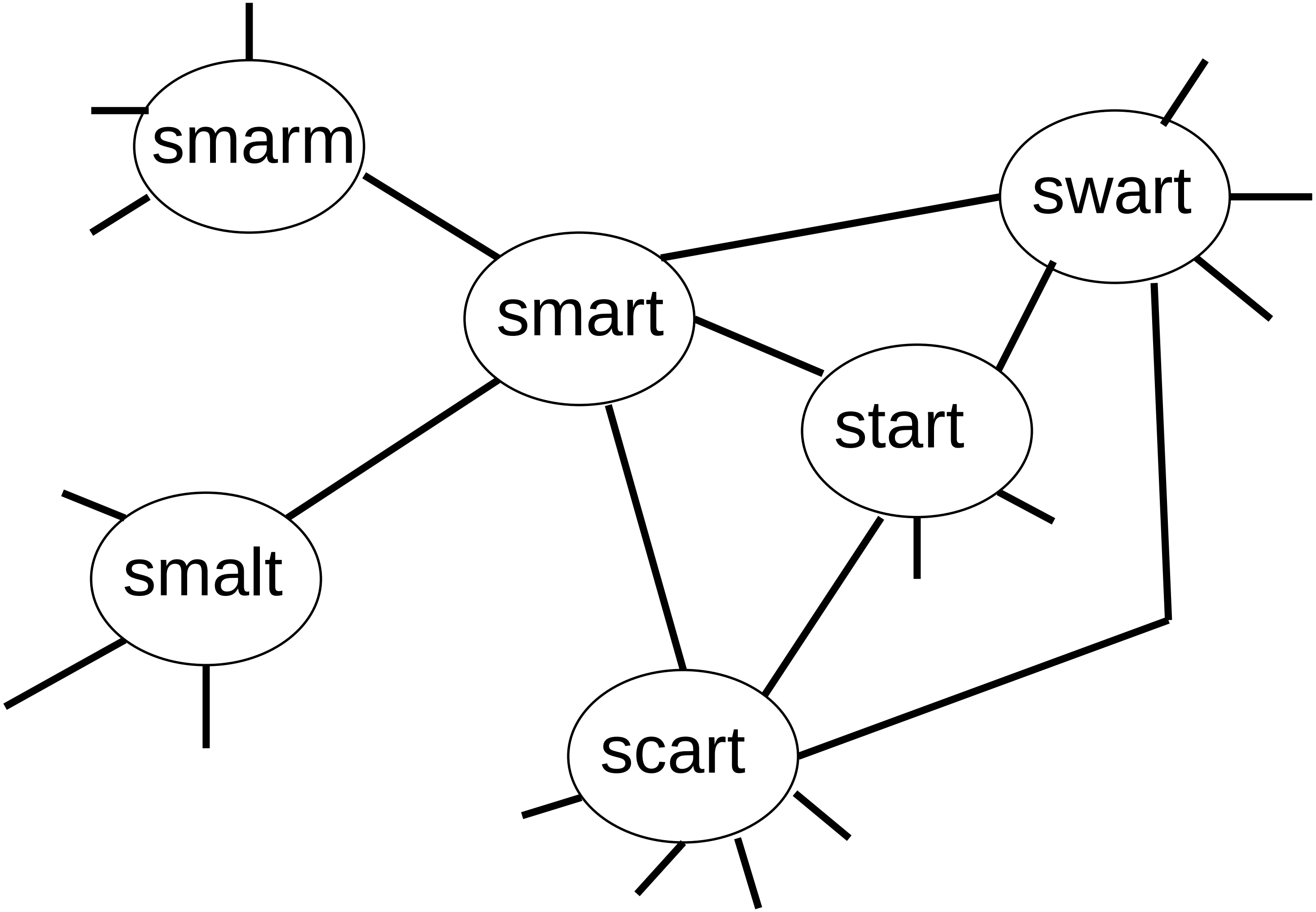
- Agree upon dictionary
- Start word and end word of same length
- Change one letter at a time to form step
- Step must also be a word
- Example: Start = silly, end = funny

```
silly
sully
sulky
hulky
hunky
funky
funny
```

Clicker 3 - Graph Representation

- What are the vertices and when does an edge exist between two vertices?

	Vertices	Edges
A.	Letters	Words
B.	Words	Words that share one or more letters
C.	Letters	Words that share one or more letters
D.	Words	Words that differ by one letter
E.	Words	Letters



Portion of Graph

Clicker 4 - Size of Graph

- Number of vertices and edges depends on dictionary
- Modified Scrabble dictionary, 5 letter words
- Words are vertices
  - 8660 words, 7915 words that are one letter different from at least one other word
- Edge exists between words if they are one letter different
  - 24,942 edges

Is this graph sparse or dense?

A. Sparse

B. Dense

Max number of edges =
N * (N - 1) / 2
37,493,470

Clicker 5 - Unweighted Shortest Path Algorithm

- Problem: Find the **shortest word ladder** between two words if one exists
- What kind of search should we use?

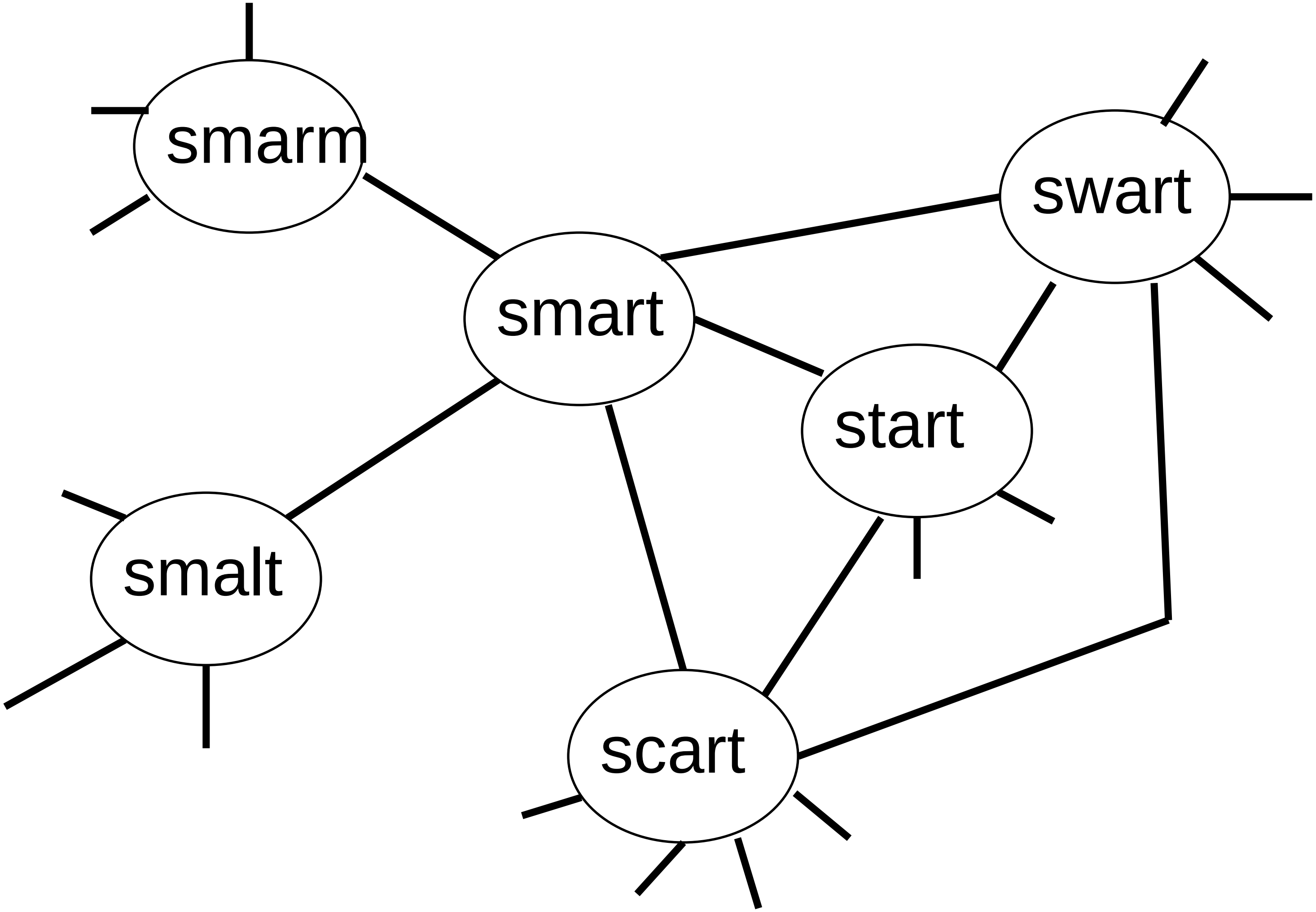
A. Breadth First Search

B. Depth First Search

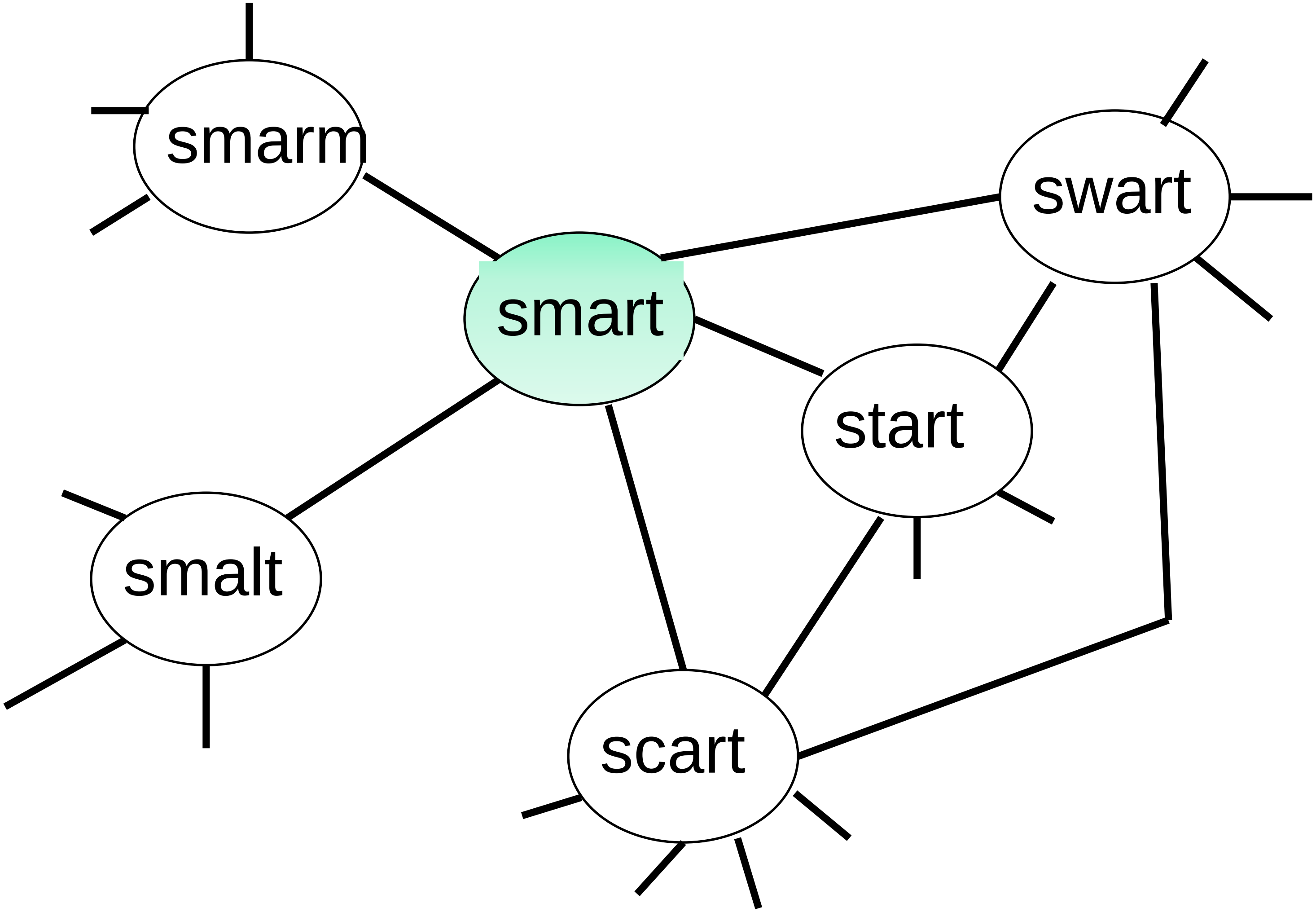
C. Either one

Unweighted Shortest Path Algorithm

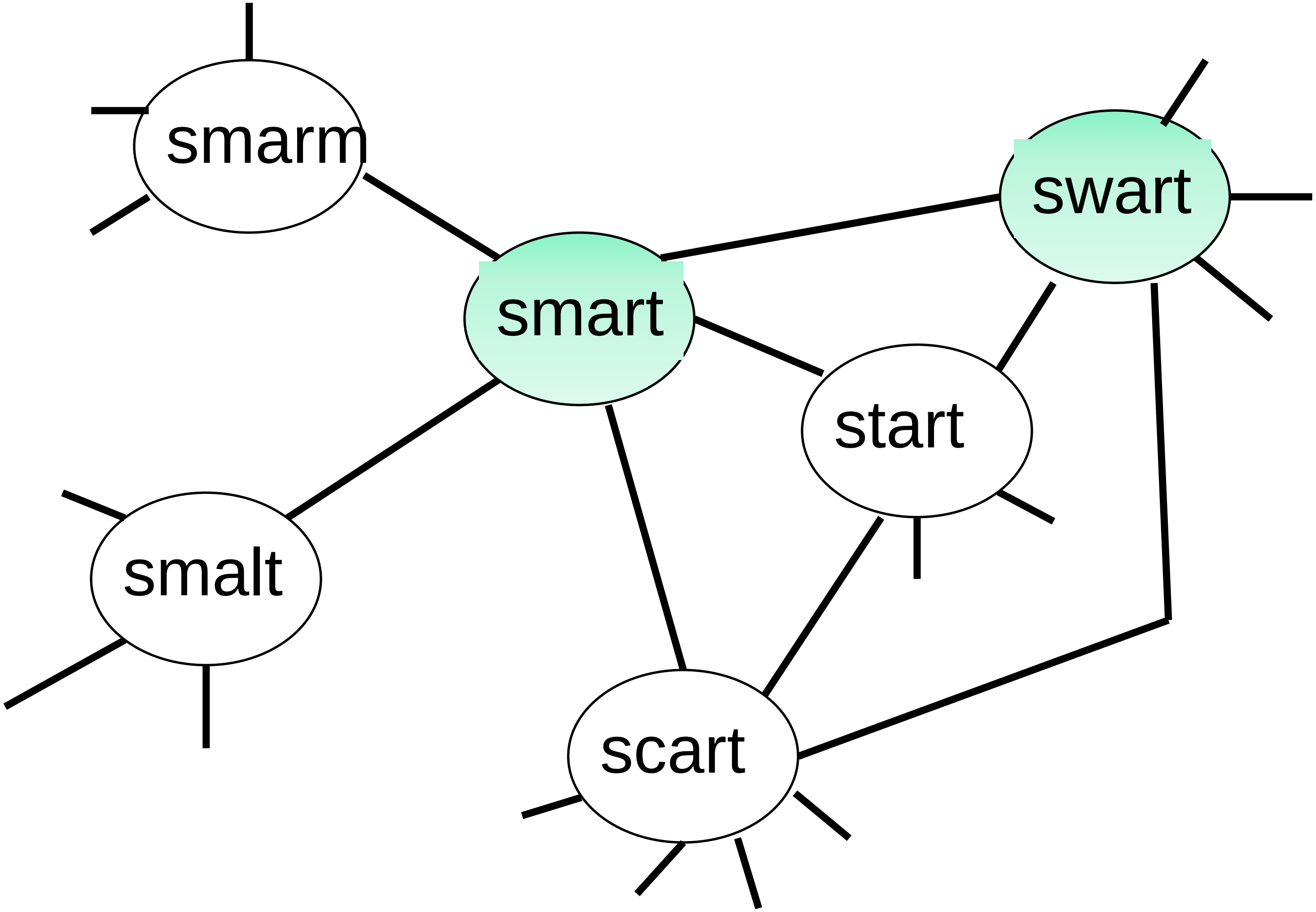
- Set distance of start to itself to 0
- Create a queue and add the start vertex
- while the queue is not empty
  - – remove front
  - – loop through all edges of current vertex
    - get vertex edge connects to
    - if this vertex has not been visited (have not found path to the destination of the edge)
      - – sets its distance to current distance + 1
      - – sets its previous vertex to current vertex
      - – add new vertex to queue



Portion of Graph

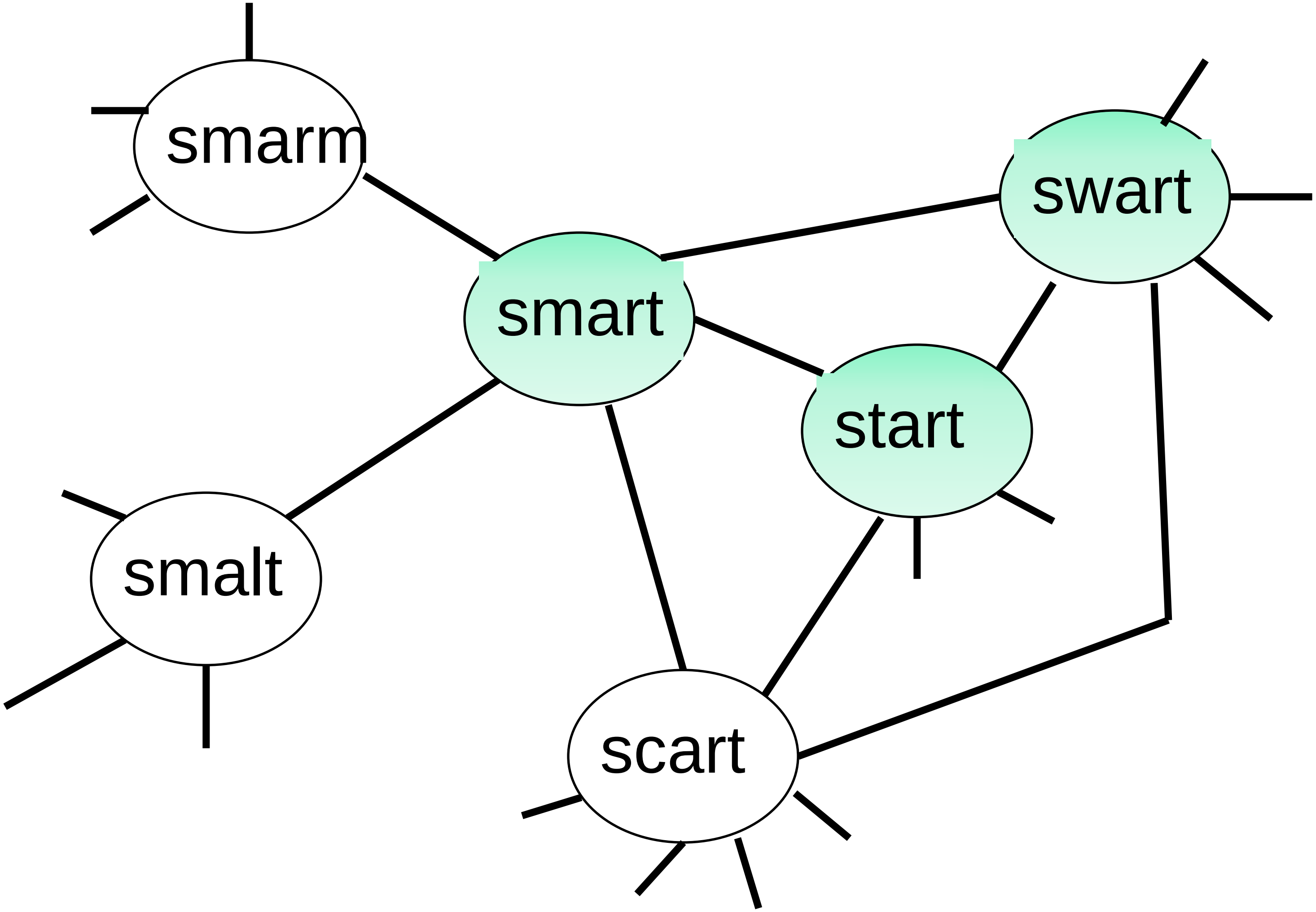


Start at "smart" and enqueue it
[smart]



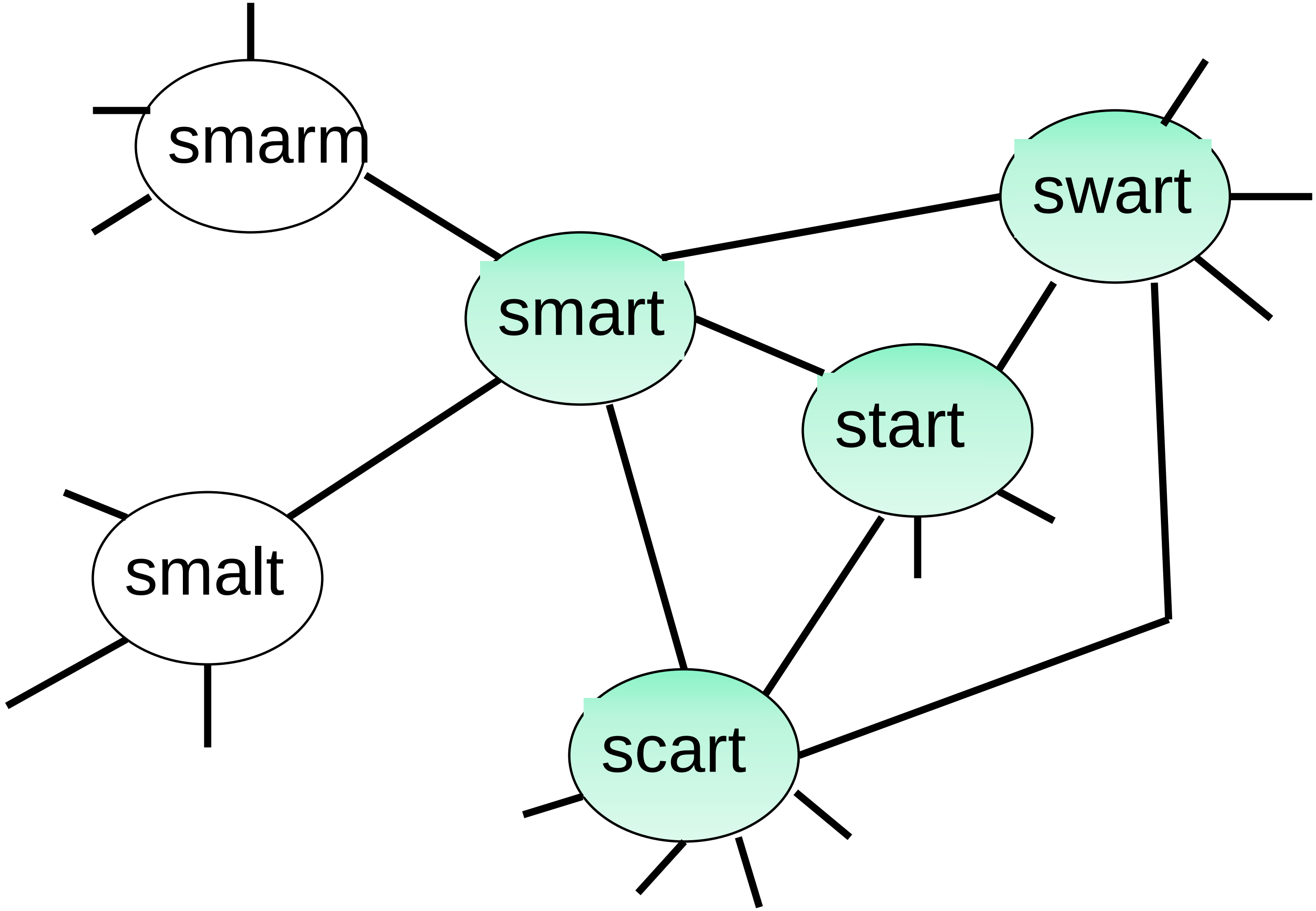
Dequeue (smart), loop through edges

[swart]



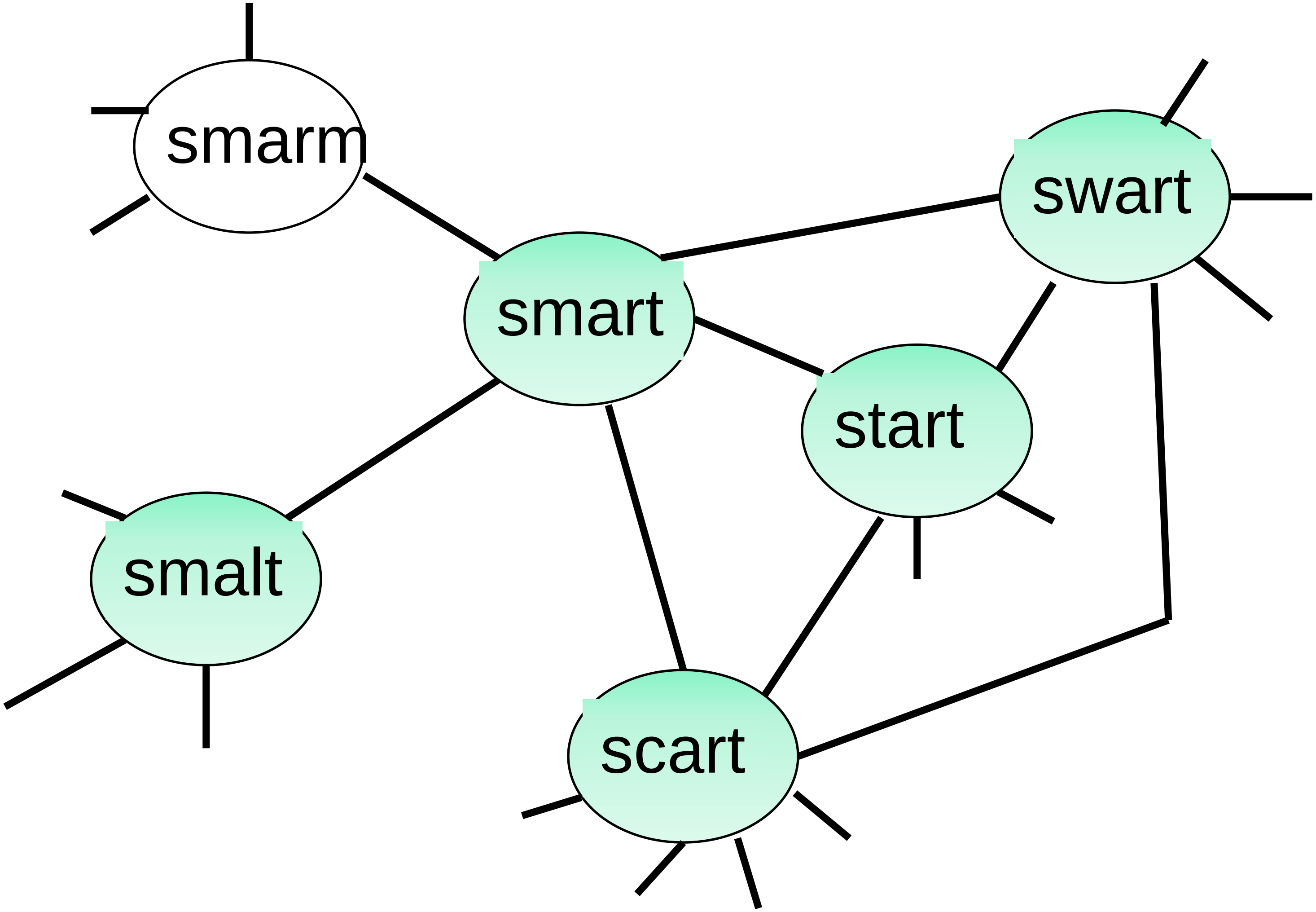
Dequeue (smart), loop through edges

[swart, start]



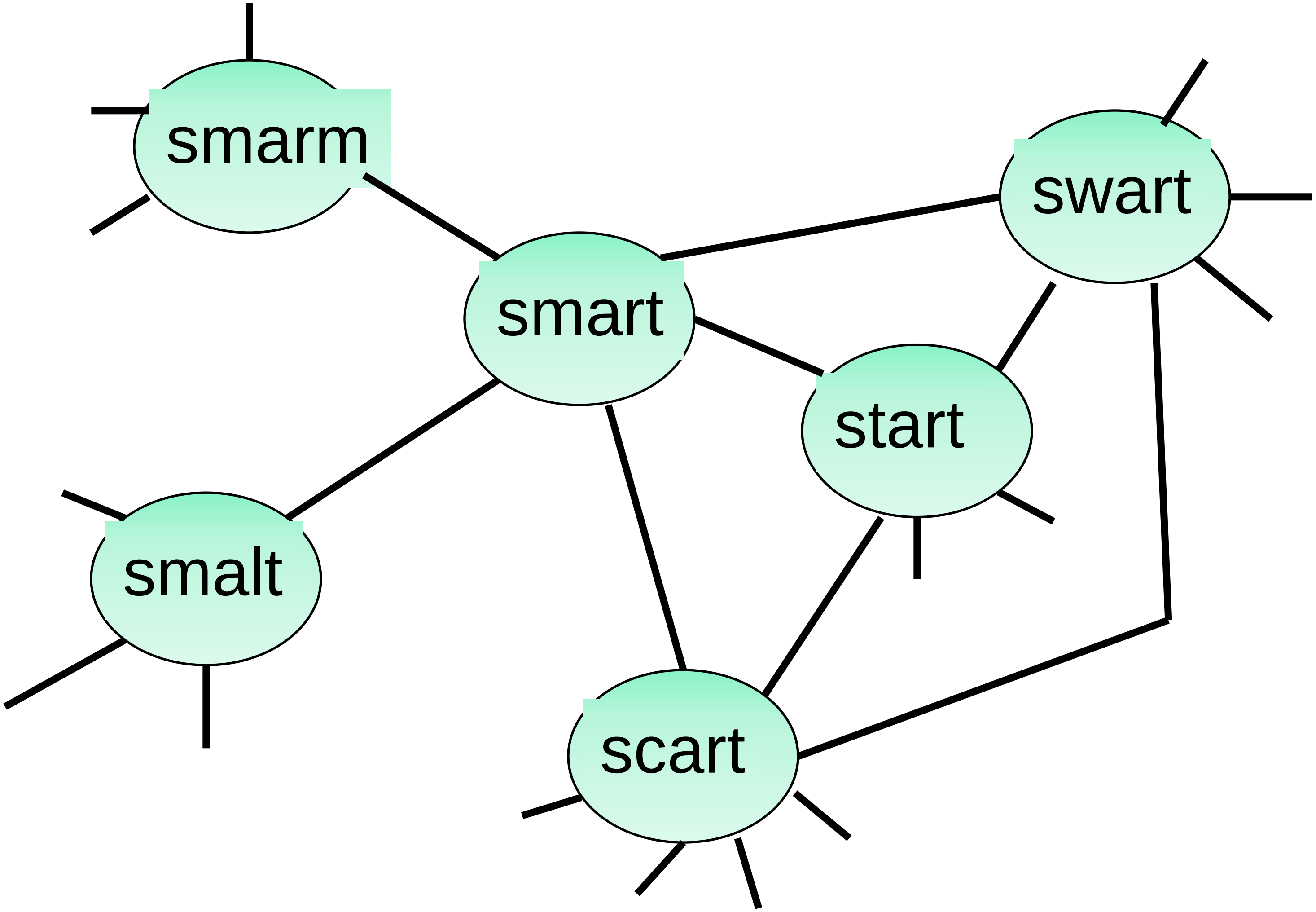
Dequeue (smart), loop through edges

[swart, start, scart]



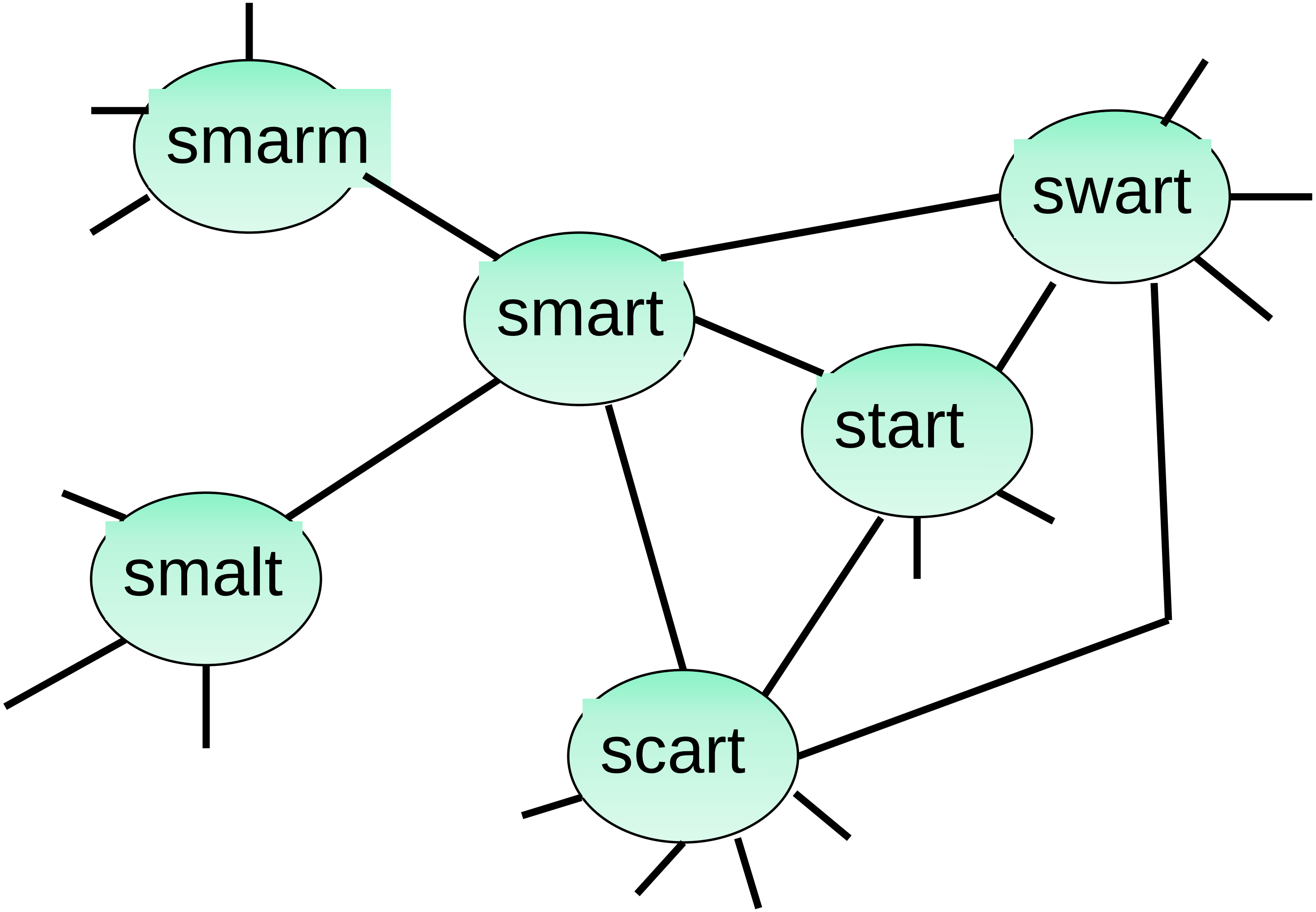
Dequeue (smart), loop through edges

[swart, start, scart, smalt]



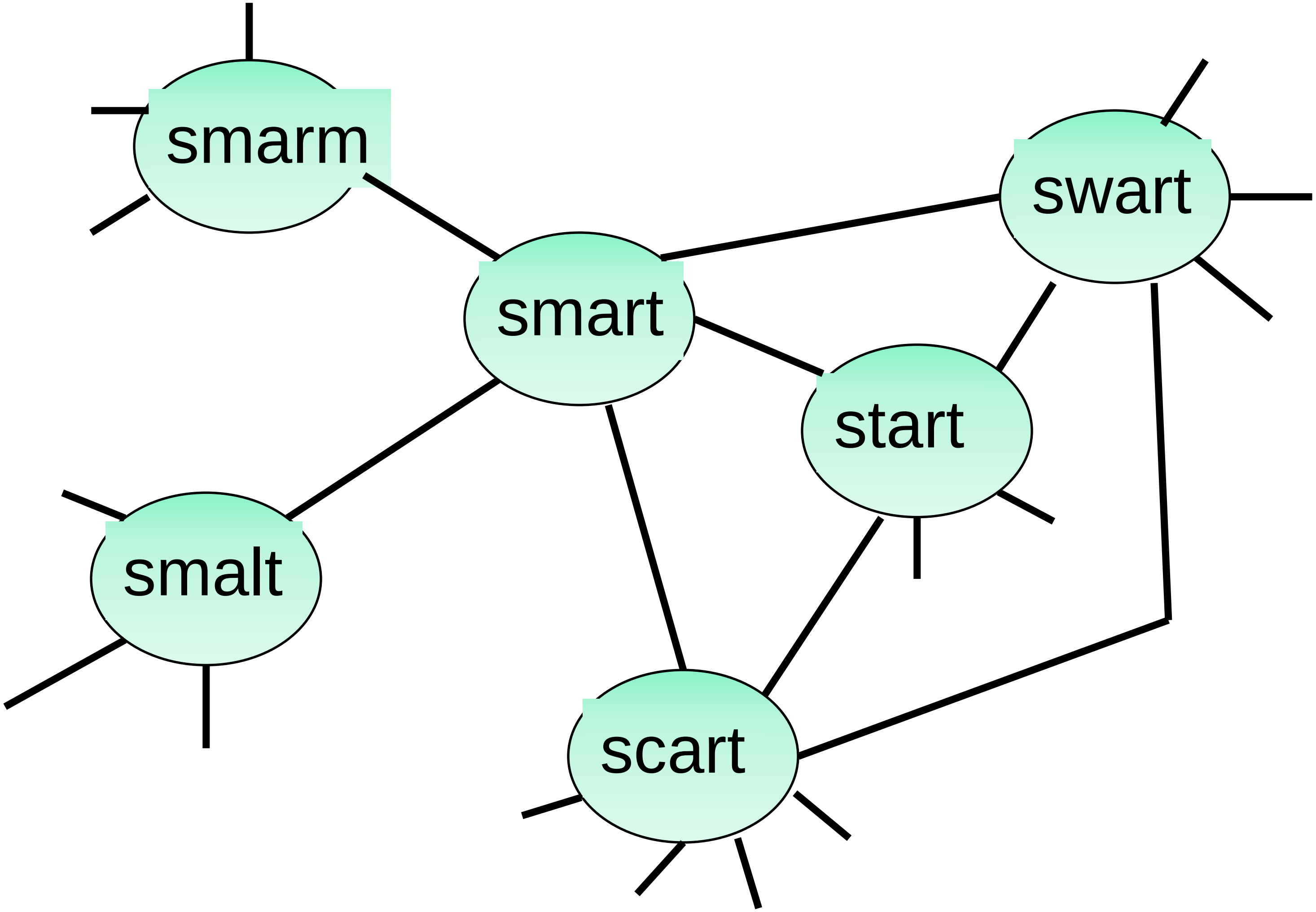
Dequeue (smart), loop through edges

[swart, start, scart, smalt, smarm]



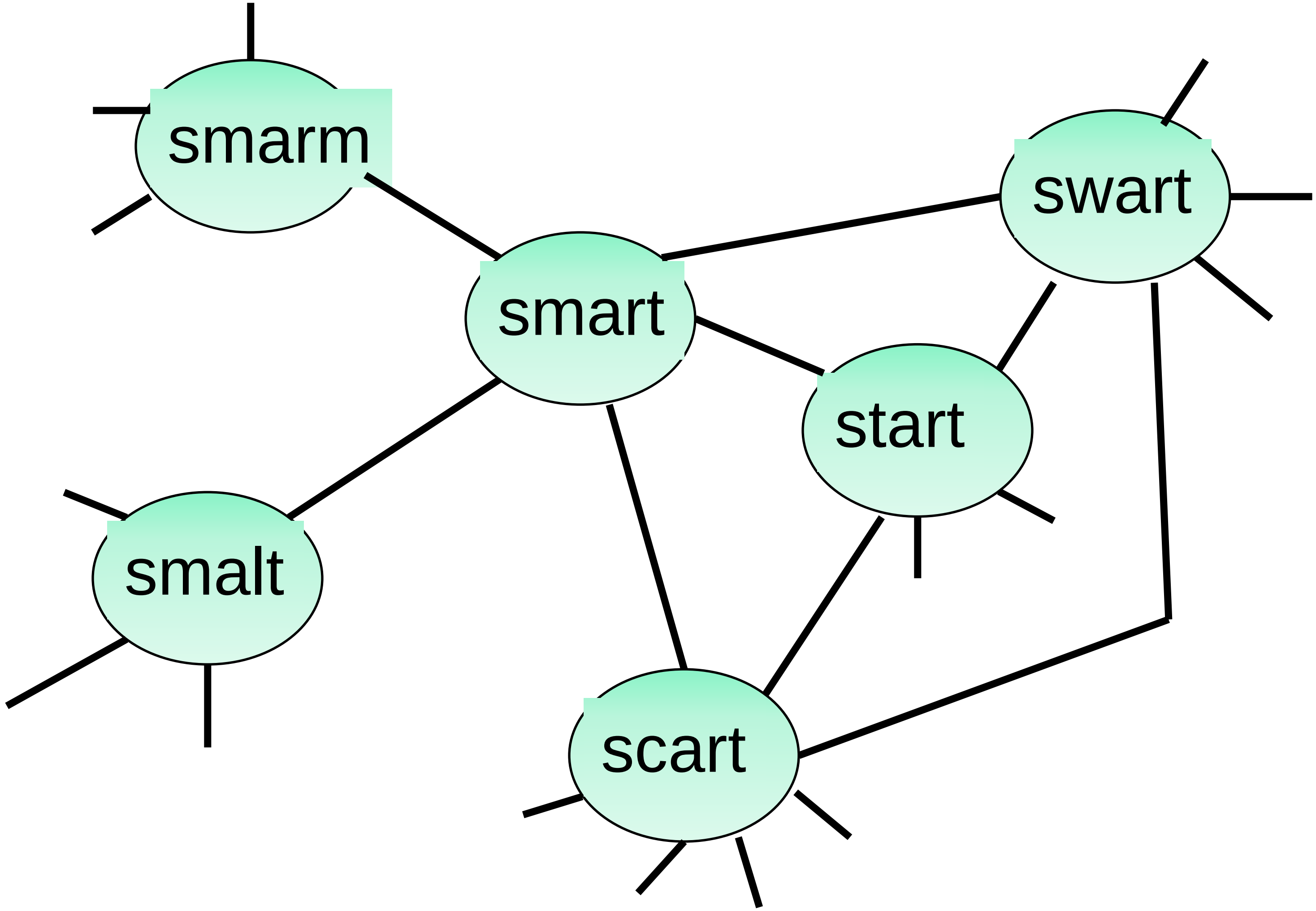
Done with smart, dequeue (swart)

[start, scart, smalt, smarm]



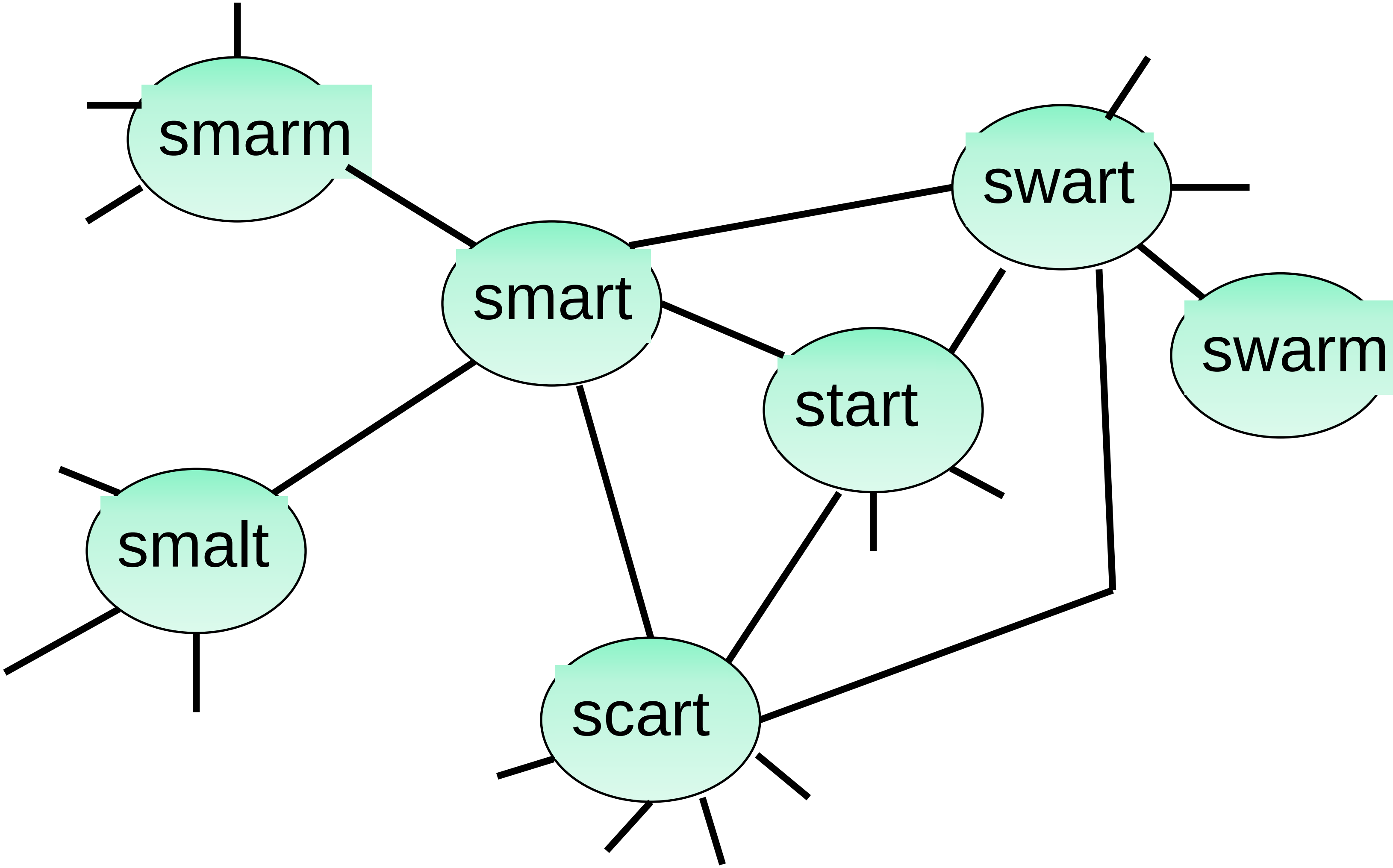
loop through edges of swart (start already present)

[start, scart, smalt, smarm]



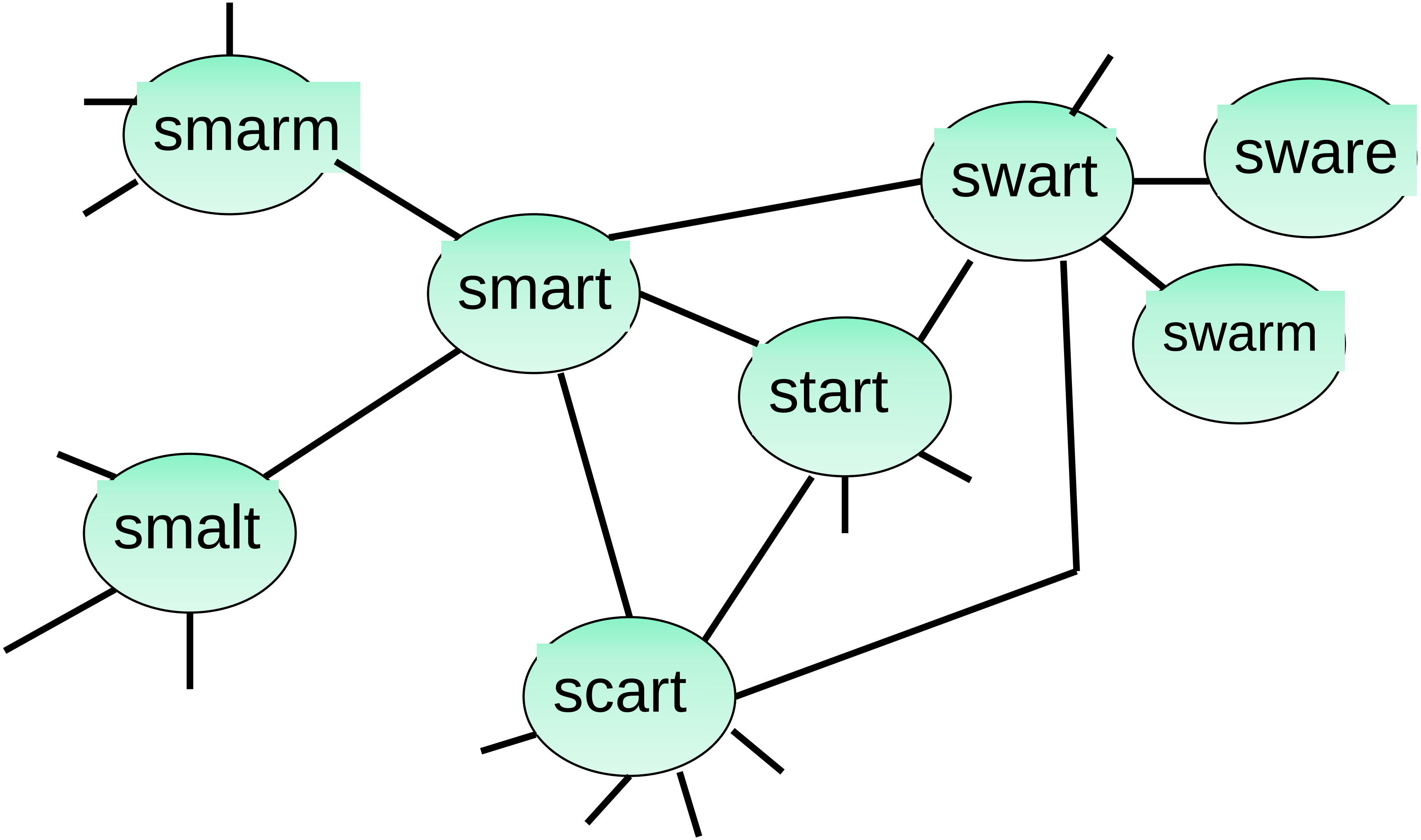
loop through edges of swart (scart already present)

[start, scart, smalt, smarm]



loop through edges of swart

[start, scart, smalt, smarm, swarm]



loop through edges of swart

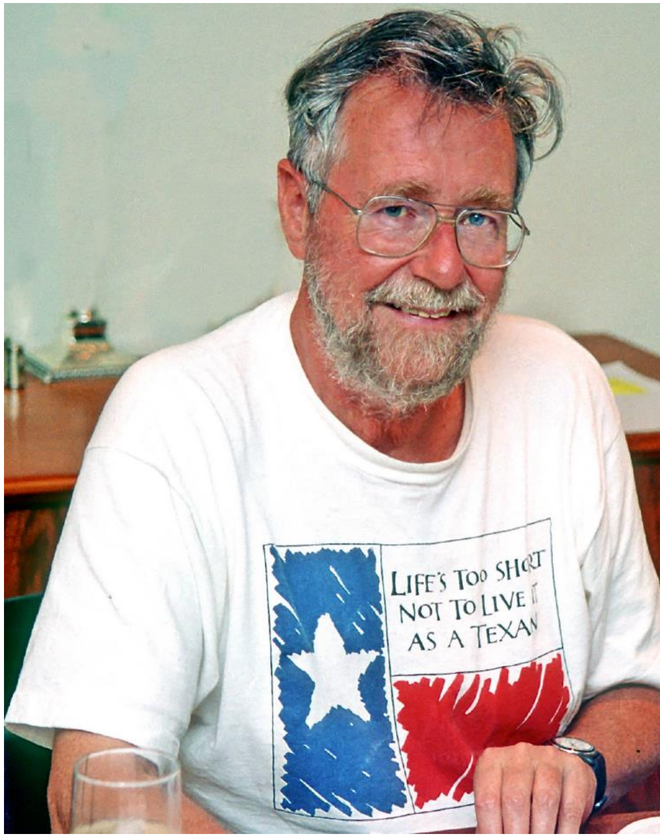
[start, scart, smalt, smarm, swarm, sware]

Unweighted Shortest Path

- Implement method
- demo
- how is path printed?
- The *diameter* of a graph is the longest shortest past in the graph
- How to find?
- How to find *center* of graph?
  - many measures of centrality
  - ours: vertex connected to the largest number of other vertices with the shortest average path length

Positive Weighted Shortest Path

- Edges in graph are weighted and all weights are positive
- Similar solution to unweighted shortest path
- Dijkstra's algorithm
- Edsger W. Dijkstra, 1930–2002
- UT Professor 1984 - 2000
- Algorithm developed in 1956 and published in 1959.
  - other algorithms developed independently around this time



Dijkstra on Creating the Algorithm

- What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which **I designed in about twenty minutes**. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. **One of the reasons that it is so nice was that I designed it without pencil and paper**. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. **Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame.**
- — Edsger Dijkstra, in an interview with Philip L. Frana, Communications of the ACM, 2001 (wiki page on the algorithm)

Vertex Class (nested in Graph)

```
private static class Vertex
     private String name;
     private List<Edge> adjacent;

     public Vertex(String n)

     // for shortest path algorithms
     private double distance;
     private Vertex prev;
     private int scratch;

     // call before finding new paths
     public void reset()
```

Dijkstra's Algorithm

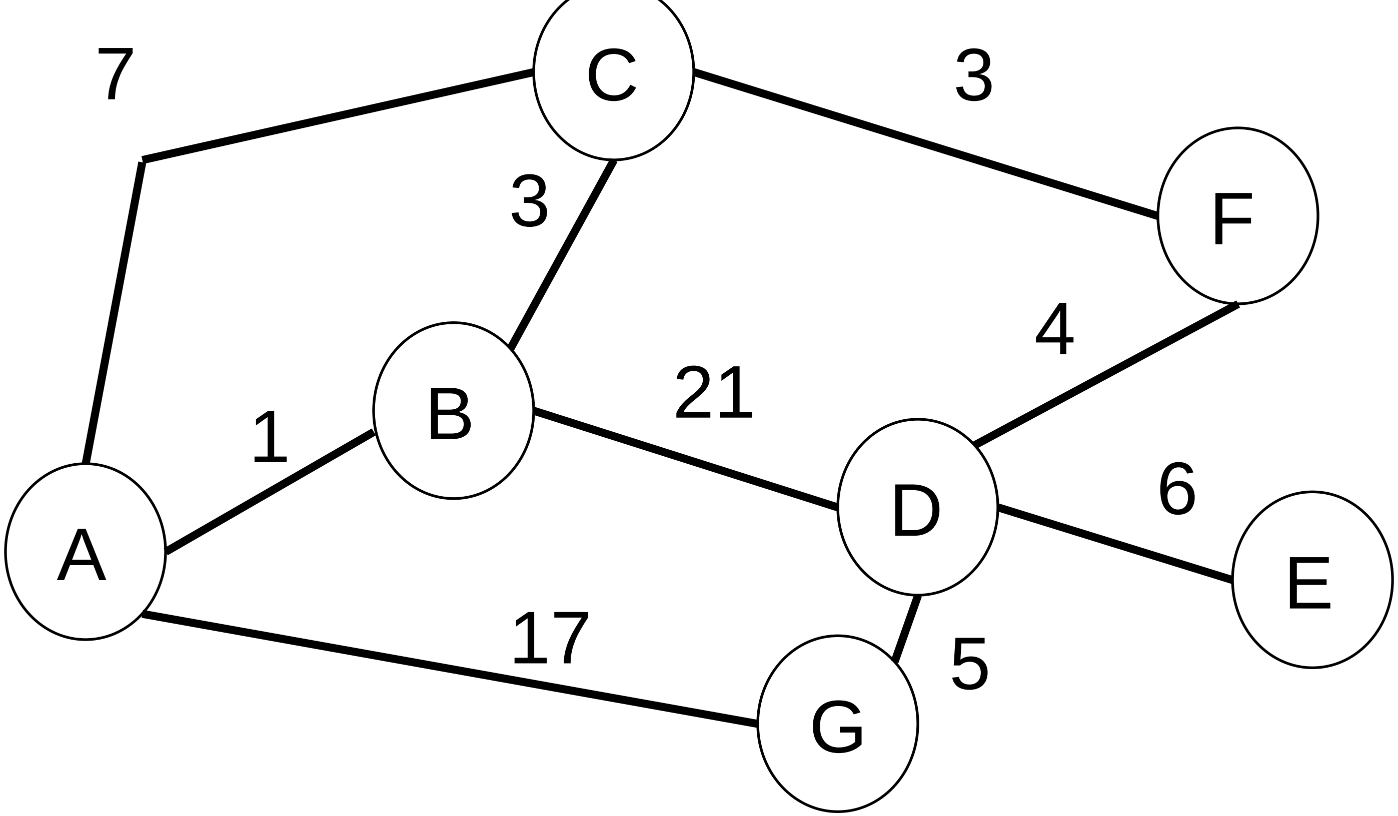
- Pick the start vertex
- Set the distance of the start vertex to 0 and all other vertices to INFINITY
- While there are unvisited vertices:
  - Let the current vertex be the vertex with the lowest cost path from start to it that has **not yet been visited**
  - mark current vertex as visited
  - for each edge from the current vertex
    - if the sum of the cost of the current vertex and the cost of the edge is less than the cost of the destination vertex
      - update the cost of the destination vertex
      - set the previous of the destination vertex to the current vertex
      - enqueue this path (not vertex) to the priority queue
      - **THIS IS NOT VISITING THE NEIGHBORING VERTEX**

Dijkstra's Algorithm

- Example of a *Greedy Algorithm*
  - A Greedy Algorithm does what appears to be the best thing at each stage of solving a problem
- Gives best solution in Dijkstra's Algorithm
- Does NOT always lead to best answer
- Fair teams:
  - (10, 10, 8, 8, 8), 2 teams
- Making change with fewest coins

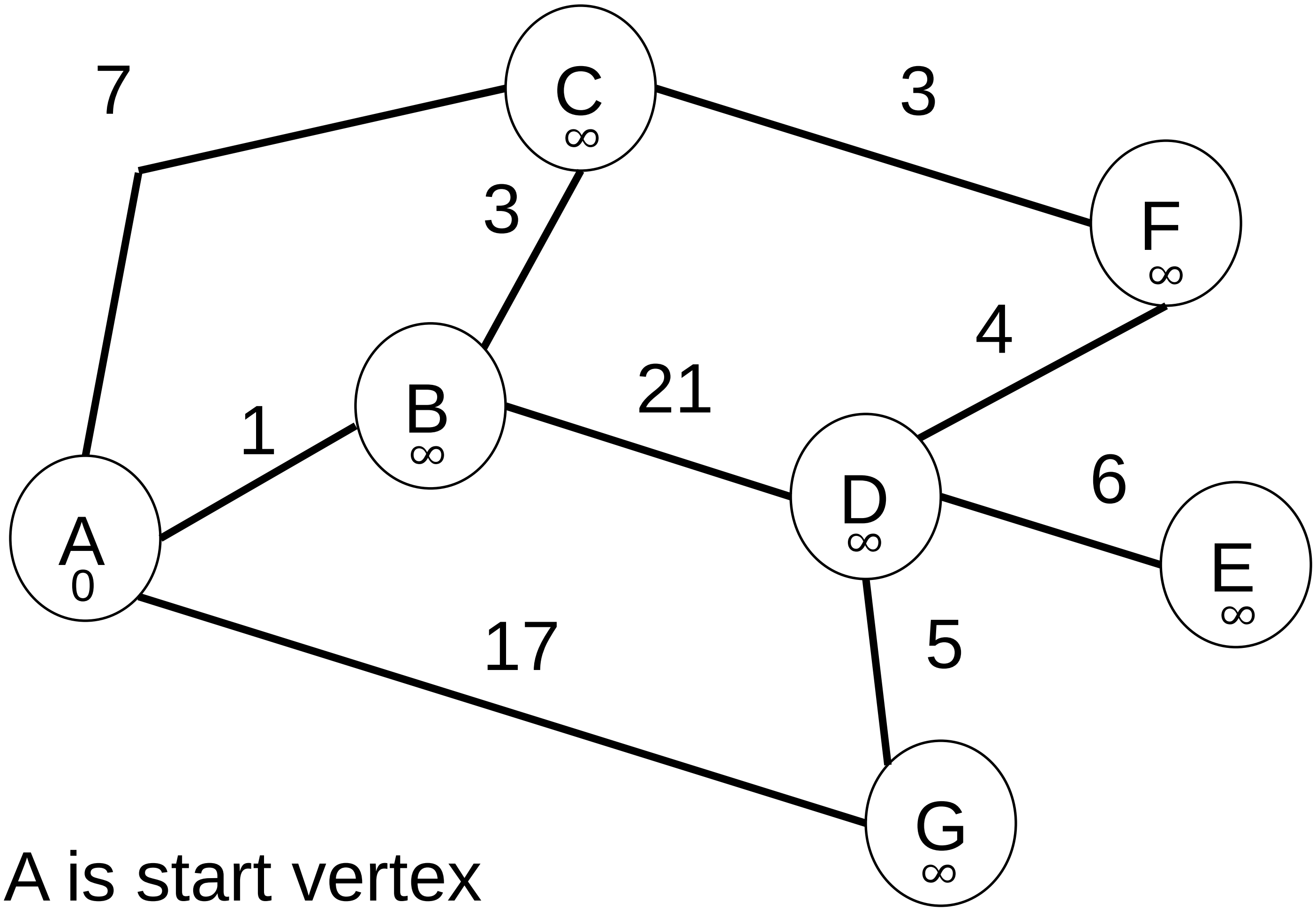
  (1, 5, 10) 15 cents

  (1, 5, 12) 15 cents



Clicker 6 - What is the cost of the lowest cost path from A to E?

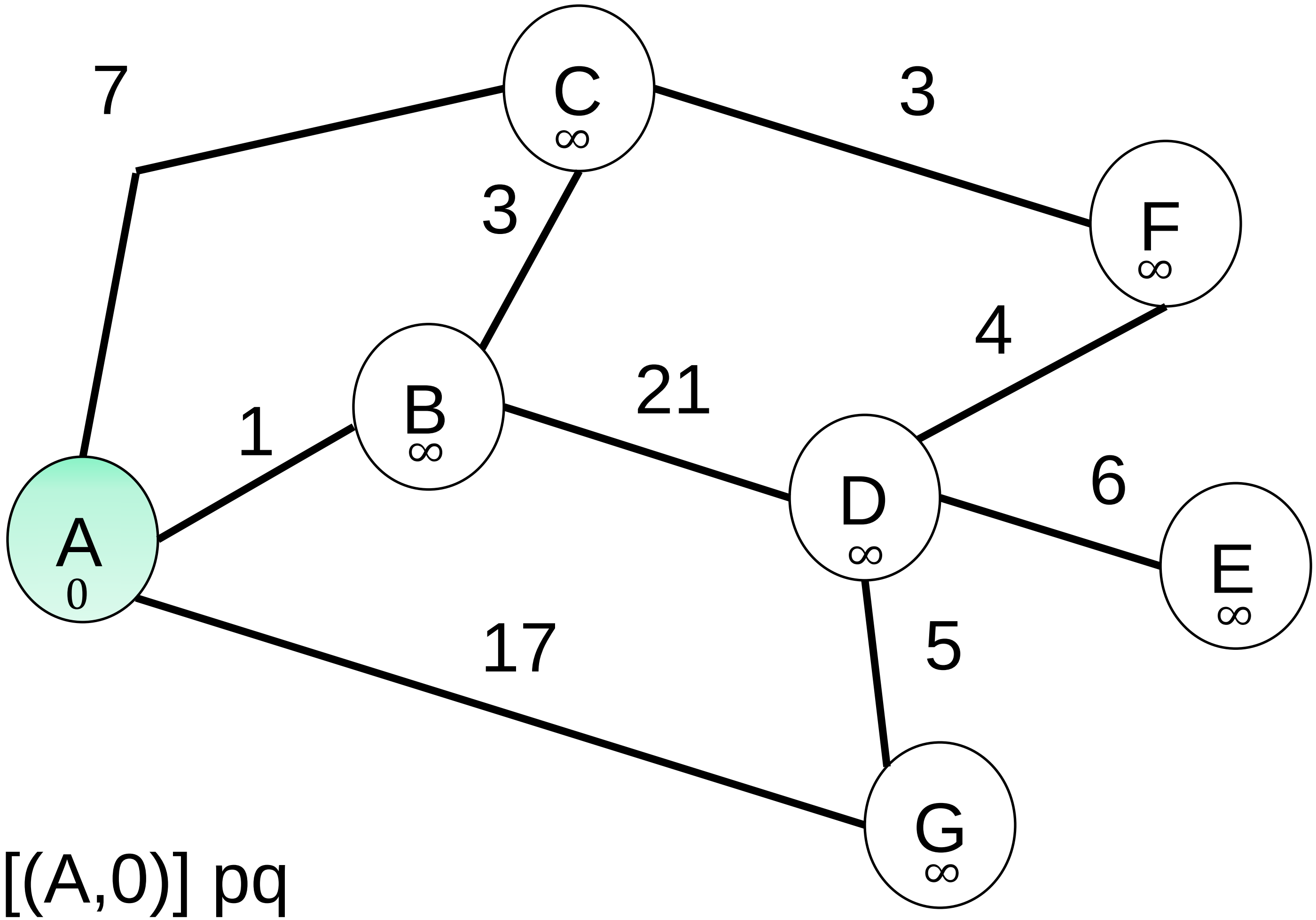
A. 5
B. 17
C. 20
D. 28
E. 37



A is start vertex

Set cost of A to 0, all others to INFINITY

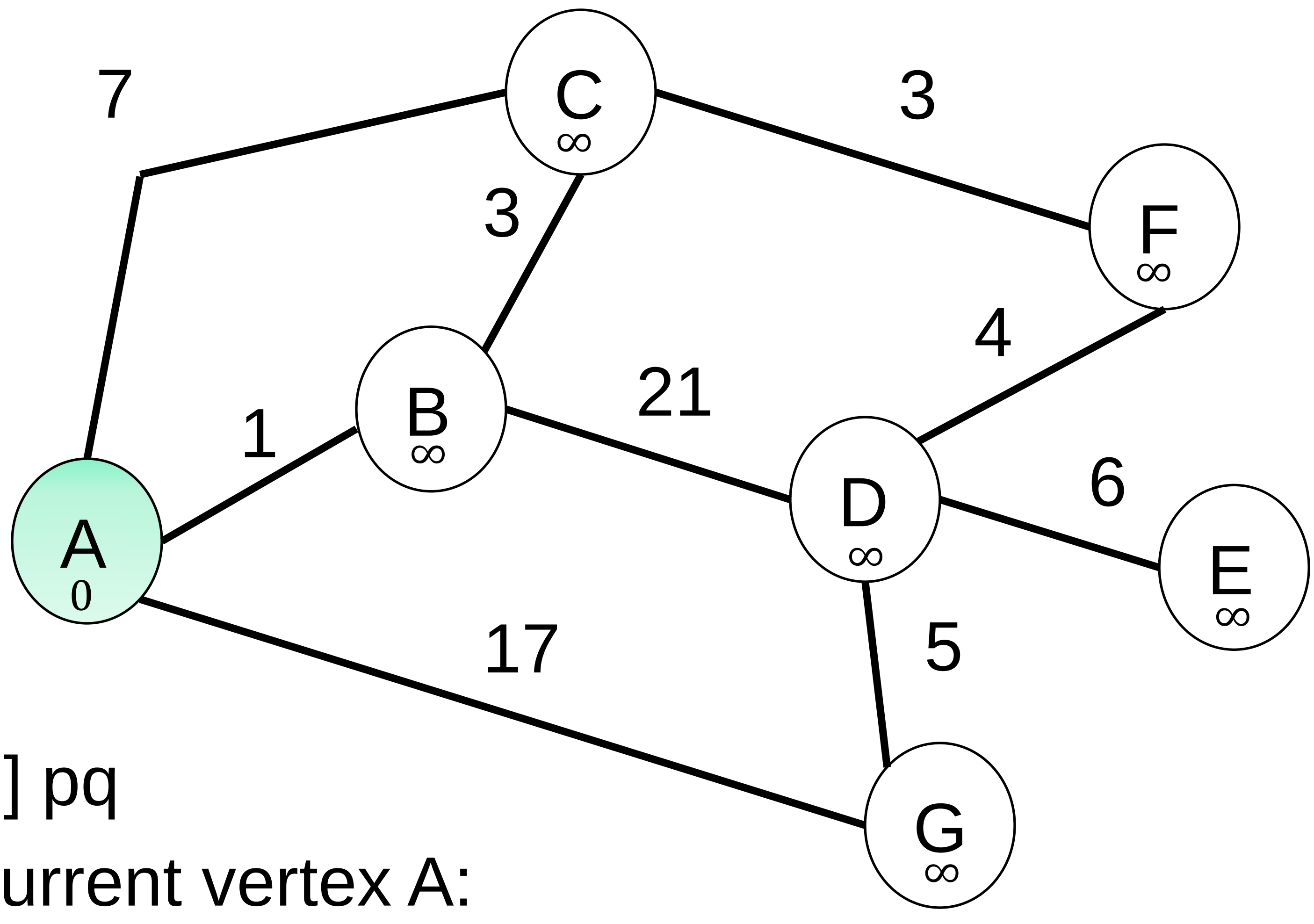
Place A in a priority queue



[(A,0)] pq

dequeue (A,0)

Mark A as visited



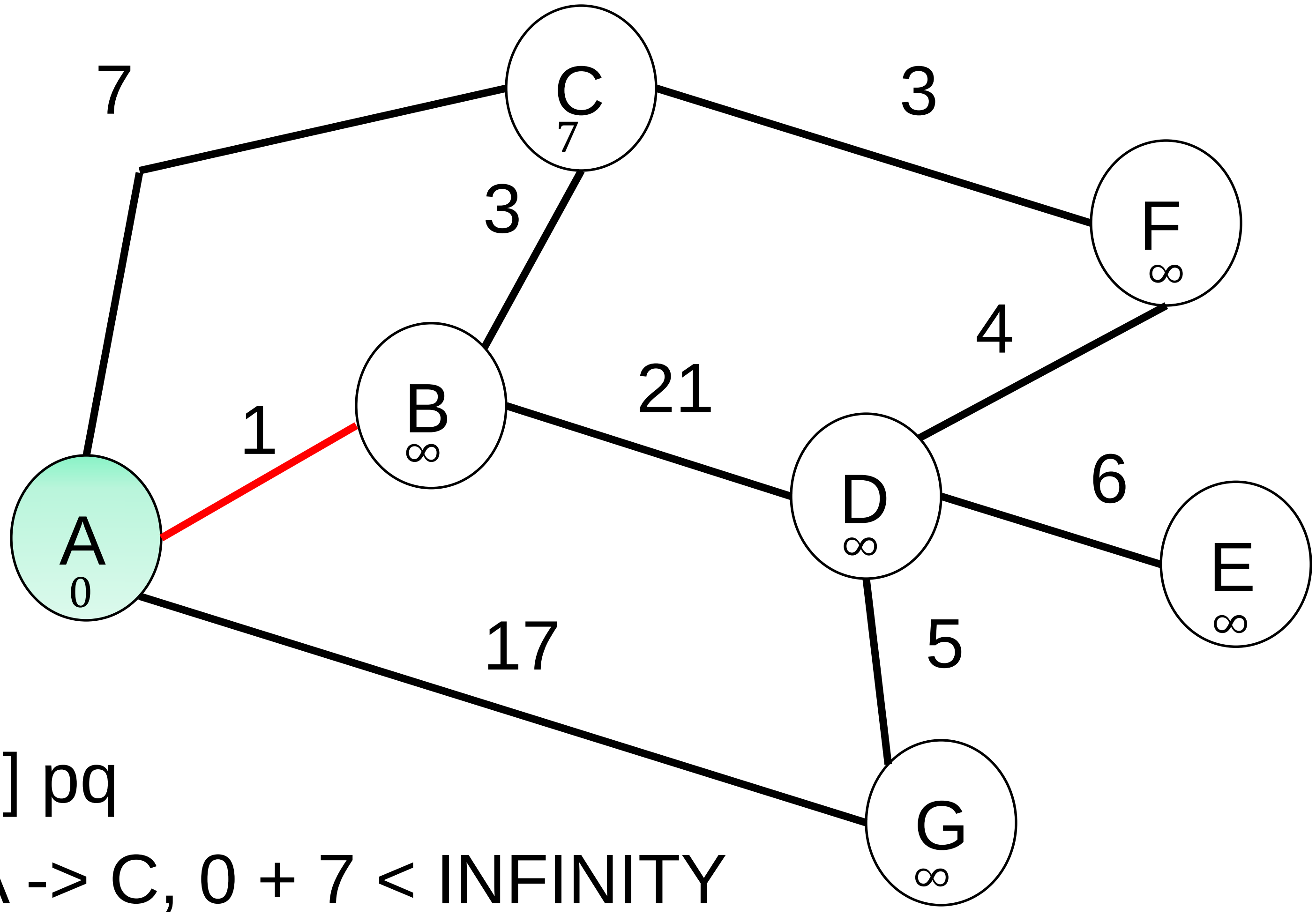
[ ] pq

current vertex A:

loop through A's edges

if sum of cost from A to dest is less than current cost

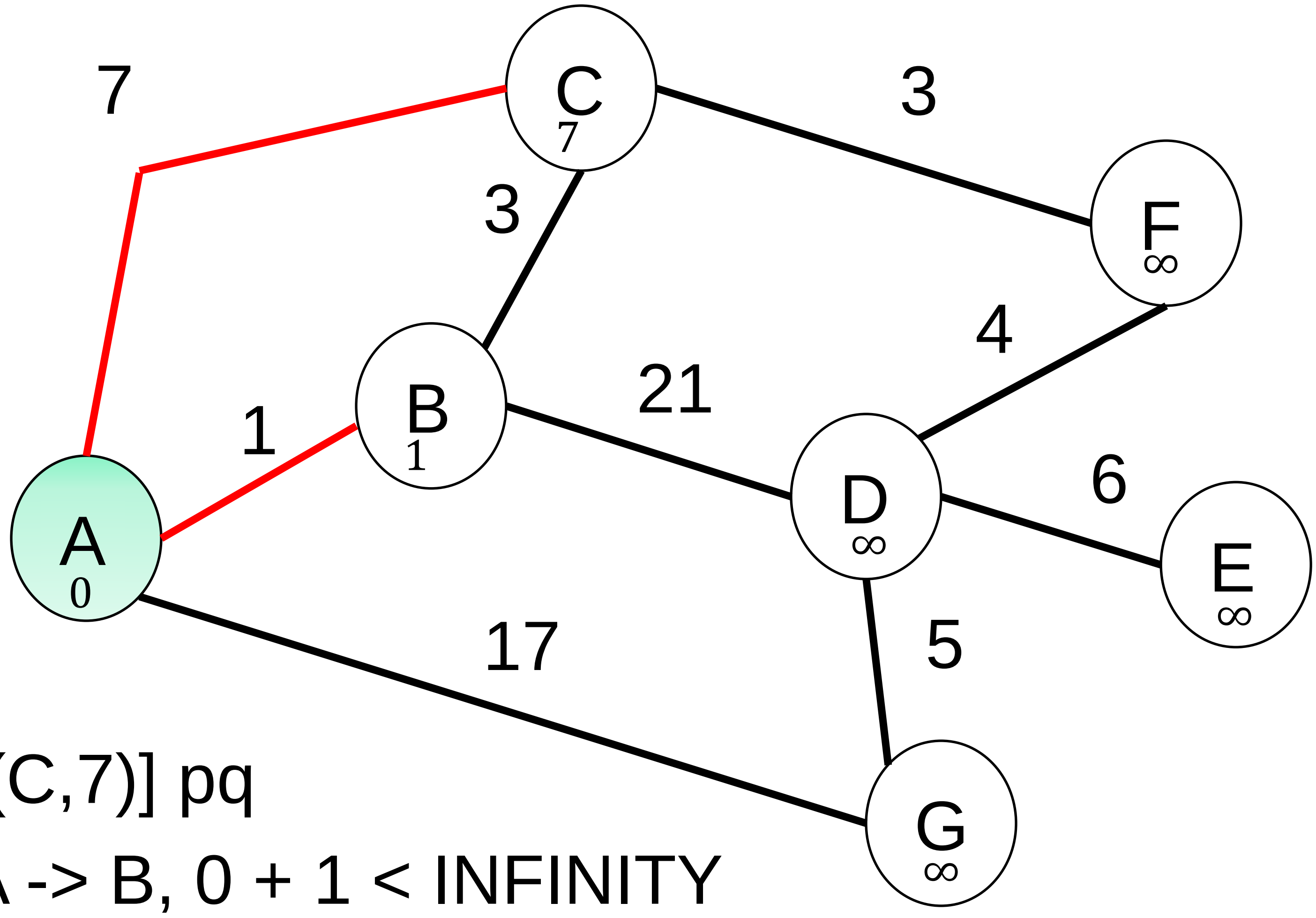
update cost and prev



[ ] pq

A -> C, 0 + 7 < INFINITY

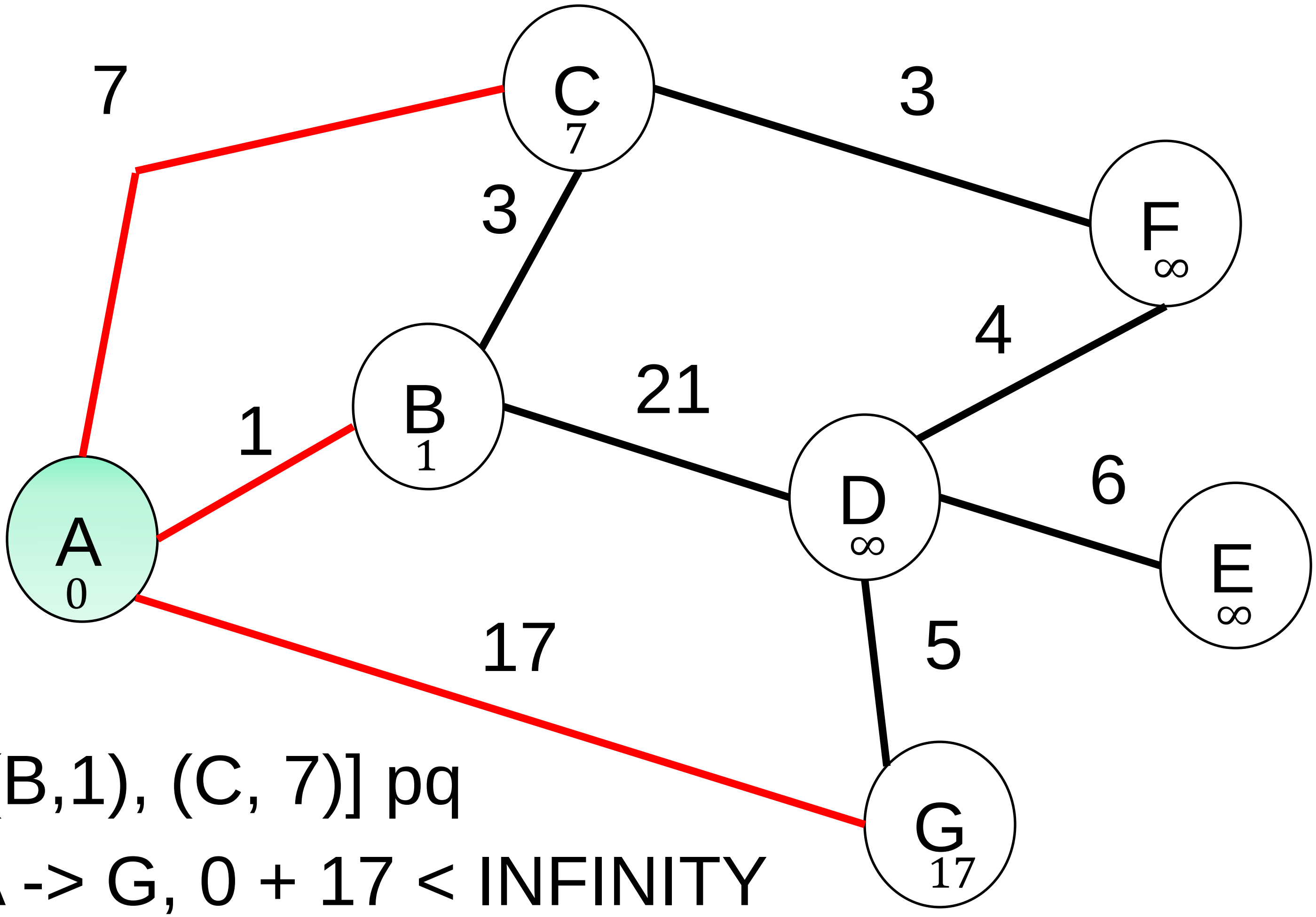
[(C,7)] pq



[(C,7)] pq

A -> B, 0 + 1 < INFINITY

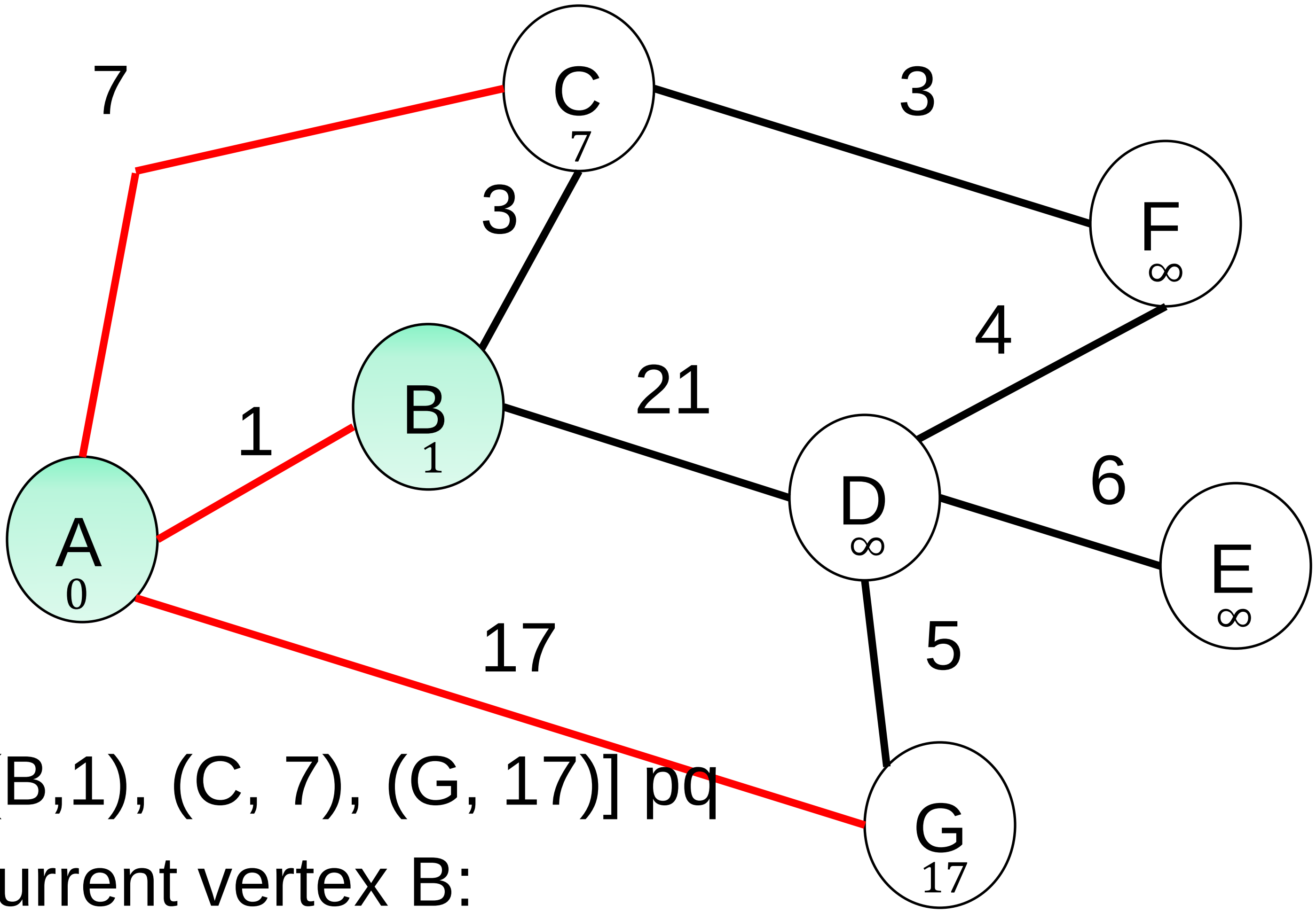
[(B,1), (C, 7)] pq (Note, the (B,1) jumps in front of (C,7)



[(B,1), (C, 7)] pq

A -> G, 0 + 17 < INFINITY

[(B,1), (C, 7), (G, 17)] pq



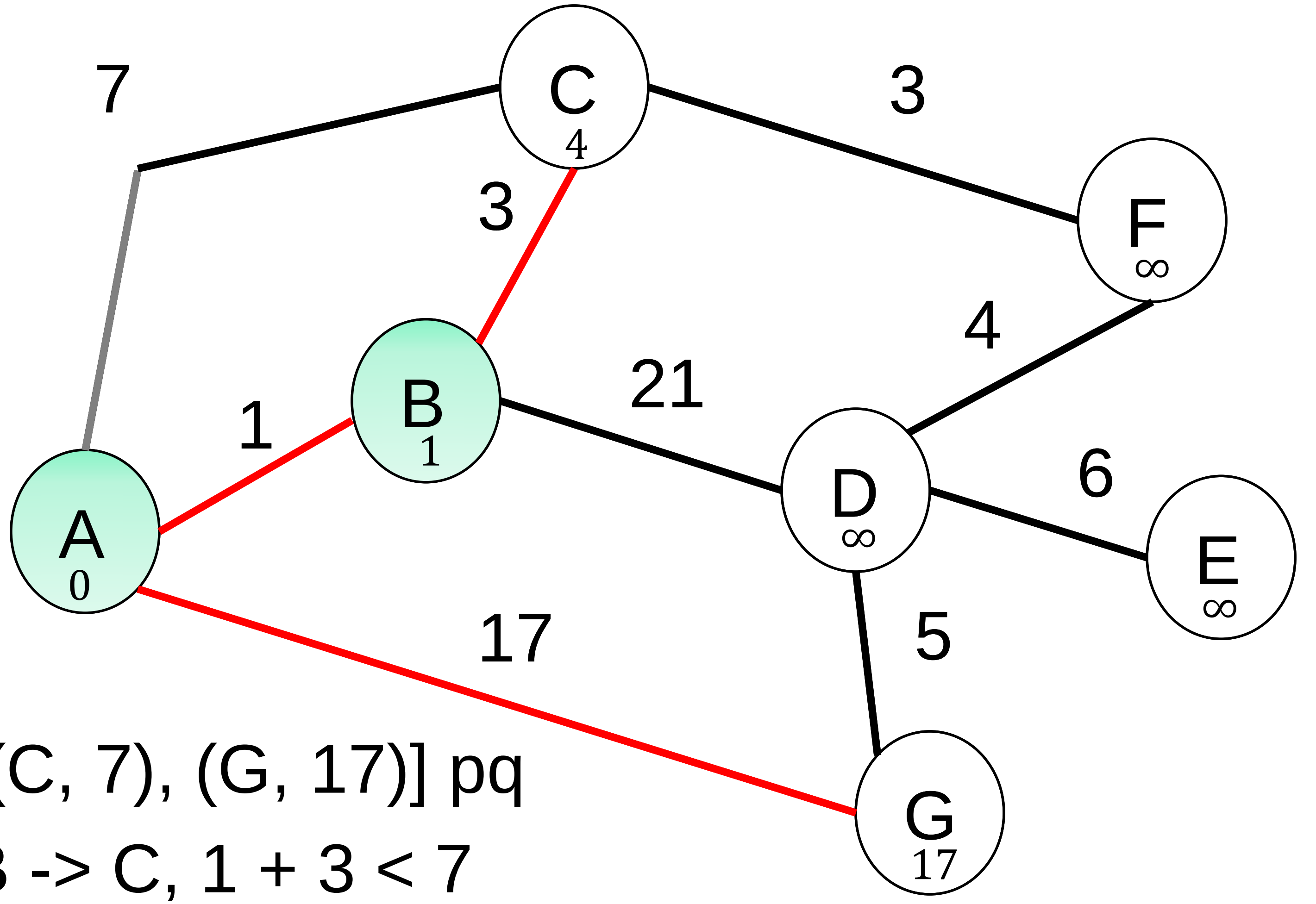
[(B,1), (C, 7), (G, 17)] pq

current vertex B:

loop through B's edges

if sum of cost from B to edge is less than current cost

update cost and prev

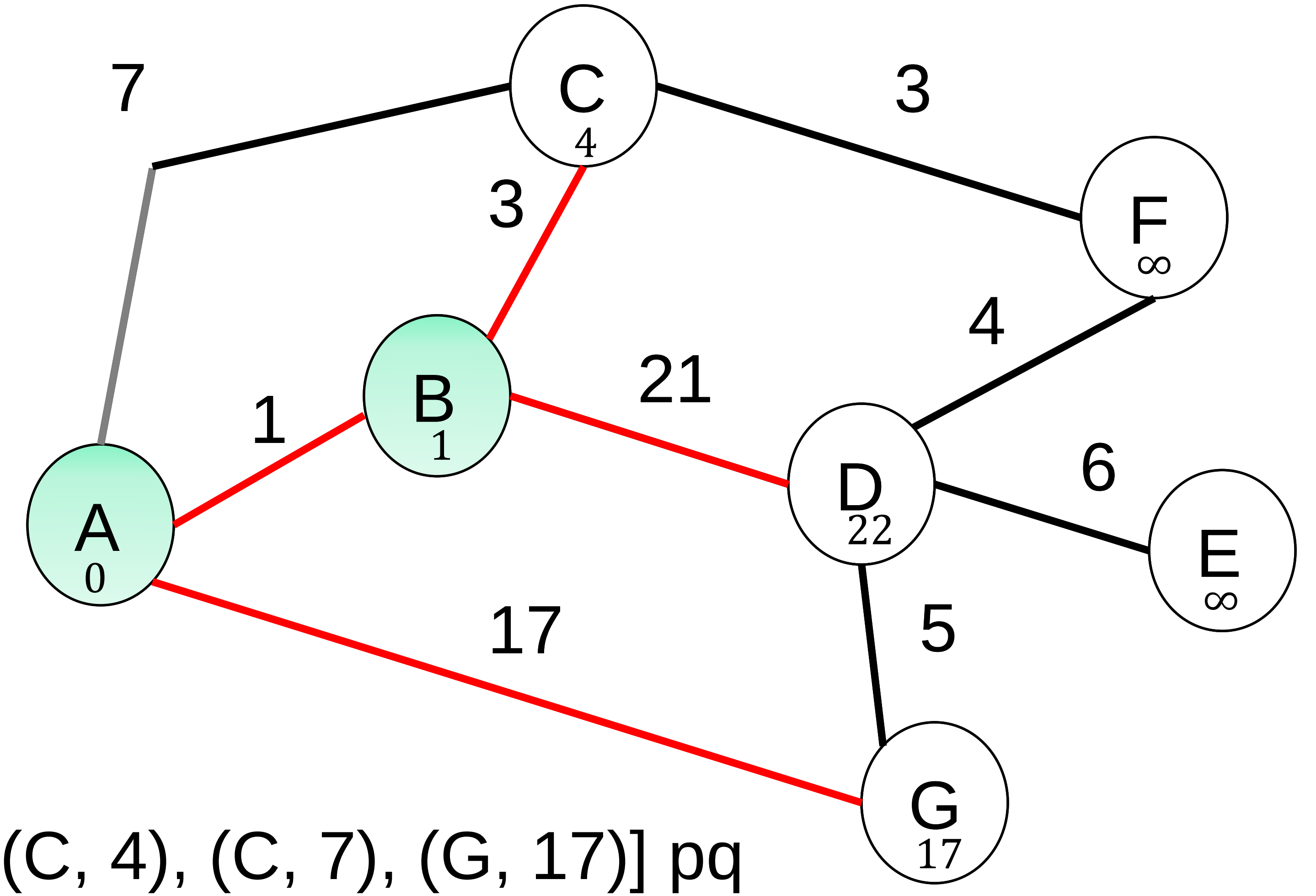


[(C, 7), (G, 17)] pq

B -> C, 1 + 3 < 7

update C's cost and previous

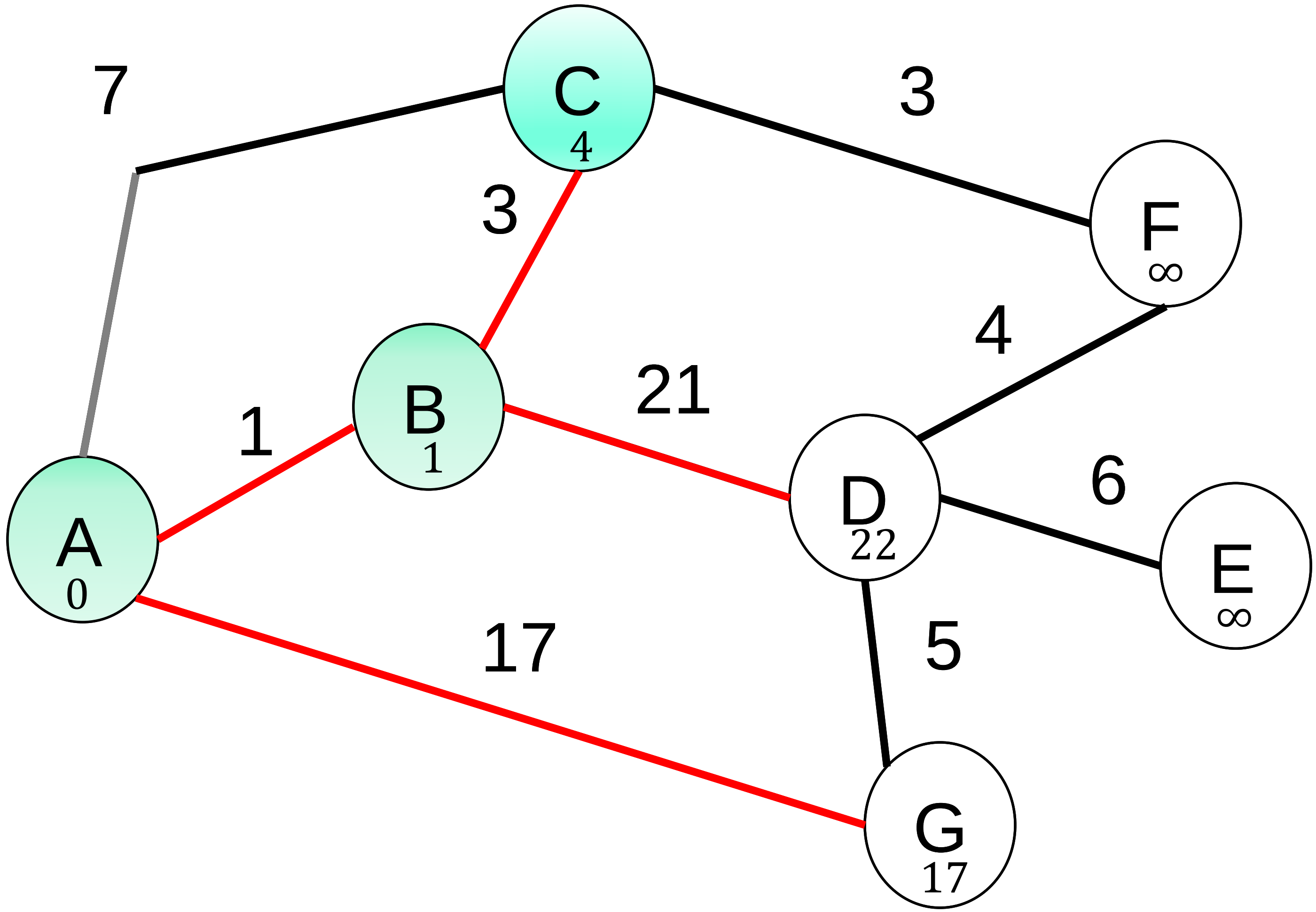
[(C, 4), (C, 7), (G, 17)] pq



[(C, 4), (C, 7), (G, 17)] pq

B -> D, 1 + 21 < INFINITY

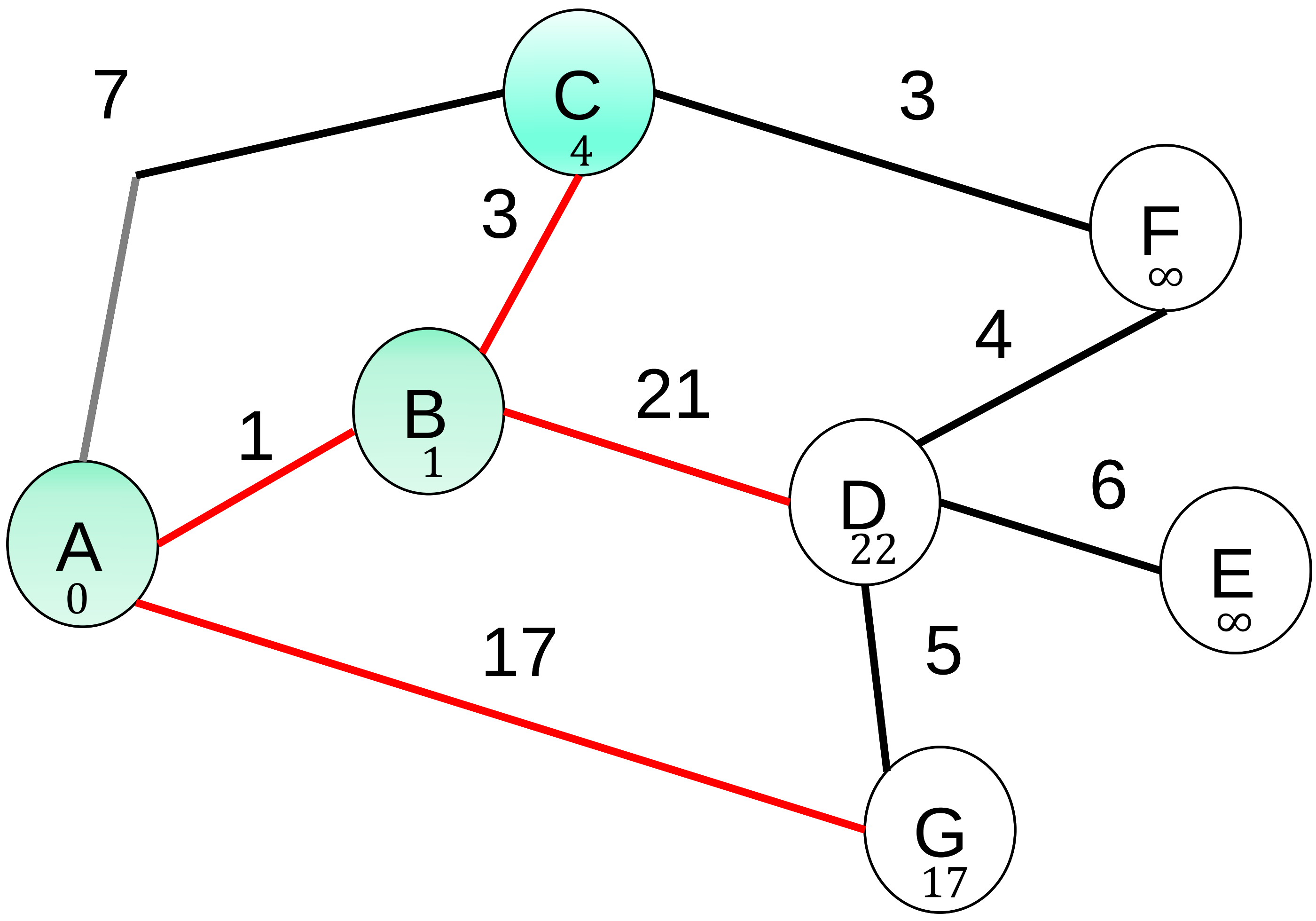
[(C, 4), (C, 7), (G, 17), (D, 22)] pq



[(C, 4), (C, 7), (G, 17), (D, 22)] pq

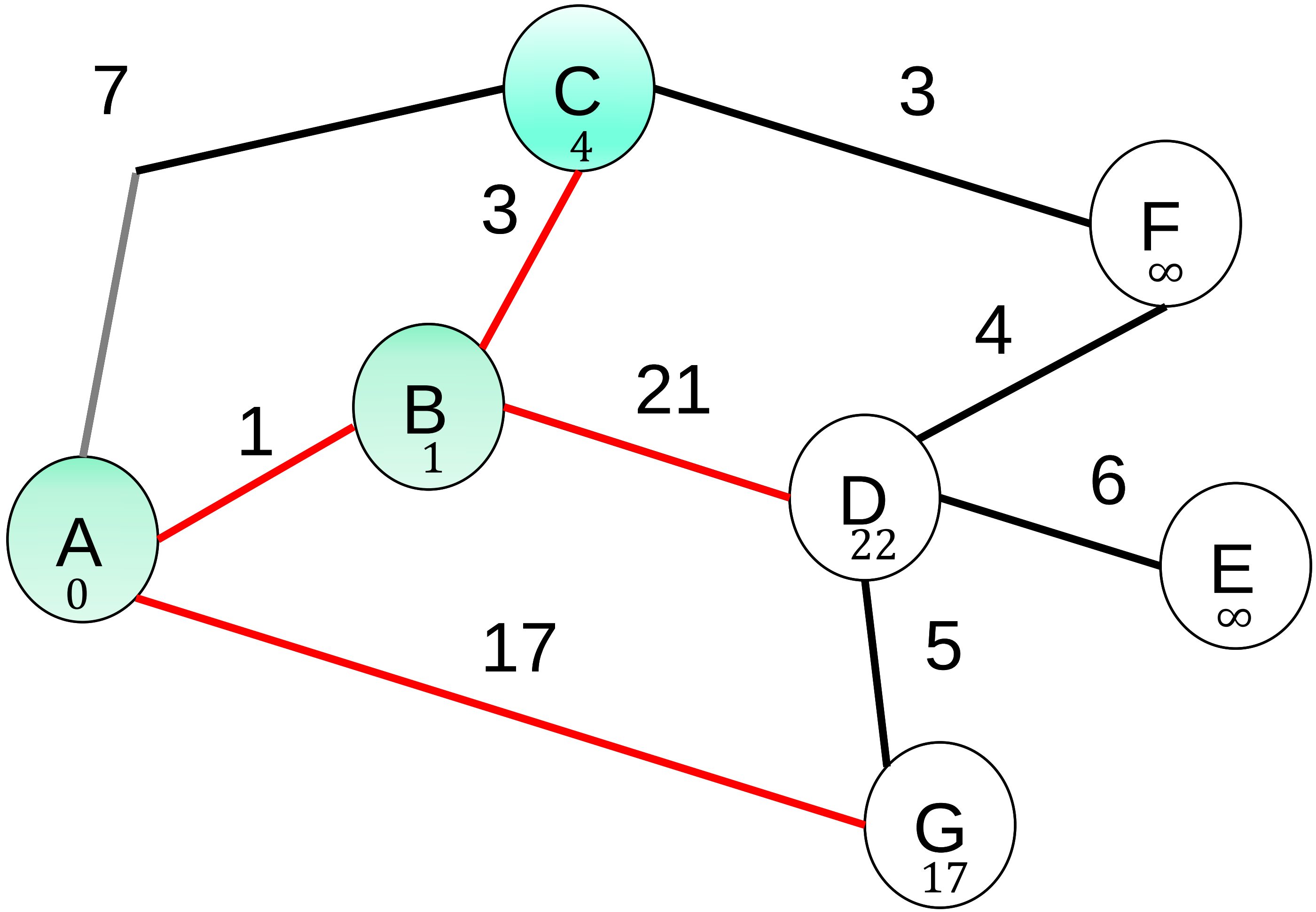
current vertex is C, cost 4

loop through C's edges



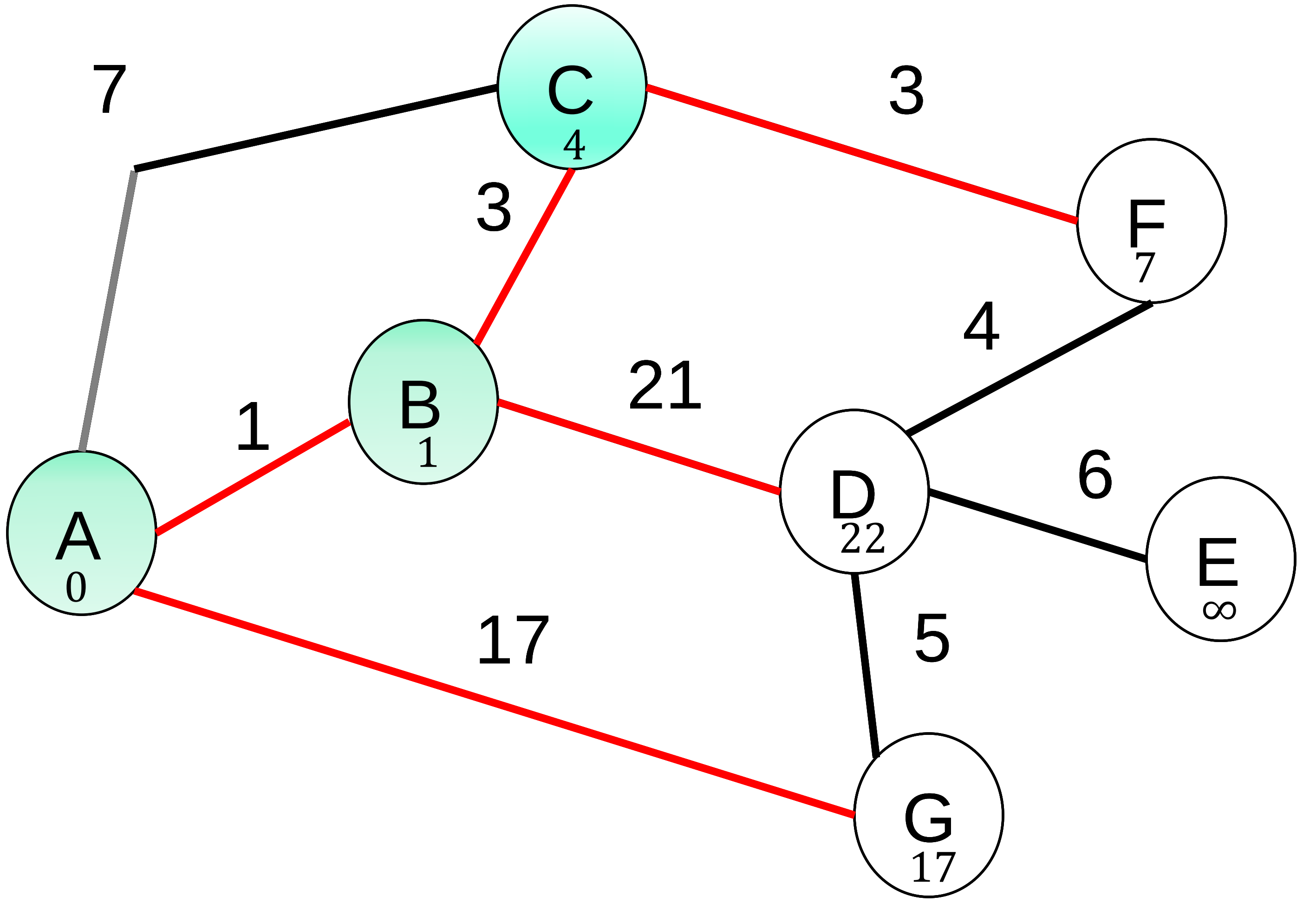
[(C, 7), (G, 17), (D, 22)] pq

C -> A, 7 + 4 !< 0, skip



[(C, 7), (G, 17), (D, 22)] pq

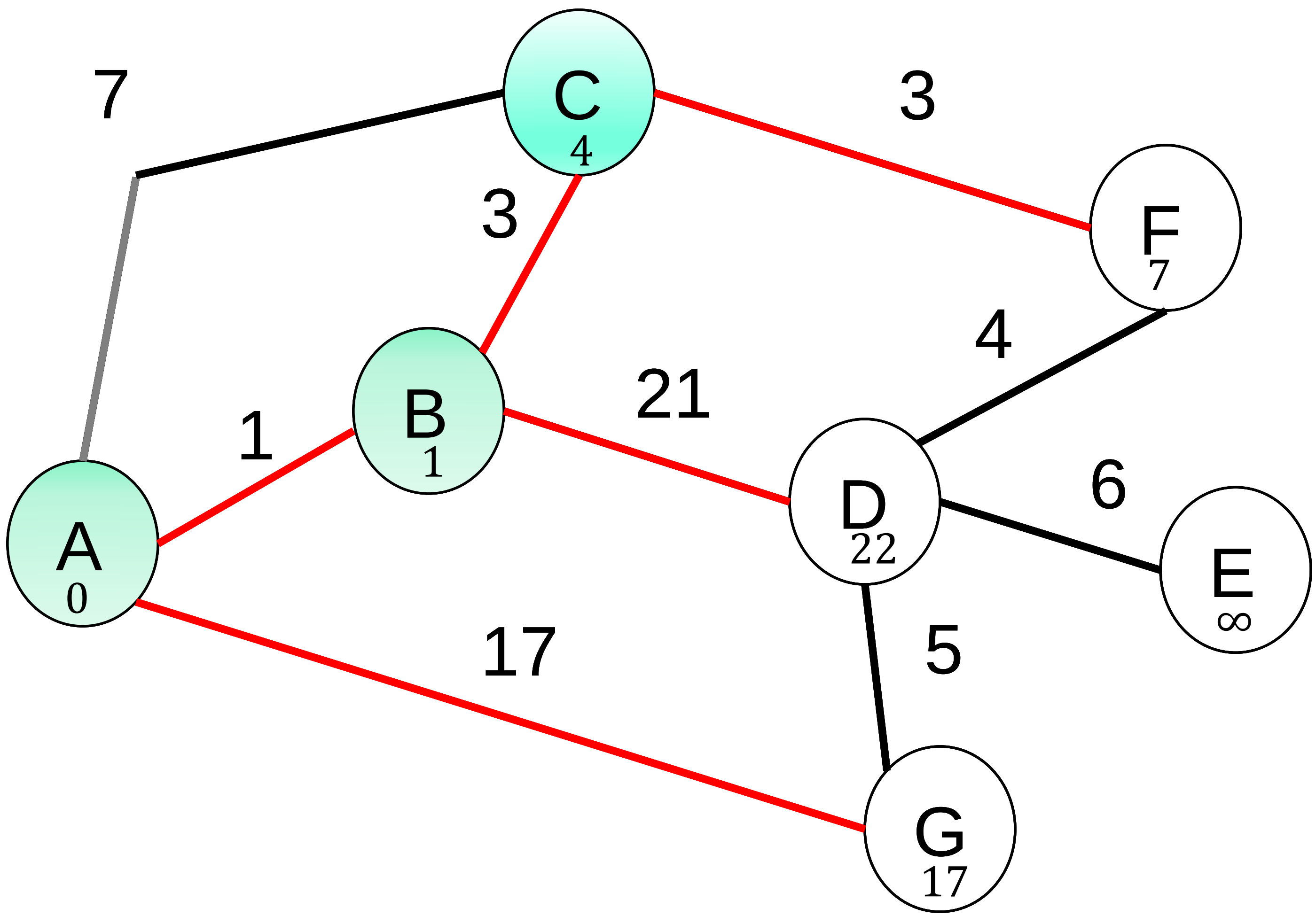
C -> B, 4 + 3 !< 1, skip



[(C, 7), (G, 17), (D, 22)] pq

C -> F, 4 + 3 < INFINITY

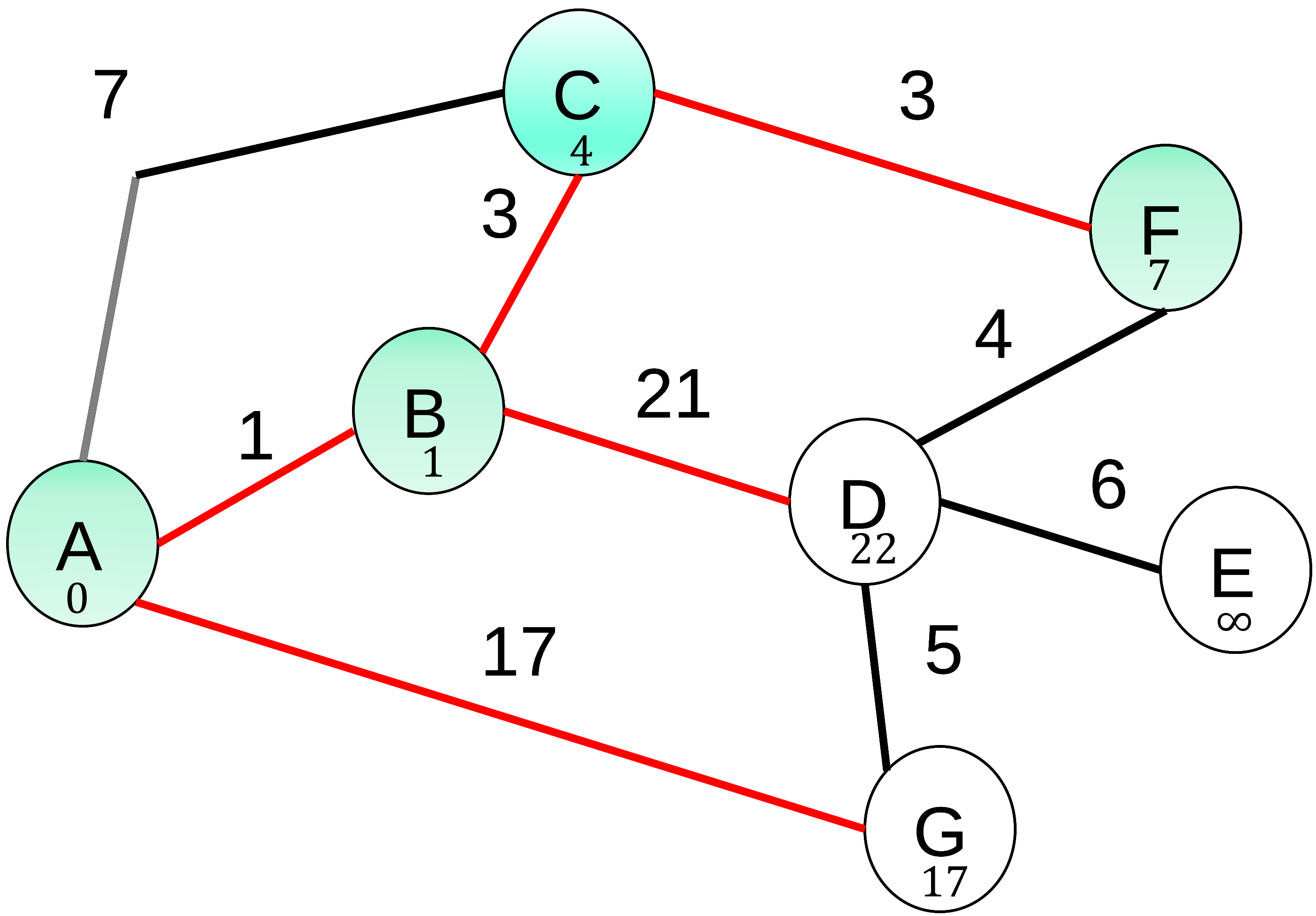
[(C, 7), (F, 7), (G, 17), (D, 22)] pq



[(C, 7), (F, 7), (G, 17), (D, 22)] pq

current vertex is C

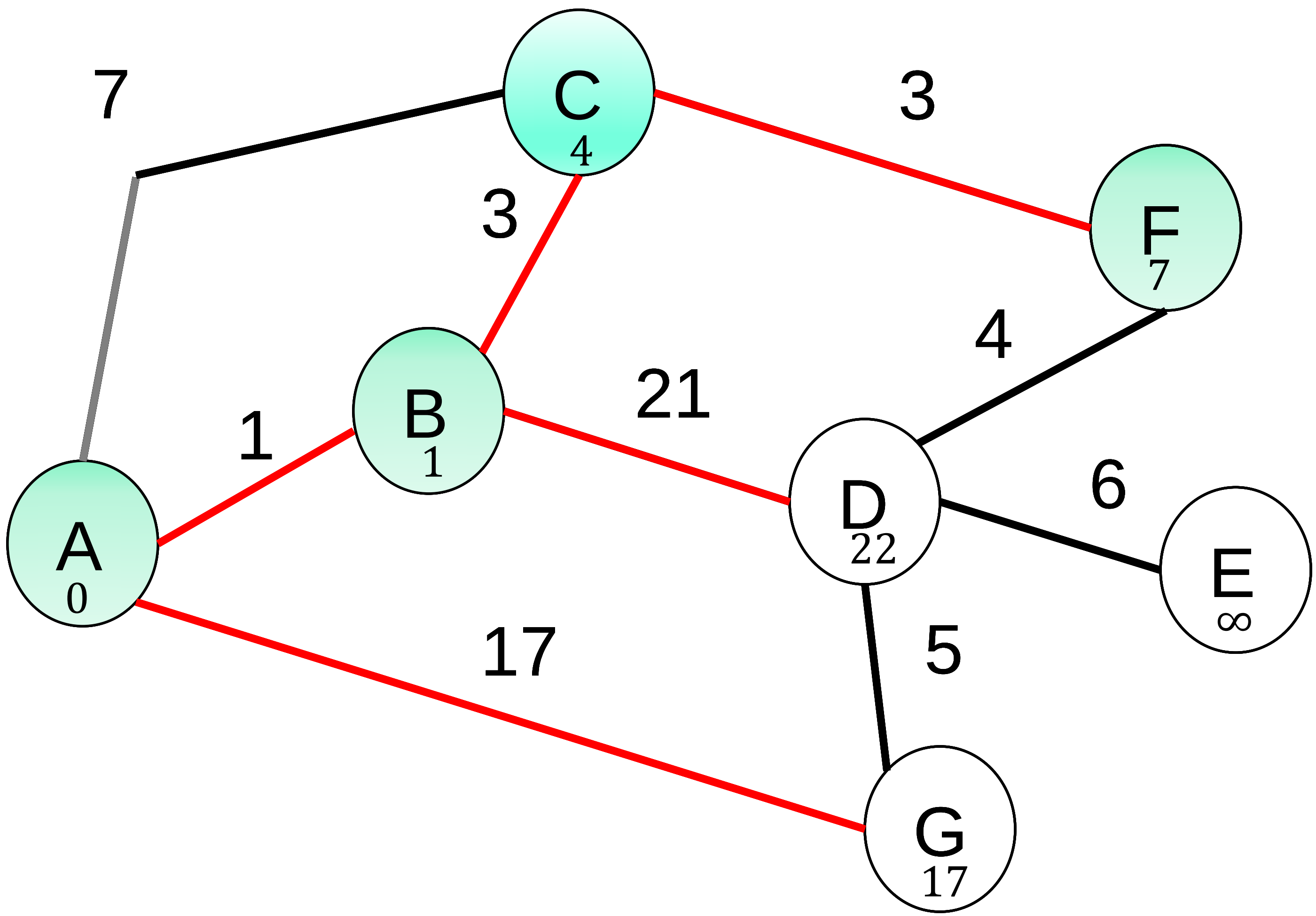
Already visited so skip



[(F, 7), (G, 17), (D, 22)] pq

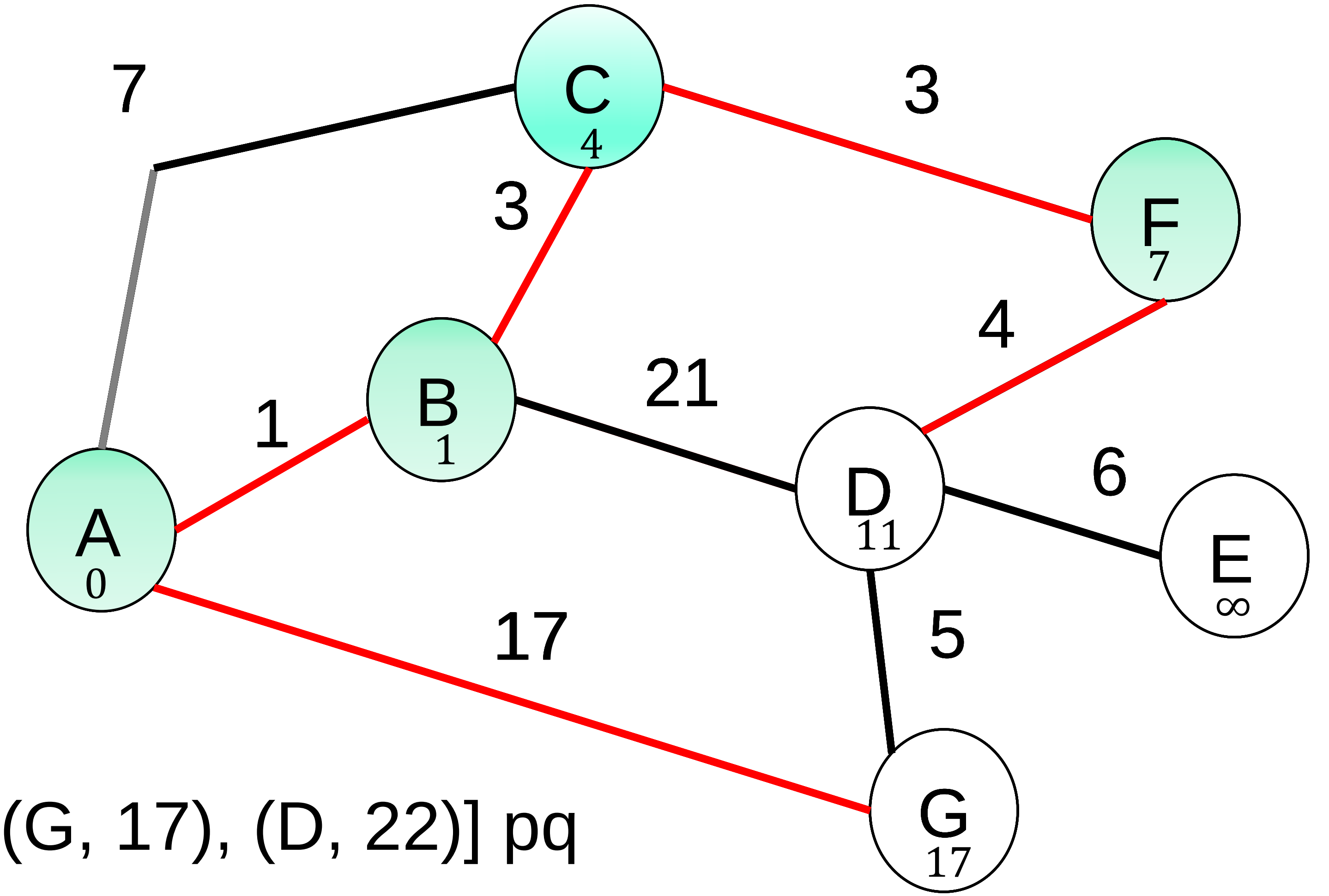
current vertex is F

loop through F's edges



[(G, 17), (D, 22)] pq

F -> C, 7 + 3 !< 4, so skip



[(G, 17), (D, 22)] pq

F -> D, 7 + 4 < 22

update D's cost and previous

[(D, 11), (G, 17), (D, 22)] pq

Aside - Implementing Dijkstra's

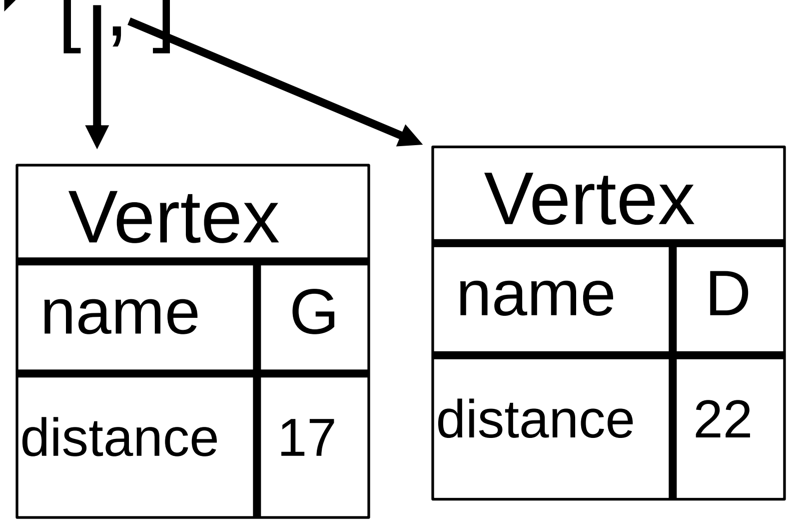
- Create a Path class to allow for multiple paths and distances (costs) to a given vertex

```
private static class Path
    implements Comparable<Path> {

    private Vertex dest;
    private double cost;
```

- Use a priority queue of Paths to store the vertices and distances

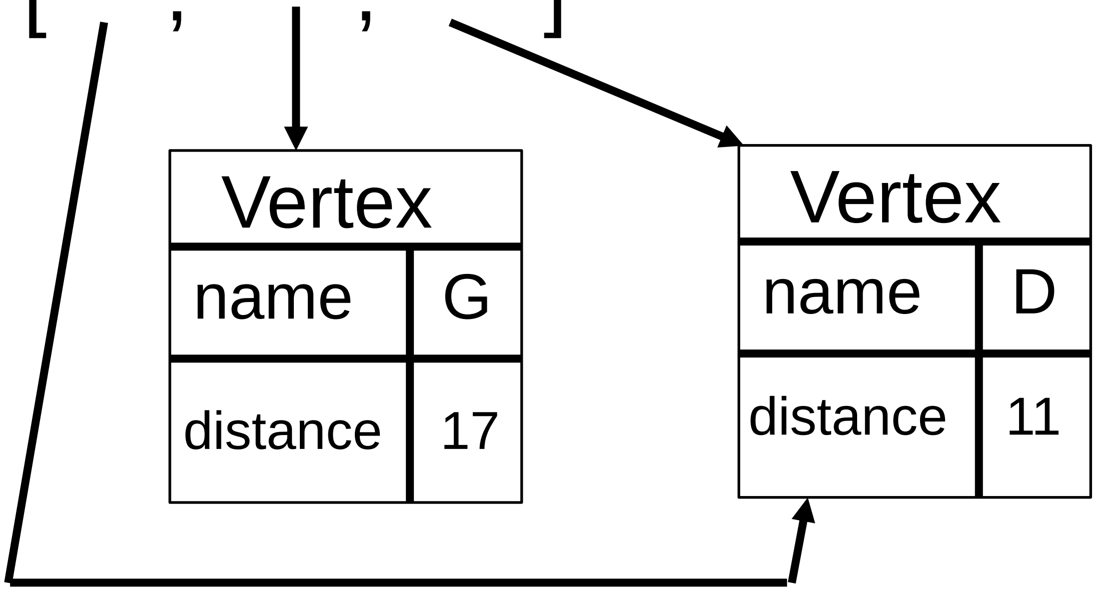
Why? References!!!

- Slide 74 and 78, adding new, lower cost path to Vertex D
- Abstractly: [(G, 17), (D, 22)] becomes
  [(D, 11) (G, 17), (D, 22)]
- What does priority queue store? ***References to Vertex Objects***
- [,]

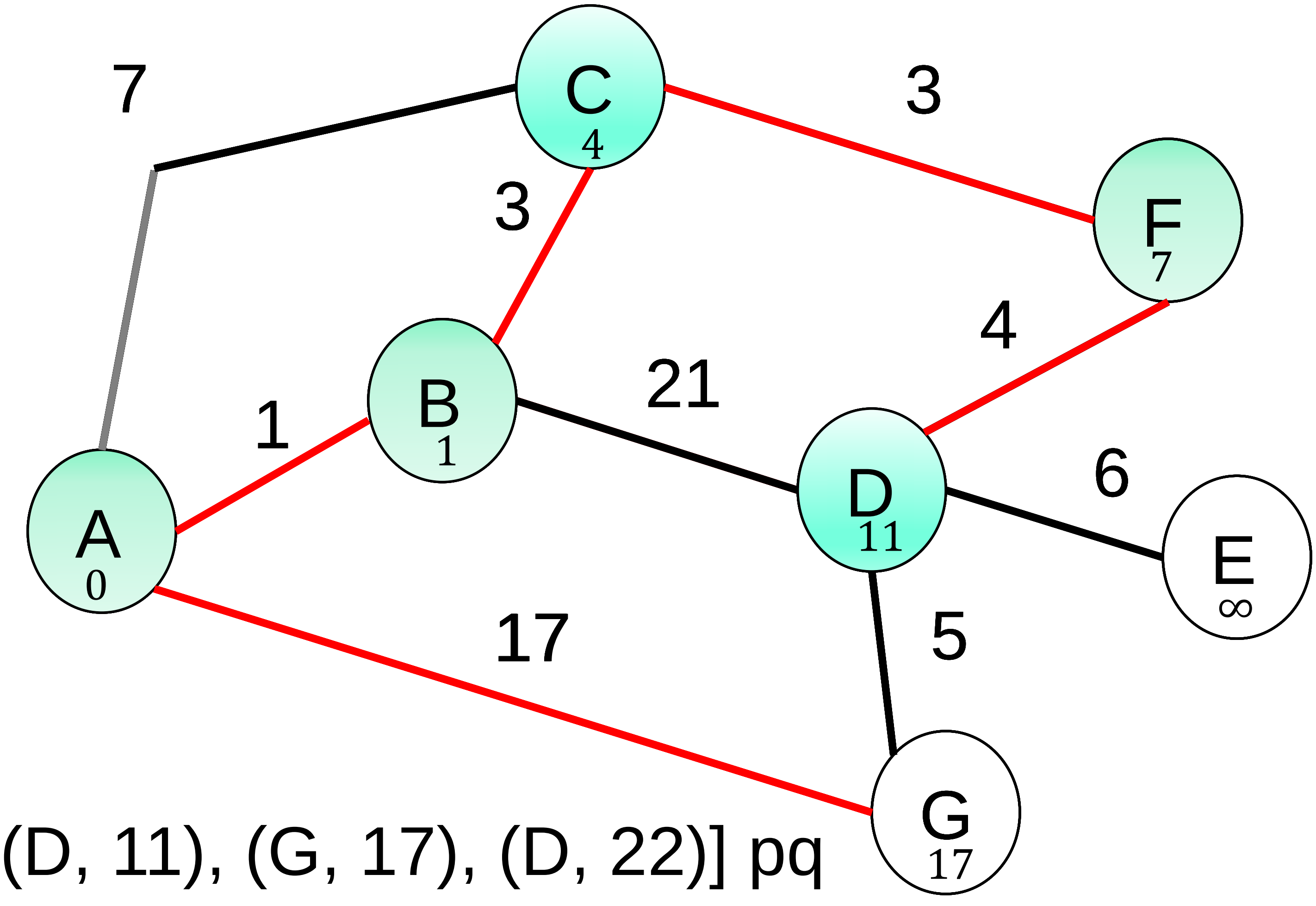


Lower Cost Path to D

- New, lower cost path to D. Alter Vertex D's distance to 11 and add to priority queue
- [, ,]



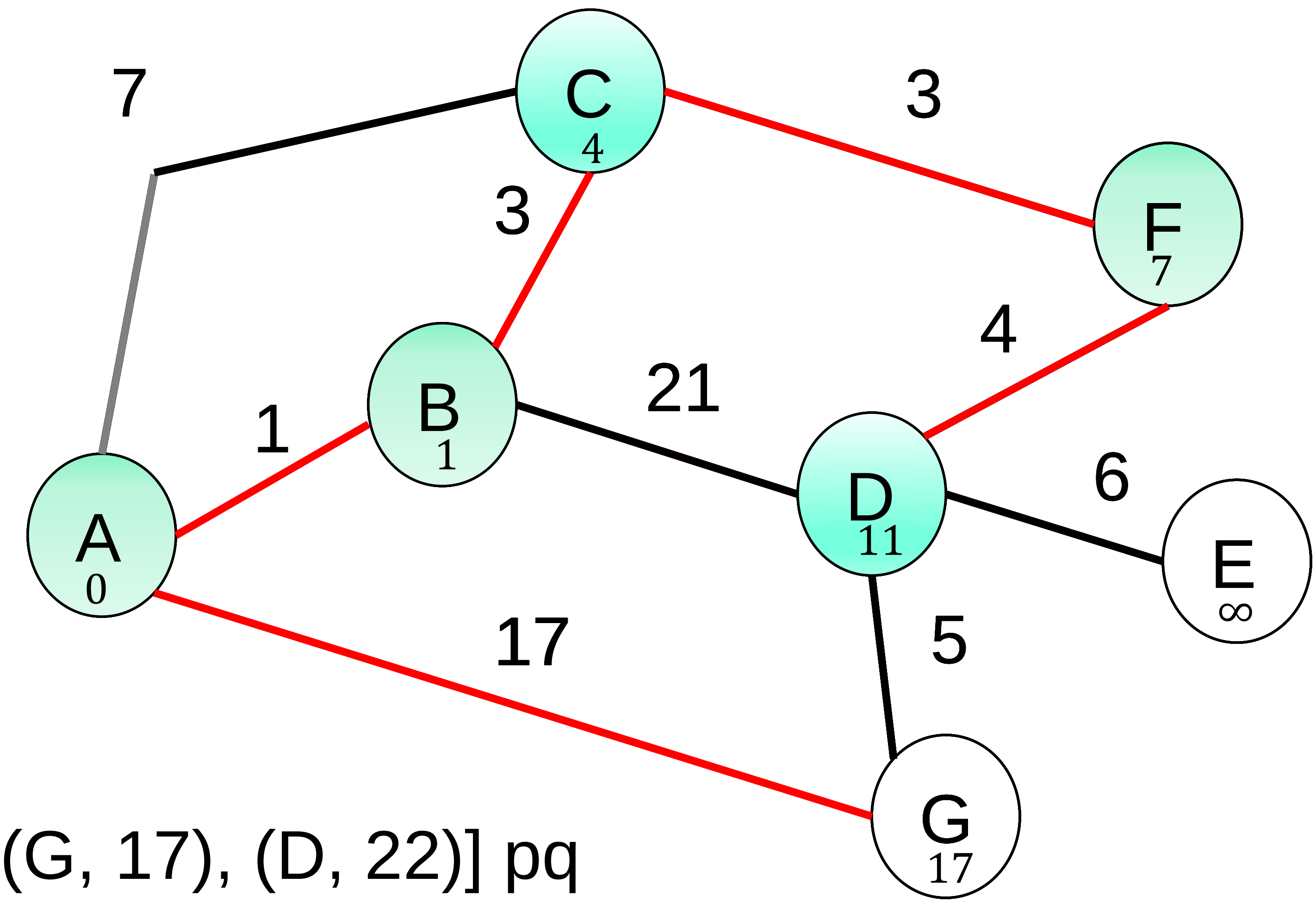
- PROBLEMS?????
- Abstractly [(D, 11), (G, 17), (D, 11)]



[(D, 11), (G, 17), (D, 22)] pq

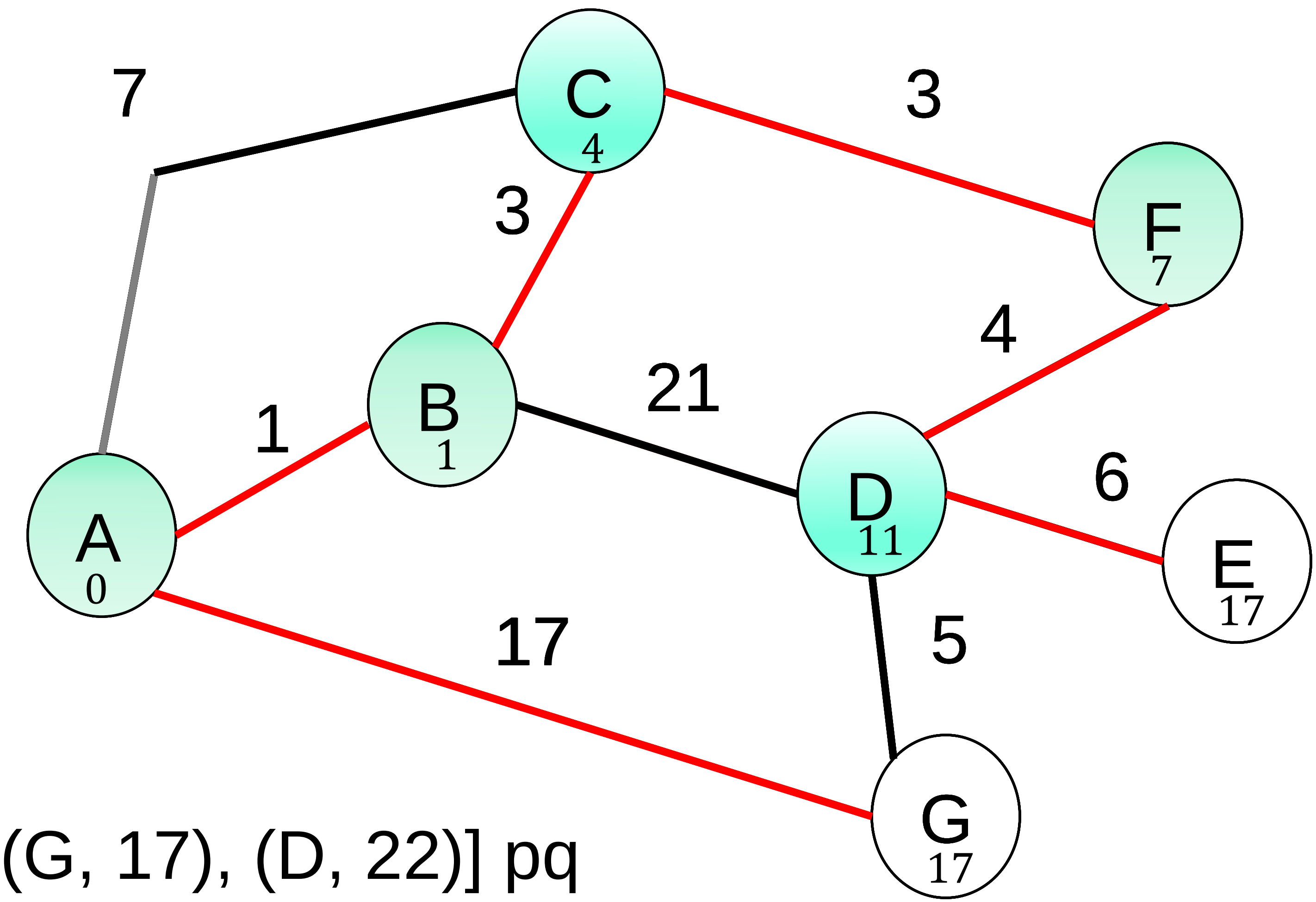
current vertex is D

loop through D's edges



[(G, 17), (D, 22)] pq

D -> B, 11 + 21 !< 1, so skip

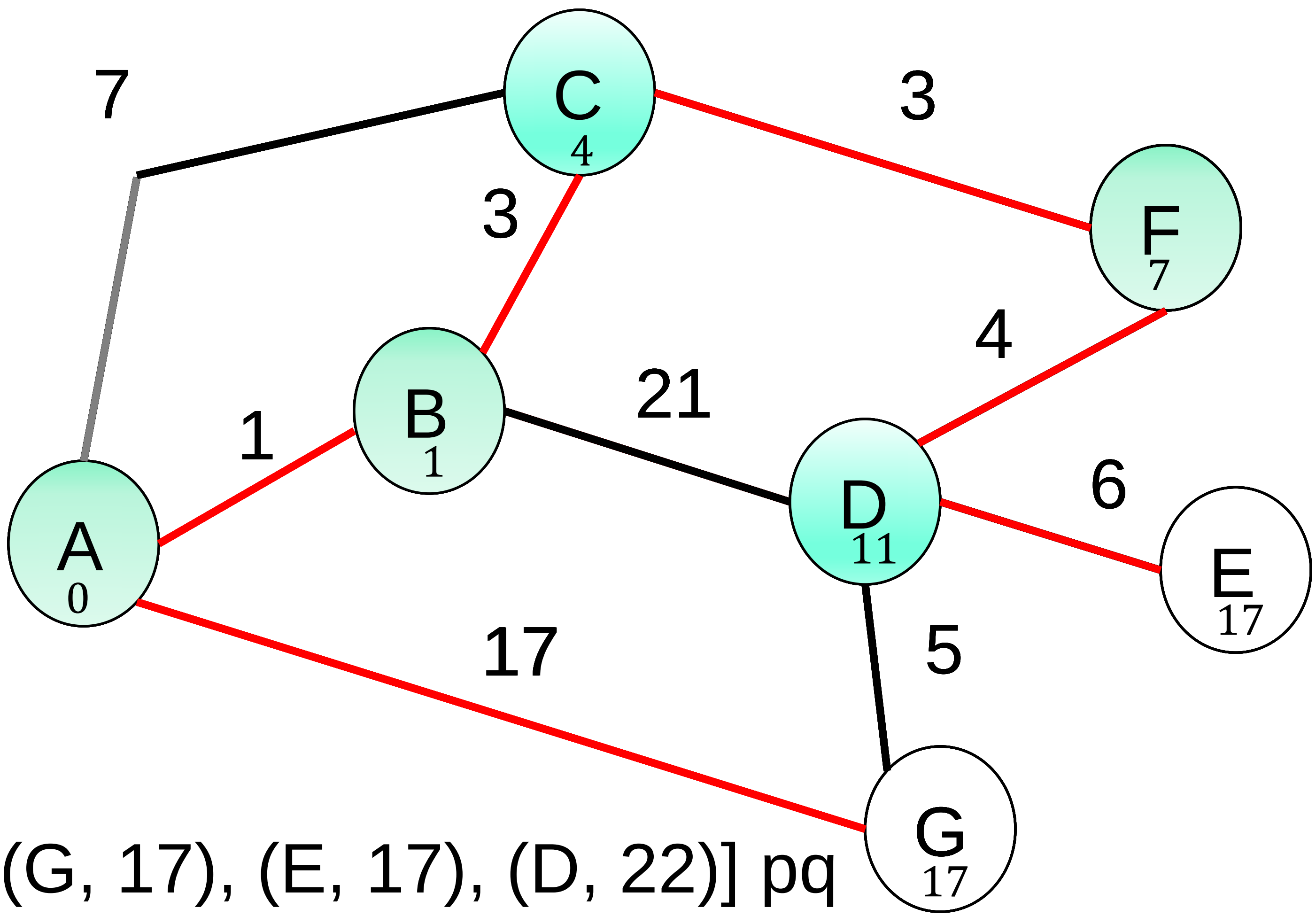


[(G, 17), (D, 22)] pq

D -> E, 11 + 6 < INFINITY

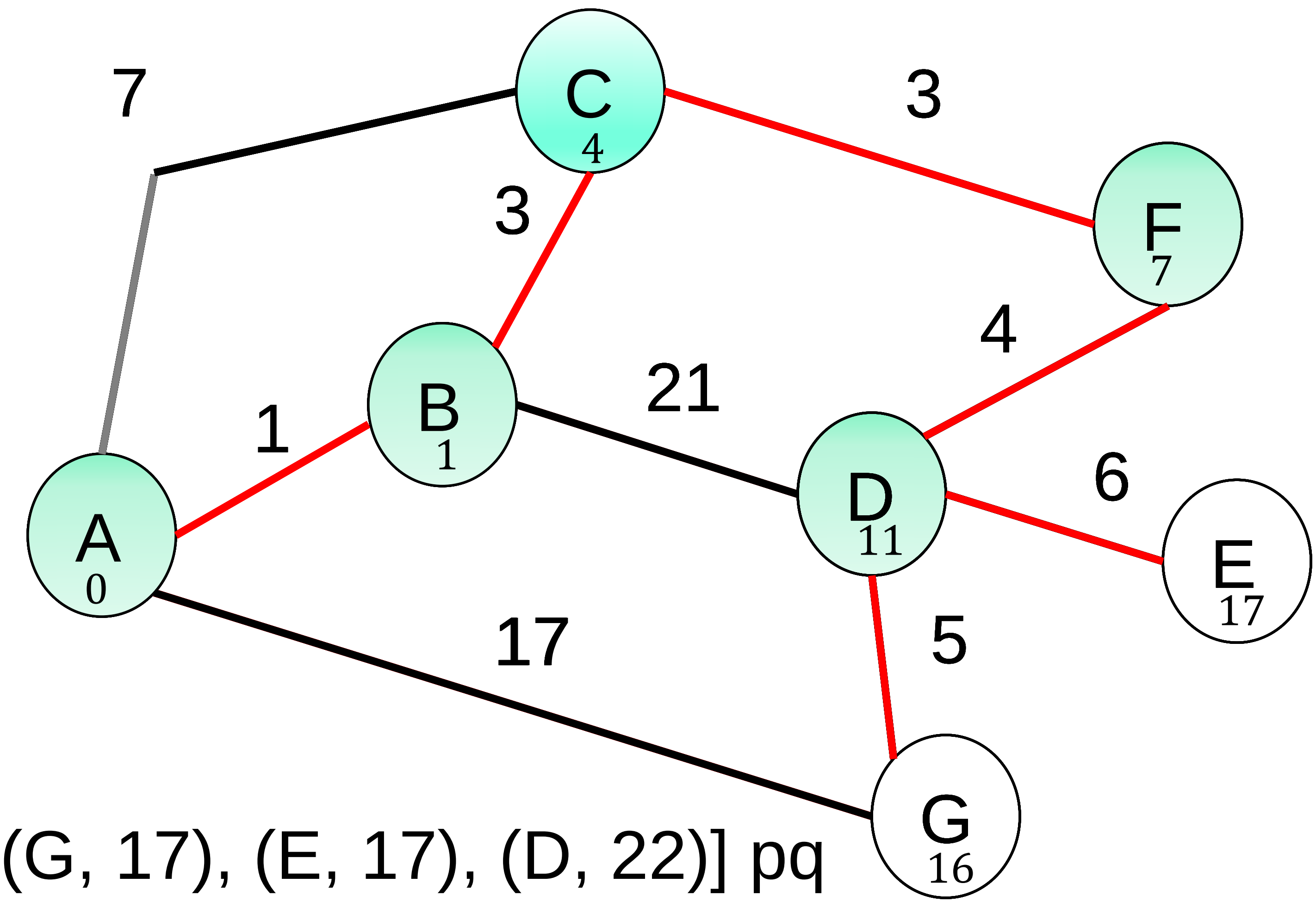
update E's cost and previous

[(G, 17), (E, 17), (D, 22)] pq



[(G, 17), (E, 17), (D, 22)] pq

D -> F, 4 + 11 !< 7, so skip

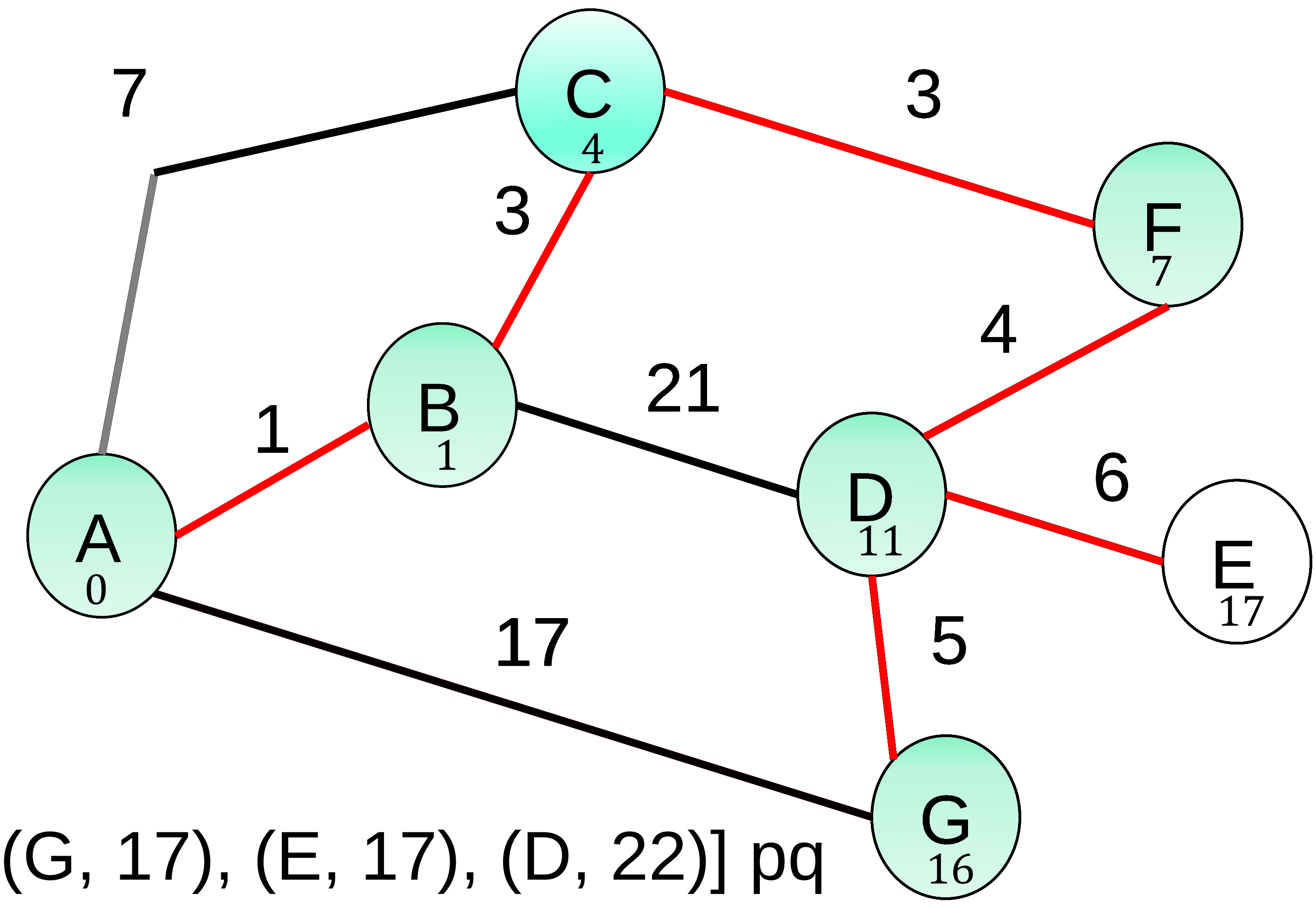


[(G, 17), (E, 17), (D, 22)] pq

D -> G, 11 + 5 < 17

update G's cost and previous

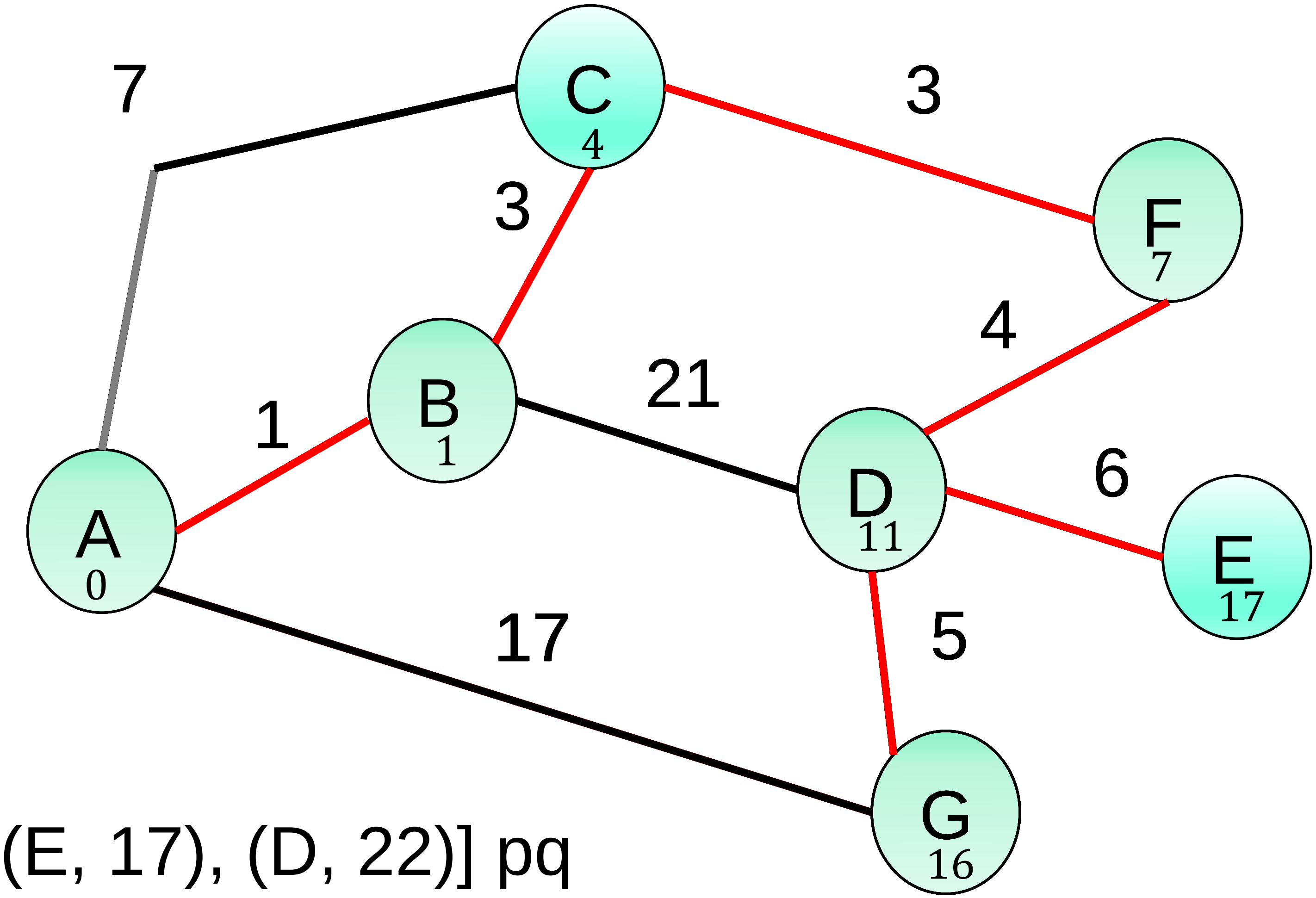
[(G, 16), (G, 17), (E, 17), (D, 22)] pq



[(G, 17), (E, 17), (D, 22)] pq

current vertex is G

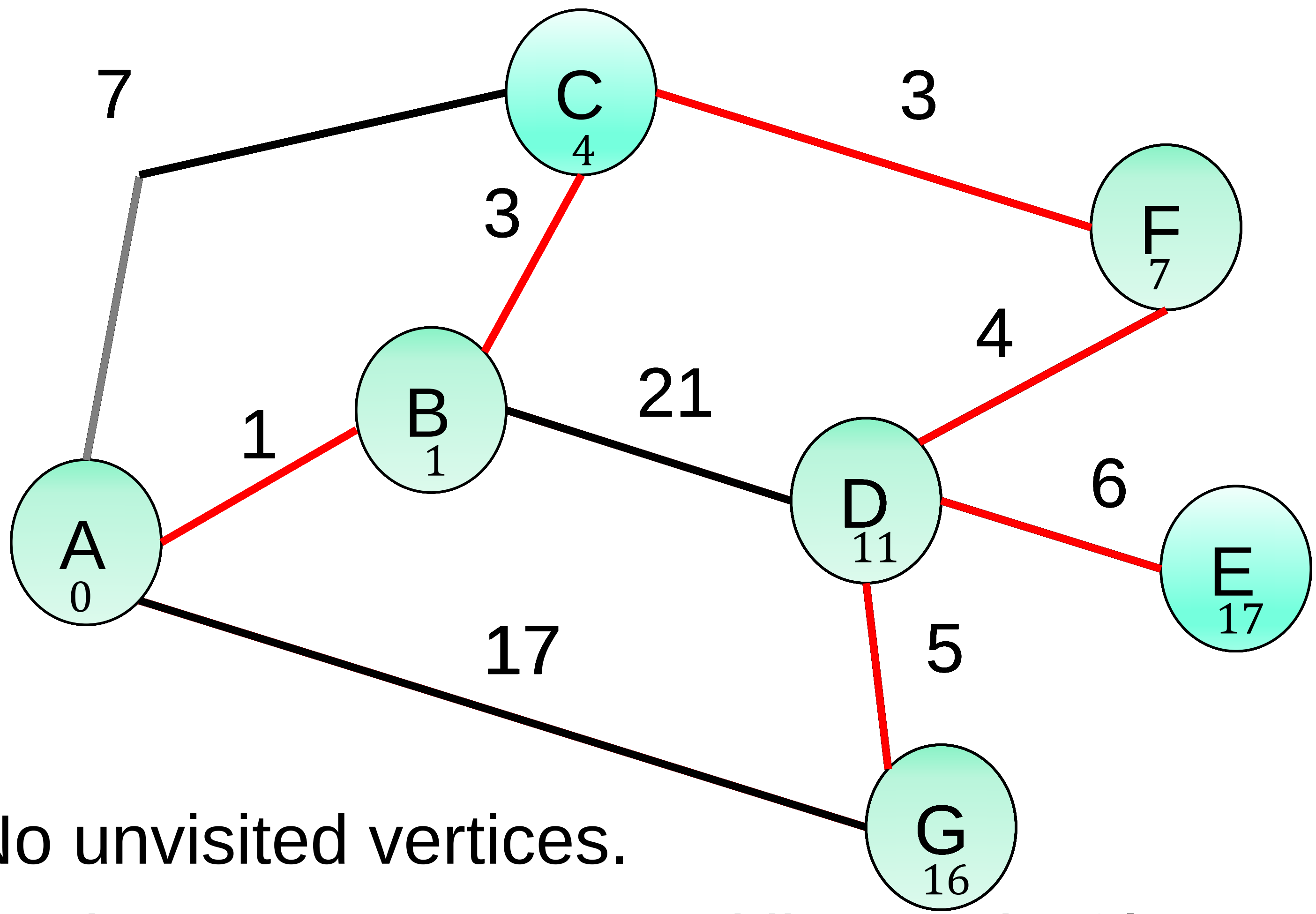
loop though edges, already visited all neighbors



[(E, 17), (D, 22)] pq

current vertex is E

loop though edges, already visited all neighbors



No unvisited vertices.

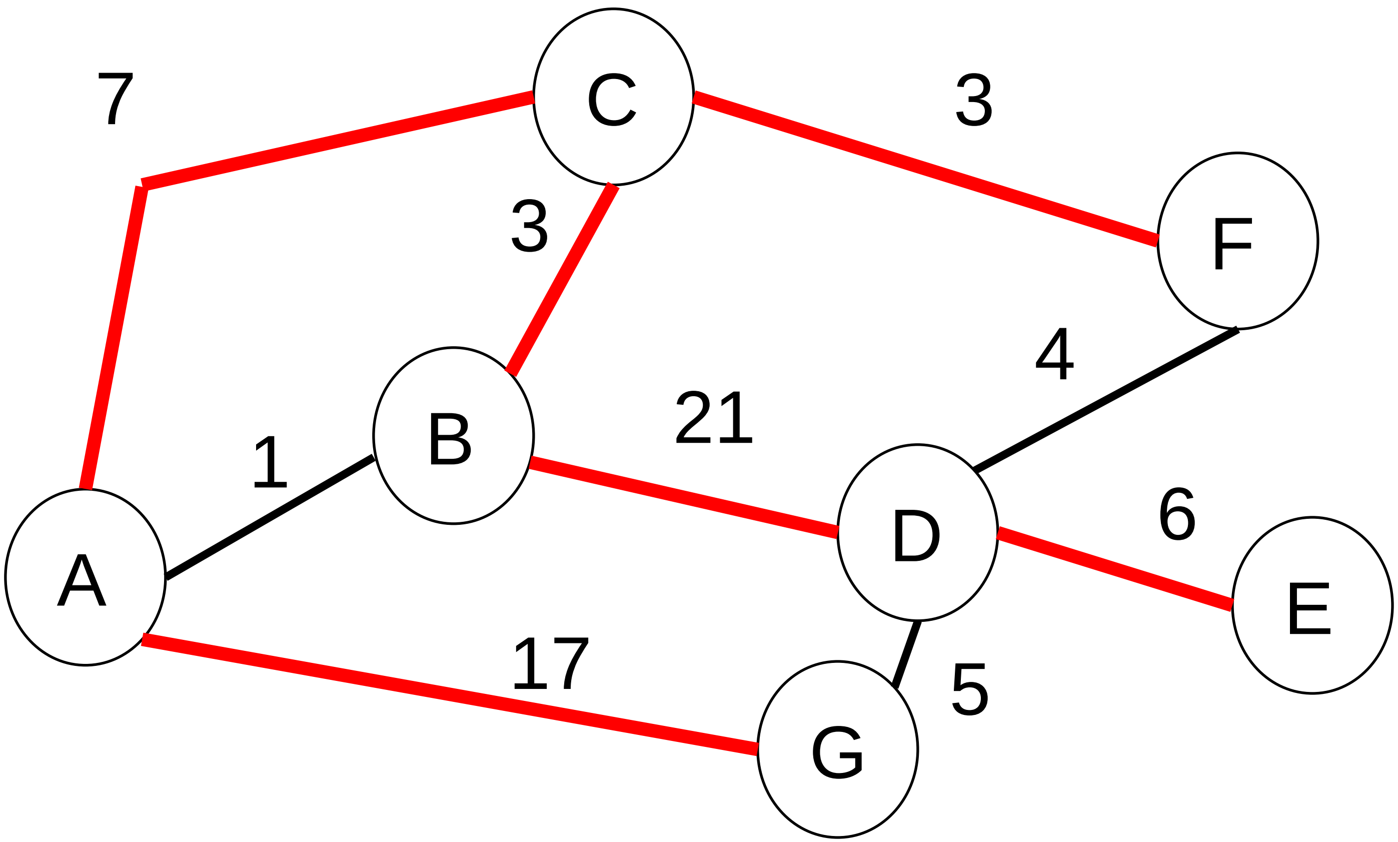
Each Vertex stores cost (distance) of lowest cost path from start Vertex to itself and previous vertex in path from start vertex to itself.

Alternatives to Dijkstra's Algorithm

- A*, pronounced "A Star"
- A heuristic, goal of finding shortest weighted path from single start vertex to goal vertex
- Uses actual distance like Dijkstra's but also estimates *remaining cost or distance*
  – *distance is set to current distance from start PLUS the **estimated distance** to the goal*
- For example when finding a path between towns, estimate the remaining distance as the straight-line (*as the crow flies)* distance between current location and goal.

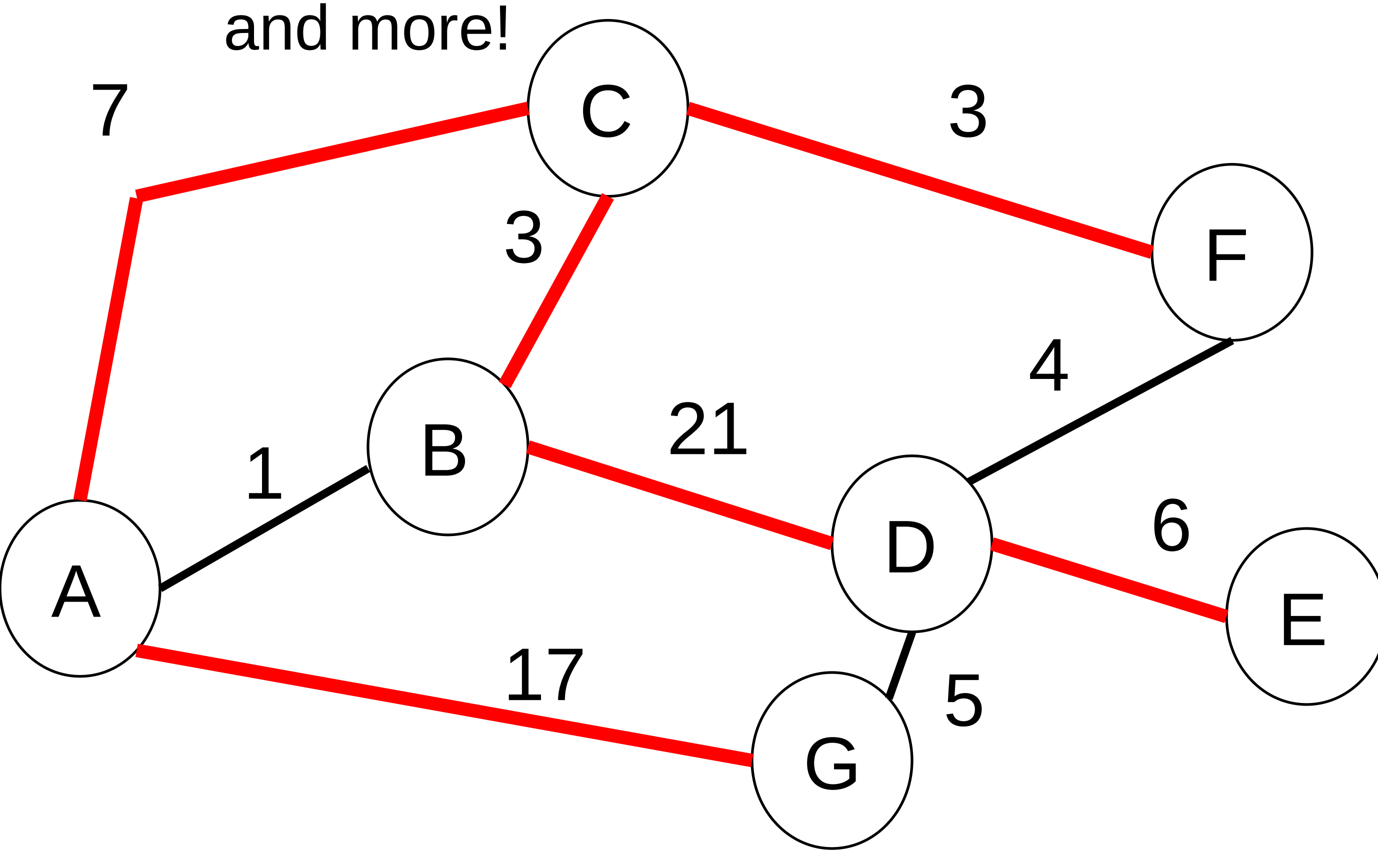
Spanning Tree

- *Spanning Tree:* A tree of edges that connects all the vertices in a graph



Clicker 7 -
Minimum Spanning Tree

- *Minimum Spanning Tree*: A spanning tree in a weighted graph with the lowest total cost
  - used in network design, taxonomy, Image registration, and more!



Cost of spanning tree shown?

A. 6

B. 7

C. 29

D. 61

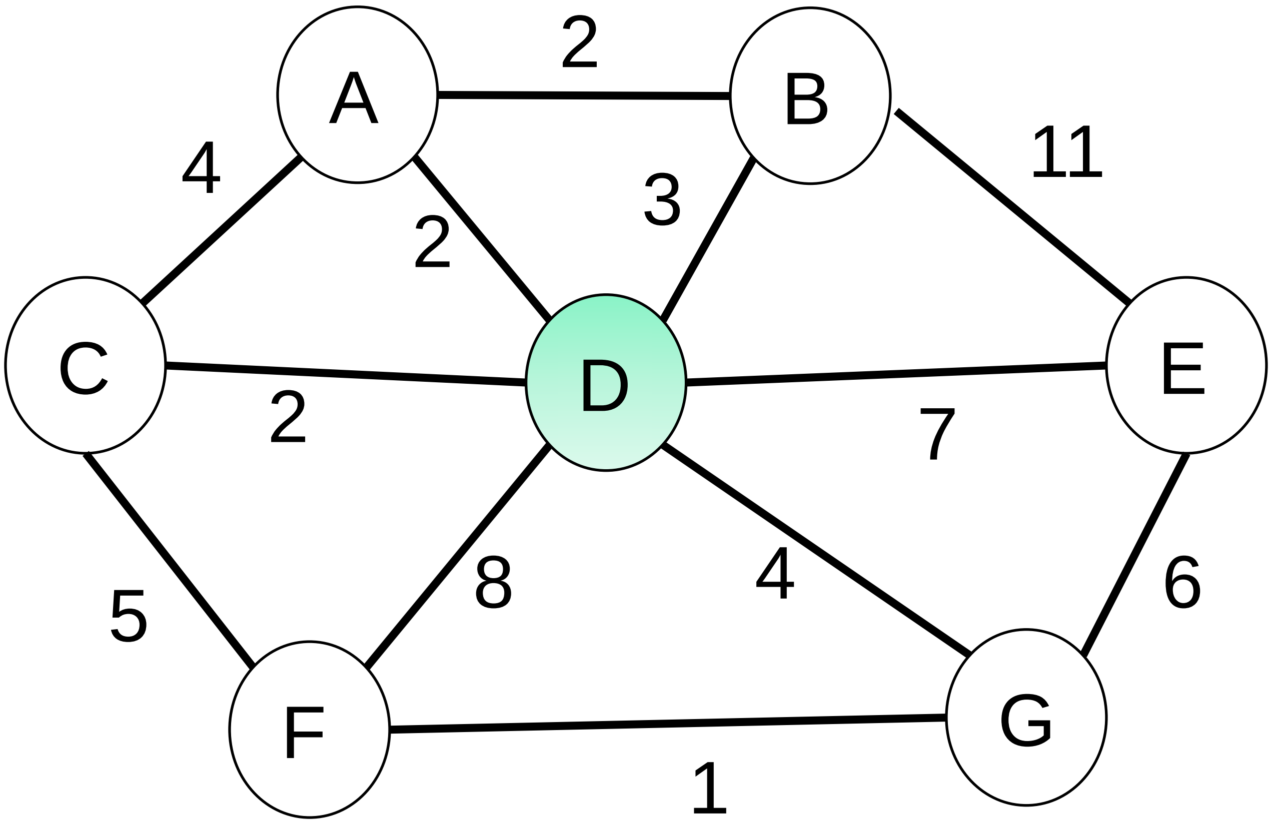
E. None of These

Is it a MST for the graph?

Prim's Algorithm

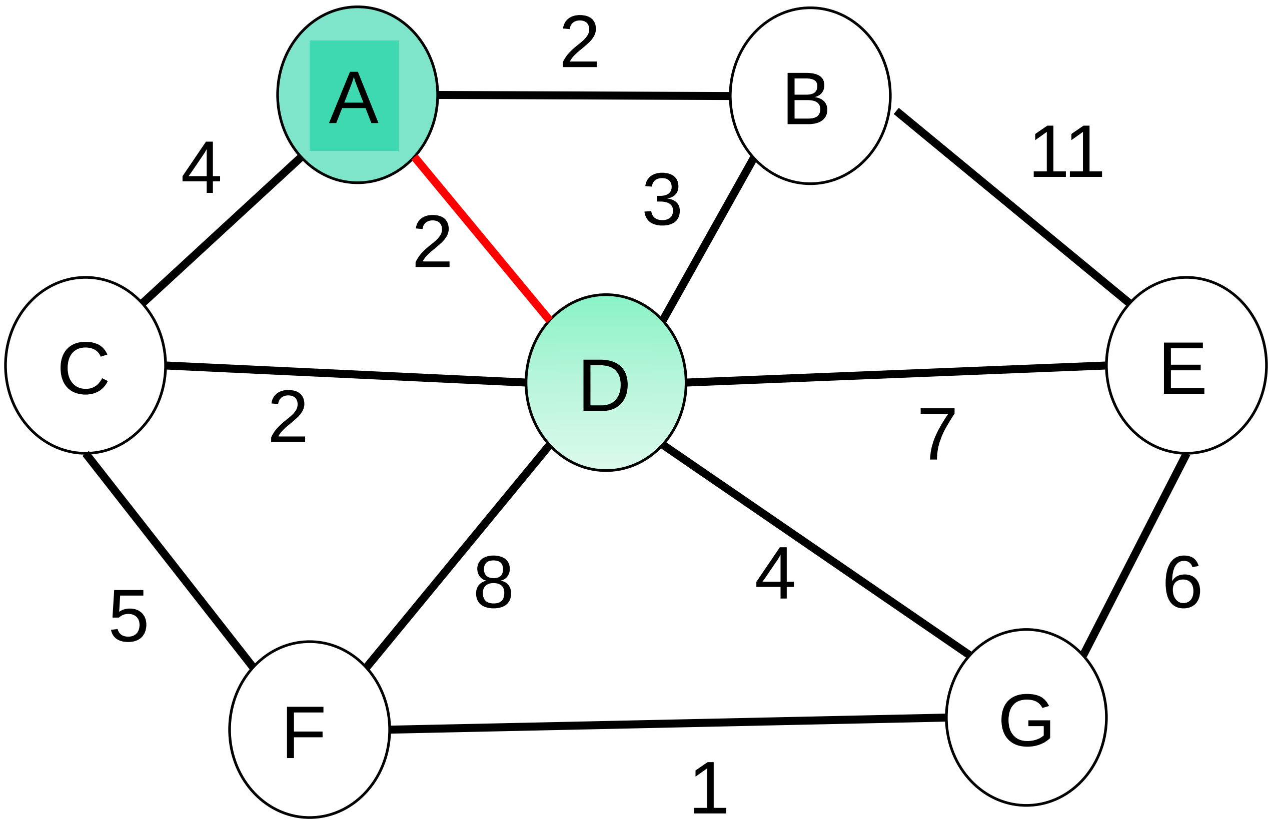
- Initially created by Vojtěch Jarník
- Rediscovered by Prim (of Sweetwater, TX) and Dijkstra
- Pick a vertex arbitrarily from graph
  - In other words, it doesn't matter which one
- Add lowest cost edge between the tree and a vertex that is not part of the tree UNTIL every vertex is part of the tree
- Greedy Algorithm, very similar to Dijkstra's

Prim's Algorithm



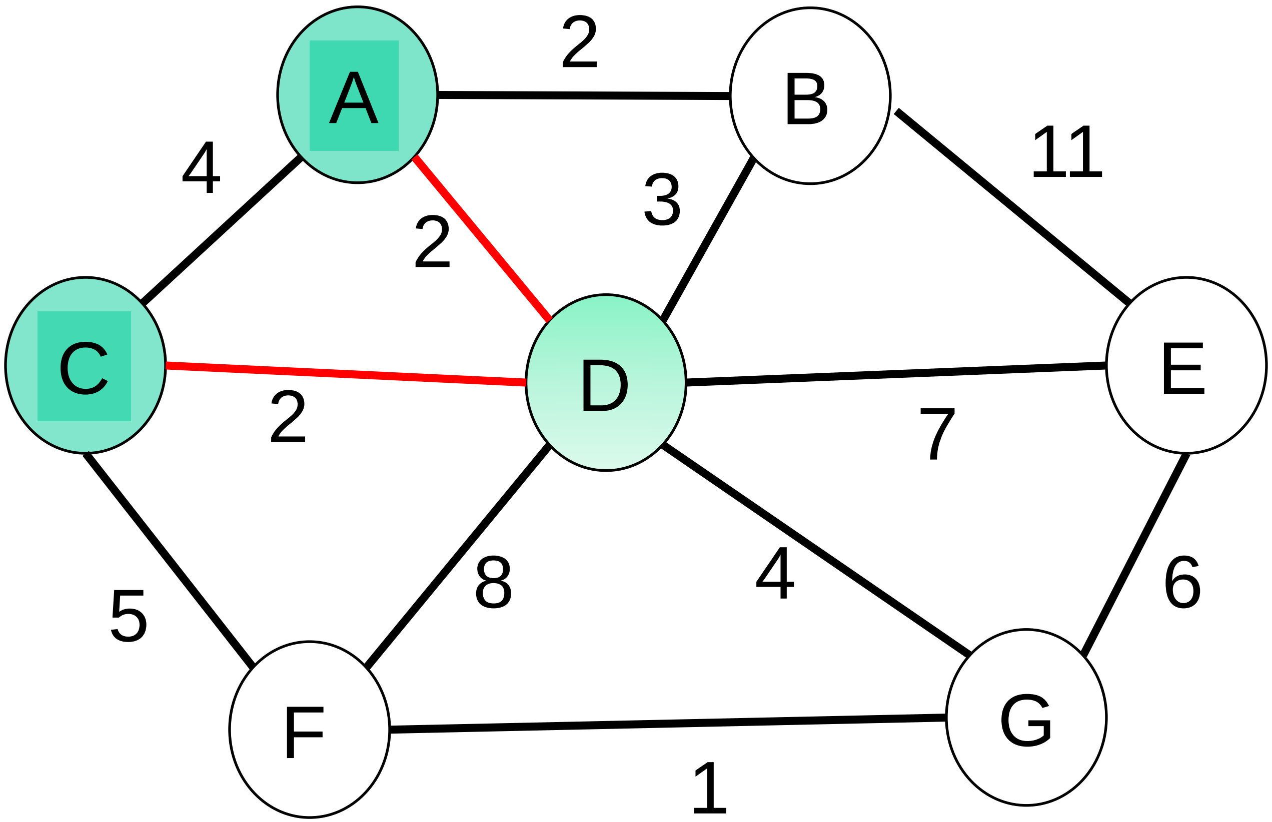
Pick D as root

Prim's Algorithm



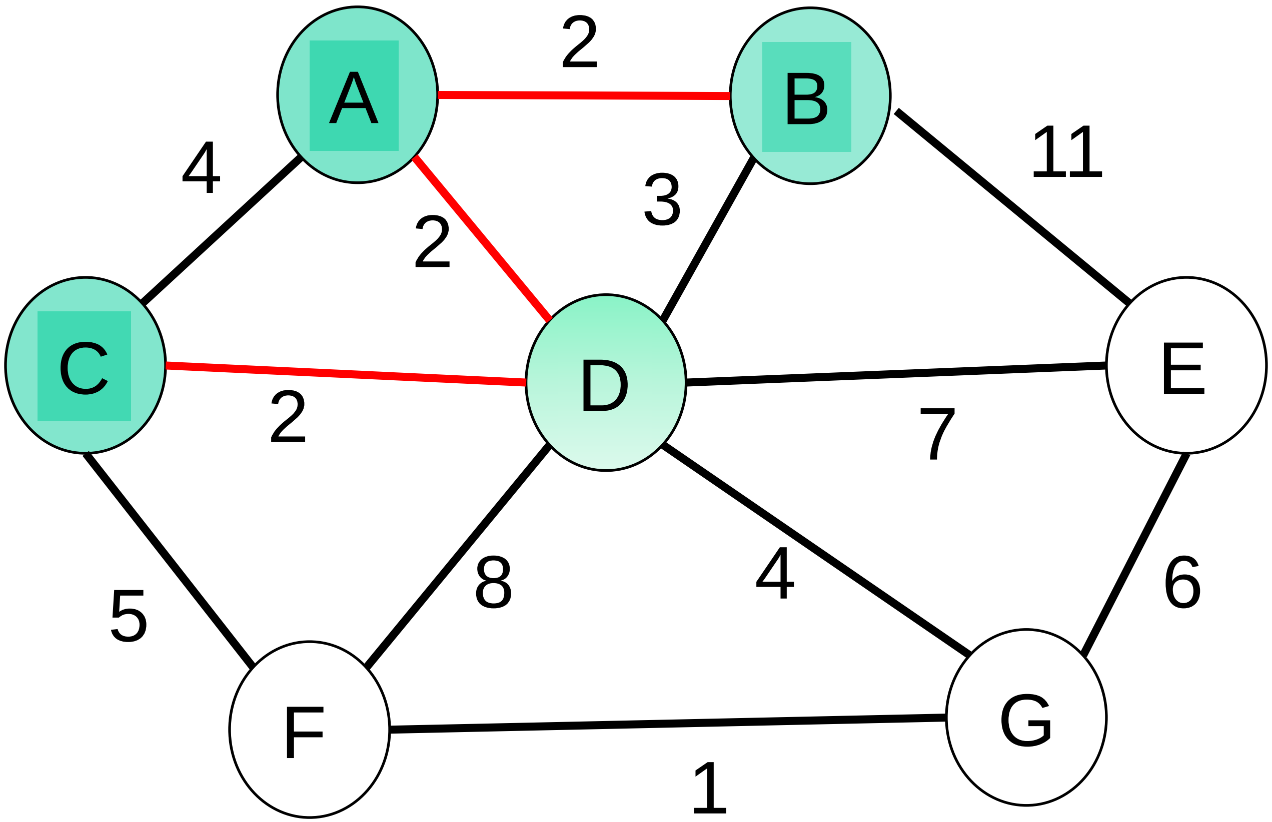
Lowest cost edge from tree to vertex not in Tree?
2 from D to A (or C)

Prim's Algorithm



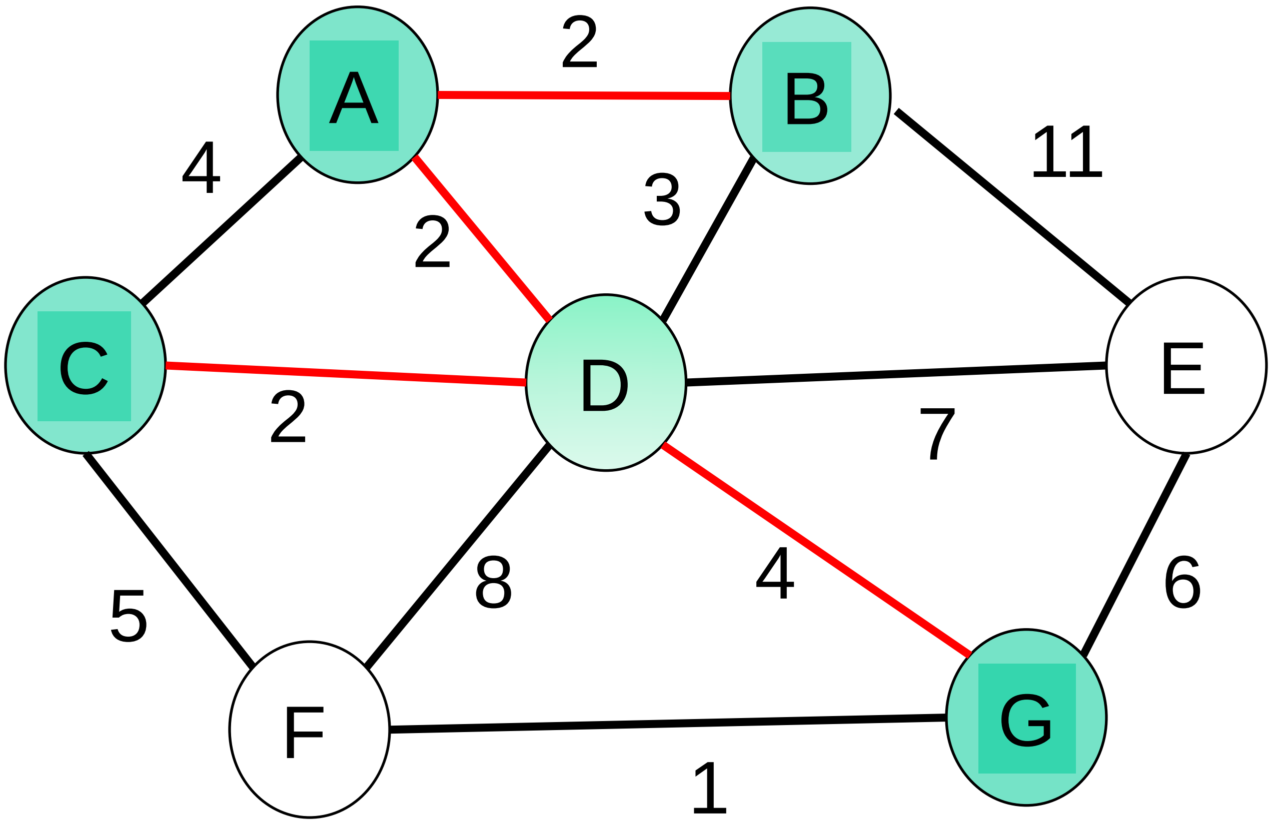
Lowest cost edge from tree to vertex not in Tree?
2 from D to C (OR from A to B)

Prim's Algorithm



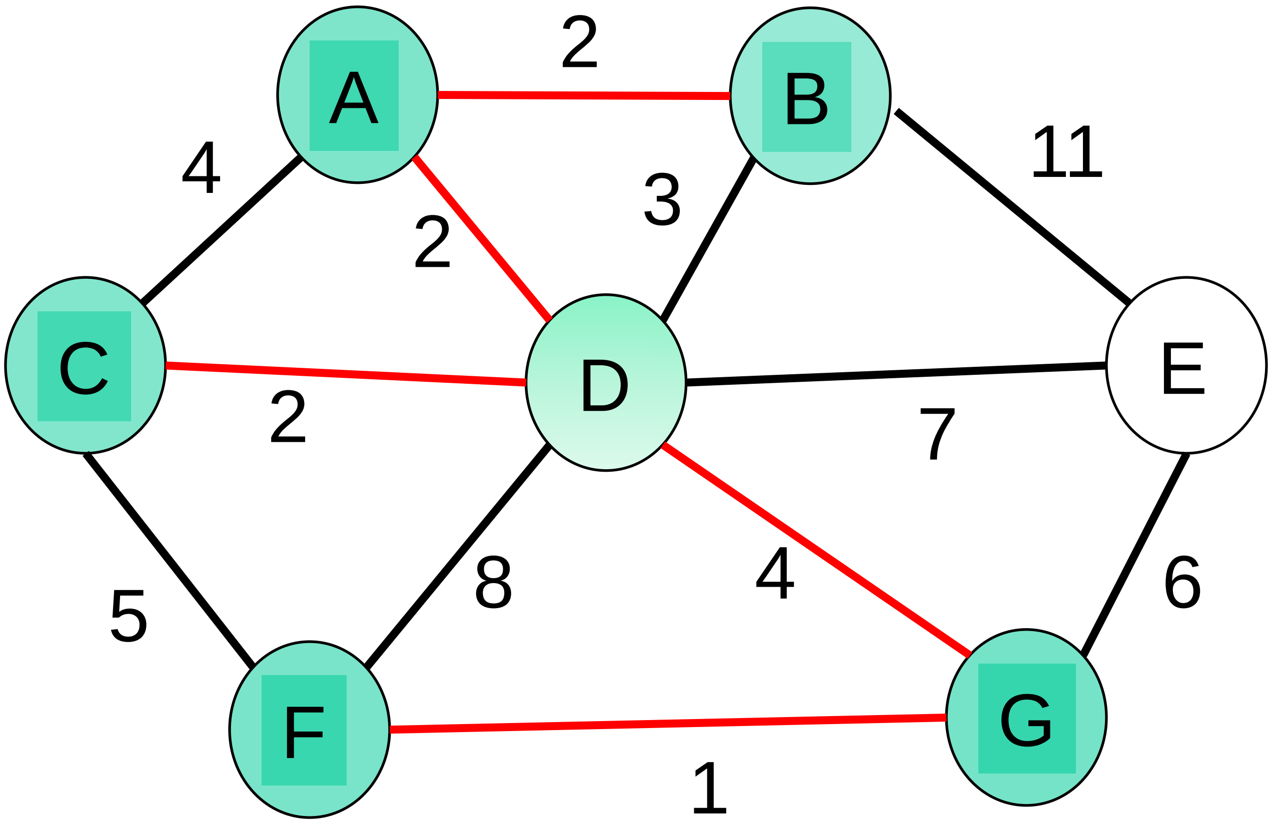
Lowest cost edge from tree to vertex not in Tree?
2 from A to B

Prim's Algorithm



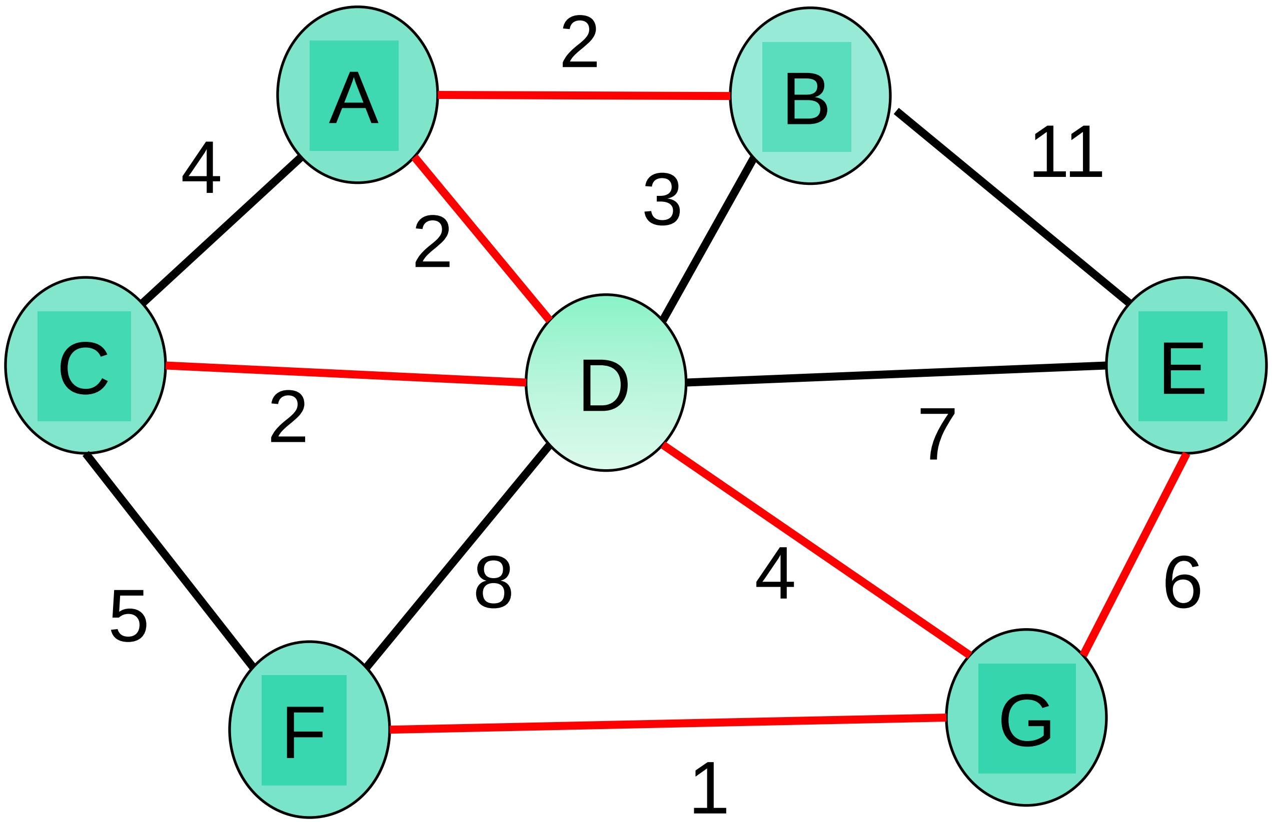
Lowest cost edge from tree to vertex not in Tree?
5 from D to G

Prim's Algorithm



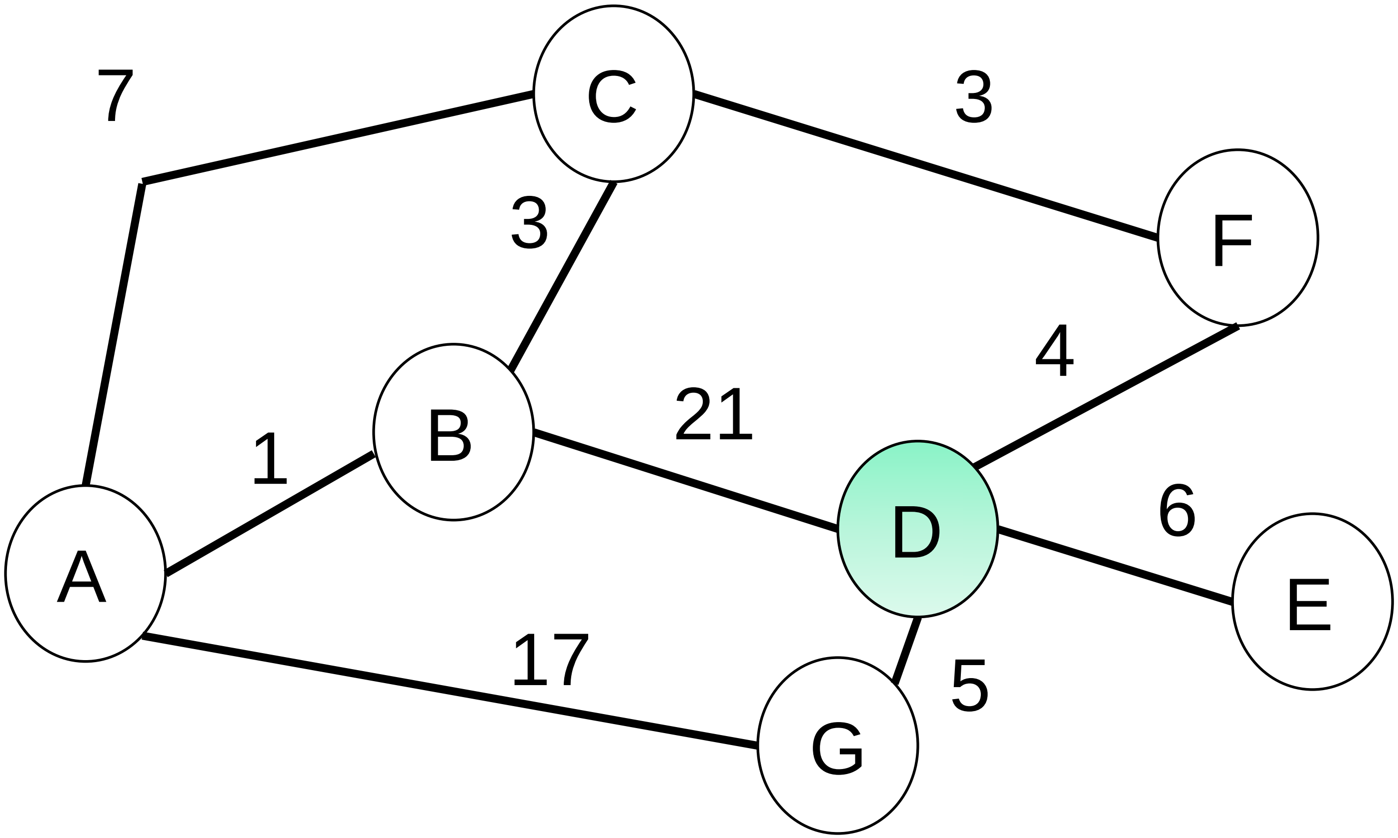
Lowest cost edge from tree to vertex not in Tree?
1 from G to F

Prim's Algorithm



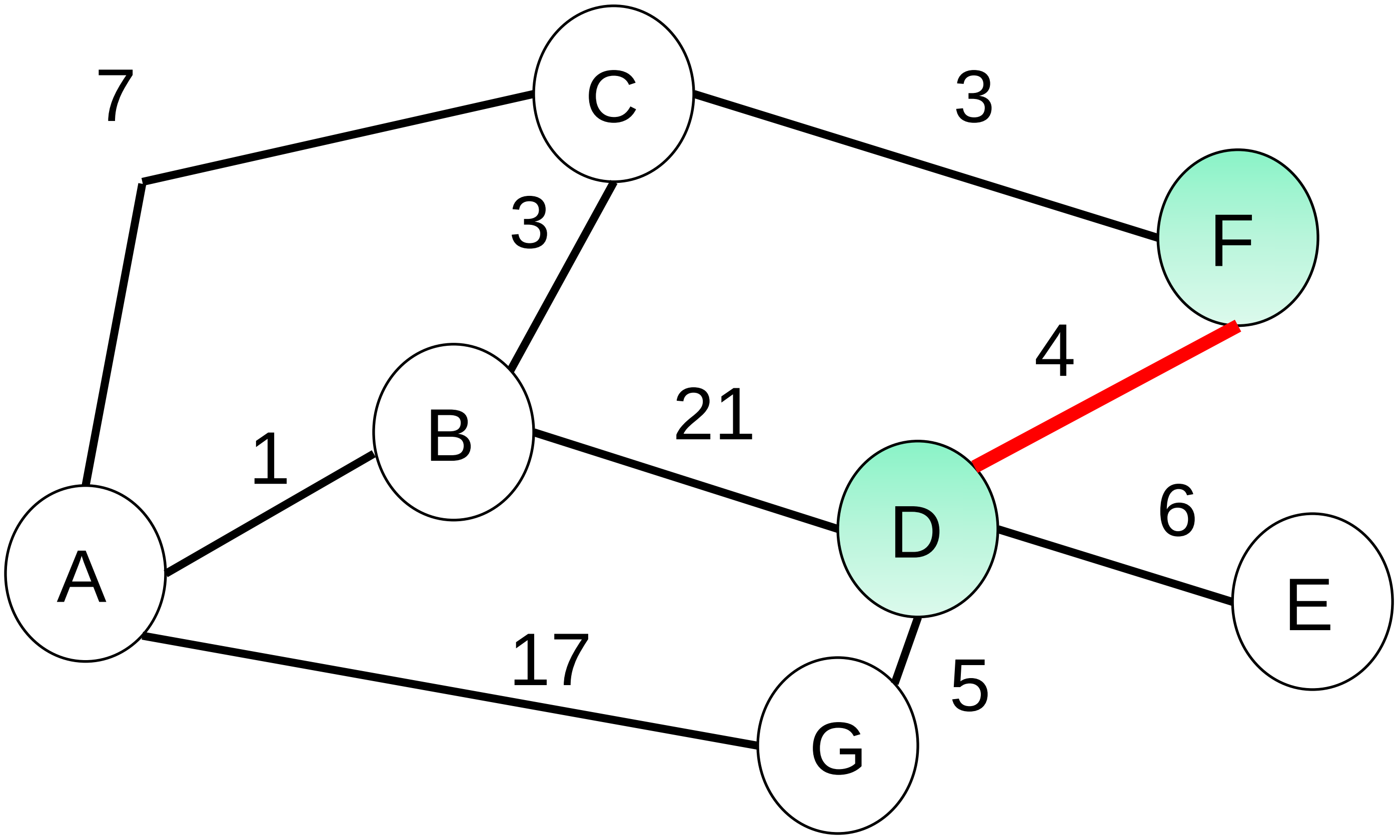
Lowest cost edge from tree to vertex not in Tree?
6 from G to E

Prim's Algorithm



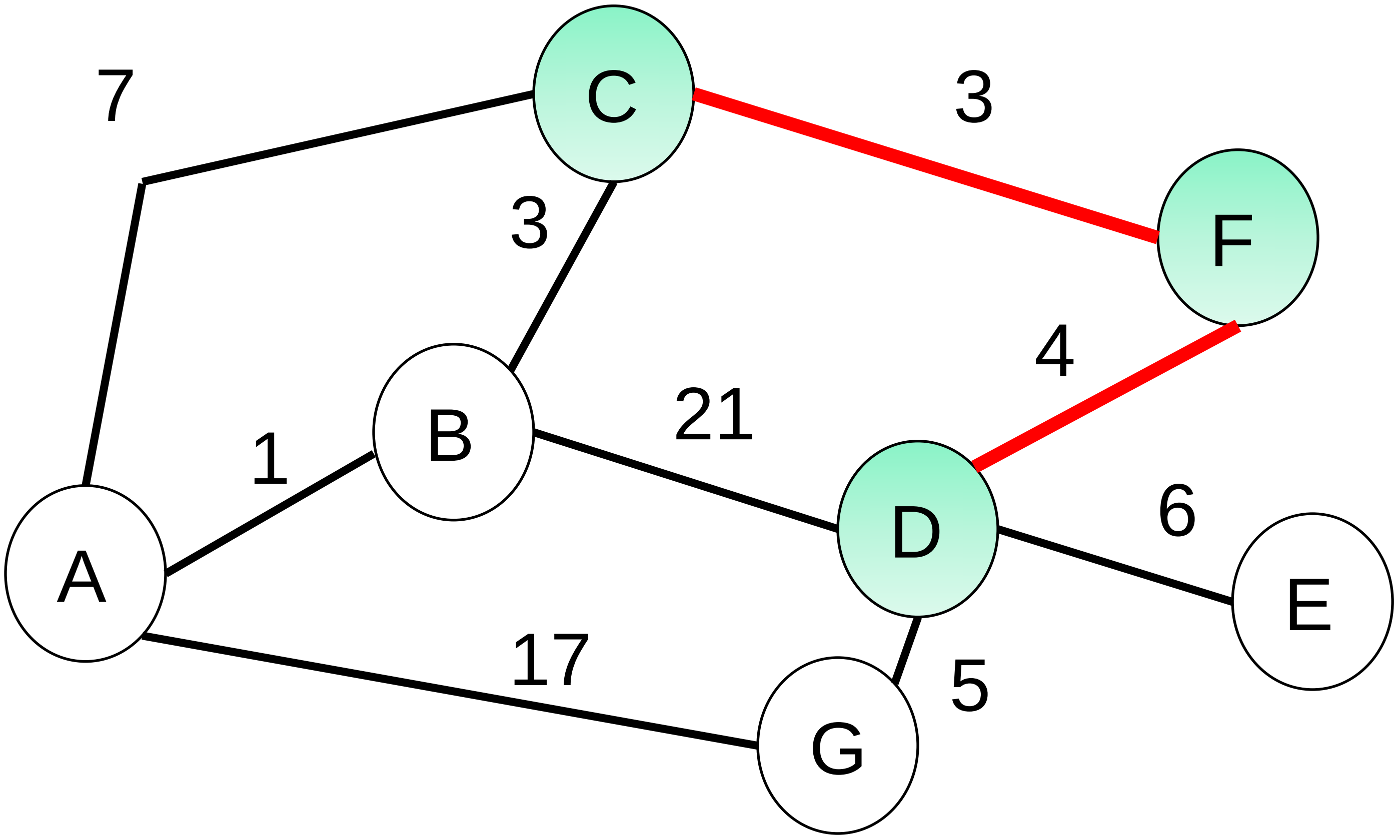
Pick D as root

Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?
4 from D to F

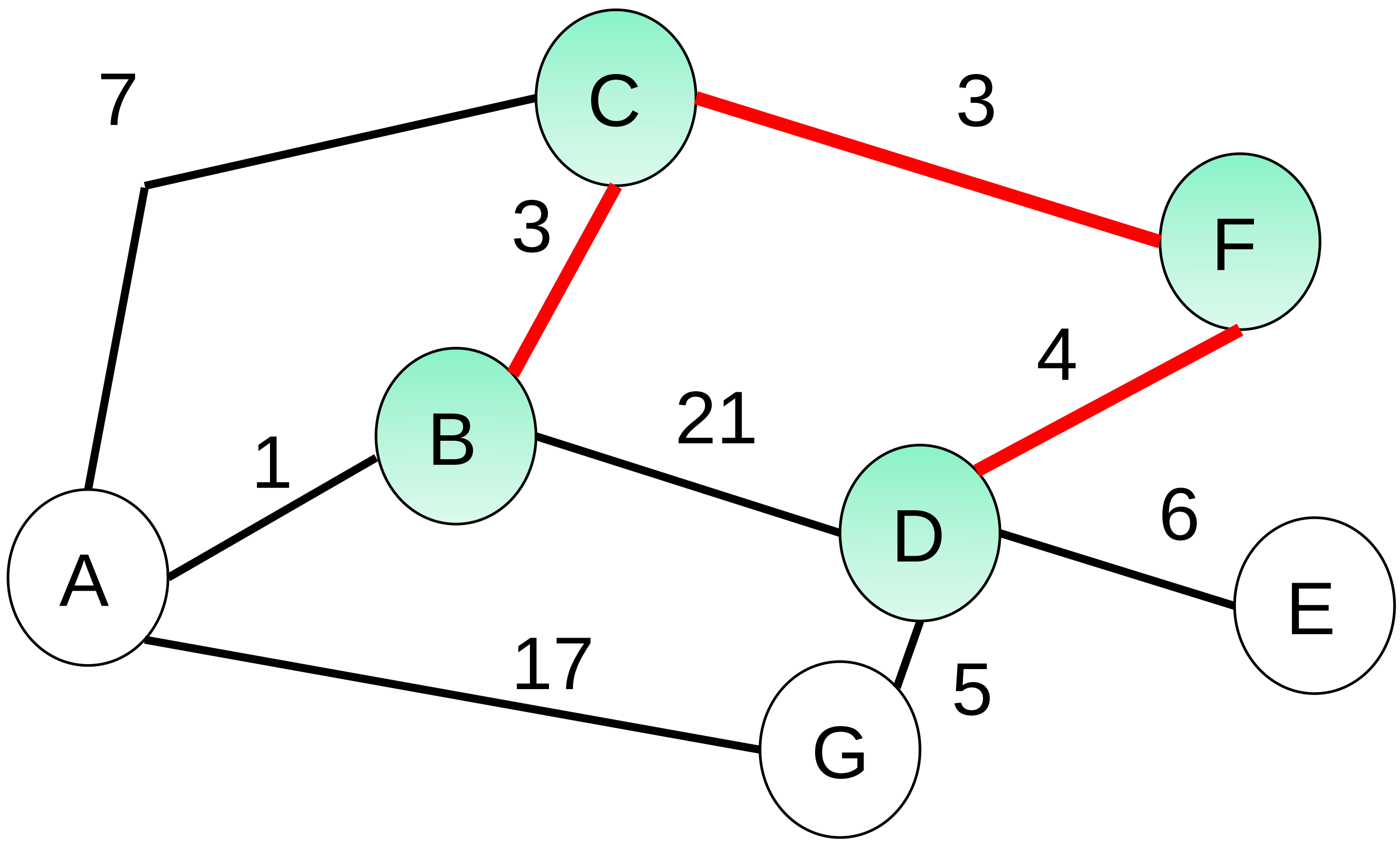
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

3 from F to C

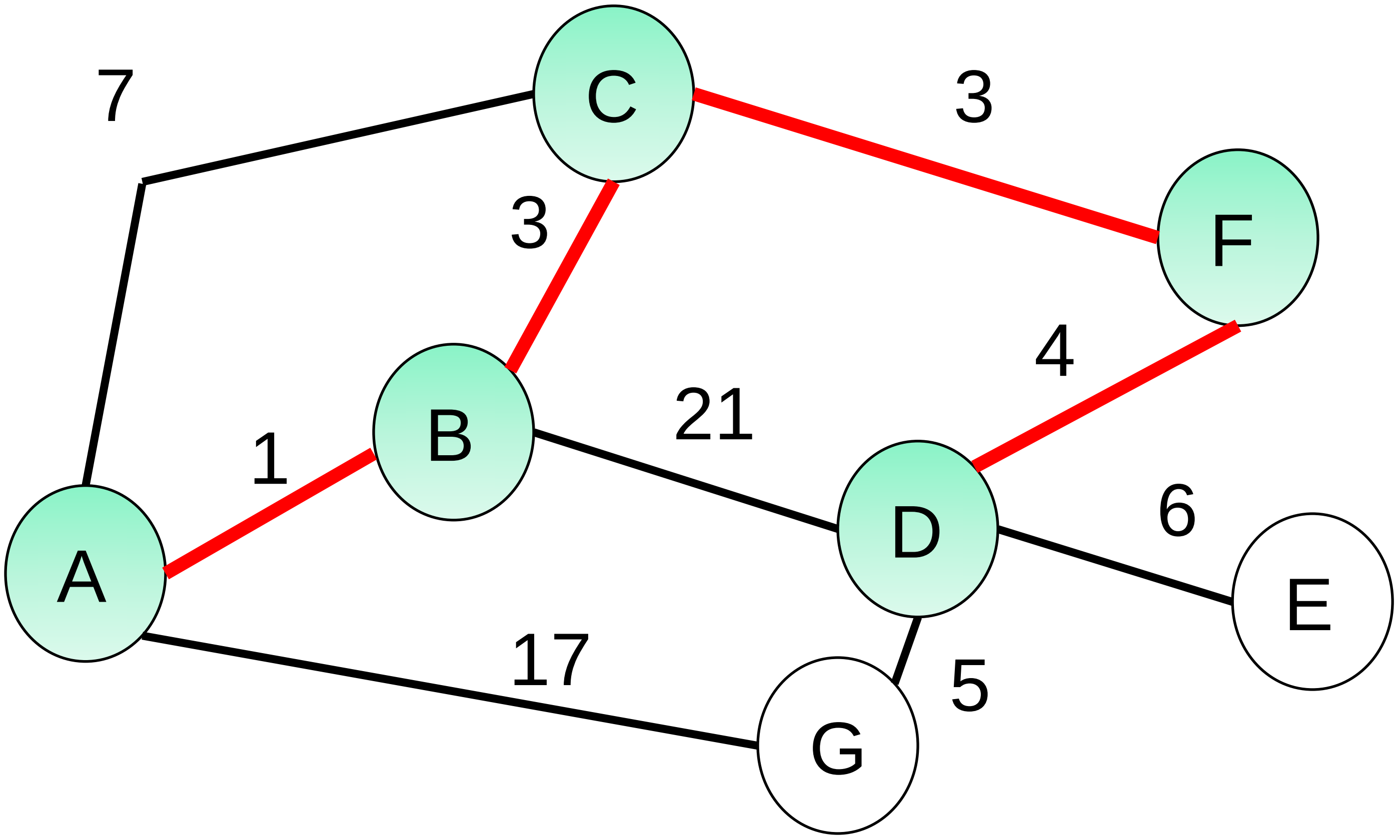
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

3 from C to B

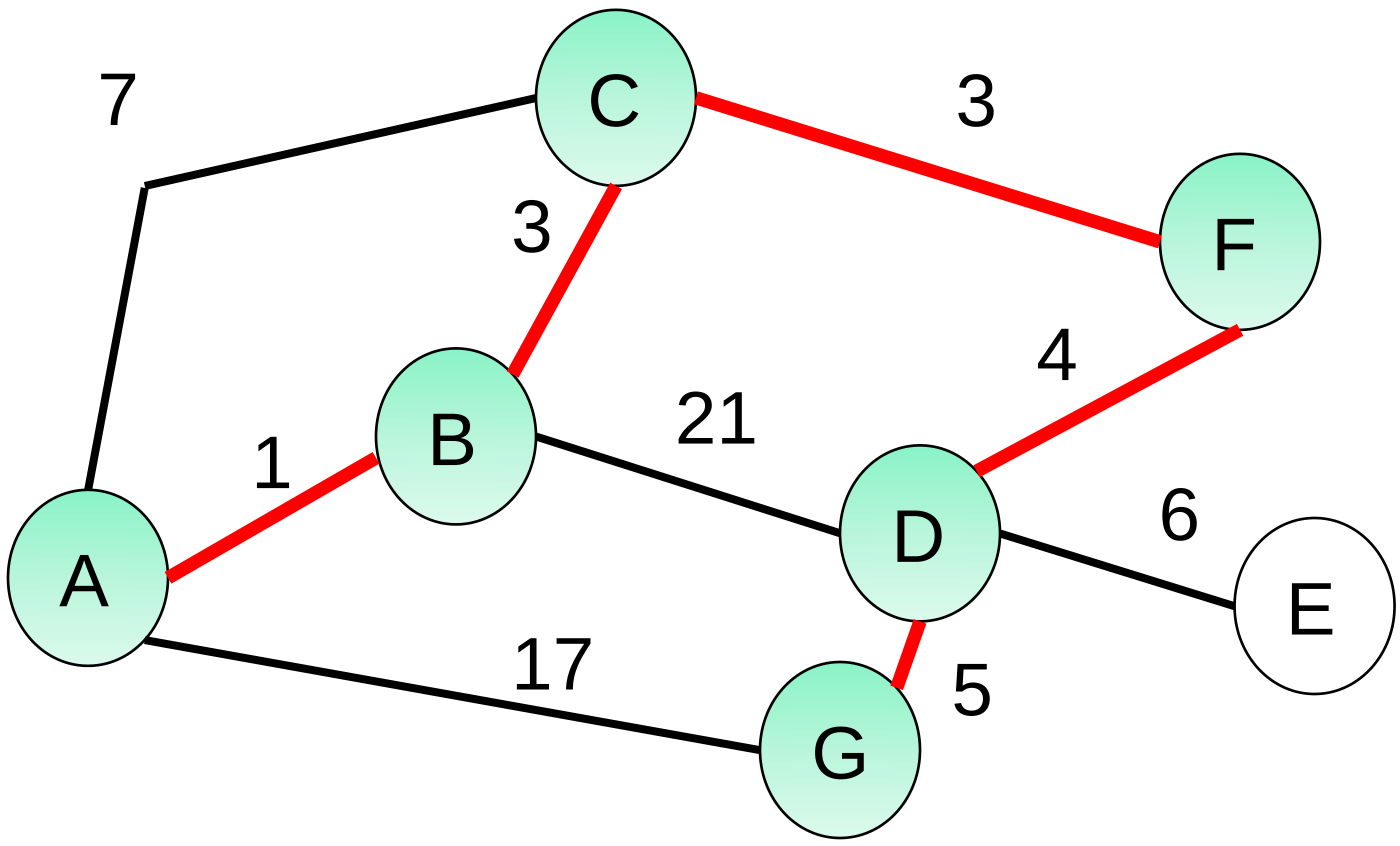
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

1 from B to A

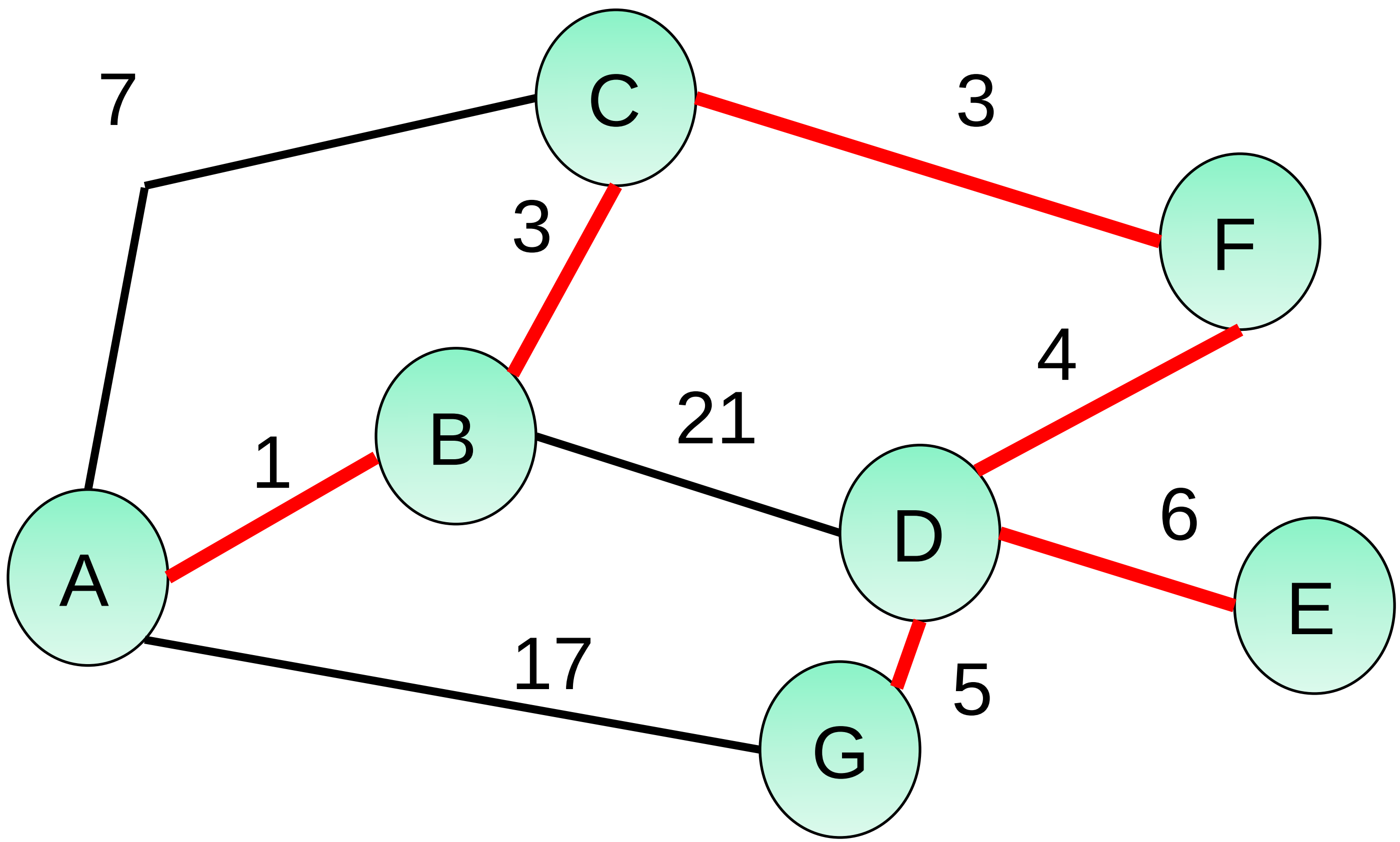
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

5 from D to G

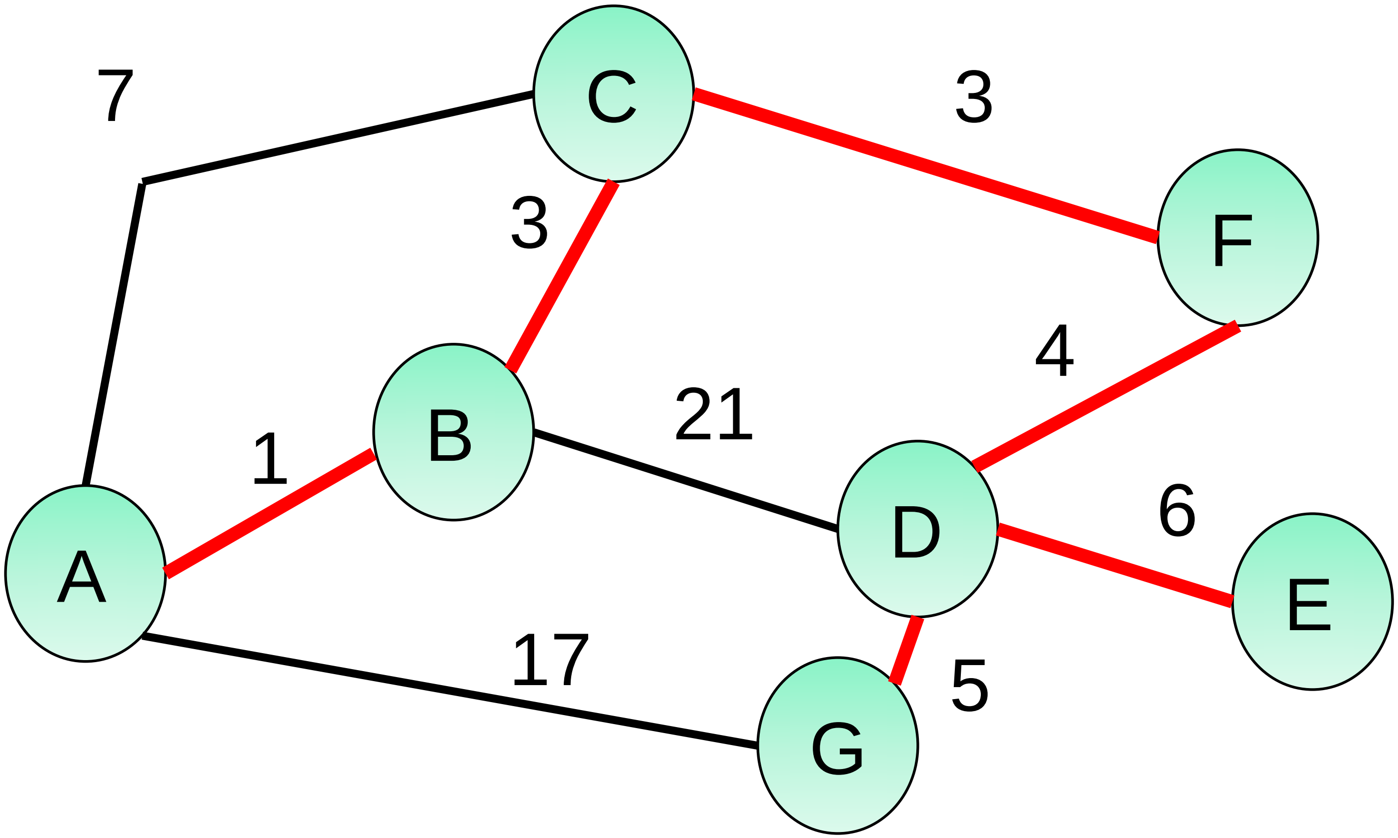
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

6 from D to E

Prim's Algorithm



Cost of Spanning Tree?

Other Graph Algorithms

- Lots!
- http://en.wikipedia.org/wiki/Category:Graph_algorithms

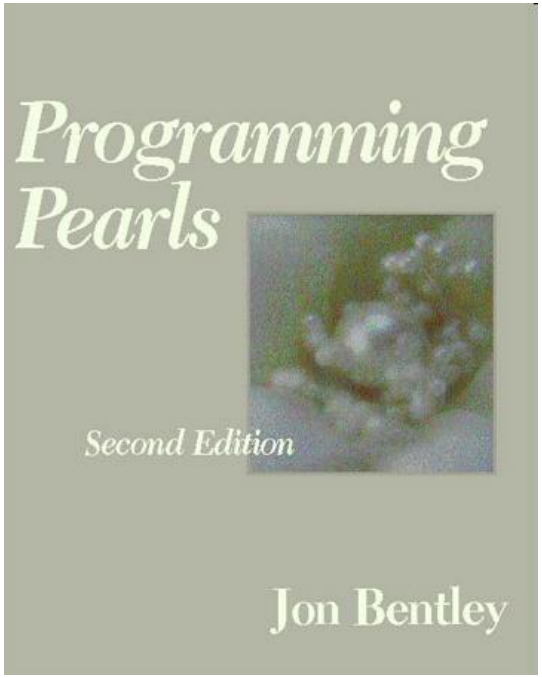
Topic 23
Hash Tables

"**hash collision** n. [from the techspeak] (var. `hash clash') When used of people, signifies a confusion in associative memory or imagination, especially a persistent one (see **thinko**).

True story: One of us was once on the phone with a friend about to move out to Berkeley. When asked what he expected Berkeley to be like, the friend replied: 'Well, I have this mental picture of naked people throwing Molotov cocktails, but I think that's just a collision in my hash tables.'"

-The Hacker's Dictionary

Programming Pearls by Jon Bentley



- Jon was *senior programmer* on a large programming project.
- Senior programmer spend a lot of time helping junior programmers.
- Junior programmer to Jon: "I need help writing a *sorting algorithm*."

A Problem

- From *Programming Pearls (Jon in Italics)*

Why do you want to write your own sort at all? Why not use a sort provided by your system?
I need the sort in the middle of a large system, and for obscure technical reasons, I can't use the system file-sorting program.
What exactly are you sorting? How many records are in the file? What is the format of each record?
The file contains at most ten million records; each record is a seven-digit integer.
Wait a minute. If the file is that small, why bother going to disk at all? Why not just sort it in main memory?
Although the machine has **many megabytes of main memory**, this function is part of a big system. I expect that I'll have only about a megabyte free at that point.
Is there anything else you can tell me about the records?
Each one is a seven-digit positive integer with no other associated data, and no integer can appear more than once.

System Sort

```
pisces% cat simple.txt
zoo
apple
bee
Apple
Zoo
Yacht
Soccer
2410 Speedway
Dorr
!!
pisces%
```

```
pisces% sort simple.txt
!!
2410 Speedway
apple
Apple
bee
Dorr
Soccer
Yacht
zoo
Zoo
```

Starting Other Programs

getRuntime

```
public static Runtime getRuntime()
```

Returns the runtime object associated with the current Java application. Most of the methods of class `Runtime` are instance methods and must be invoked with respect to the current runtime object.

Returns:

the `Runtime` object associated with the current Java application.

Starting Other Programs

exec

```
public Process exec(String command) throws IOException
```

Executes the specified string command in a separate process.

This is a convenience method. An invocation of the form `exec(command)` behaves in exactly the same way as the invocation `exec(command, null, null)`.

Parameters:

`command` - a specified system command.

Returns:

A new `Process` object for managing the subprocess

Clicker 1 and 2

- When did this conversation take place?

A. circa 1965

B. circa 1975

C. circa 1985

D. circa 1995

E. circa 2005

- What were they sorting?

A. SSNs. B. Random values C. Street Addresses

D. Personal Incomes   E. Phone Numbers

A Solution

```
/* phase 1: initialize set to empty */
for i = [0, n)
    bit[i] = 0

/* phase 2: insert present elements into the set */
for each num_in_file in the input file
    bit[num_in_file] = 1

/* phase 3: write sorted output */
for i = [0, n)
    if bit[i] == 1 write i on the output file
```

Some Structures so Far

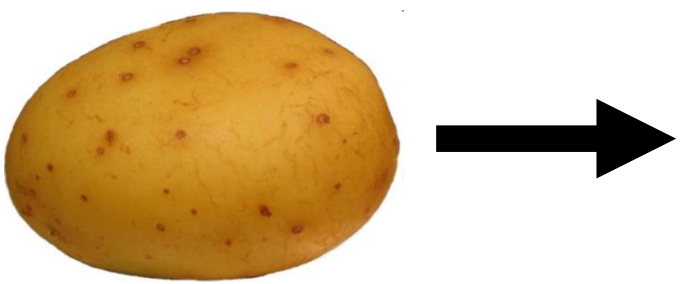
- ArrayLists
  – O(1) access
  – O(N) insertion (average case), better at end
  – O(N) deletion (average case)
- LinkedLists
  – O(N) access
  – O(N) insertion (average case), better at front and back
  – O(N) deletion (average case), better at front and back
- Binary Search Trees
  – O(log N) access if balanced
  – O(log N) insertion if balanced
  – O(log N) deletion if balanced

Why are Binary Trees Better?

- Divide and Conquer - splitting problem into smaller problems
- Can we reduce the work by a bigger factor? 3? 10? More?
- An ArrayList does this in a way when *accessing* elements
  - *but must use an integer value*
  - *each position holds a single element*
  - ***given the index in an array, I can access that element rather quickly***
  - ***determining the address of the element requires a multiply op and an add op***

Hash Tables

- Hash Tables maintaining the fast access of arrays but improve the order for insertion, and deletion compare to array based lists.



- Hash tables use an *array* and *hash functions* to determine the index for each element.

Hash Functions

- Hash: "From the French hatcher, which means 'to chop'. "
- *to hash* to mix randomly or shuffle (To cut up, to slash or hack about; to mangle)
- Hash Function: Take a piece of data and transforms it to a different piece of data (typically smaller), usually a single integer.
  - A function or algorithm
  - The input need not be integers!

Hash Function

5/17/1971

555389085

Manchester, VT

"Mike Scott"

mscott61729@gmail.com

"Kelly"

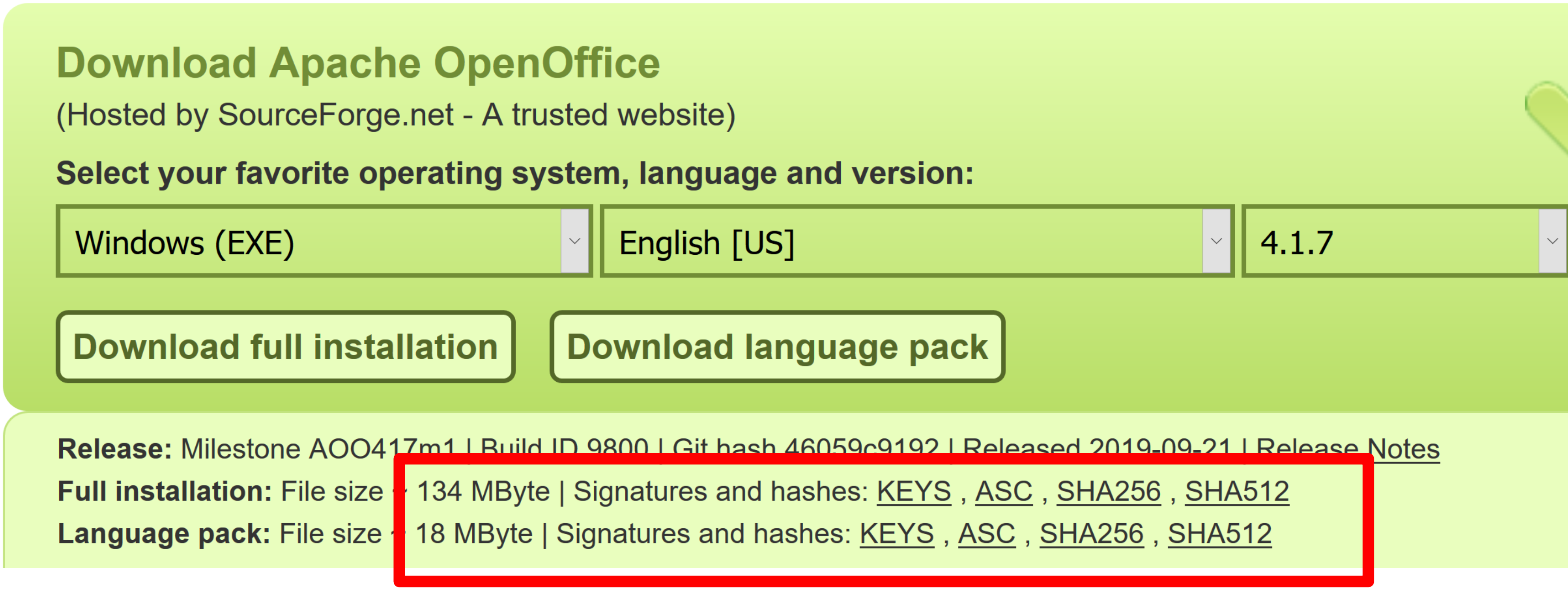
"Olivia"

"Isabelle"

$\xrightarrow[\text{hash function}]{}$ 12

Hash Functions

- Like a fingerprint



- 134 Megabytes

Hash Function

- SHA 512 Hash code

```
c3e6ef13fb80e34981304754717 4a35367fb19ee
0ba704a27ab748ed99a646367 1bcd1bef348bb42
a5e883301ef786970fe303d0fd3f677f8d0a8fd0
4aeba798
*Apache_OpenOffice_4.1.7_Win_x86_install
_en-US.exe
```

Simple Example

- Assume we are using names as our *key*
  - *take 3rd letter of name, take int value of letter (a = 0, b = 1, ...), divide by 6 and take remainder*
- What does "Bellers" hash to?
- L -> 11 -> 11 % 6 = 5

Result of Hash Function

- Mike = (10 % 6) = 4
- Kelly = (11 % 6) = 5
- Olivia = (8 % 6) = 2
- Isabelle = (0 % 6) = 0
- David = (21 % 6) = 3
- Margaret = (17 % 6) = 5 (uh oh)
- Wendy = (13 % 6) = 1
- This is an imperfect hash function. A perfect hash function yields a one to one mapping from the keys to the hash values.
- What is the maximum number of values this function can hash perfectly?

Clicker 3 - Hash Function

- Assume the hash function for String adds up the Unicode value for each character.

```
public static int hashCode(String s) {
    int result = 0;
    for (int i = 0; i < s.length(); i++)
        result += s.charAt(i);
    return result;
}
```

- Hashcode for "DAB" and "BAD"?

A. 301 103
B. 4 4
C. 412 214
D. 5 5
E. 199 199

More on Hash Functions

- transform the key (which may not be an integer) into an integer value
- The transformation can use one of four techniques
  – Mapping
  – Folding
  – Shifting
  – Casting

Hashing Techniques

- Mapping
  - As seen in the example
  - integer values or things that can be easily converted to integer values in key
- Folding
  - partition key into several parts and the integer values for the various parts are combined
  - the parts may be hashed first
  - combine using addition, multiplication, shifting, logical exclusive OR

Shifting

- More complicated with shifting

```
int hashVal = 0;
int i = str.length() - 1;
while(i > 0)
{ hashVal = (hashVal << 1) + (int) str.charAt(i);
  i--;
}
```

different answers for "dog" and "god"

Shifting may give a better range of hash values when compared to just folding

Casts

- Very simple
  - essentially casting as part of fold and shift when working with chars.

The Java String class hashCode method

```
public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        char[] val = value;
        for (int i = 0; i < val.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}
```

Mapping Results

- Transform hashed key value into a legal index in the hash table
- Hash table is normally uses an array as its underlying storage container
- Normally get location on table by taking result of hash function, dividing by size of table, and taking remainder

  index = key mod n

  n is size of hash table

  empirical evidence shows a prime number is best

  10 element hash table, move up to 11 or 13 elements

Mapping Results

"Isabelle" → 230492619

hashCode method

230492619 % 997 = 177

0 1 2 3 ....177............ 996

"Isabelle"

Handling Collisions

- What to do when inserting an element and already something present?

Open Addressing

- Could search forward or backwards for an open space
- Linear probing:
  - move forward 1 spot. Open?, 2 spots, 3 spots
  - reach the end?
  - When removing, insert a blank
  - null if never occupied, blank if once occupied
- Quadratic probing
  - 1 spot, 2 spots, 4 spots, 8 spots, 16 spots
- Resize when *load factor* reaches some limit

Closed Addressing: Chaining

- Each element of hash table be another data structure
  - linked list, balanced binary tree
  - More space, but somewhat easier
  - everything goes in its spot
- What happens when resizing?
  - Why don't things just collide again?

Hash Tables in Java

- `hashCode` method in `Object`
- `hashCode` and `equals`
  - "If two objects are equal according to the equals (`Object`) method, then calling the `hashCode` method on each of the two objects must produce the same integer result. "
  - if you override `equals` you need to override `hashCode`
- Overriding one of `equals` and `hashCode`, but not the other, can cause logic errors that are difficult to track down if objects added to hash tables.

Hash Tables in Java

- HashTable class
- HashSet class
  - implements Set interface with internal storage container that is a HashTable
  - compare to TreeSet class, internal storage container is a Red Black Tree
- HashMap class
  - implements the Map interface, internal storage container for keys is a hash table

Comparison

- Compare these data structures for speed:
- Java HashSet
- Java TreeSet
- our naïve Binary Search Tree
- our HashTable
- Insert random ints

Clicker 4

- What will be order from fastest to slowest?

A. HashSet TreeSet HashTable314 BST

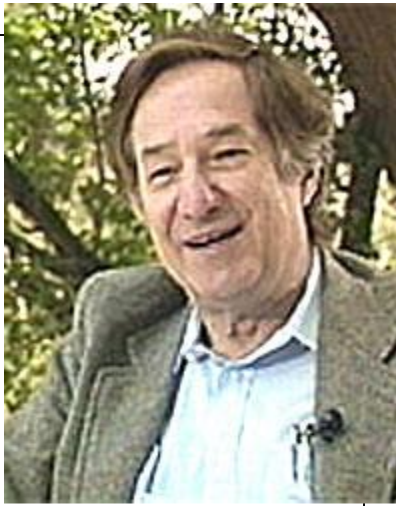
B. HashSet HashTable314 TreeSet BST

C. TreeSet HashSet BST HashTable314

D. HashTable314 HashSet BST TreeSet

E. None of these

Topic 24
Tries



"In 1959, (Edward) Fredkin recommended that BBN (Bolt, Beranek and Newman, now BBN Technologies) purchase the very first PDP-1 to support research projects at BBN. ***The PDP-1 came with no software whatsoever.***

Fredkin wrote a PDP-1 assembler called FRAP (Free of Rules Assembly Program);"

Tries were first described by René de la Briandais in *File searching using variable length keys*.

Clicker 1

- How would you pronounce “Trie”

A. “tree”

B. “tri – ee”

C. “try”

D. “tiara”

E. something else

Tries aka Prefix Trees

- Pronunciation:
- From re**<u>trie</u>**val
- Name coined by Computer Scientist Edward Fredkin
- Re**trie**val so "tree"
- … but that is very confusing so most people pronounce it "try"

Predictive Text and AutoComplete

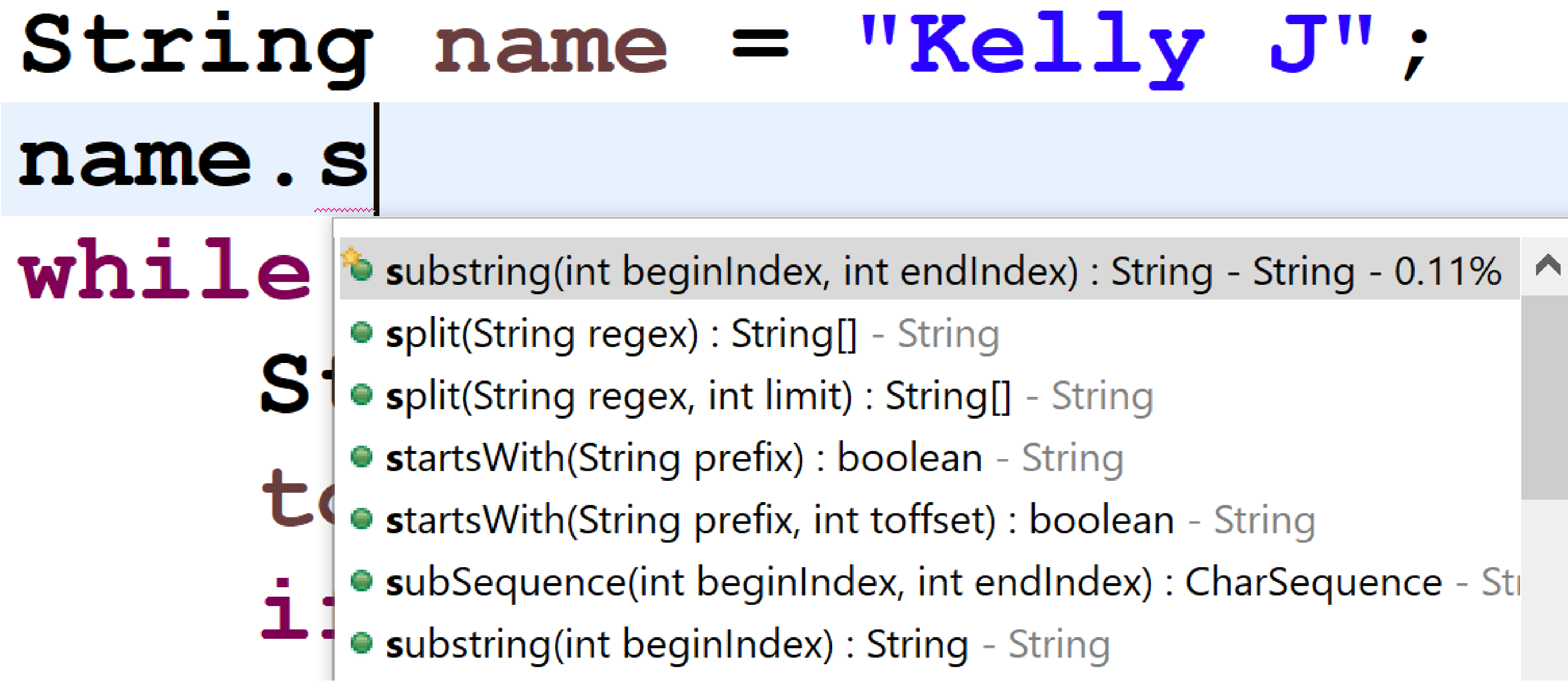
- Search engines and texting applications guess what you want after typing only a few characters

Hel

hel**lo**
hel**lboy**
hel**lo fresh**
hel**en keller**
hel**ena christensen**
hel**lo may**
hel**l or high water**
hel**lo neighbor**
hel**zberg**
hel**p synonym**

AutoComplete

- So do other programs such as IDEs



Searching a Dictionary

- How?
- Could search a set for all values that start with the given prefix.
- Naively O(N) (search the whole data structure).
- Could improve if possible to do a binary search for prefix and then localize search to that location.

Tries

- A general tree (more than 2 children possible)
- Root node (or possibly a list of root nodes)
- Nodes can have many children
  – not a binary tree
- In simplest form each node stores a character and a data structure (list?) to refer to its children
- "Stores" all the words or phrases in a dictionary.
- How?

René de la Briandais Original Paper

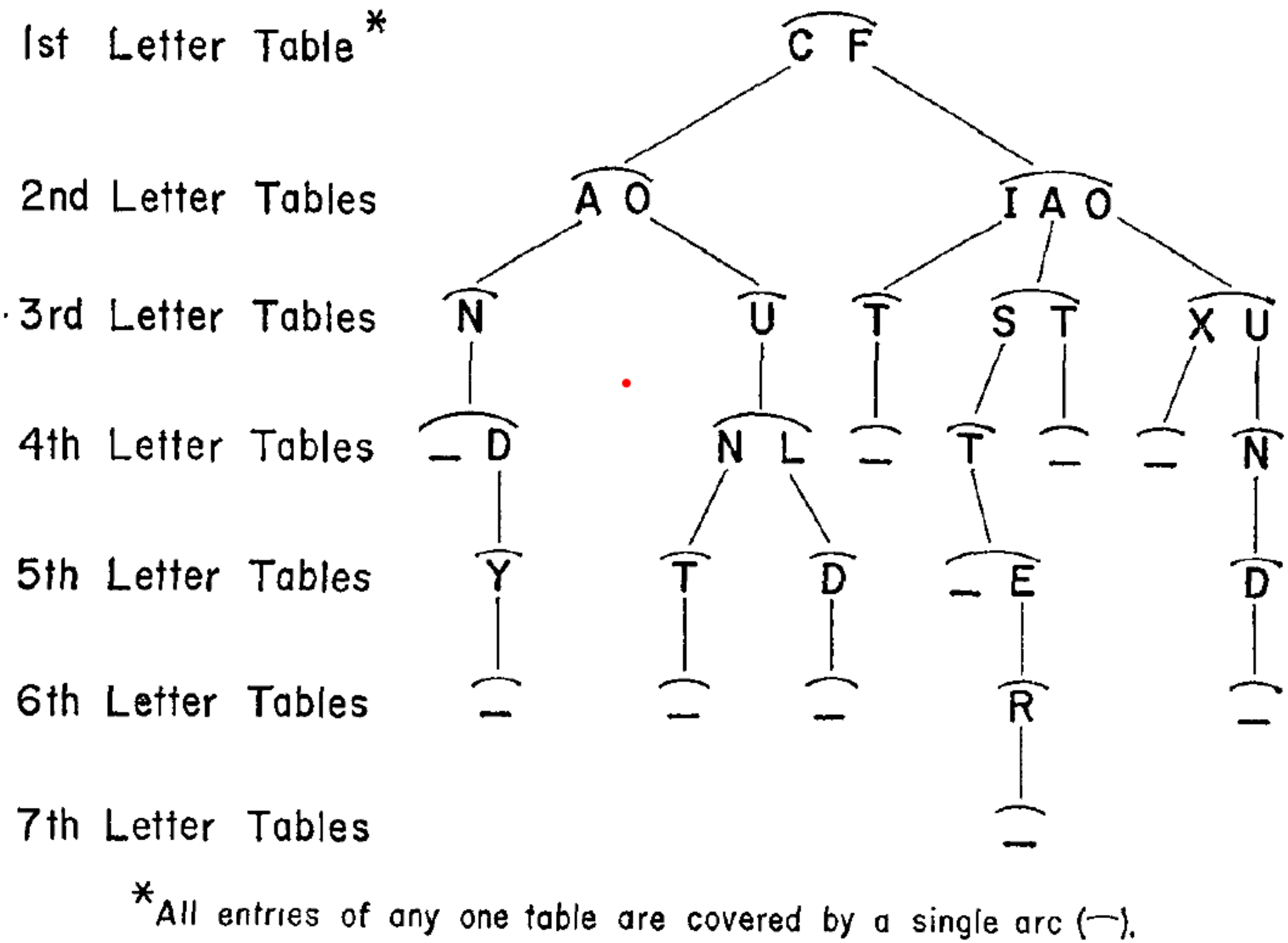
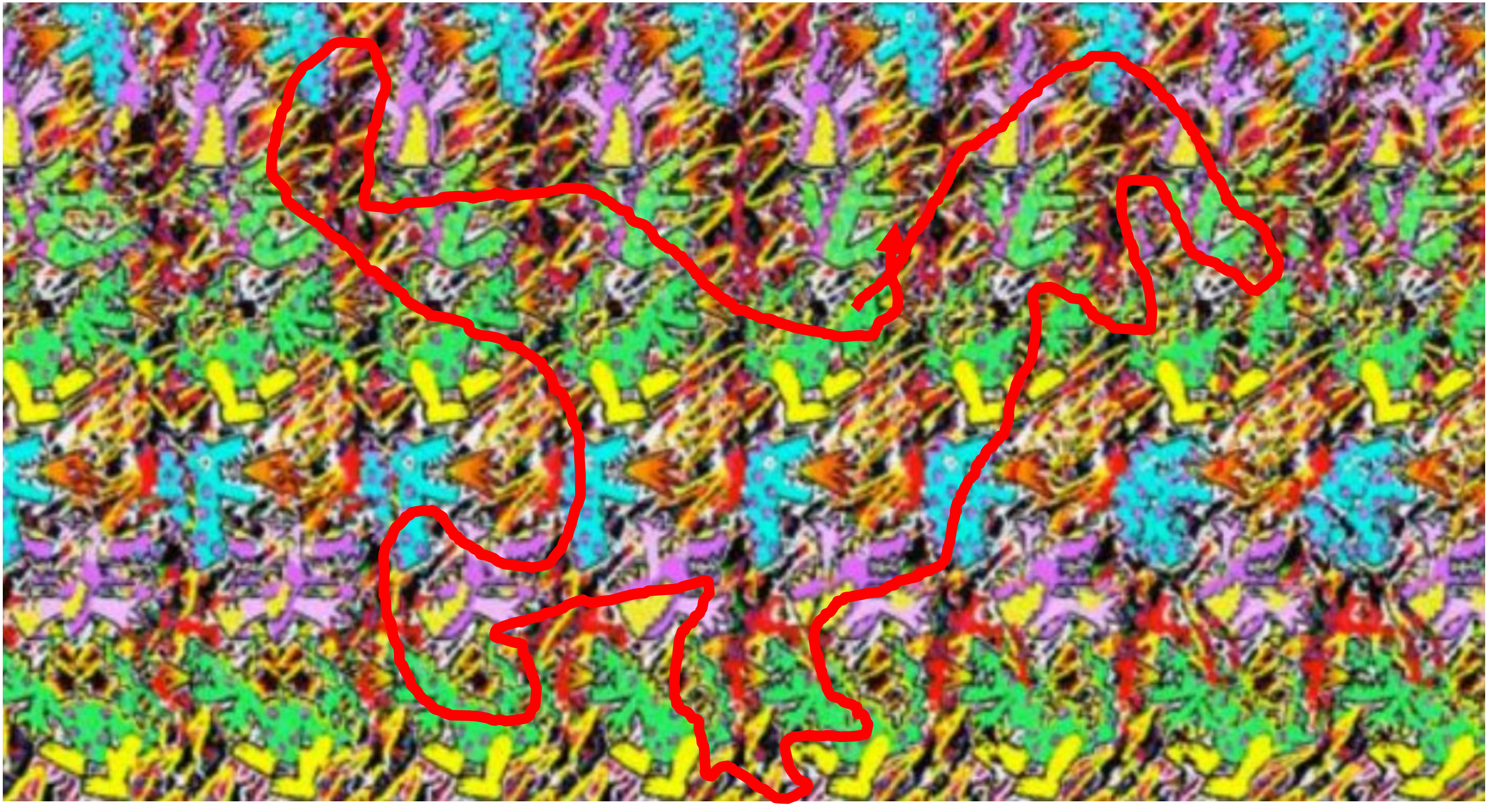


Fig. 1—Formation of a set of tables.

????

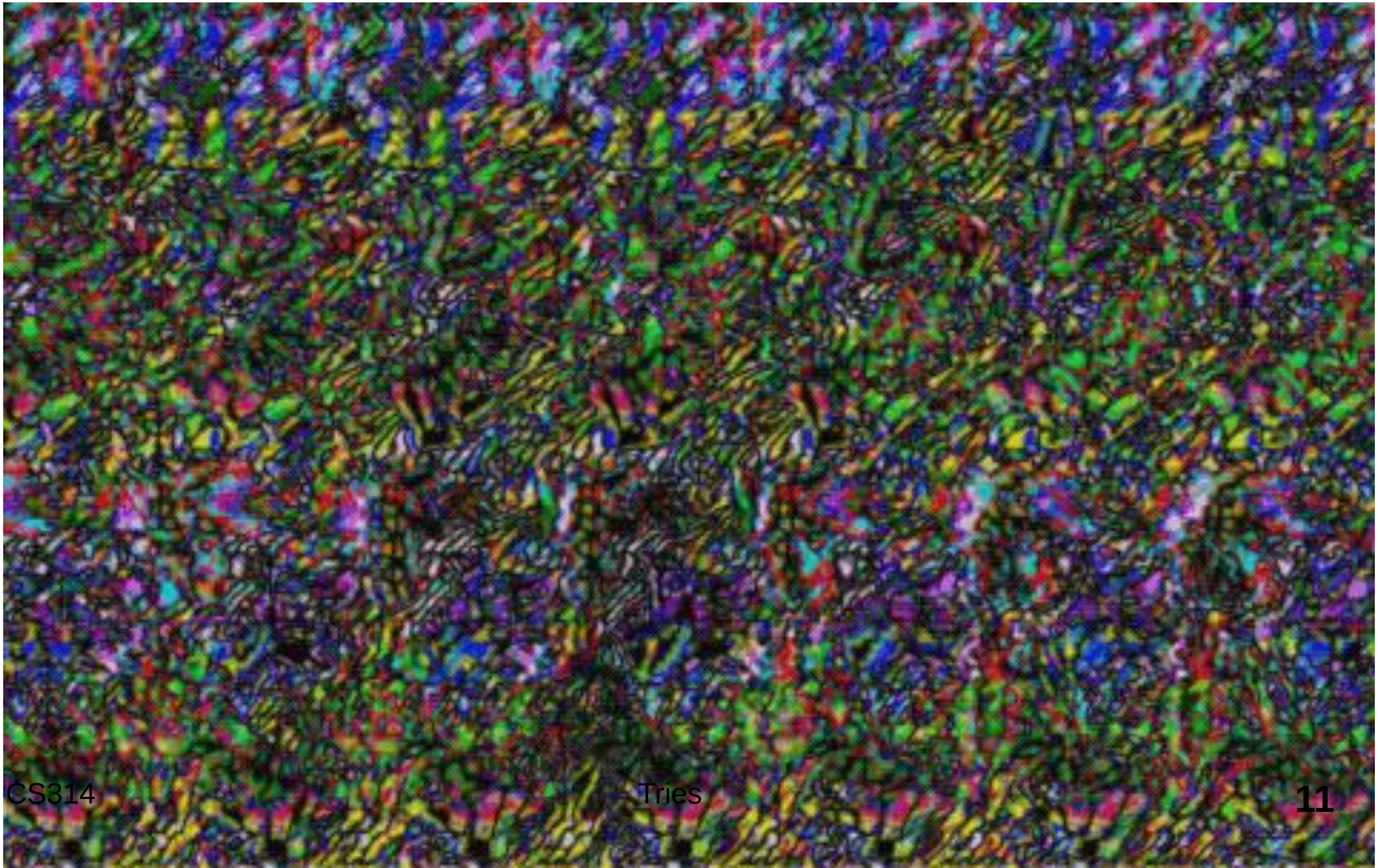
????



Picture of a Dinosaur

Fall 2022 - Ryan P.

Created with Procreate: https://procreate.art/



Can

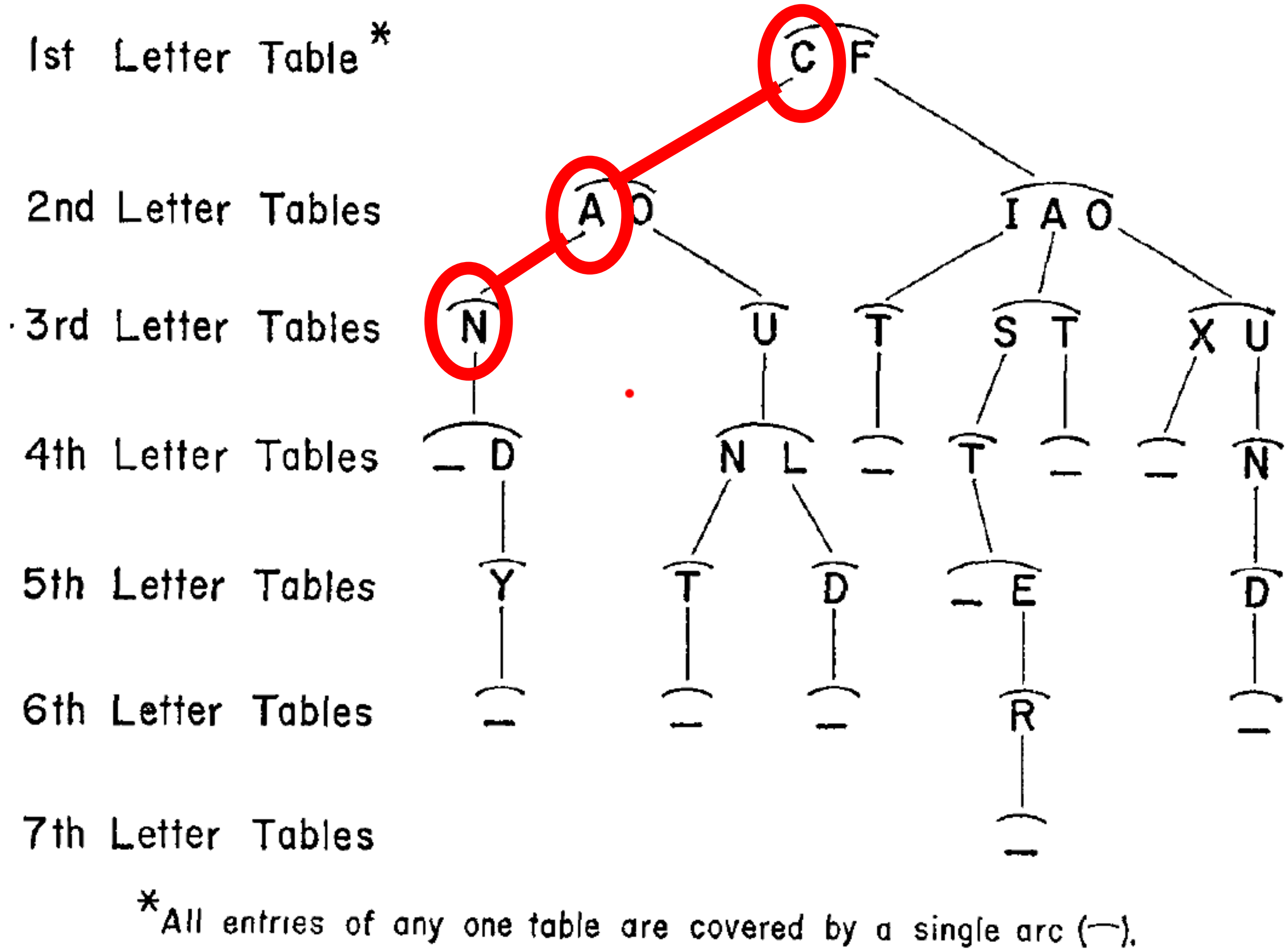


Fig. 1—Formation of a set of tables.

Candy

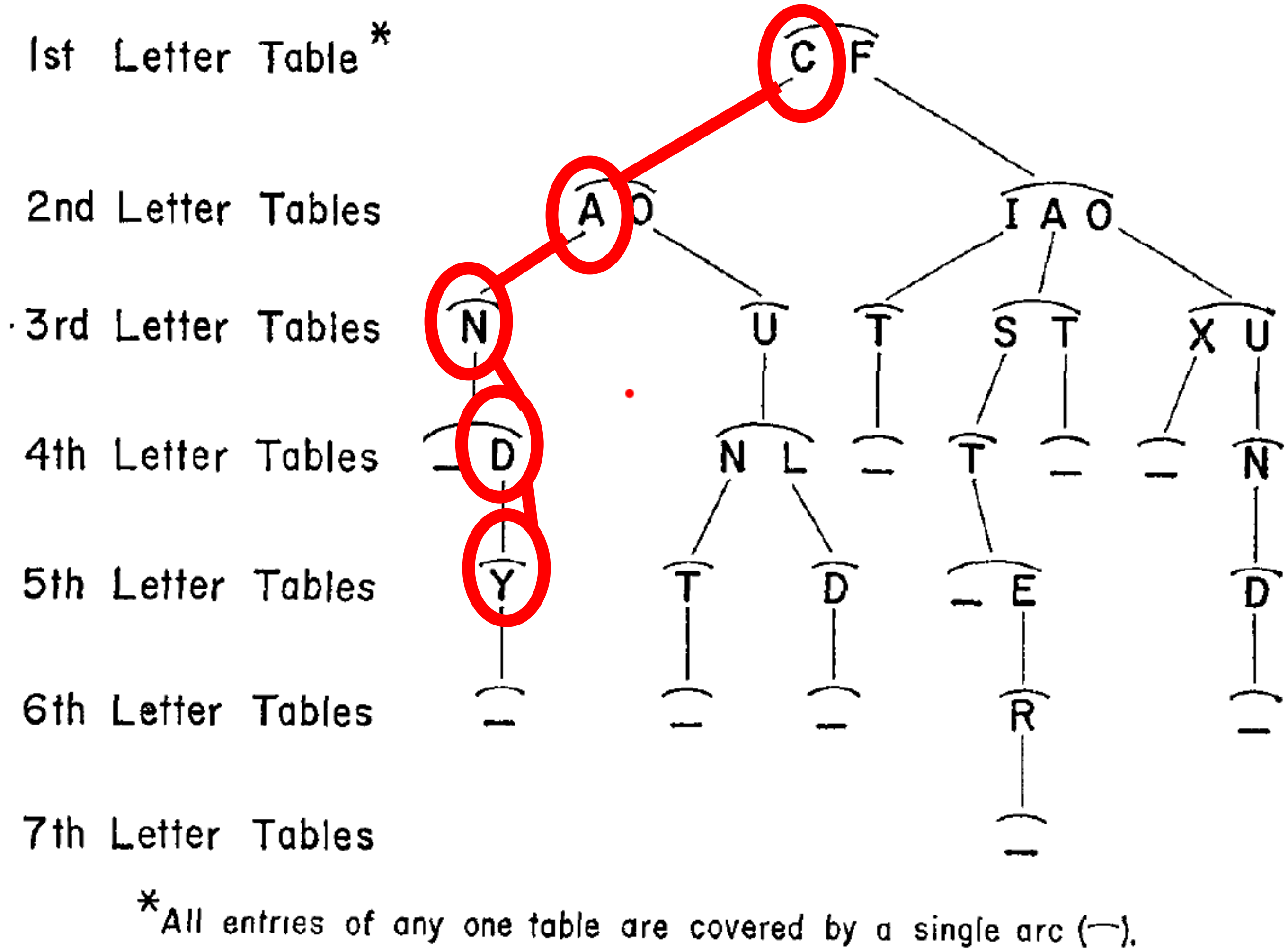


Fig. 1—Formation of a set of tables.

Fox

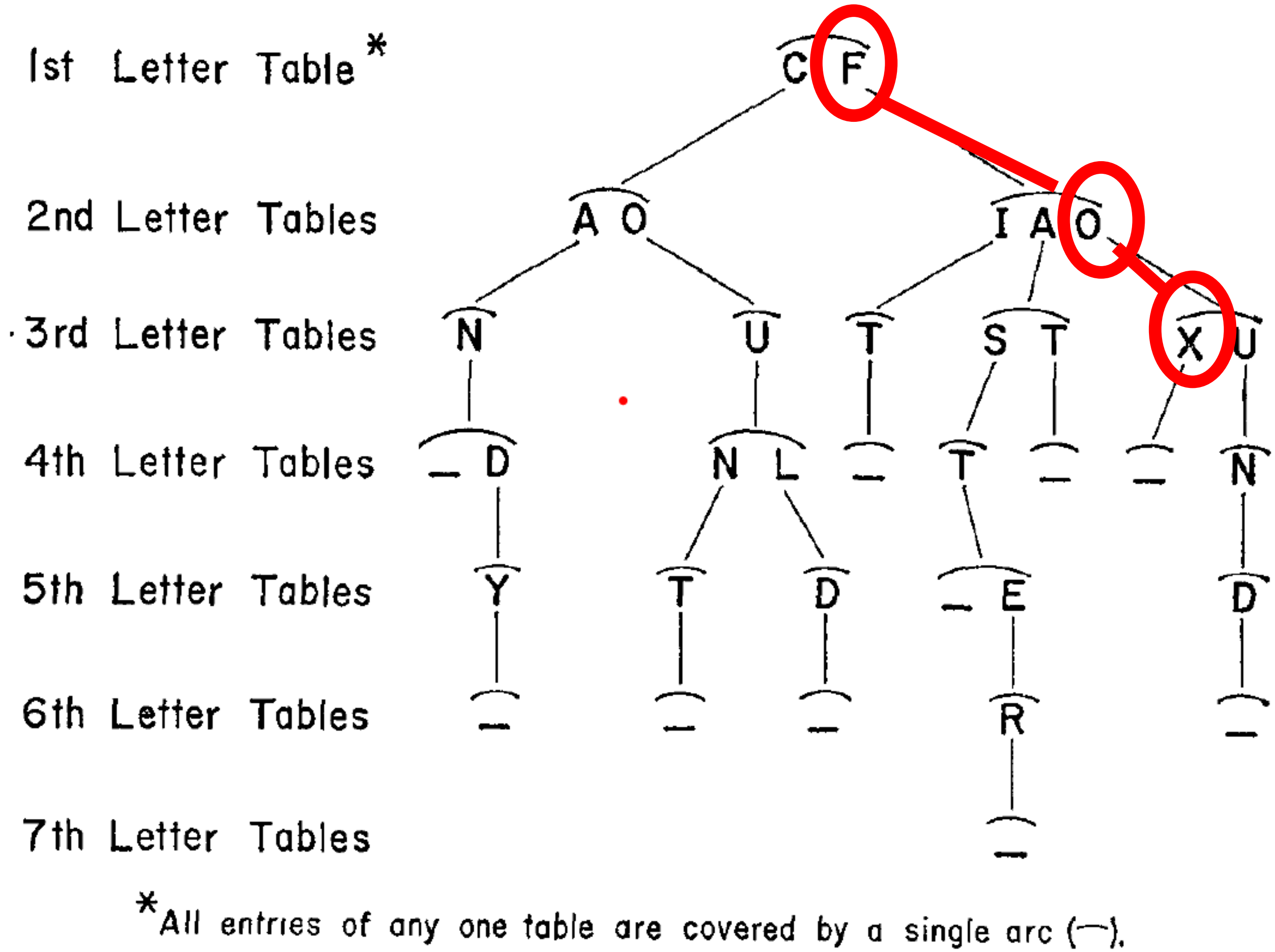


Fig. 1—Formation of a set of tables.

Clicker 2

- Is "fast" in the dictionary represented by this Trie?

A. No

B. Yes

C. It depends

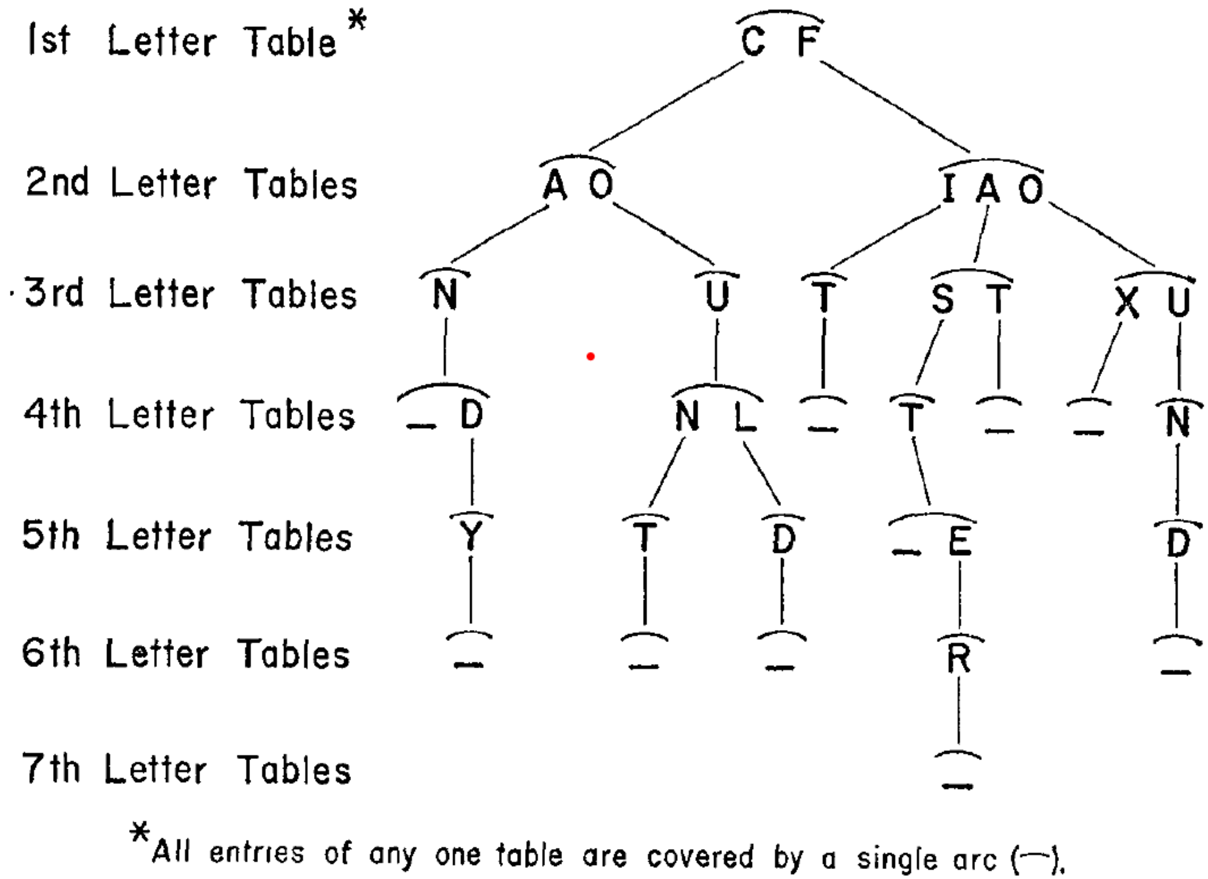


Fig. 1—Formation of a set of tables.

Clicker 3

- Is "fist" in the dictionary represented by this Trie?

A. No

B. Yes

C. It depends

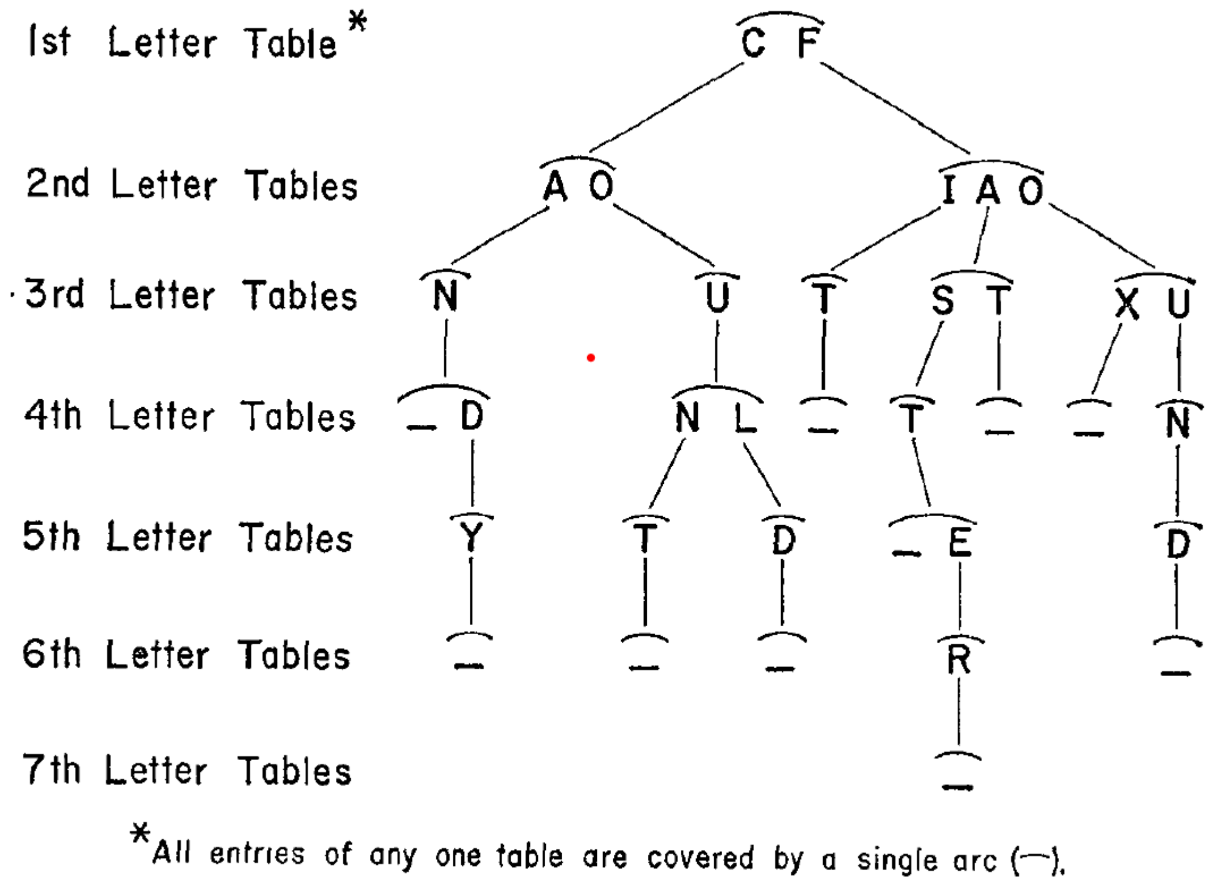
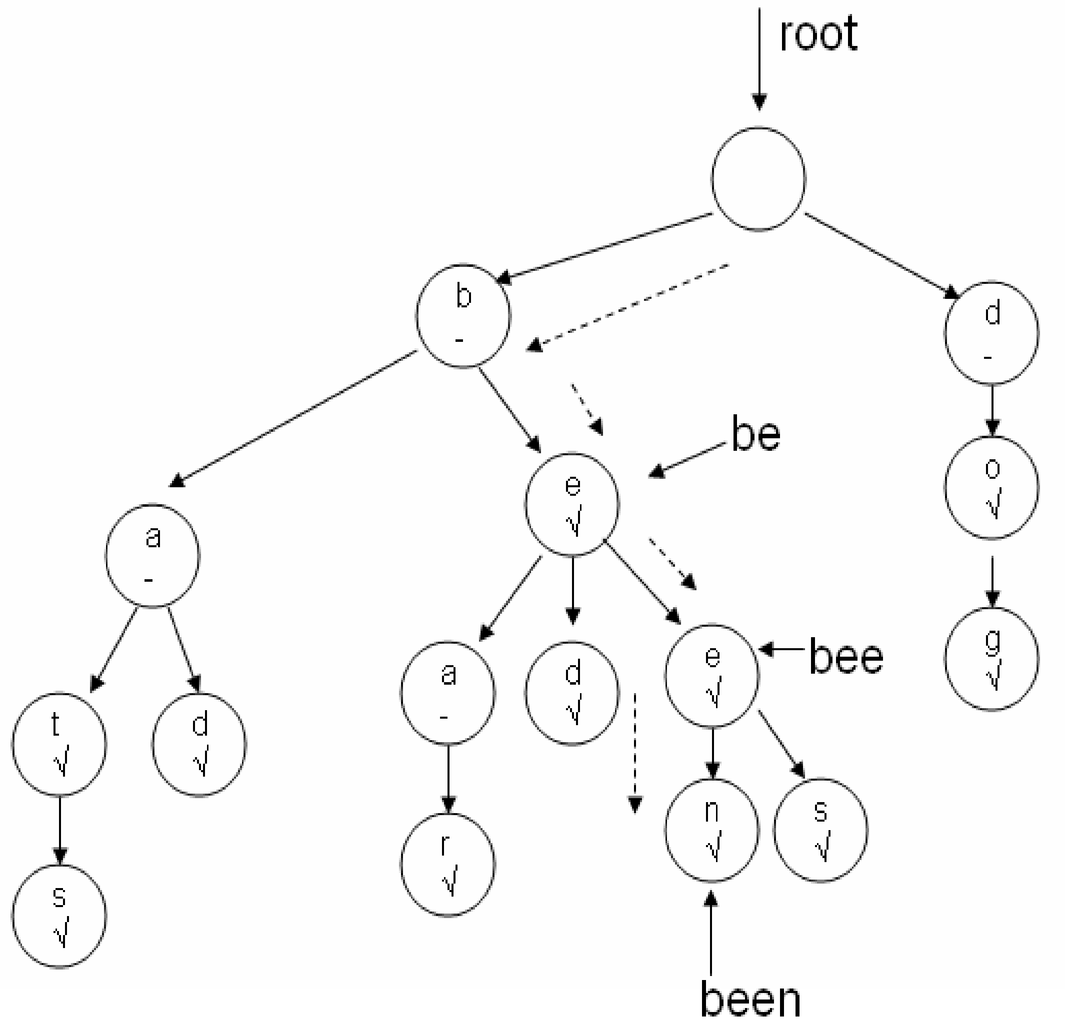


Fig. 1—Formation of a set of tables.

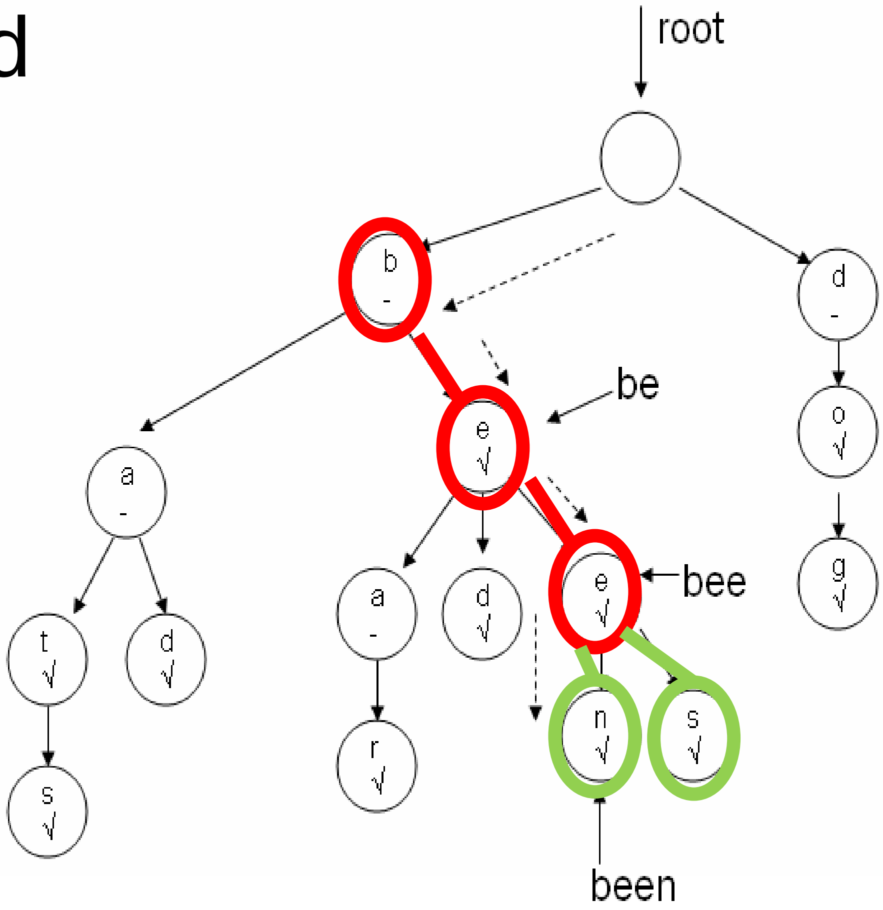
Tries

- Another example of a Trie
- Each node stores:
  – A char
  – A boolean indicating if the string ending at that node is a word
  – A list of children



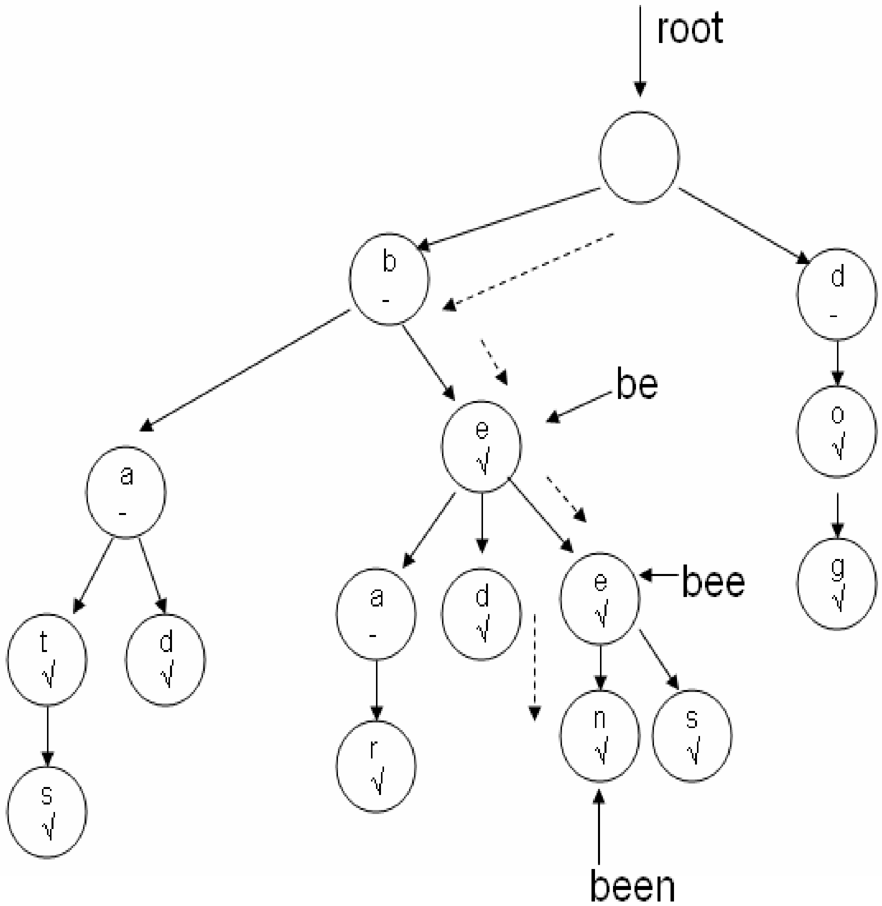
Predictive Text and AutoComplete

- As characters are entered we descend the Trie
- … and from the current node …
- … <u>we can descend to terminators and leaves to see all possible words based on current prefix</u>
- b, e, e -> bee, been, bees



Tries

- Stores words and phrases.
  - other values possible, but typically Strings
- The whole word or phrase is not actually stored in a single node.
- … rather the path in the tree represents the word.



Implementing a Trie

```
public class Trie {

    private TNode root;
    private int size; // number of words
    private int numNodes;

    public Trie() {
        root = new TNode();
        numNodes = 1;
```

TNode Class

```
private static class TNode {
        private boolean word;
        private char ch;
        private LinkedList<TNode> children;
```

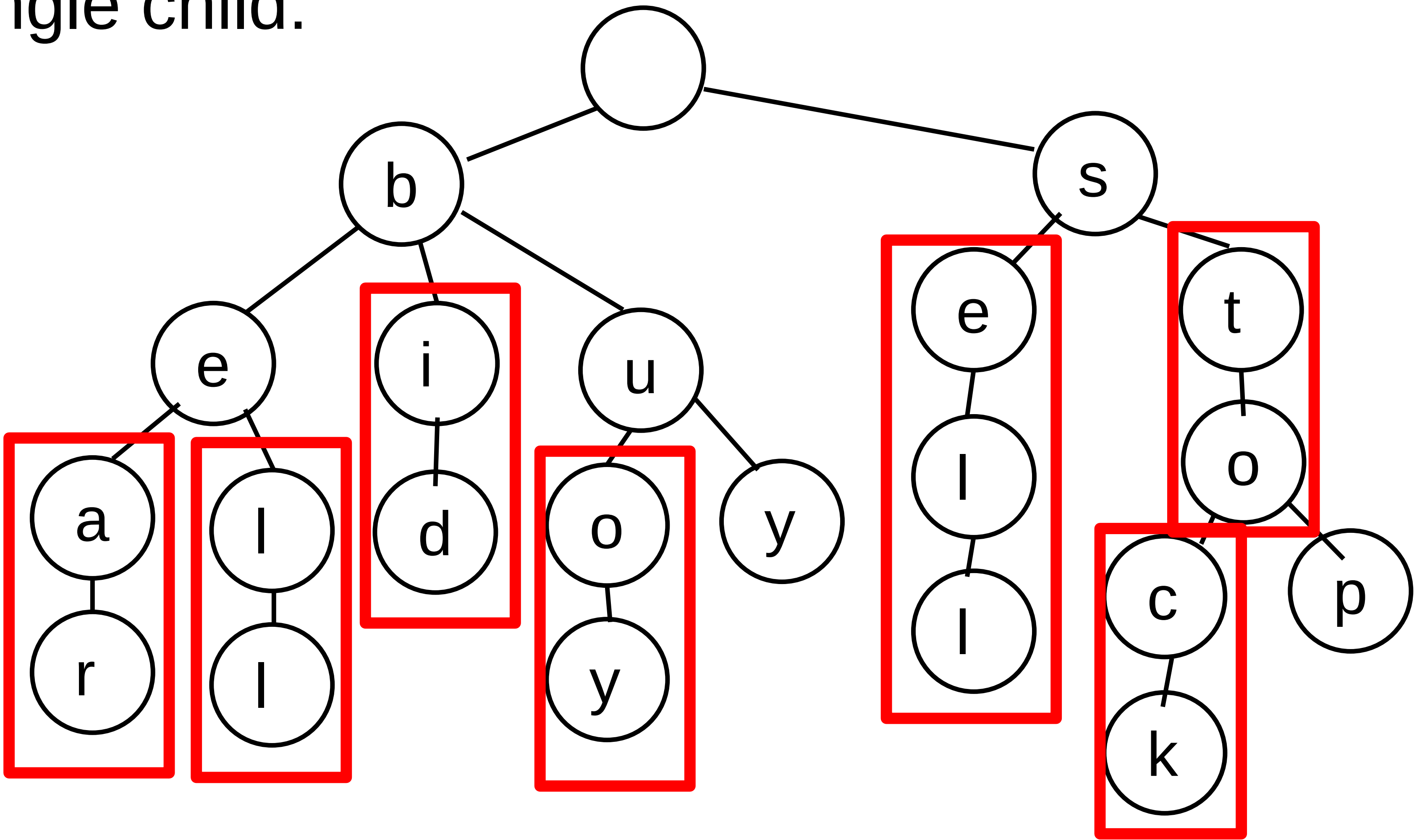
- Basic implementation uses a LinkedList of TNode objects for children
- Other options?
  - ArrayList?
  - Something more exotic?

Basic Operations

- Adding a word to the Trie
- Getting all words with given prefix
- Demo in IDE

Compressed Tries

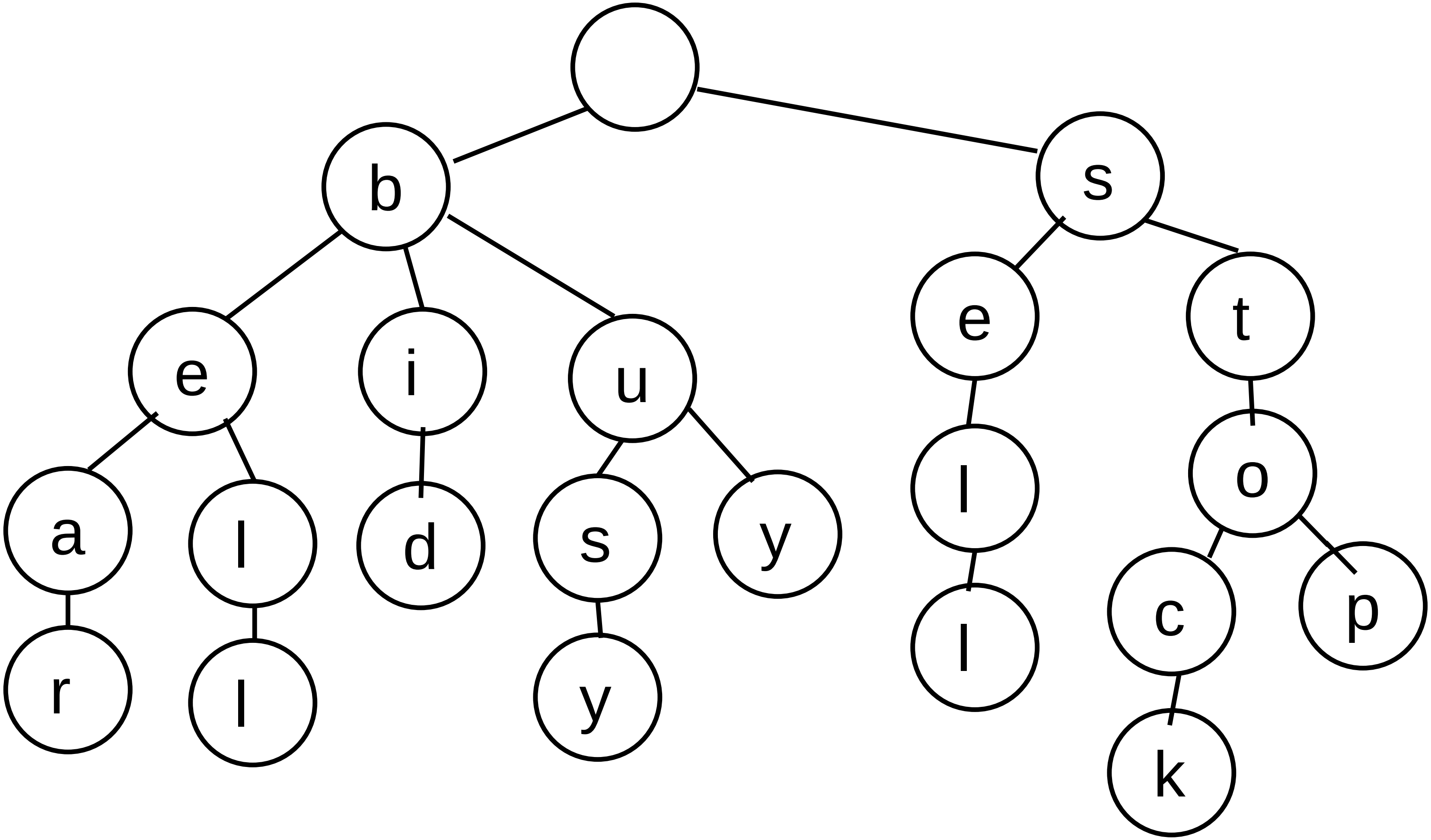
- Some words, especially long ones, lead to a chain of nodes with single child, followed by single child:



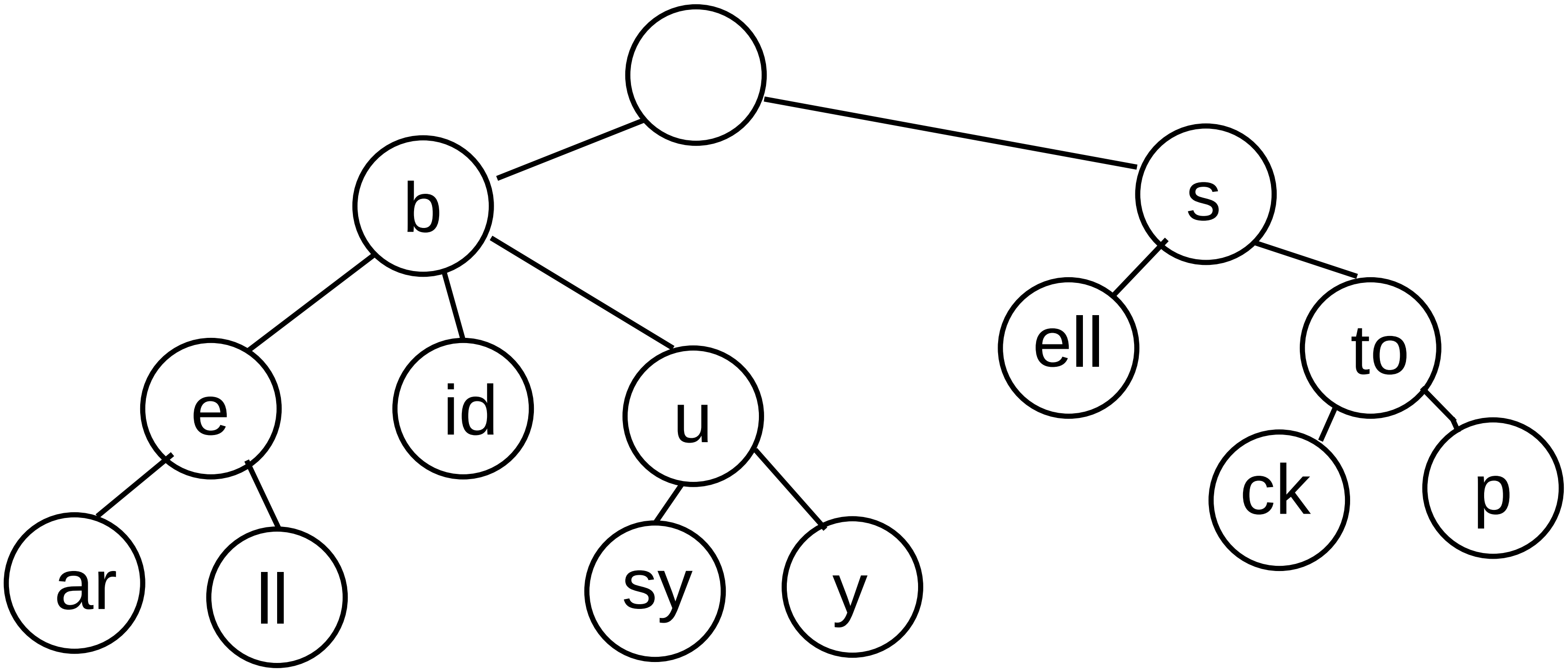
Compressed Trie

- Reduce number of nodes, by having nodes store Strings
- A chain of single child followed by single child (followed by single child …) is compressed to a single node with that String
- Does not have to be a chain that terminates in a leaf node
  - Can be an internal chain of nodes

Original, Uncompressed



Compressed Version



8 fewer nodes compared to uncompressed version

s – t – o – c - k

Topic 25
Heaps

"You think you know when you can **learn**,
are more sure when you can **write,**
even more when you can **teach**,
but certain when you can **program**."

- Alan Perlis

Priority Queue

- Recall priority queue
  - elements enqueued based on priority
  - dequeue removes the highest priority item
- Options?
  - List? Binary Search Tree? Clicker 1

	Array List enqueue	BST enqueue
A.	O(N)	O(1)
B.	O(N)	O(logN)
C.	O(N)	O(N)
D.	O(logN)	O(logN)
E.	O(1)	O(logN)

Another Option

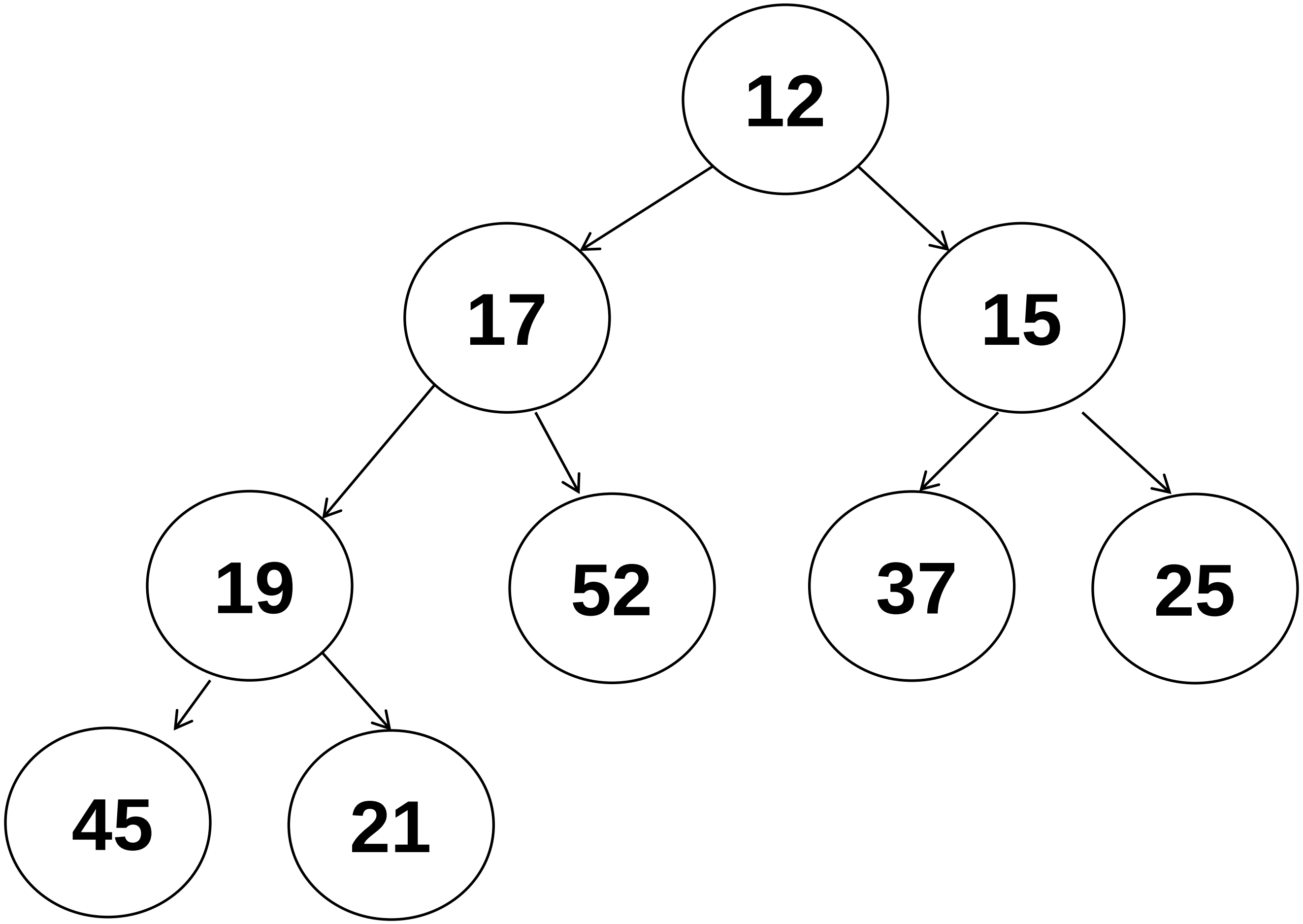
- The *heap* data structure
  - not to be confused with the runtime heap (portion of memory for dynamically allocated variables)
- Typically a complete binary tree (variations with more than 2 children possible)
  - all levels have maximum number of nodes except deepest where nodes are filled in from left to right
- Maintains the *heap order property*
  - in a min heap the value in the root of any subtree is less than or equal to all other values in the subtree

Clicker 2

- In a max heap with no duplicates where is the largest value?

A. the root of the tree
B. in the left-most node
C. in the right-most node
D. a node in the lowest level
E. none of these

Example Min Heap

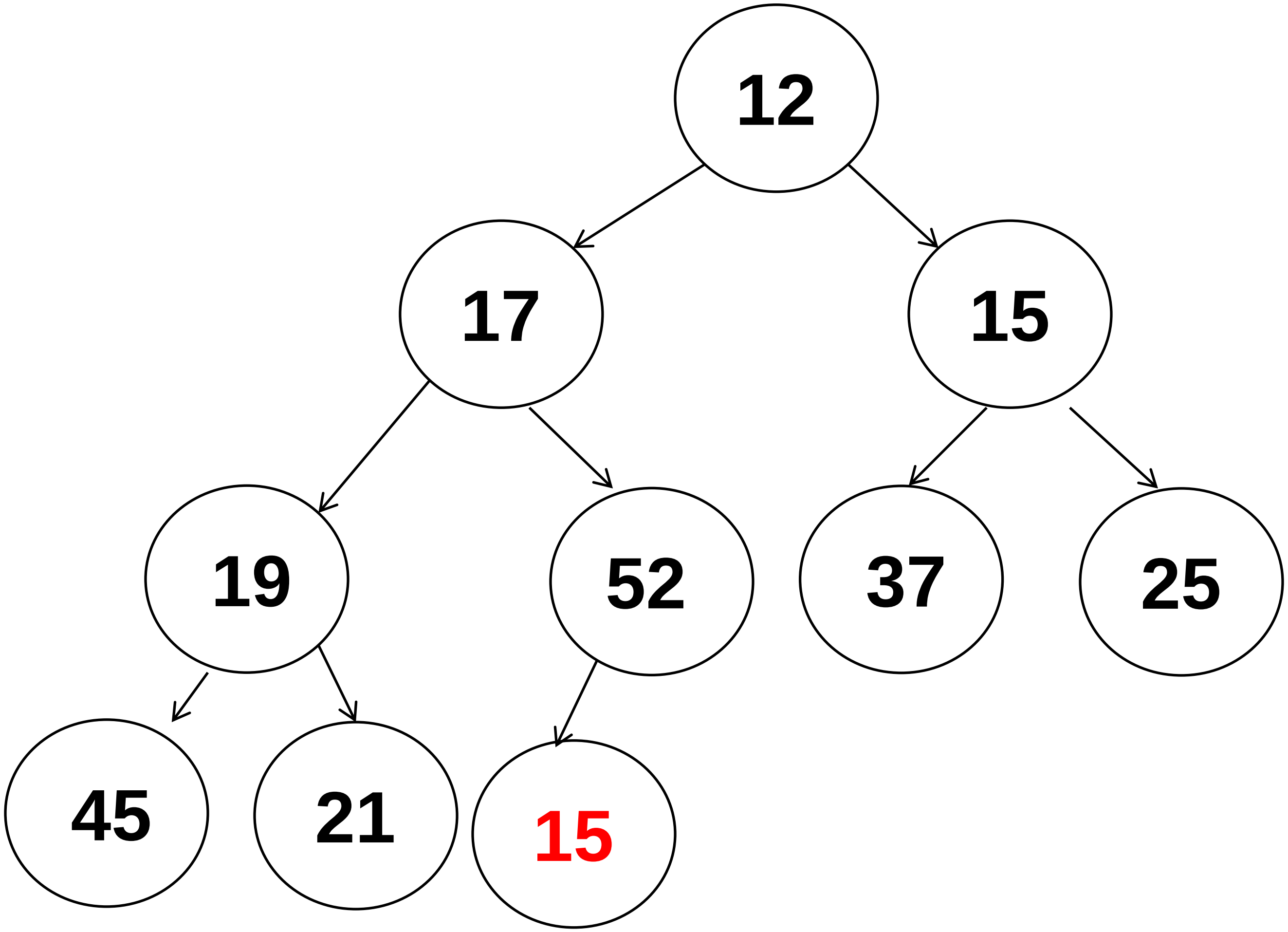


Add Operation

- Add new element to next open spot in array
- Swap with parent if new value is less than parent
- Continue back up the tree as long as the new value is less than new parent node

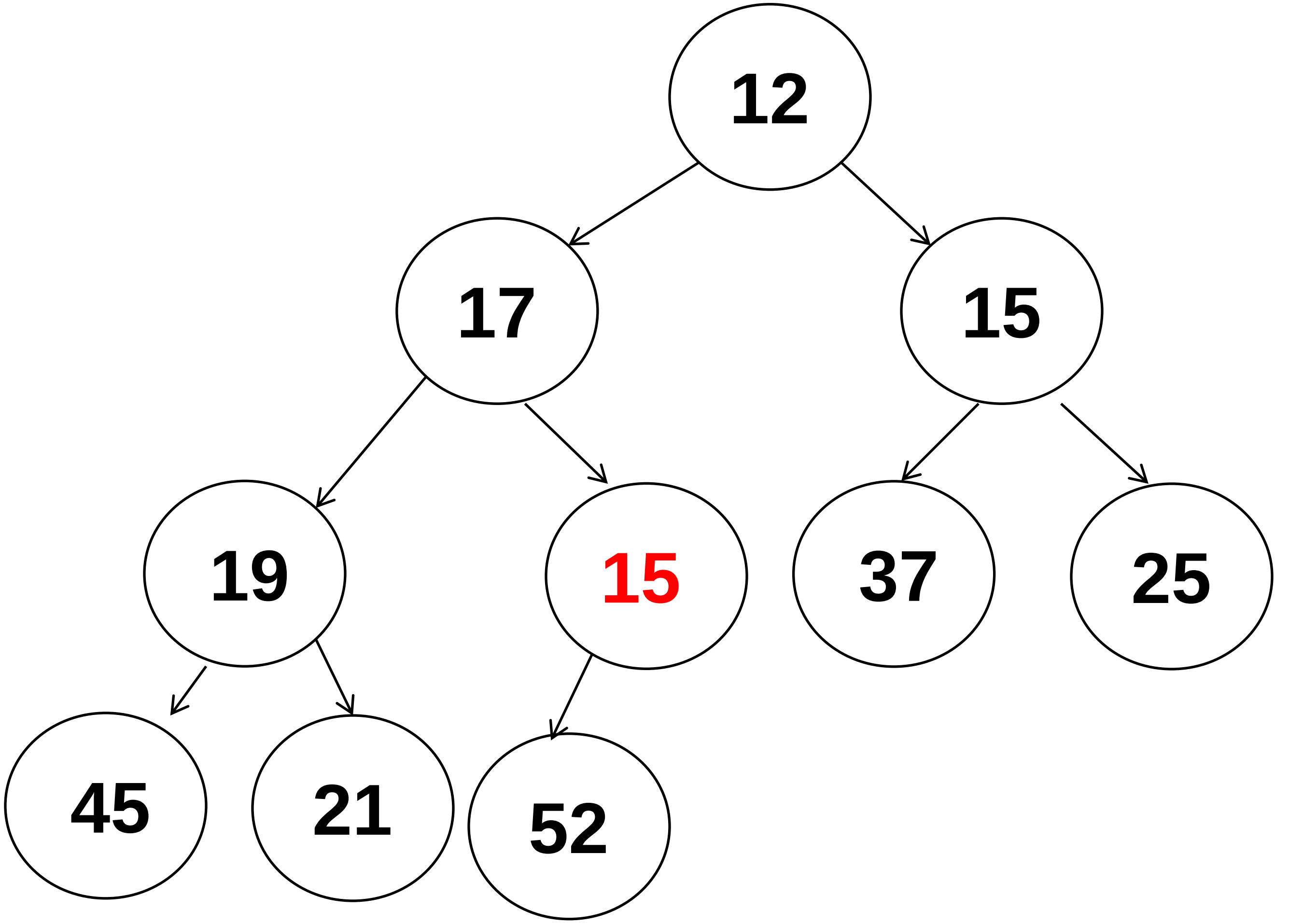
Add Example

- Add 15 to heap (initially next left most node)



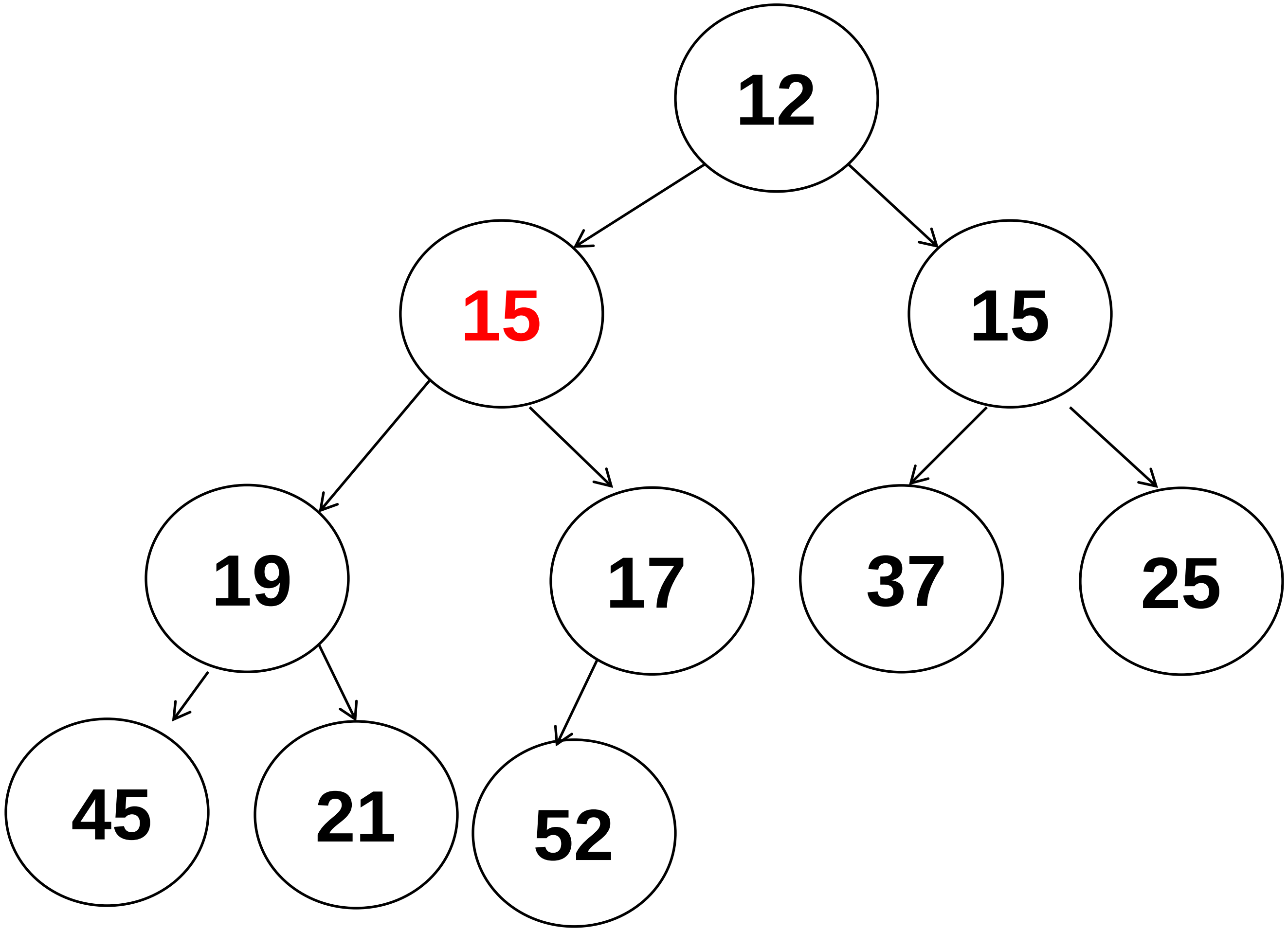
Add Example

- Swap 15 and 52



Enqueue Example

- Swap 15 and 17, then stop

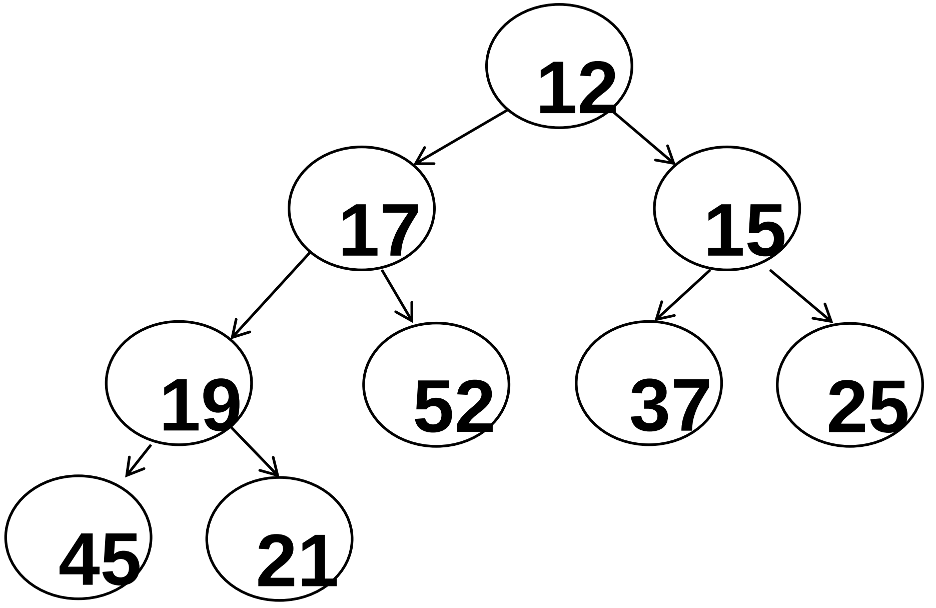


Add Example

- Insert the following values 1 at a time into a min heap:
  16 9 5 8 13 8 8 5 5 19 27 9 3

Internal Storage

- Interestingly heaps are often implemented with an array instead of nodes



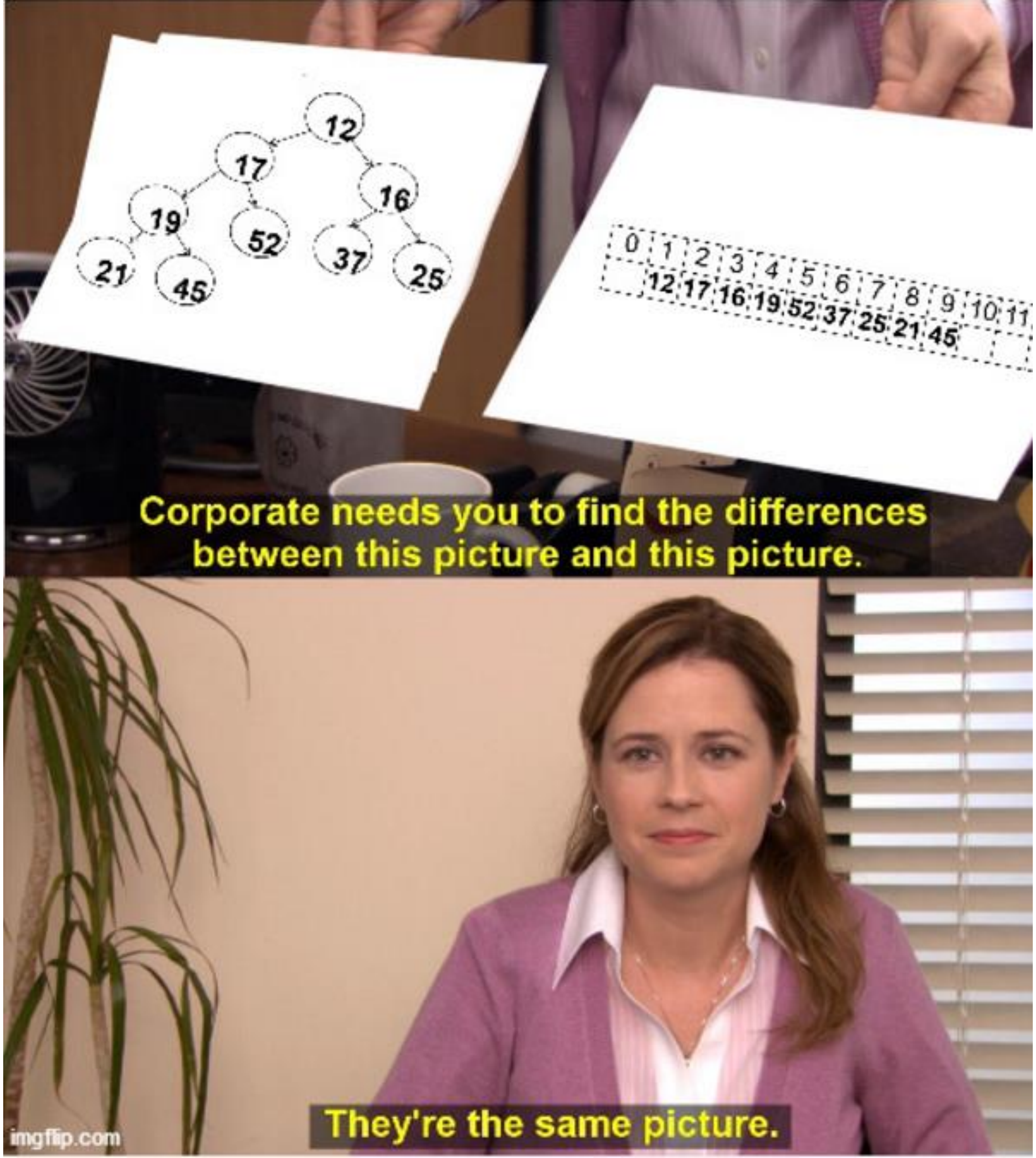
for element at index i:

parent index: *i / 2*

left child index: *i * 2*

right child index: *i * 2 + 1*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	**17**	**15**	**19**	**52**	**37**	**25**	**45**	**21**						



In Honor of Elijah,
The Meme King,
Spring 2020

PriorityQueue Class

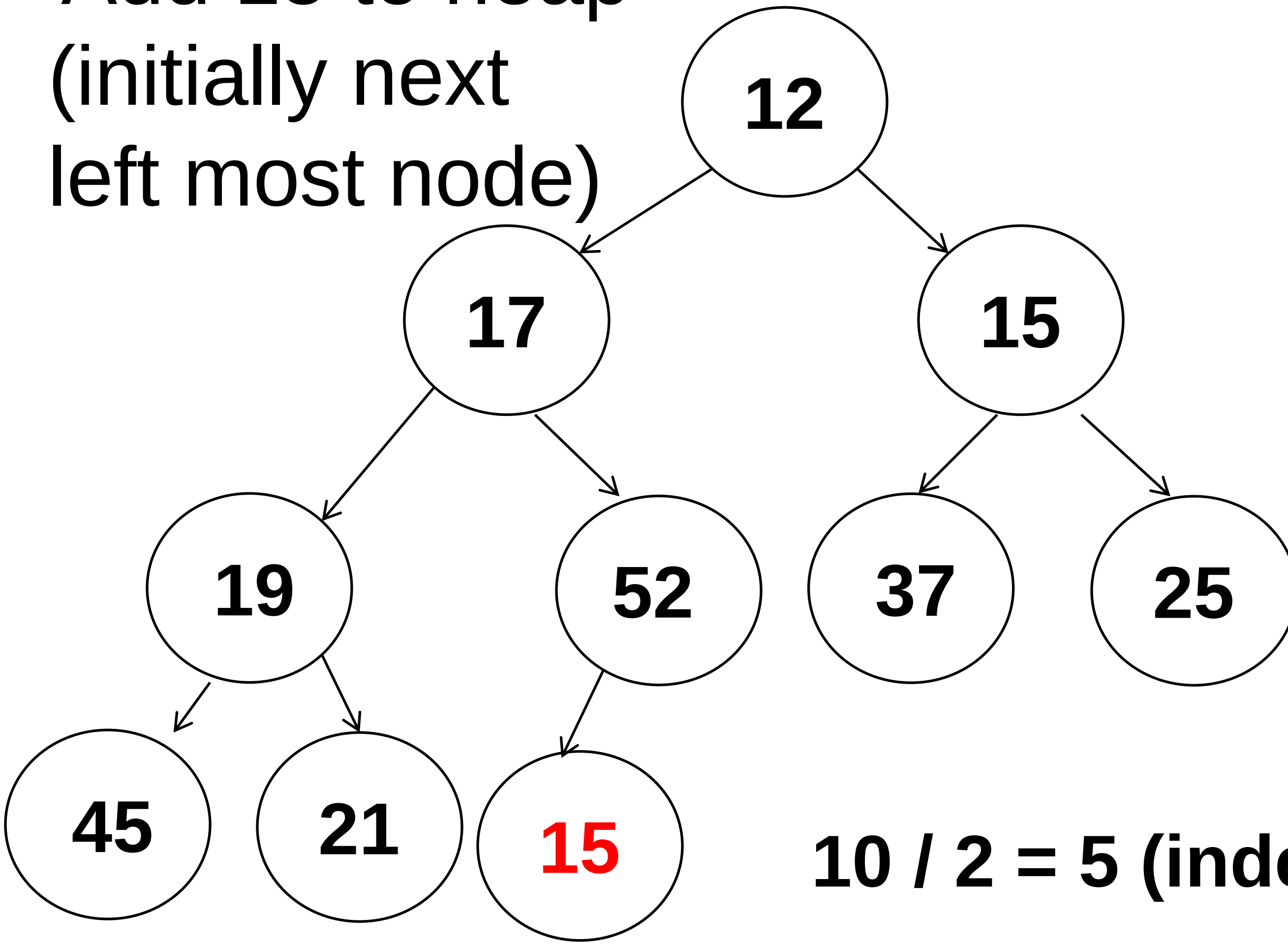
```
public class PriorityQueue<E extends Comparable<? super E>>
{

    private int size;
    private E[] con;

    public PriorityQueue() {
        con = getArray(2);
    }

    private E[] getArray(int size) {
        return (E[]) (new Comparable[size]);
    }
```

PriorityQueue enqueue / add

```
public void enqueue(E val) {
    if ( size >= con.length - 1 )
        enlargeArray(con.length * 2);

    size++;
    int indexToPlace = size;
    while ( indexToPlace > 1
            && val.compareTo( con[indexToPlace / 2] ) < 0 ) {

        con[indexToPlace] = con[indexToPlace / 2]; // swap
        indexToPlace /= 2; // change indexToPlace to parent
    }
    con[indexToPlace] = val;
}

private void enlargeArray(int newSize) {
    E[] temp = getArray(newSize);
    System.arraycopy(con, 1, temp, 1, size);
    con = temp;
}
```

Enqueue / add Example With Array Shown

- Add 15 to heap (initially next left most node)

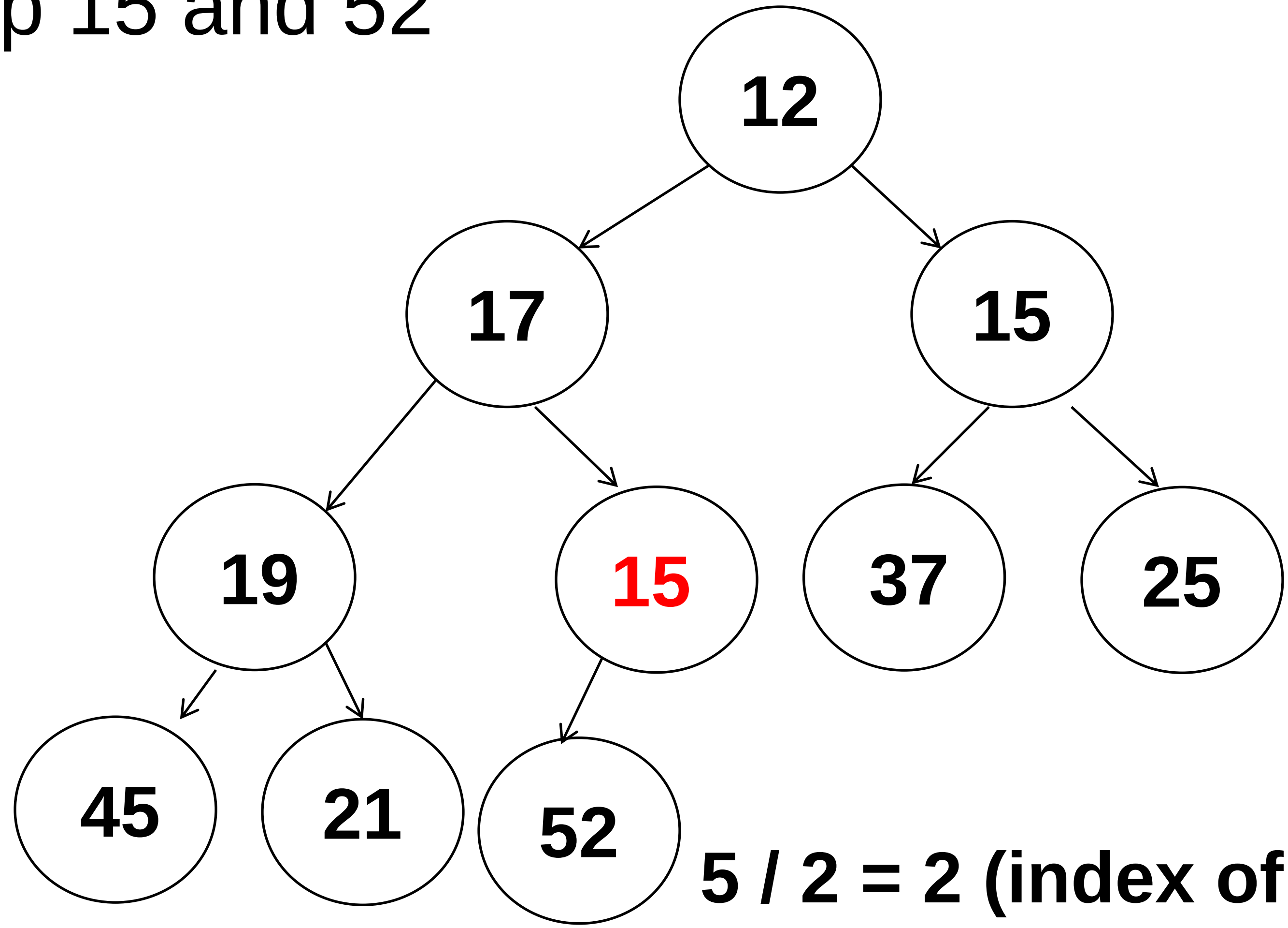


10 / 2 = 5 (index of parent)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	**17**	**15**	**19**	**52**	**37**	**25**	**45**	**21**	**15**					

Enqueue Example
With Array Shown

- Swap 15 and 52

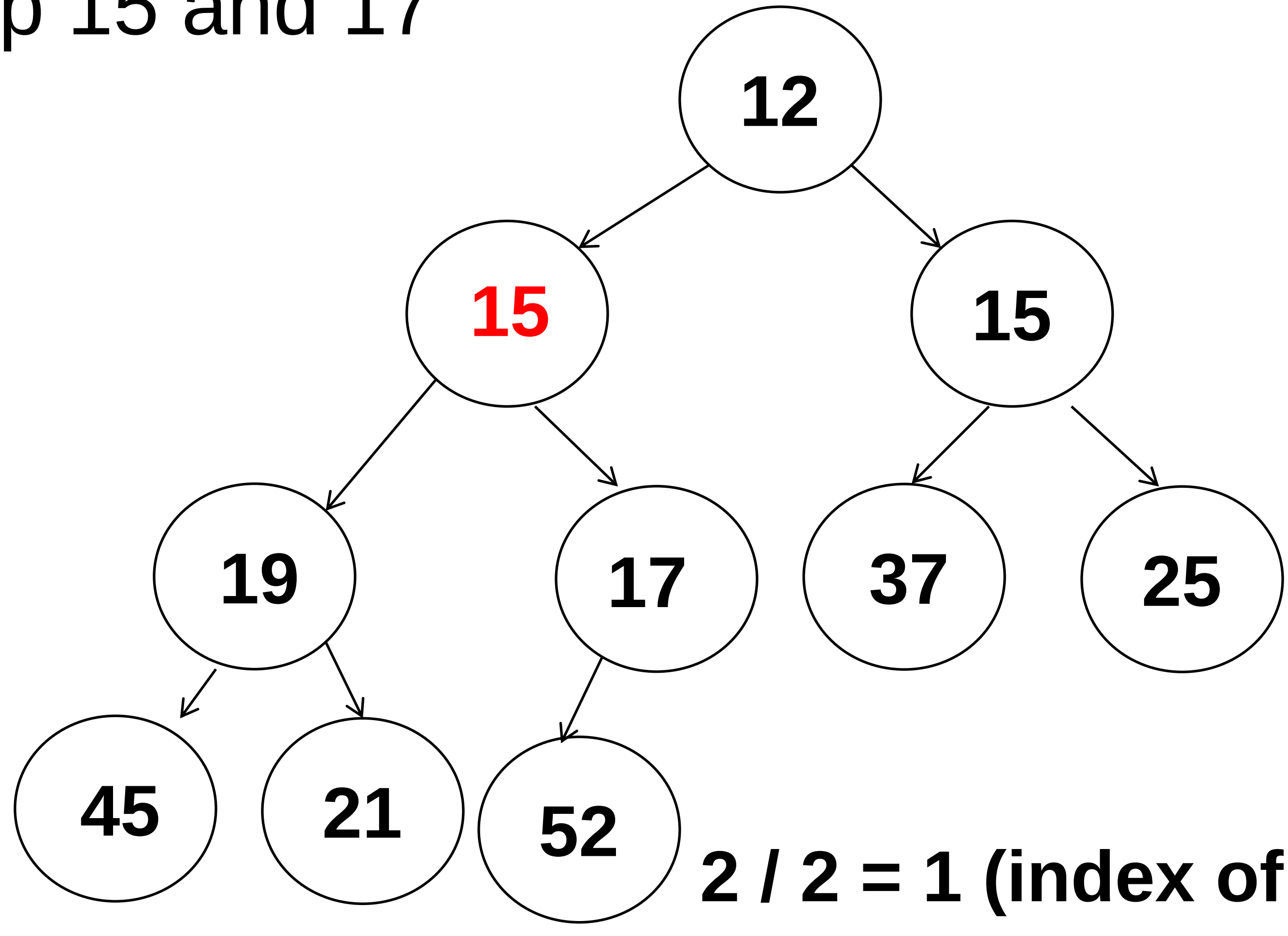


5 / 2 = 2 (index of parent)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	**17**	**15**	**19**	**15**	**37**	**25**	**45**	**21**	**52**					

Enqueue Example
With Array Shown

- Swap 15 and 17

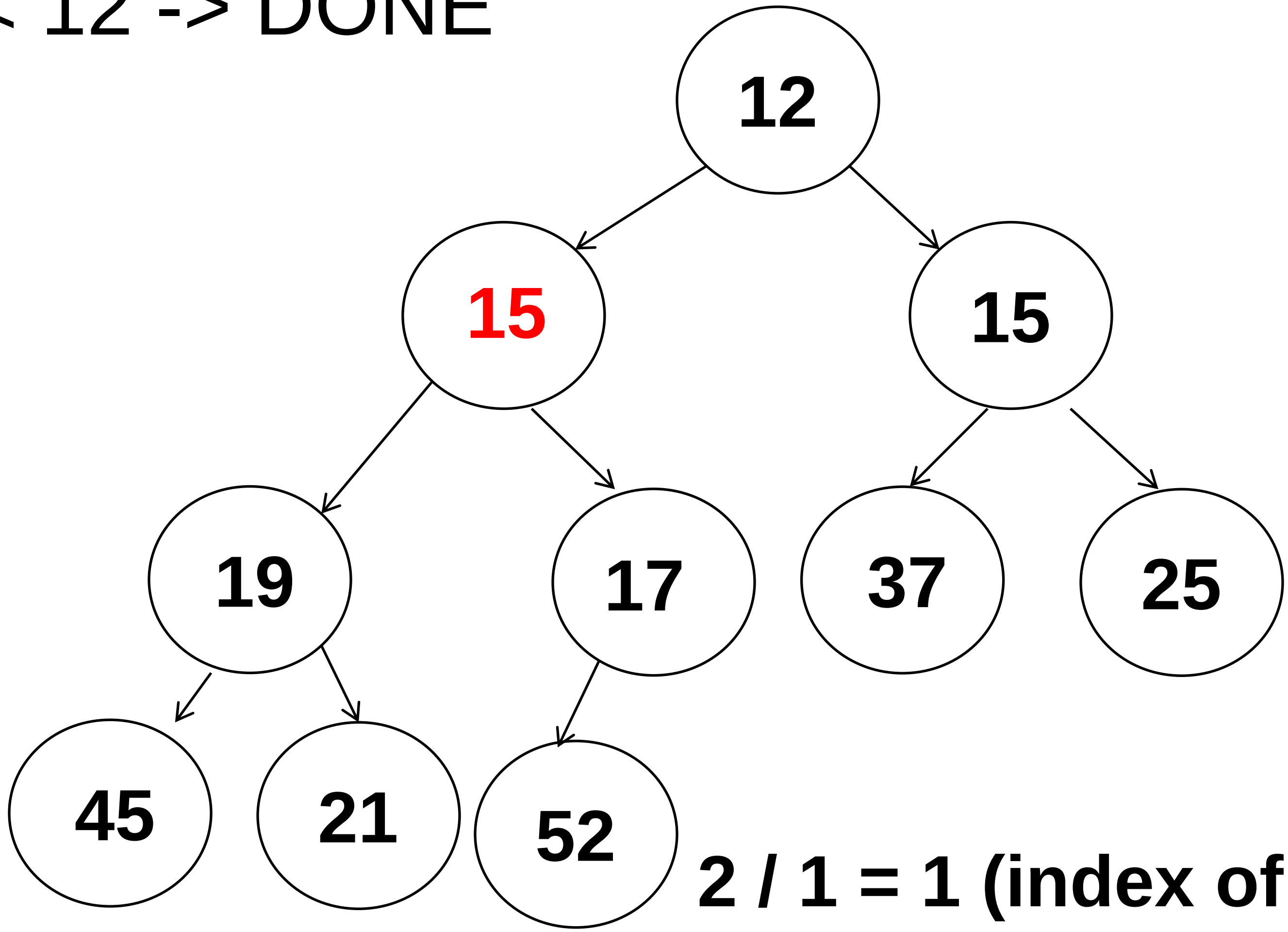


2 / 2 = 1 (index of parent)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	**15**	**15**	**19**	**17**	**37**	**25**	**45**	**21**	**52**					

Enqueue Example
With Array Shown

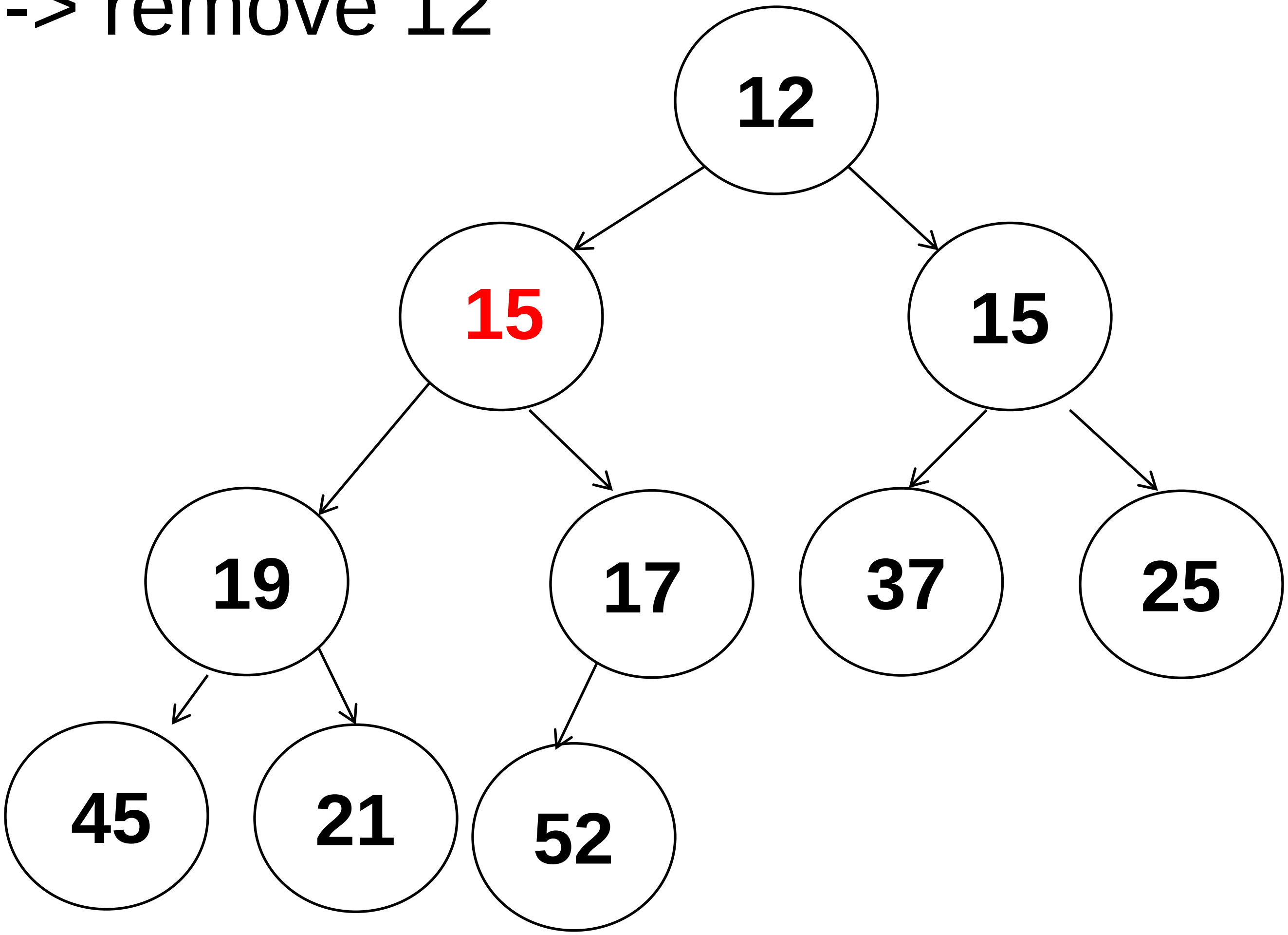
- 15 !< 12 -> DONE



2 / 1 = 1 (index of parent)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	**15**	**16**	**19**	**17**	**37**	**25**	**45**	**21**	**52**					

- Remove -> remove 12

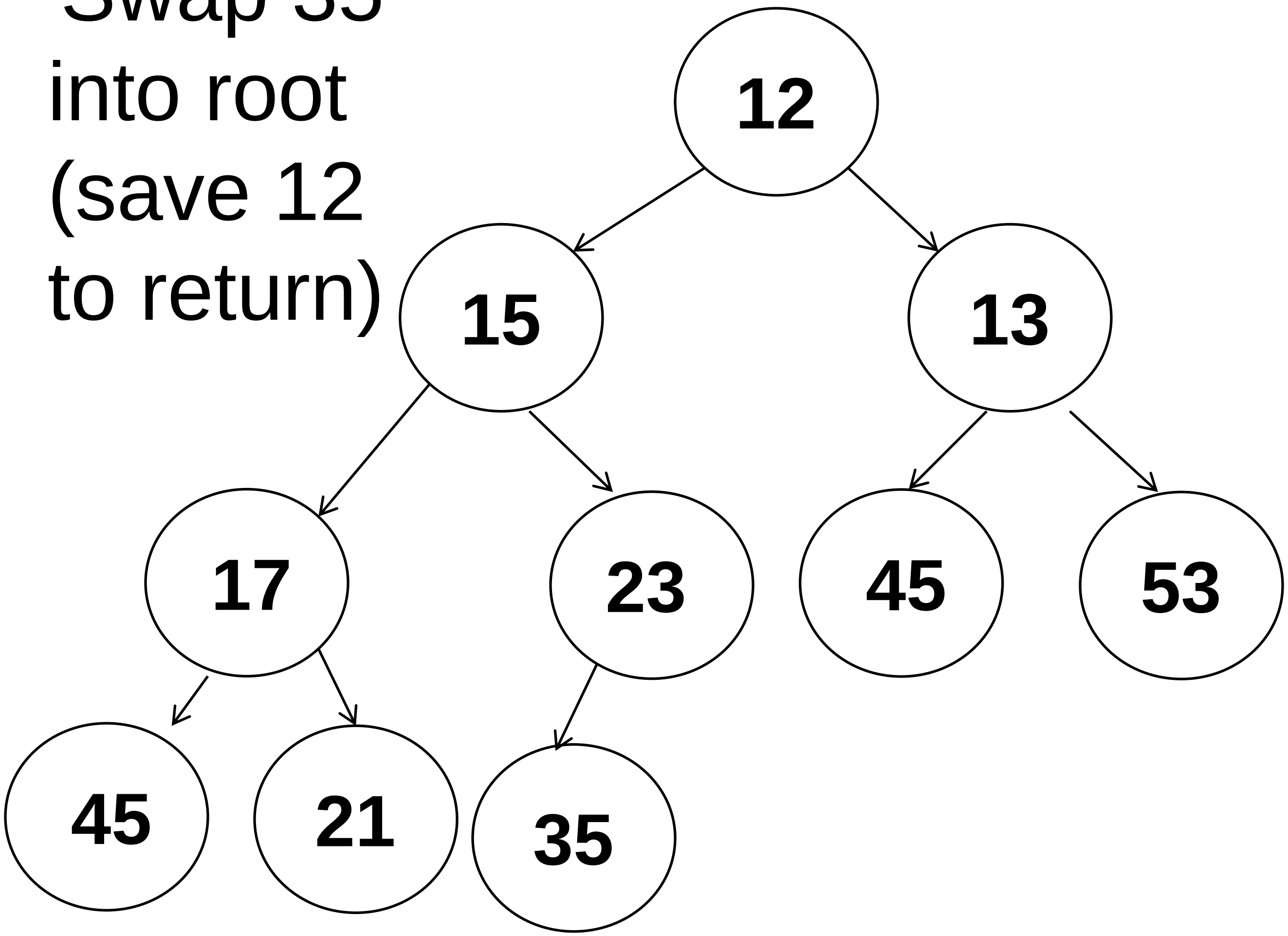


Remove / Dequeue

- min value / front of queue is in root of tree
- swap value from last node to root and move down swapping with smaller child unless values is smaller than both children

Dequeue Example

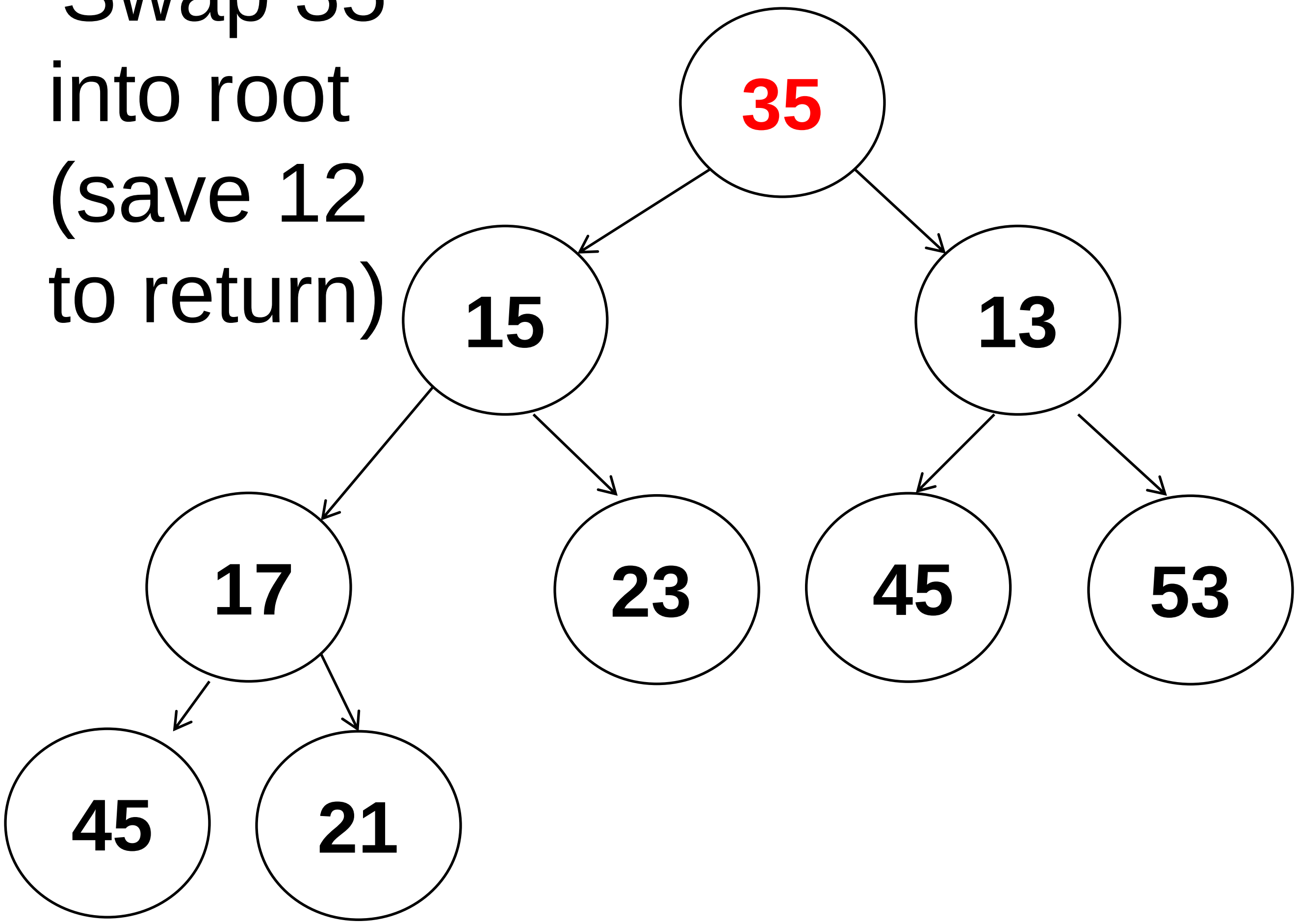
- Swap 35 into root (save 12 to return)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	**15**	**13**	**17**	**23**	**45**	**53**	**45**	**21**	**35**					

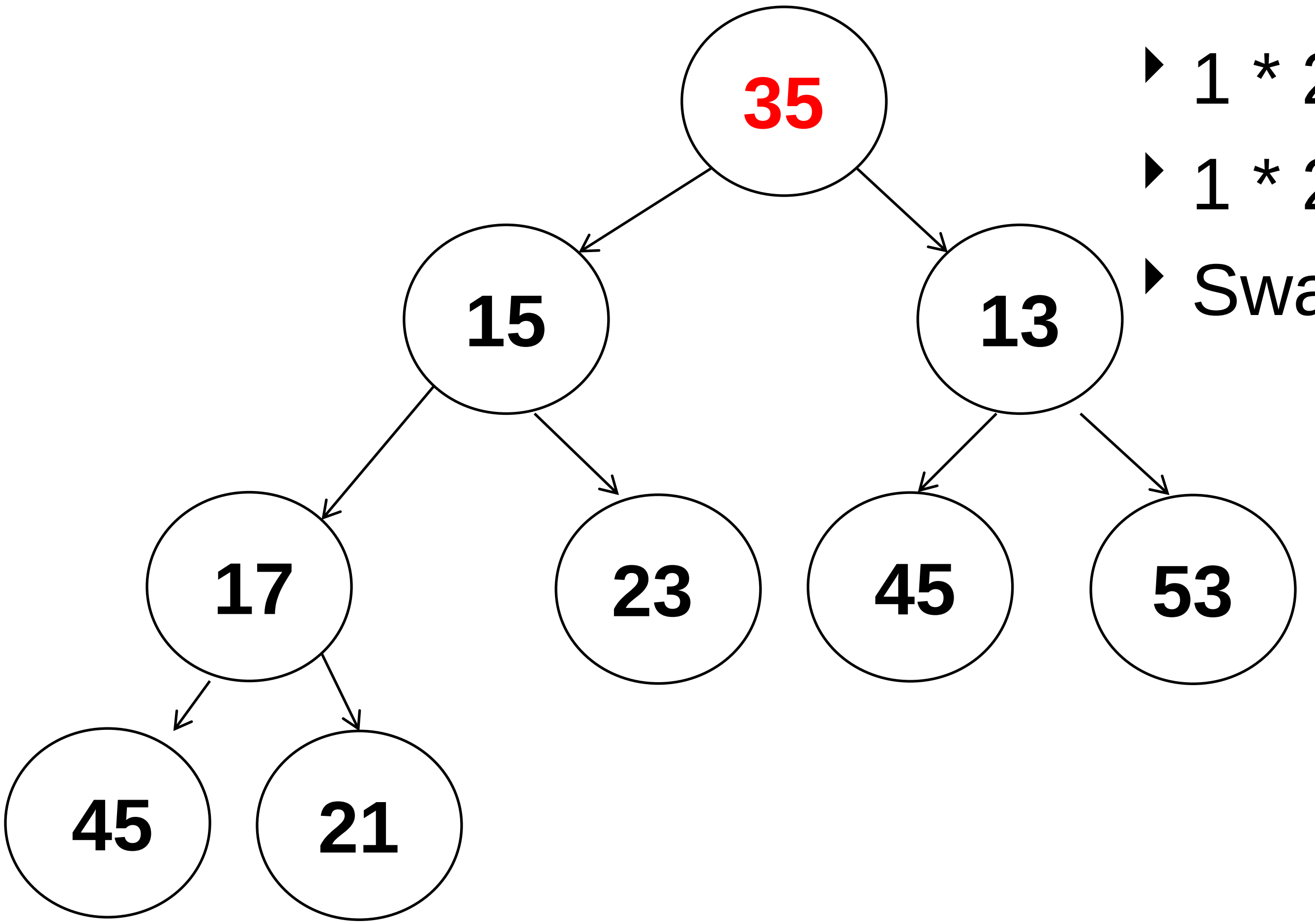
Dequeue Example

- Swap 35 into root (save 12 to return)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	35	**15**	**13**	**17**	**23**	**45**	**53**	**45**	**21**						

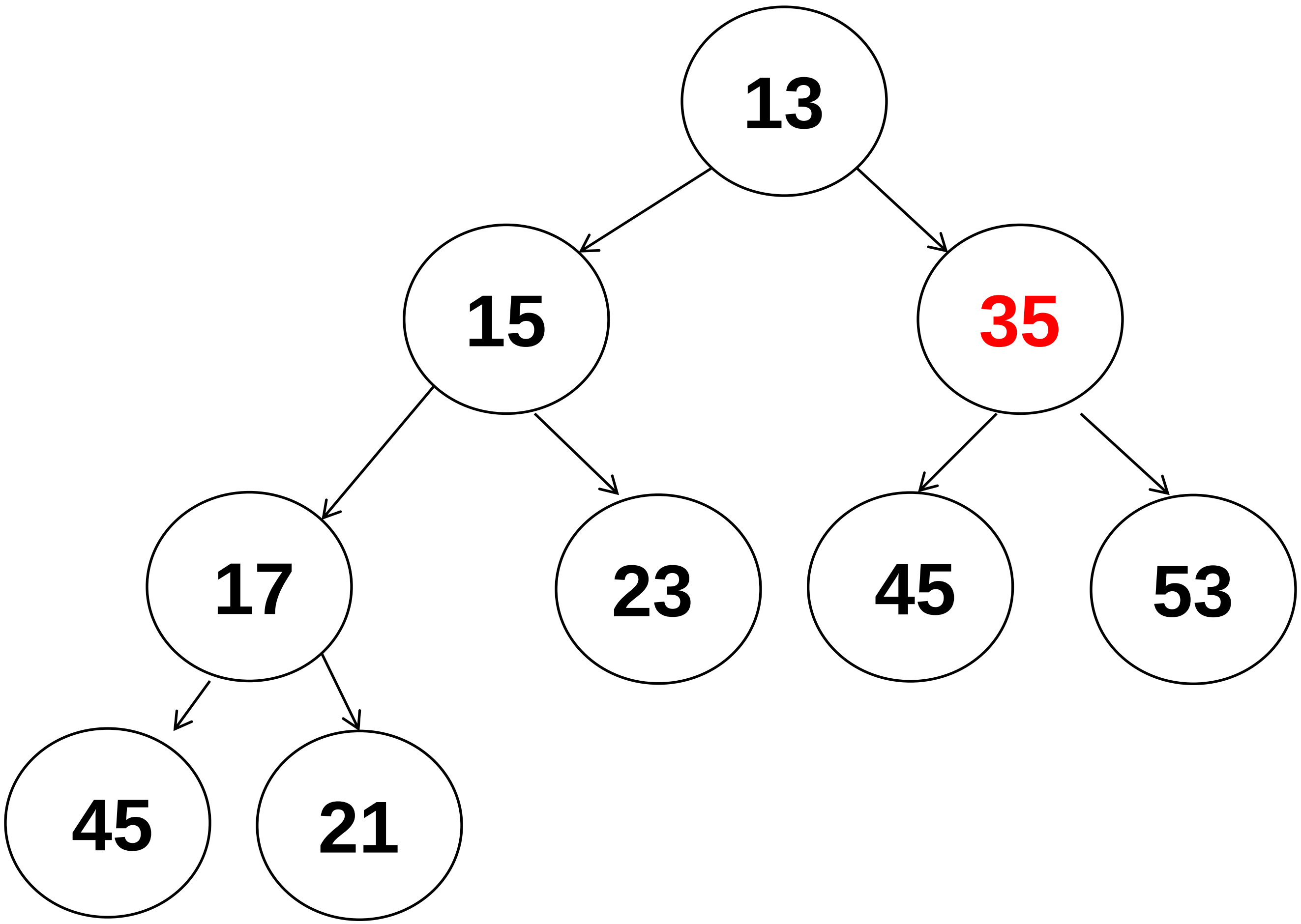
Dequeue Example



- Min child?
- 1 * 2 = 2 -> 15
- 1 * 2 + 1 = 3 -> 13
- Swap with 13

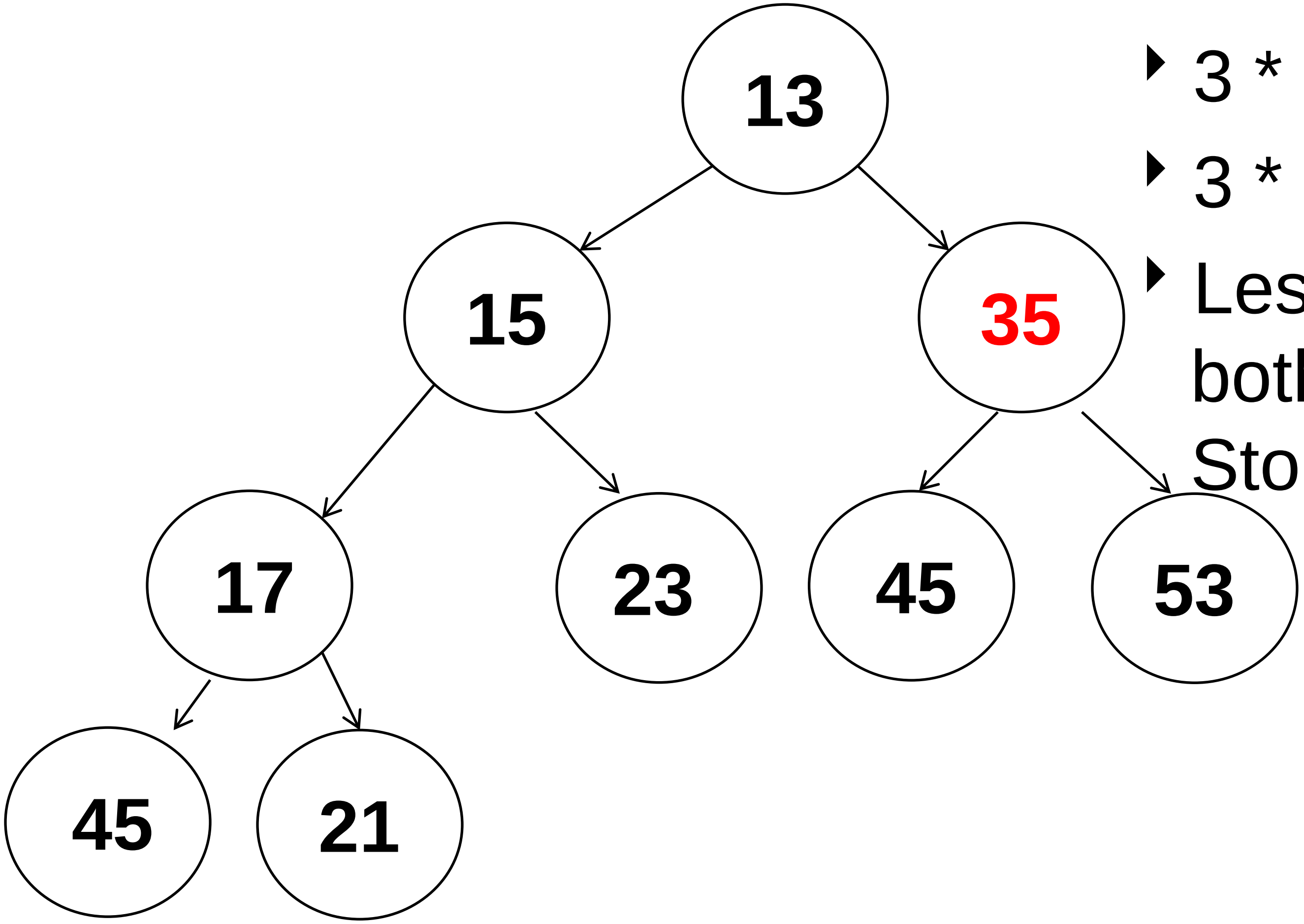
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	35	**15**	**13**	**17**	**23**	**45**	**53**	**45**	**21**						

Dequeue Example



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	13	**15**	**35**	**17**	**23**	**45**	**53**	**45**	**21**						

Dequeue Example



- Min child?
- 3 * 2 = 6 -> 45
- 3 * 2 + 1 = 7 -> 53
- Less than or equal to both of my children! Stop!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	13	**15**	**35**	**17**	**23**	**45**	**53**	**45**	**21**						

Dequeue Code

```
public E dequeue( ) {
    E top = con[1];
    int hole = 1;
    boolean done = false;
    while ( hole * 2 < size && ! done ) {
        int child = hole * 2;
        // see which child is smaller
        if ( con[child].compareTo( con[child + 1] ) > 0 )
            child++;    // child now points to smaller

        // is replacement value bigger than child?
        if (con[size].compareTo( con[child] ) > 0 ) {
            con[hole] = con[child];
            hole = child;
        }
        else
            done = true;
    }
    con[hole] = con[size];
    size--;
    return top;
}
```

Clicker 3 - PriorityQueue Comparison

- Run a Stress test of PQ implemented with Heap and PQ implemented with BinarySearchTree
- What will result be?

A. Heap takes half the time or less of BST

B. Heap faster, but not twice as fast

C. About the same

D. BST faster, but not twice as fast

E. BST takes half the time or less of Heap

Data Structures

- Data structures we have studied
  – arrays, array based lists, linked lists, maps, sets, stacks, queues, trees, binary search trees, graphs, hash tables, red-black trees, priority queues, heaps, tries
- Most program languages have some built in data structures, native or library
- Must be familiar with performance of data structures
  – best learned by implementing them yourself

Data Structures

- We have ***not*** covered every data structure

Abstract data types [edit source | edit beta]

- Container
- Map/Associative array/Dictionary
- Multimap
- List
- Set
- Multiset
- Priority queue
- Queue
- Deque
- Stack
- String
- Tree
- Graph

Some properties of abstract data types:

Structure	Stable	Unique	Cells per Node
Bag (multiset)	no	no	1
Set	no	yes	1
List	yes	no	1
Map	no	yes	2

"Stable" means that input order is retained. Other stru

Arrays [edit source | edit beta]

- Array
- Bidirectional map
- Bit array
- Bit field
- Bitboard
- Bitmap
- Circular buffer
- Control table
- Image
- Dynamic array
- Gap buffer
- Hashed array tree
- Heightmap
- Lookup table
- Matrix
- Parallel array
- Sorted array
- Sparse array
- Sparse matrix
- Iliffe vector
- Variable-length array

Lists [edit source | edit beta]

- Doubly linked list
- Linked list
- Self-organizing list
- Skip list
- Unrolled linked list
- VList
- Xor linked list
- Zipper
- Doubly connected edge list
- Difference list

Heaps [edit source | edit

- Heap
- Binary heap
- Weak heap
- Binomial heap
- Fibonacci heap
  - AF-heap
- 2-3 heap
- Soft heap
- Pairing heap
- Leftist heap
- Treap
- Beap
- Skew heap
- Ternary heap
- D-ary heap

Trees [edit source | edit t

In these data structures eacl

- Tree
- Radix tree
- Suffix tree
- Suffix array
- Compressed suffix array
- FM-index
- Generalised suffix tree
- B-tree
- Judy array
- X-fast tree
- Y-fast tree
- Ctree

Multiway trees [edit source

Graphs [edit source | edit beta]

- Graph
- Adjacency list
- Adjacency matrix
- Graph-structured stack
- Scene graph
- Binary decision diagram
- Zero suppressed decision diagram
- And-inverter graph
- Directed graph
- Directed acyclic graph
- Propositional directed acyclic graph
- Multigraph
- Hypergraph

Other [edit source | edit beta]

- Lightmap
- Winged edge
- Doubly connected edge list
- Quad-edge
- Routing table
- Symbol table

http://en.wikipedia.org/wiki/List_of_data_structures

Data Structures

- deque, b-trees, quad-trees, binary space partition trees, skip list, sparse list, sparse matrix, union-find data structure, Bloom filters, AVL trees, 2-3-4 trees, and more!
- Must be able to learn new and apply new data structures

Topic 26
Dynamic Programming

"Thus, I thought ***dynamic programming*** was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities"

- Richard E. Bellman

Origins

- A method for solving complex problems by breaking them into smaller, easier, sub problems
- Term *Dynamic Programming* coined by mathematician Richard Bellman in early 1950s
  - employed by Rand Corporation
  - Rand had many, large military contracts
  - Secretary of Defense, Charles Wilson "against research, especially mathematical research"
  - how could any one oppose "dynamic"?

Dynamic Programming

- Break big problem up into smaller problems ...

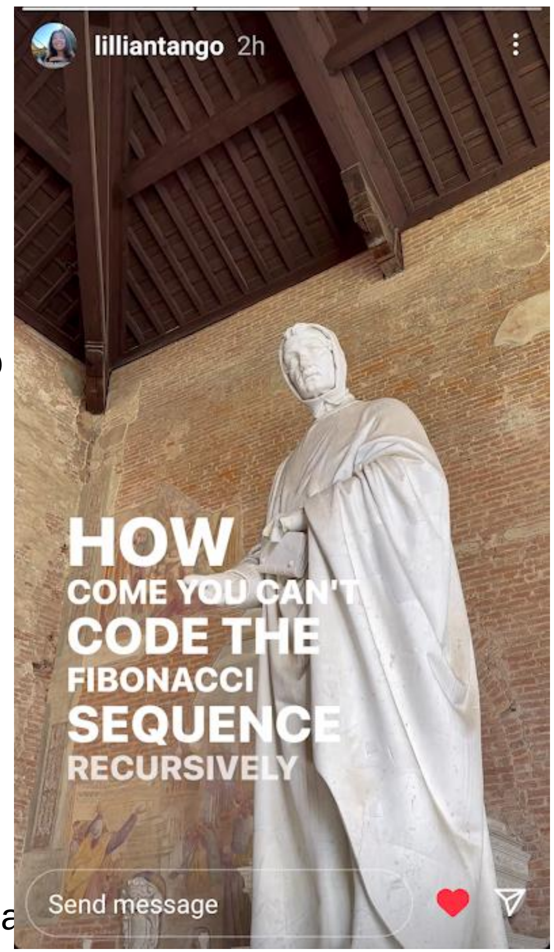
- Sound familiar?

- Recursion?
  N! = 1 for N == 0
  N! = N * (N - 1)! for N > 0

Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 114, …
- $F_1 = 1$
- $F_2 = 1$
- $F_N = F_{N-1} + F_{N-2}$

- Recursive Solution?



Failing Spectacularly

- Naïve recursive method

```
// pre: n > 0
// post: return the nth Fibonacci number
public int fib(int n) {
    if (n <= 2)
        return 1;
    else
        return fib(n – 1) + fib (n – 2);
}
```

- Clicker 1 - Order of this method?

A. O(1) B. O(log N) C. O(N) D. O(N^2) E. O(2^N)

Failing Spectacularly

```
1th fibonnaci number: 1 -  Time: 4.467E-6
2th fibonnaci number: 1 -  Time: 4.47E-7
3th fibonnaci number: 2 -  Time: 4.46E-7
4th fibonnaci number: 3 -  Time: 4.46E-7
5th fibonnaci number: 5 -  Time: 4.47E-7
6th fibonnaci number: 8 -  Time: 4.47E-7
7th fibonnaci number: 13 -  Time: 1.34E-6
8th fibonnaci number: 21 -  Time: 1.787E-6
9th fibonnaci number: 34 -  Time: 2.233E-6
10th fibonnaci number: 55 -  Time: 3.573E-6
11th fibonnaci number: 89 -  Time: 1.2953E-5
12th fibonnaci number: 144 -  Time: 8.934E-6
13th fibonnaci number: 233 -  Time: 2.9033E-5
14th fibonnaci number: 377 -  Time: 3.7966E-5
15th fibonnaci number: 610 -  Time: 5.0919E-5
16th fibonnaci number: 987 -  Time: 7.1464E-5
17th fibonnaci number: 1597 -  Time: 1.08984E-4
```

Failing Spectacularly

```
36th fibonnaci number: 14930352 -  Time: 0.045372057
37th fibonnaci number: 24157817 -  Time: 0.071195386
38th fibonnaci number: 39088169 -  Time: 0.116922086
39th fibonnaci number: 63245986 -  Time: 0.186926245
40th fibonnaci number: 102334155 -  Time: 0.308602967
41th fibonnaci number: 165580141 -  Time: 0.498588795
42th fibonnaci number: 267914296 -  Time: 0.793824734
43th fibonnaci number: 433494437 -  Time: 1.323325593
44th fibonnaci number: 701408733 -  Time: 2.098209943
45th fibonnaci number: 1134903170 -  Time: 3.392917489
46th fibonnaci number: 1836311903 -  Time: 5.506675921
47th fibonnaci number: -1323752223 -  Time: 8.803592621
48th fibonnaci number: 512559680 -  Time: 14.295023778
49th fibonnaci number: -811192543 -  Time: 23.030062974
50th fibonnaci number: -298632863 -  Time: 37.217244704
51th fibonnaci number: -1109825406 -  Time: 60.224418869
```

Clicker 2 - Failing Spectacularly

```
50th fibonnaci number: -298632863 -  Time: 37.217
```

- How long to calculate the 70th Fibonacci Number with this method?

A. 37 seconds

B. 74 seconds

C. 740 seconds

D. 14,800 seconds

E. None of these

Aside - Overflow

- at 47th Fibonacci number overflows int
- Could use BigInteger class instead

```
private static final BigInteger one
        = new BigInteger("1");

private static final BigInteger two
        = new BigInteger("2");

public static BigInteger fib(BigInteger n) {
        if (n.compareTo(two) <= 0)
                return one;
        else {
                BigInteger firstTerm = fib(n.subtract(two));
                BigInteger secondTerm = fib(n.subtract(one));
                return firstTerm.add(secondTerm);
        }
}
```

Aside - BigInteger

- Answers correct beyond 46th Fibonacci number
- Even slower, math on BigIntegers, object creation, and garbage collection

```
37th fibonnaci number: 24157817 -  Time: 2.406739213
38th fibonnaci number: 39088169 -  Time: 3.680196724
39th fibonnaci number: 63245986 -  Time: 5.941275208
40th fibonnaci number: 102334155 -  Time: 9.63855468
41th fibonnaci number: 165580141 -  Time: 15.659745756
42th fibonnaci number: 267914296 -  Time: 25.404417949
43th fibonnaci number: 433494437 -  Time: 40.867030512
44th fibonnaci number: 701408733 -  Time: 66.391845965
45th fibonnaci number: 1134903170 -  Time: 106.964369924
46th fibonnaci number: 1836311903 -  Time: 178.981819822
47th fibonnaci number: 2971215073 -  Time: 287.052365326
```

Slow Fibonacci

- Why so slow?
- Algorithm keeps calculating the same value over and over
- When calculating the 40th Fibonacci number the algorithm calculates the 4th Fibonacci number **<u>24,157,817</u>** times!!!

Fast Fibonacci

- Instead of starting with the big problem and working down to the small problems
- ... start with the small problem and work up to the big problem

```
public static BigInteger fastFib(int n) {
	BigInteger smallTerm = one;
	BigInteger largeTerm = one;
	for (int i = 3; i <= n; i++) {
		BigInteger temp = largeTerm;
		largeTerm = largeTerm.add(smallTerm);
		smallTerm = temp;
	}
	return largeTerm;
}
```

Fast Fibonacci

```
1th fibonnaci number: 1 -  Time: 4.467E-6
2th fibonnaci number: 1 -  Time: 4.47E-7
3th fibonnaci number: 2 -  Time: 7.146E-6
4th fibonnaci number: 3 -  Time: 2.68E-6
5th fibonnaci number: 5 -  Time: 2.68E-6
6th fibonnaci number: 8 -  Time: 2.679E-6
7th fibonnaci number: 13 -  Time: 3.573E-6
8th fibonnaci number: 21 -  Time: 4.02E-6
9th fibonnaci number: 34 -  Time: 4.466E-6
10th fibonnaci number: 55 -  Time: 4.467E-6
11th fibonnaci number: 89 -  Time: 4.913E-6
12th fibonnaci number: 144 -  Time: 6.253E-6
13th fibonnaci number: 233 -  Time: 6.253E-6
14th fibonnaci number: 377 -  Time: 5.806E-6
15th fibonnaci number: 610 -  Time: 6.7E-6
16th fibonnaci number: 987 -  Time: 7.146E-6
17th fibonnaci number: 1597 -  Time: 7.146E-6
```

Fast Fibonacci

```
45th fibonnaci number: 1134903170 -  Time: 1.7419E-5
46th fibonnaci number: 1836311903 -  Time: 1.6972E-5
47th fibonnaci number: 2971215073 -  Time: 1.6973E-5
48th fibonnaci number: 4807526976 -  Time: 2.3673E-5
49th fibonnaci number: 7778742049 -  Time: 1.9653E-5
50th fibonnaci number: 12586269025 -  Time: 2.01E-5
51th fibonnaci number: 20365011074 -  Time: 1.9207E-5
52th fibonnaci number: 32951280099 -  Time: 2.0546E-5
```

```
67th fibonnaci number: 44945570212853 -  Time: 2.3673E-5
68th fibonnaci number: 72723460248141 -  Time: 2.3673E-5
69th fibonnaci number: 117669030460994 -  Time: 2.412E-5
70th fibonnaci number: 190392490709135 -  Time: 2.4566E-5
71th fibonnaci number: 308061521170129 -  Time: 2.4566E-5
72th fibonnaci number: 498454011879264 -  Time: 2.5906E-5
73th fibonnaci number: 806515533049393 -  Time: 2.5459E-5
74th fibonnaci number: 1304969544928657 -  Time: 2.546E-5
```

```
200th fibonnaci number: 280571172992510140037611932413038677189525 -  Time: 1.0273E-5
```

Memoization

- Store (cache) results from computations for later lookup
- Memoization of Fibonacci Numbers

```
public class FibMemo {

	private static List<BigInteger> lookupTable;

	private static final BigInteger ONE
		= new BigInteger("1");

	static {
		lookupTable = new ArrayList<>();
		lookupTable.add(null);
		lookupTable.add(ONE);
		lookupTable.add(ONE);
	}
```

Fibonacci Memoization

```
public static BigInteger fib(int n) {
    // check lookup table
    if (n < lookupTable.size()) {
        return lookupTable.get(n);
    }

    // Calculate nth Fibonacci.
    // Don't repeat work. Start with the last known.
    BigInteger smallTerm
        = lookupTable.get(lookupTable.size() - 2);
    BigInteger largeTerm
        = lookupTable.get(lookupTable.size() - 1);
    for(int i = lookupTable.size(); i <= n; i++) {
        BigInteger temp = largeTerm;
        largeTerm = largeTerm.add(smallTerm);
        lookupTable.add(largeTerm); // memo
        smallTerm = temp;
    }
    return largeTerm;
}
```

Dynamic Programming

- When to use?
- When a big problem can be broken up into sub problems.
- **Solution to original problem can be calculated from results of smaller problems.**
  - larger problems depend on previous solutions
- **Sub problems must have a natural ordering from smallest to largest (simplest to hardest)**
- Multiple techniques within DP

DP Algorithms

- Step 1: Define the *meaning* of the subproblems (in English for sure, Mathematically as well if you find it helpful).
- Step 2: Show where the solution will be found.
- Step 3: Show how to set the first subproblem.
- Step 4: Define the order in which the subproblems are solved.
- Step 5: Show how to compute the answer to each subproblem using the previously computed subproblems. (This step is typically polynomial, once the other subproblems are solved.)

Dynamic Programming Requires:

- overlapping sub problems:
  - problem can be broken down into sub problems
  - obvious with Fibonacci
  - Fib(N) = Fib(N - 2) + Fib(N - 1) for N >= 3
- optimal substructure:
  - the optimal solution for a problem can be constructed from optimal solutions of its sub problems
  - In Fibonacci just sub problems, no optimality
  - min coins opt(36) = 1_{12} + opt(24) [1, 5, 12]

Dynamic Programing Example

- Another simple example
- Finding the best solution involves finding the best answer to simpler problems
- Given a set of coins with values (V_1, V_2, … V_N) and a target sum S, find the fewest coins required to equal S
- What is Greedy Algorithm approach?
- Does it always work?
- {1, 5, 12} and target sum = 15 (12, 1, 1, 1)
- Could use recursive backtracking …

Minimum Number of Coins

- To find minimum number of coins to sum to 15 with values {1, 5, 12} start with sum 0
  – recursive backtracking would likely start with 15
- Let M(S) = minimum number of coins to sum to S
- At each step look at target sum, coins available, and previous sums
  – pick the smallest option

Minimum Number of Coins

- M(0) = 0 coins
- M(1) = 1 coin (1 coin)
- M(2) = 2 coins (1 coin + M(1))
- M(3) = 3 coins (1 coin + M(2))
- M(4) = 4 coins (1 coin + M(3))
- M(5) = interesting, 2 options available:

  1 + others OR single 5

if 1 then 1 + M(4) = 5, if 5 then 1 + M(0) = 1

clearly better to pick the coin worth 5

Minimum Number of Coins

- M(0) = 0
- M(1) = 1 (1 coin)
- M(2) = 2 (1 coin + M(1))
- M(3) = 3 (1 coin + M(2))
- M(4) = 4 (1 coin + M(3))
- M(5) = 1 (1 coin + M(0))
- M(6) = 2 (1 coin + M(5))
- M(7) = 3 (1 coin + M(6))
- M(8) = 4 (1 coin + M(7))
- M(9) = 5 (1 coin + M(8))
- M(10) = 2 (1 coin + M(5))
  options: 1, 5
- M(11) = 2 (1 coin + M(10))
  options: 1, 5
- M(12) = 1 (1 coin + M(0))
  options: 1, 5, 12
- M(13) = 2 (1 coin + M(12))
  options: 1, 12
- M(14) = 3 (1 coin + M(13))
  options: 1, 12
- M(15) = 3 (1 coin + M(10))
  options: 1, 5, 12

KNAPSACK PROBLEM - RECURSIVE BACKTRACKING AND DYNAMIC PROGRAMMING

Knapsack Problem

- A variation of a *bin packing* problem
- Similar to fair teams problem from recursion assignment
- You have a set of items
- Each item has a weight and a value
- You have a knapsack with a weight limit
- Goal: Maximize the ***value*** of the items you put in the knapsack without exceeding the weight limit

Knapsack Example

- Items:

Item Number	Weight of Item	Value of Item	Value per unit Weight
1	1	6	6.0
2	2	11	5.5
3	4	1	0.25
4	4	12	3.0
5	6	19	3.167
6	7	12	1.714

- Weight Limit = 8
- A greedy solution: Take the highest ratio item that will fit: (1, 6), (2, 11), and (4, 12)
- Total value = 6 + 11 + 12 = 29
- Clicker 3 - Is this optimal? A. No B. Yes

Knapsack - Recursive Backtracking

```
private static int knapsack(ArrayList<Item> items,
        int current, int capacity) {

    int result = 0;
    if (current < items.size()) {
        // don't use item
        int withoutItem
            = knapsack(items, current + 1, capacity);
        int withItem = 0;
        // if current item will fit, try it
        Item currentItem = items.get(current);
        if (currentItem.weight <= capacity) {
            withItem += currentItem.value;
            withItem += knapsack(items, current + 1,
                    capacity - currentItem.weight);
        }
        result = Math.max(withoutItem, withItem);
    }
    return result;
}
```

Knapsack - Dynamic Programming

- Recursive backtracking starts with max capacity and makes choice for items: choices are:
  - take the item if it fits
  - don't take the item
- Dynamic Programming, start with simpler problems
- Reduce number of items available
- … AND Reduce weight limit on knapsack
- Creates a 2d array of possibilities

Knapsack - Optimal Function

- OptimalSolution(items, weight) is best solution given a subset of items and a weight limit
- 2 options:
- OptimalSolution does not select i^{th} item
  - select best solution for items 1 to i - 1with weight limit of w
- OptimalSolution selects i^{th} item
  - New weight limit = w - weight of i^{th} item
  - select best solution for items 1 to i - 1with new weight limit

Knapsack Optimal Function

- OptimalSolution(items, weight limit) =

  0 if 0 items

  OptimalSolution(items - 1, weight) if weight of ith item is greater than allowed weight
  $w_i > w$ (In others i^{th} item doesn't fit)

  max of (OptimalSolution(items - 1, w),
  value of i^{th} item +
  OptimalSolution(items - 1, $w - w_i$)

Knapsack - Algorithm

- Create a 2d array to store value of best option given subset of items and possible weights

Item Number	Weight of Item	Value of Item
1	1	6
2	2	11
3	4	1
4	4	12
5	6	19
6	7	12

- In our example 0 to 6 items and weight limits of of 0 to 8
- Fill in table using OptimalSolution Function

Knapsack Algorithm

Given N items and WeightLimit

Create Matrix M with N + 1 rows and WeightLimit + 1 columns

```
For weight = 0 to WeightLimit
        M[0, w] = 0

For item = 1 to N
        for weight = 1 to WeightLimit
                if(weight of ith item > weight)
                        M[item, weight] = M[item - 1, weight]
                else
                        M[item, weight] = max of
                        M[item - 1, weight] AND
                        value of item + M[item - 1, weight - weight of item]
```

Knapsack - Table

Item	Weight	Value
1	1	6
2	2	11
3	4	1
4	4	12
5	6	19
6	7	12

items / capacity	0	1	2	3	4	5	6	7	8
{}	0	0	0	0	0	0	0	0	0
{1}									
{1,2}									
{1, 2, 3}									
{1, 2, 3, 4}									
{1, 2, 3, 4, 5}									
{1, 2, 3, 4, 5, 6}									

Knapsack - Completed Table

items / weight	0	1	2	3	4	5	6	7	8
{}	0	0	0	0	0	0	0	0	0
{1} [1, 6]	0	6	6	6	6	6	6	6	6
{1,2} [2, 11]	0	6	11	17	17	17	17	17	17
{1, 2, 3} [4, 1]	0	6	11	17	17	17	17	18	18
{1, 2, 3, 4} [4, 12]	0	6	11	17	17	18	23	29	29
{1, 2, 3, 4, 5} [6, 19]	0	6	11	17	17	18	23	29	30
{1, 2, 3, 4, 5, 6} [7, 12]	0	6	11	17	17	18	23	29	30

Knapsack - Items to Take

items / weight	0	1	2	3	4	5	6	7	8
{}	0	0	0	0	0	0	0	0	0
{1} [1, 6]	0	6	6	6	6	6	6	6	6
{1,2} [2, 11]	0	6	11	17	17	17	17	17	17
{1, 2, 3} [4, 1]	0	6	11	17	17	17	17	17	17
{1, 2, 3, 4} [4, 12]	0	6	11	17	17	18	23	29	29
{1, 2, 3, 4, 5} [6, 19]	0	6	11	17	17	18	23	29	30
{1, 2, 3, 4, 5, 6} [7, 12]	0	6	11	17	17	18	23	29	30

Dynamic Knapsack

```
// dynamic programming approach
public static int knapsack(ArrayList<Item> items, int maxCapacity) {
    final int ROWS = items.size() + 1;
    final int COLS = maxCapacity + 1;
    int[][] partialSolutions = new int[ROWS][COLS];
    // first row and first column all zeros

    for(int item = 1; item <= items.size(); item++) {
        for(int capacity = 1; capacity <= maxCapacity; capacity++) {
            Item currentItem = items.get(item - 1);
            int bestSoFar = partialSolutions[item - 1][capacity];
            if( currentItem.weight <= capacity) {
                int withItem = currentItem.value;
                int capLeft = capacity - currentItem.weight;
                withItem += partialSolutions[item - 1][capLeft];
                if (withItem > bestSoFar) {
                    bestSoFar = withItem;
                }
            }
            partialSolutions[item][capacity] = bestSoFar;
        }
    }
    return partialSolutions[ROWS - 1][COLS - 1];
}
```

Dynamic vs. Recursive Backtracking Timing Data

Number of items: 32. Capacity: 123
Recursive knapsack. Answer: 740, time: 10.0268025
Dynamic knapsack.   Answer: 740, time: 3.43999E-4

Number of items: 33. Capacity: 210
Recursive knapsack. Answer: 893, time: 23.0677814
Dynamic knapsack.   Answer: 893, time: 6.76899E-4

Number of items: 34. Capacity: 173
Recursive knapsack. Answer: 941, time: 89.8400178
Dynamic knapsack.   Answer: 941, time: 0.0015702

Number of items: 35. Capacity: 93
Recursive knapsack. Answer: 638, time: 81.0132219
Dynamic knapsack.   Answer: 638, time: 2.95601E-4

Clicker 4

- Which approach to the knapsack problem uses more memory?

A. the recursive backtracking approach

B. the dynamic programming approach

C. they use about the same amount of memory

Topic 27
Functional Programming

Functional Programming with Java 8

"**It's a long-standing principle of programming style that the functional elements of a program should not be too large.** If some component of a program grows beyond the stage where it's readily comprehensible, it becomes a mass of complexity which conceals errors as easily as a big city conceals fugitives. **Such software will be hard to read, hard to test, and hard to debug**." – Paul Graham

What is FP?

- **functional programming:** A style of programming that emphasizes the use of **functions** (methods) to decompose a complex task into subtasks.
  - Examples of functional languages:
    LISP, Scheme, ML, Haskell, Erlang, F#, Clojure, ...

- Java is considered an object-oriented language, not a functional language.

- But Java 8 added several language features to facilitate a partial functional programming style.
  - Popular contemporary languages tend to be
    Multi Paradigm Languages

Java 8 FP features

- 1. Effect-free programming
- 2. First-class functions
- 3. Processing structured data via functions
- 4. Function closures
- 5. Higher-order operations on collections

Effect-free code (19.1)

- **side effect**: A change to the state of an object or program variable produced by a call on a function (i.e., a method).
  - example: modifying the value of a variable
  - example: printing output to System.out
  - example: reading/writing data to a file, collection, or network

```
int result = f(x) + f(x);
int result = 2 * f(x);
```

- Are the two above statements equivalent?
  - Yes, **if** the function f() has no *side effects.*
  - One goal of functional programming is to minimize side effects.

Code w/ side effects

```
public class SideEffect {

    public static int x;

    public static int f(int n) {
        x = x * 2;
        return x + n;
    }

    // what if it were 2 * f(x)?
    public static void main(String[] args) {
        x = 5;
        int result = f(x) + f(x);
        System.out.println(result);
    }
}
```

First-class functions (19.2)

- **first-class citizen**: An element of a programming language that is tightly integrated with the language and supports the full range of operations generally available to other entities in the language.

- In functional programming, functions (methods) are treated as first-class citizens of the languages.
  - can store a function in a variable
  - can pass a function as a parameter to another function
  - can return a function as a value from another function
  - can create a collection of functions
  - ...

Lambda expressions

- **lambda expression** ("lambda"): Expression that describes a function by specifying its parameters and return value.
  - Java 8 adds support for lambda expressions.
  - Essentially an anonymous function (aka method)
- Syntax:

  (***parameters***) -> ***expression***

- Example:

```
(x) -> x * x        // squares a number
```

  - The above is roughly equivalent to:

```
public static int squared(int x) {
    return x * x;
}
```

MathMatrix add / subtract

- Recall the MathMatrix class:

```
public MathMatrix add(MathMatrix rhs) {
  int[][] res = new int[cells.length][cells[0].length];
  for (int r = 0; r < res.length; r++)
    for (int c = 0; c < res[0].length; c++)
      res[r][c] = cells[r][c] + rhs.cells[r][c];
  return new MathMatrix(res);
}

public MathMatrix subtract(MathMatrix rhs) {
  int[][] res = new int[cells.length][cells[0].length];
  for (int r = 0; r < res.length; r++)
    for (int c = 0; c < res[0].length; c++)
      res[r][c] = cells[r][c] - rhs.cells[r][c];
  return new MathMatrix(res);
}
```

MathMatrix add / subtract

- GACK!!!

- How do we generalize the idea of "add or subtract"?
  - How much work would it be to add other operators?
  - Can functional programming help remove the repetitive code?

Code w/ lambdas

- We can represent the math operation as a lambda:

```
public MathMatrix add(MathMatrix rhs) {
   return getMat(rhs, (x, y) -> x + y);
}


public MathMatrix subtract(MathMatrix rhs) {
   return getMat(rhs, (x, y) -> x - y);
}
```

getMat method

```
private MathMatrix getMat(MathMatrix rhs,
                          IntBinaryOperator operator) {

  int[][] res = new int[cells.length][cells[0].length];

  for (int r = 0; r < cells.length; r++) {
    for (int c = 0; c < cells[0].length; c++) {
      int temp1 = cells[r][c];
      int temp2 = rhs.cells[r][c];
      res[r][c] = operator.applyAsInt(temp1, temp2);
    }
  }
  return new MathMatrix(res);
}
```

[// IntBinaryOperator Documentation]()

Clicker 1

- Which of the following is a lambda that checks if x divides evenly into y?

A. `(x, y) -> y / x == 0`

B. `(x, y) -> x / y == 0`

C. `(x, y) -> y % x == 0`

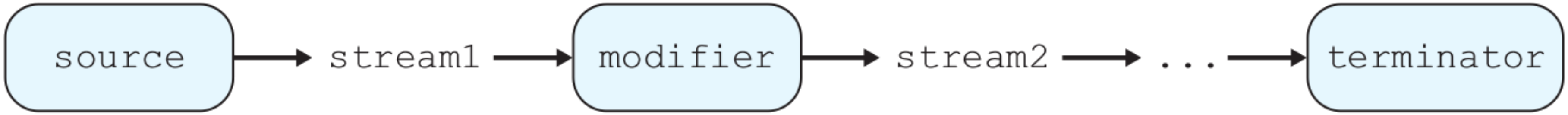
D. `(x, y) -> x % y == 0`

E. `(x, y) -> y * x == 0`

Streams (19.3)

- **stream**: A sequence of elements from a data source that supports aggregate operations.

- Streams operate on a data source and modify it:



  - example: print each element of a collection
  - example: sum each integer in a file
  - example: concatenate strings together into one large string
  - example: find the largest value in a collection
  - ...

Code w/o streams

- Non-functional programming sum code:

```
// compute the sum of the squares of integers 1-5
int sum = 0;
for (int i = 1; i <= 5; i++) {
    sum += i * i;
}
```

The map modifier

- The `map` modifier applies a lambda to each stream element:
  - **higher-order function**: Takes a function as an argument.
- Abstracting away loops (and data structures)

```
// compute the sum of the squares of integers 1-5
int sum = IntStream.range(1, 6)
    .map(n -> n * n)
    .sum();


// the stream operations are as follows:

IntStream.range(1, 6) -> [1, 2, 3, 4, 5]
               -> map -> [1, 4, 9, 16, 25]
               -> sum -> 55
```

The filter modifier

- The `filter` stream modifier removes/keeps elements of the stream using a boolean lambda:

```
// compute the sum of squares of odd integers
int sum =
     IntStream.of(3, 1, 4, 1, 5, 9, 2, 6, 5, 3)
    .filter(n -> n % 2 != 0)
    .map(n -> n * n)
    .sum();


// the stream operations are as follows:
IntStream.of  -> [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
    -> filter -> [3, 1, 1, 5, 9, 5, 3]
       -> map -> [9, 1, 1, 25, 81, 25, 9]
       -> sum -> 151
```

Streams and methods

- using streams as part of a regular method:

```
// Returns true if the given integer is prime.
// Assumes n >= 2.
public static boolean isPrime(int n) {
    return IntStream.range(1, n + 1)
        .filter(x -> n % x == 0)
        .count() == 2;
}
```

- How to make this method faster?

The reduce modifier

- The `reduce` modifier (method) combines elements of a stream using a lambda combination function.
  - Accepts two parameters: an initial value and a lambda to combine that initial value with each subsequent value in the stream.

```
// Returns n!, or 1 * 2 * 3 * ... * (n-1) * n.
// Assumes n is non-negative.
public static int factorial(int n) {
    return IntStream.range(2, n + 1)
            .reduce(1, (a, b) -> a * b);
}
```

Stream operators

Method name	**Description**
`anyMatch(`**f**`)`	returns true if any elements of stream match given predicate
`allMatch(`**f**`)`	returns true if all elements of stream match given predicate
`average()`	returns arithmetic mean of numbers in stream
`collect(`**f**`)`	convert stream into a collection and return it
`count()`	returns number of elements in stream
`distinct()`	returns unique elements from stream
`filter(`**f**`)`	returns the elements that match the given predicate
`forEach(`**f**`)`	performs an action on each element of stream
`limit(`**size**`)`	returns only the next **size** elements of stream
`map(`**f**`)`	applies the given function to every element of stream
`noneMatch(`**f**`)`	returns true if zero elements of stream match given predicate

Stream operators

Method name	**Description**
`parallel()`	returns a multithreaded version of this stream
`peek(`**f**`)`	examines the first element of stream only
`reduce(`**f**`)`	applies the given binary reduction function to stream elements
`sequential()`	single-threaded, opposite of parallel()
`skip(`**n**`)`	omits the next n elements from the stream
`sorted()`	returns stream's elements in sorted order
`sum()`	returns sum of elements in stream
`toArray()`	converts stream into array

Static method	**Description**
`concat(`**s1**`, `**s2**`)`	glues two streams together
`empty()`	returns a zero-element stream
`iterate(`**seed**`, `**f**`)`	returns an infinite stream with given start element
`of(`**values**`)`	converts the given values into a stream
`range(`**start**`, `**end**`)`	returns a range of integer values as a stream

Clicker 2

- What is output by the following code?

```
int x1 = IntStream.of(-2, 5, 5, 10, -6)
                  .map(x -> x / 2)
                  .filter(y -> y > 0)
                  .sum();
System.out.print(x1);
```

A. `(-2, 5, 5, 10, -6)`

B. `6`

C. `(-1, 2.5, 2.5, 5, -3)`

D. `9`

E. `20`

Optional results

- Some stream terminators like max return an "optional" result because the stream might be empty or not contain the result:

```
// print largest multiple of 10 in list
// (does not compile!)
int largest =
    IntStream.of(55, 20, 19, 31, 40, -2, 62, 30)
    .filter(n -> n % 10 == 0)
    .max();
System.out.println(largest);
```

Optional results fix

- To extract the optional result, use a "get as" terminator.
  - Converts type OptionalInt to Integer

```
// print largest multiple of 10 in list
// (this version compiles and works.)
int largest =
    IntStream.of(55, 20, 19, 31, 40, -2, 62, 30)
    .filter(n -> n % 10 == 0)
    .max()
    .getAsInt();
System.out.println(largest);
```

Method references

ClassName::methodName

- A method reference lets you pass a method where a lambda would otherwise be expected:

```
// compute sum of absolute values of even ints
int[] numbers = {3, -4, 8, 4, -2, 17,
                 9, -10, 14, 6, -12};
int sum = Arrays.stream(numbers)
    .map(Math::abs)
    .filter(n -> n % 2 == 0)
    .distinct()
    .sum();
```

Ramya, Spring 2018

- “Okay, but why?”
- Programming with Streams is an alternative to writing out the loops ourselves
- Streams “abstract away” the loop structures we have spent so much time writing
- Why didn’t we just start with these?

Stream exercises

- Write a method **`sumAbsVals`** that uses stream operations to compute the sum of the absolute values of an array of integers. For example, the sum of `{-1, 2, -4, 6, -9}` is `22`.

- Write a method **`largestEven`** that uses stream operations to find and return the largest even number from an array of integers. For example, if the array is `{5, -1, 12, 10, 2, 8}`, your method should return `12`. You may assume that the array contains at least one even integer.

Closures (19.4)

- **bound/free variable**: In a lambda expression, parameters are bound variables while variables in the outer containing scope are free variables.
- **function closure**: A block of code defining a function along with the definitions of any free variables that are defined in the containing scope.

```
// free variables: min, max, multiplier
// bound variables: x, y
int min = 10;
int max = 50;
int multiplier = 3;
compute((x, y) -> Math.max(x, min) *
                  Math.max(y, max) * multiplier);
```

(19.4) Higher Order Operations on Collections (Streams and Arrays)

- An array can be converted into a stream with Arrays.stream:

```
// compute sum of absolute values of even ints
int[] numbers = {3, -4, 8, 4, -2, 17,
                 9, -10, 14, 6, -12};
int sum = Arrays.stream(numbers)
    .map(n -> Math.abs(n))
    .filter(n -> n % 2 == 0)
    .distinct()
    .sum();
```

Method references

ClassName::methodName

- A method reference lets you pass a method where a lambda would otherwise be expected:

```
// compute sum of absolute values of even ints
int[] numbers = {3, -4, 8, 4, -2, 17,
                 9, -10, 14, 6, -12};
int sum = Arrays.stream(numbers)
    .map(Math::abs)
    .filter(n -> n % 2 == 0)
    .distinct()
    .sum();
```

Streams and lists

- A collection can be converted into a stream by calling its `stream` method:

```
// compute sum of absolute values of even ints
ArrayList<Integer> list =
        new ArrayList<Integer>();
list.add(-42);
list.add(-17);
list.add(68);
list.stream()
    .map(Math::abs)
    .forEach(System.out::println);
```

Streams and strings

```
// convert into set of lowercase words
List<String> words = Arrays.asList(
    "To", "be", "or", "Not", "to", "be");
Set<String> words2 = words.stream()
    .map(String::toLowerCase)
    .collect(Collectors.toSet());
System.out.println("word set = " + words2);
```

output:

```
word set = [not, be, or, to]
```

Streams and files

```
// find longest line in the file
int longest = Files.lines(Paths.get("haiku.txt"))
    .mapToInt(String::length)
    .max()
    .getAsInt();
```

stream operations:

```
Files.lines -> ["haiku are funny",
       "but sometimes they don't make sense",
                "refrigerator"]
-> mapToInt -> [15, 35, 12]
     -> max -> 35
```

Stream exercises

- Write a method **fiveLetterWords** that accepts a file name as a parameter and returns a count of the number of unique lines in the file that are exactly five letters long. Assume that each line in the file contains at least one word.

- Write a method using streams that finds and prints the first 5 perfect numbers. (Recall a perfect number is equal to the sum of its unique integer divisors, excluding itself.)