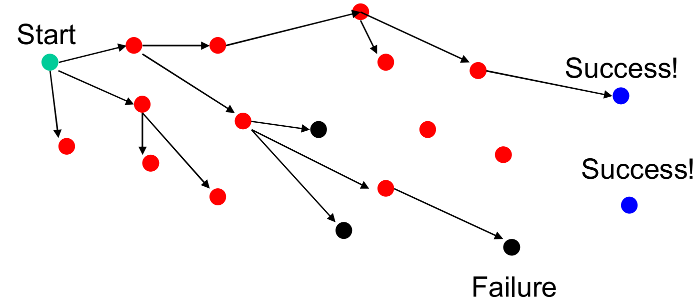
Topic 13
Recursive Backtracking

"In ancient times, before computers were invented, alchemists studied the mystical properties of numbers. Lacking computers, they had to rely on dragons to do their work for them. The dragons were clever beasts, but also lazy and bad-tempered. The worst ones would sometimes burn their keeper to a crisp with a single fiery belch. But most dragons were merely uncooperative, as violence required too much energy. This is the story of how Martin, an alchemist's apprentice, discovered recursion by outsmarting a lazy dragon."

- David S. Touretzky, *Common Lisp: A Gentle Introduction to Symbolic Computation*

Backtracking

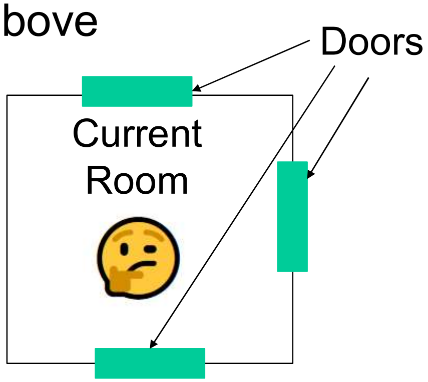


Problem space consists of states (nodes) and actions (paths that lead to new states). When in a node can can only see paths to connected nodes

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

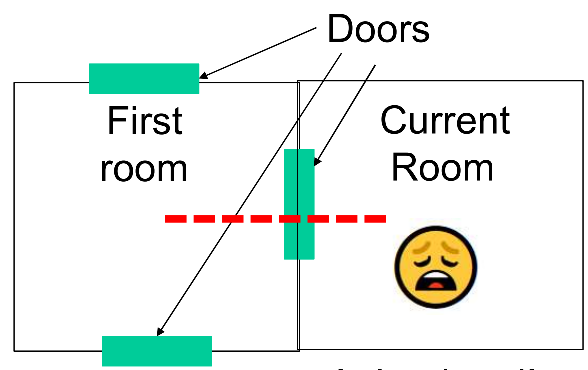
- Which door should we take?
- A view from above



Exit out there, some where … we hope

Escaping a Maze

- Try door to the east

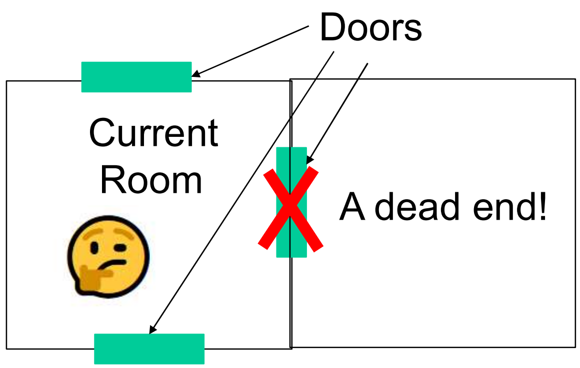


Exit out there, some where … we hope

A dead end!

Escaping a Maze

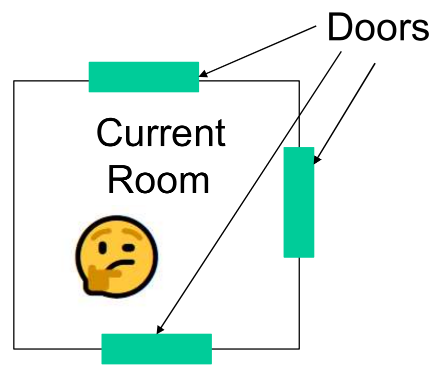
- Back we go



Exit out there, some where … we hope

Escaping a Maze

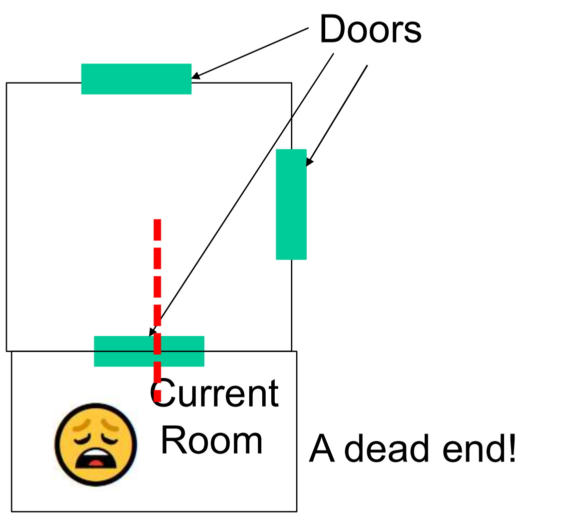
- What if we knew the exit was to the south?



Exit out there, some where to the south!

Escaping a Maze

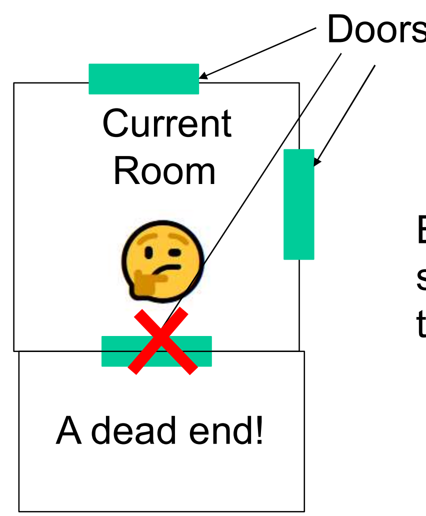
- Start over. What if we knew the exit was to the south?



Exit out there, some where to the south!

Escaping a Maze

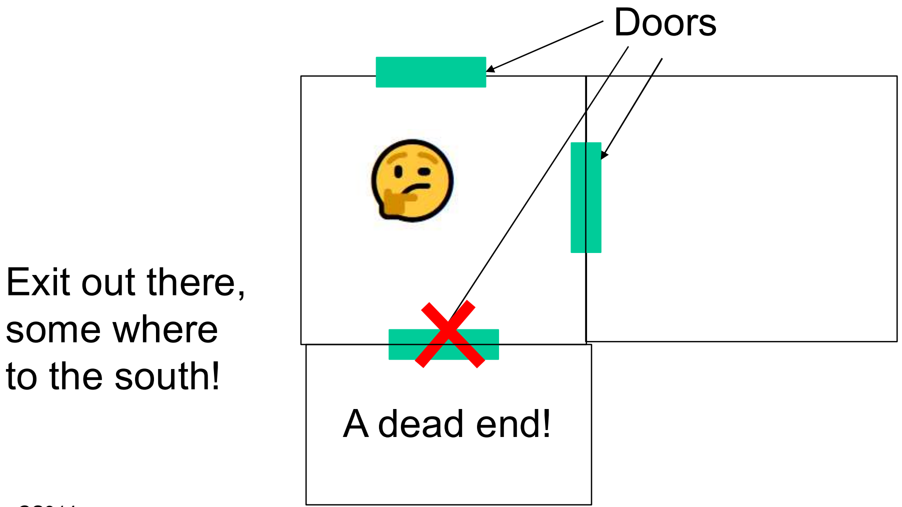
- What if we knew the exit was to the south?



Exit out there, some where to the south!

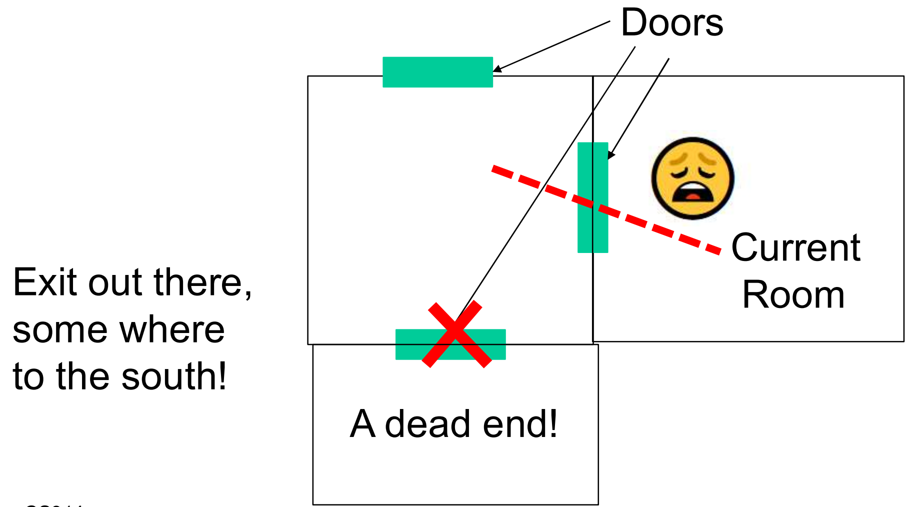
Escaping a Maze

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

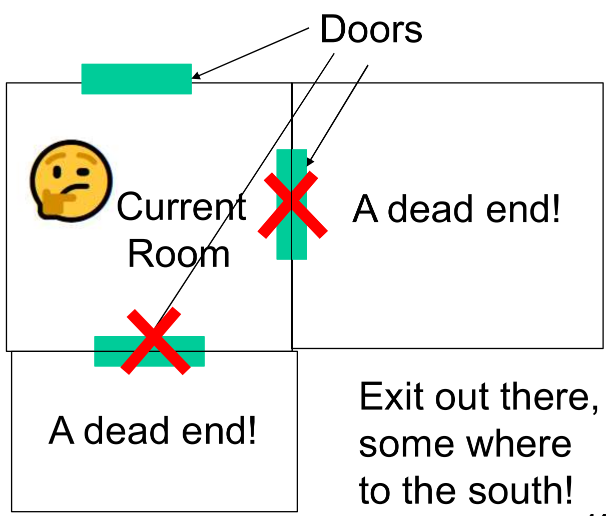


Escaping a Maze

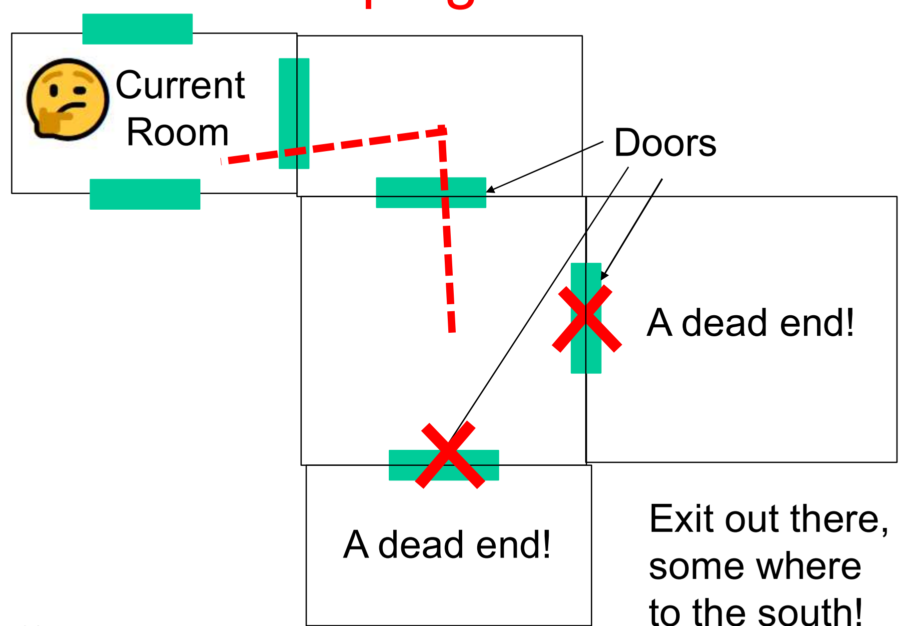
- What if we knew the exit was to the south?



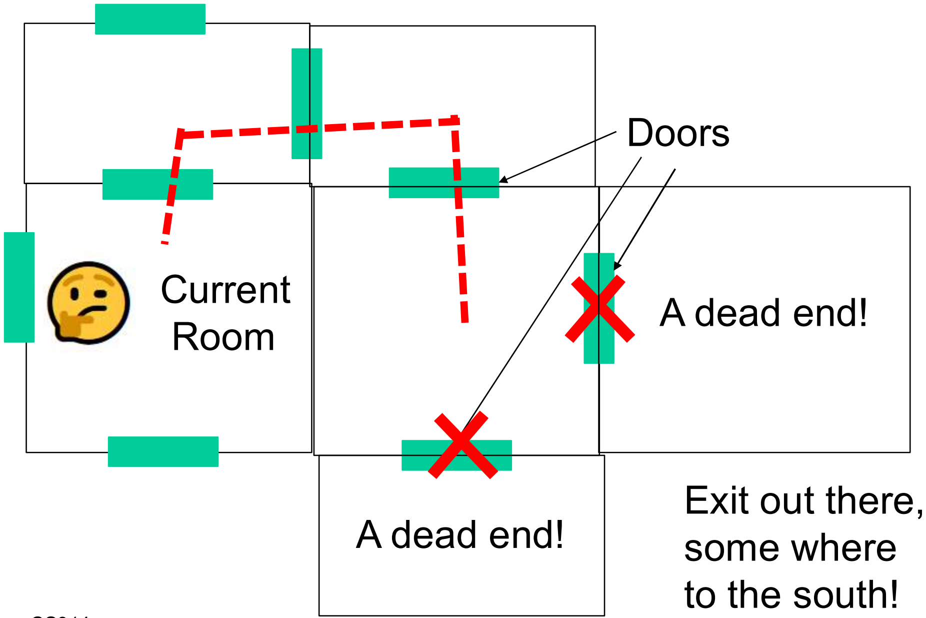
Escaping a Maze



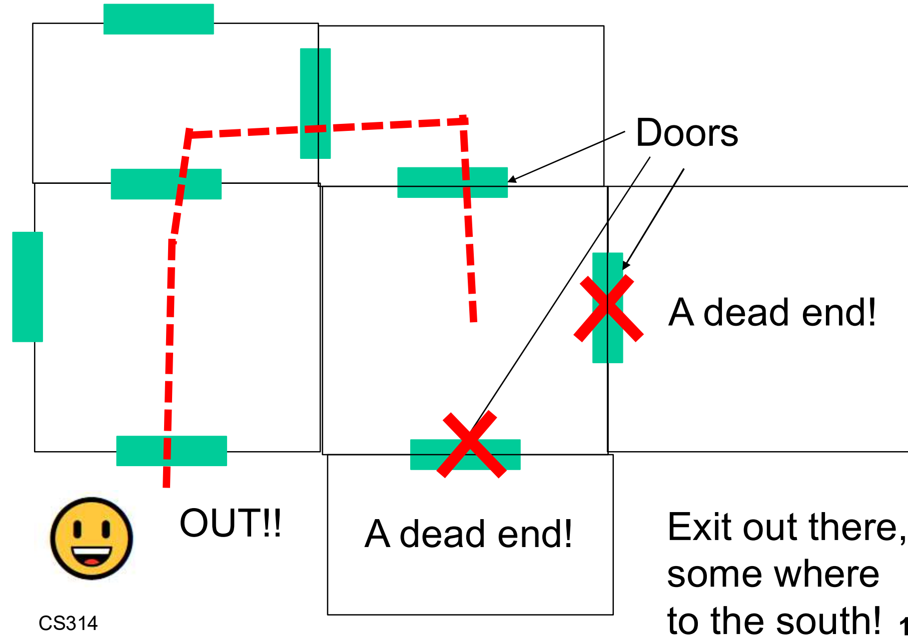
Escaping a Maze



Escaping a Maze



Escaping a Maze



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

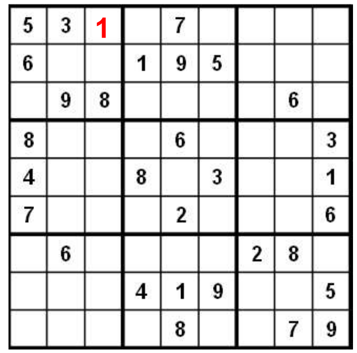
Solving Sudoku – Brute Force

- 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



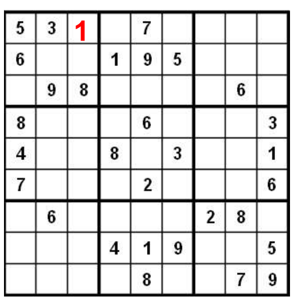
Clicker 1

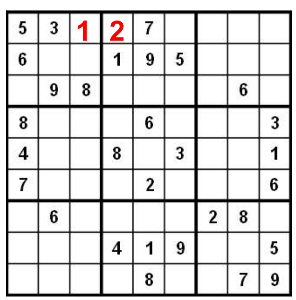
- After placing a number in a cell is the remaining problem very similar to the original problem?

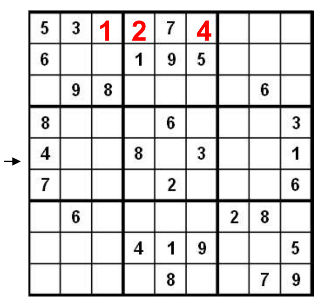
A. No

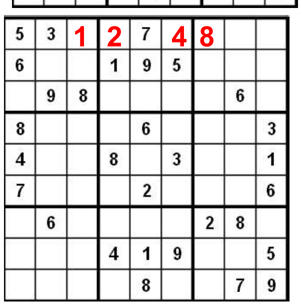
B. Yes

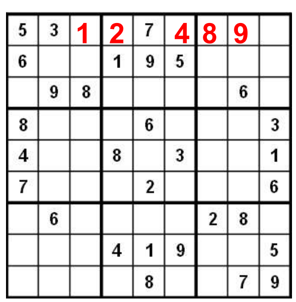
Solving Sudoku – Later Steps







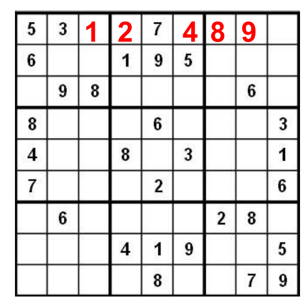




uh oh!

Sudoku – A Dead End

- We have reached a dead end in our search



- 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

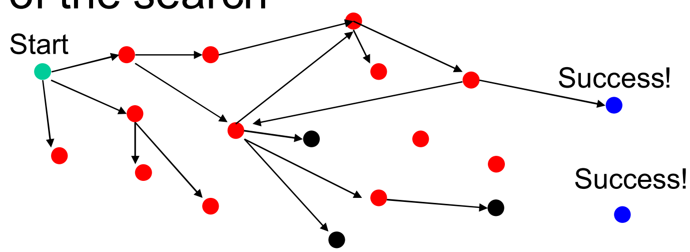
Recursive Backtracking

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

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

Goals of Backtracking

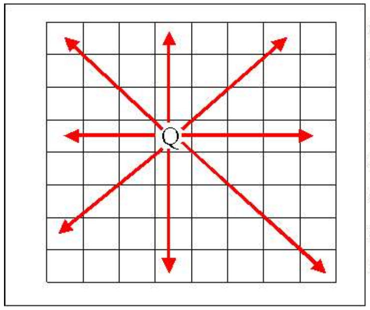
- Possible goals
  - Find a path to success
  - Find all paths to success
  - Find the best path to success
- Not all problems are exactly alike, and finding one success node may not be the end of the search

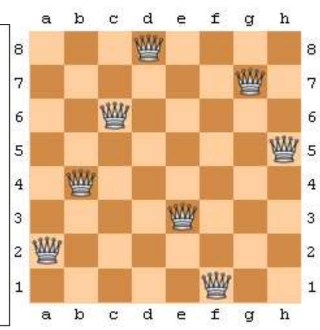


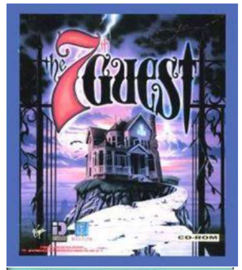
The ~~8~~ N Queens Problem

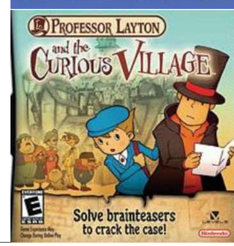

The 8 Queens Problem

- 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

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

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

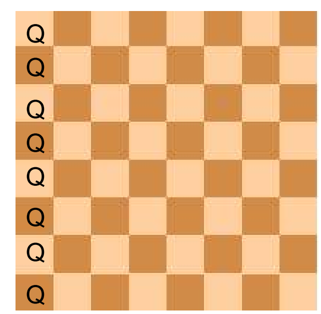
C. 2

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

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
```

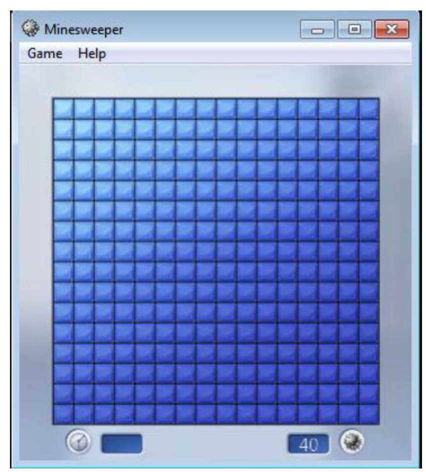

N Queens

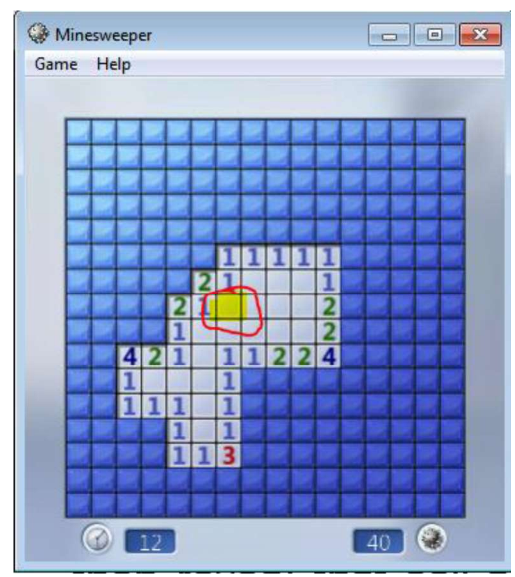
- The *problem* with N queens is you don't know how many for loops to write.
- Do the problem recursively
- Write recursive code with class and demo
  – show backtracking with breakpoint and debugging option

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

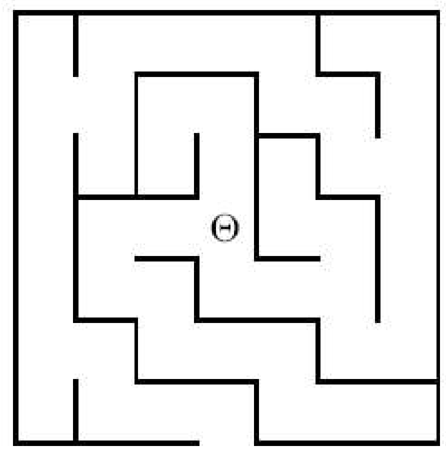




Minesweeper Reveal Algorithm

- 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

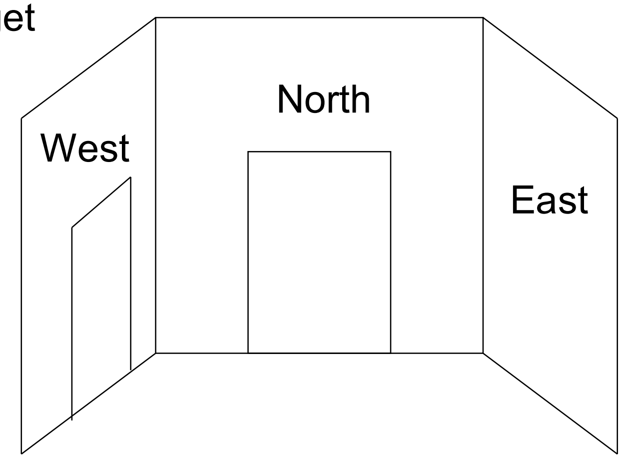
Another Backtracking Problem A Simple Maze



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.

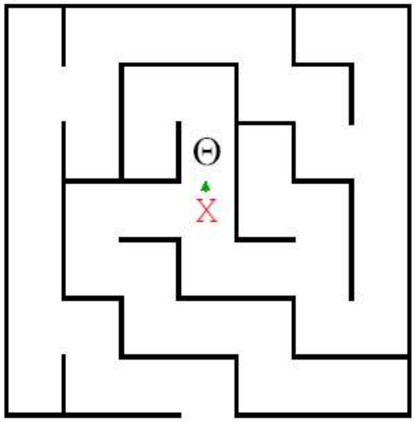
  }

  }

  Unmark the current square.

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

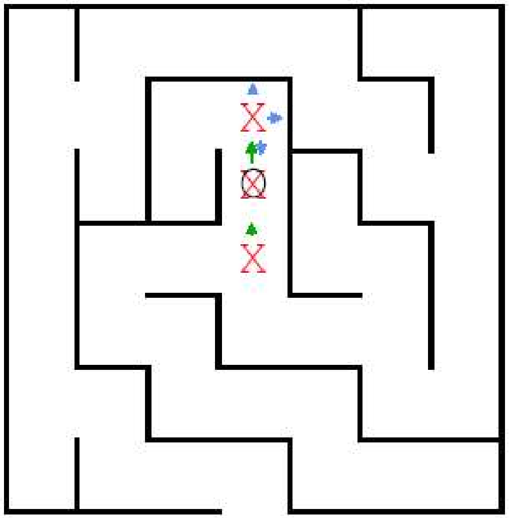
Backtracking in Action



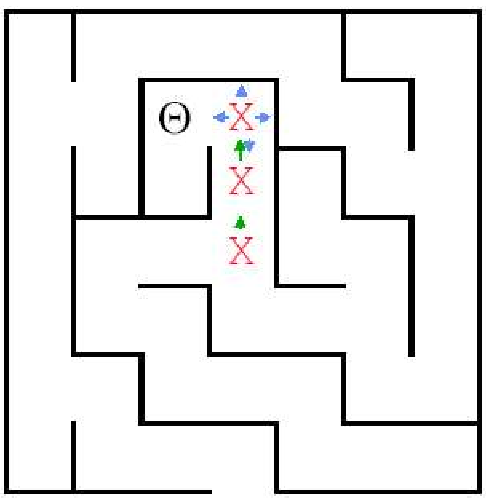
The crucial part of the algorithm is the for loop that takes us through the alternatives from the curren square. Here we have move to the North.

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


Backtracking in Action

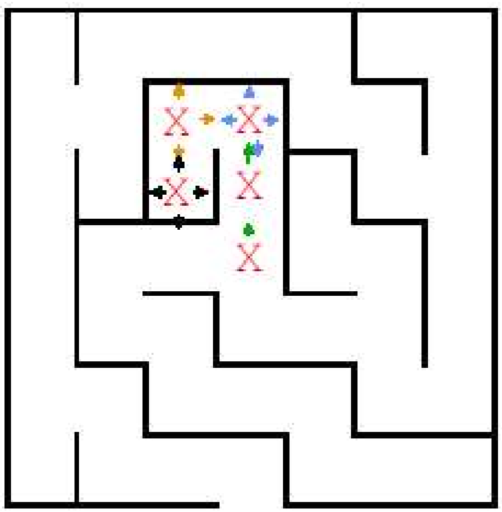


Here we have moved North again, but there is a wall to the North . East is also blocked, so we try South. That call discovers that the square is marked, so it just returns.



So the next move we can make is West.

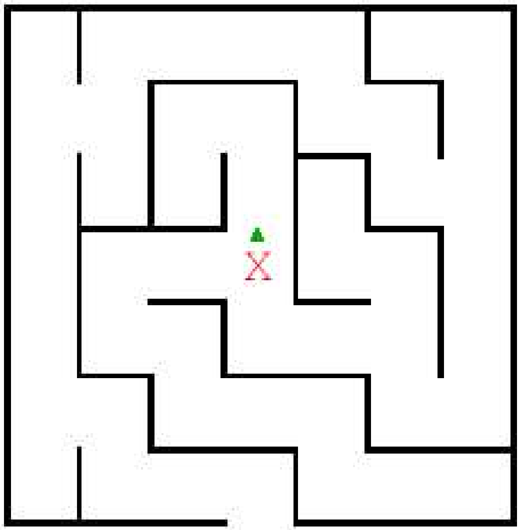
Where is this leading?



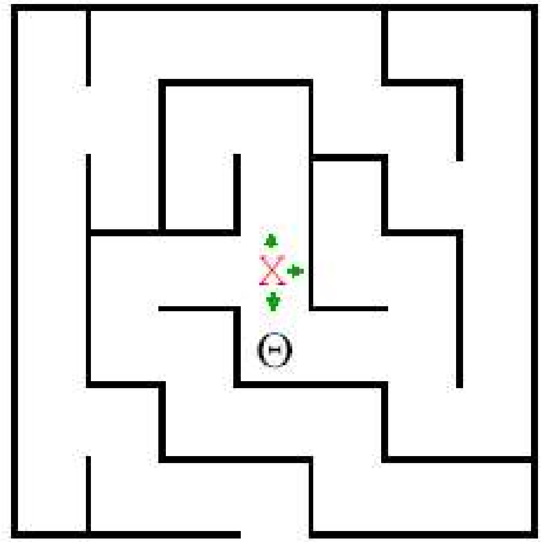
This path reaches a dead end.

Time to backtrack!

Remember the program stack!

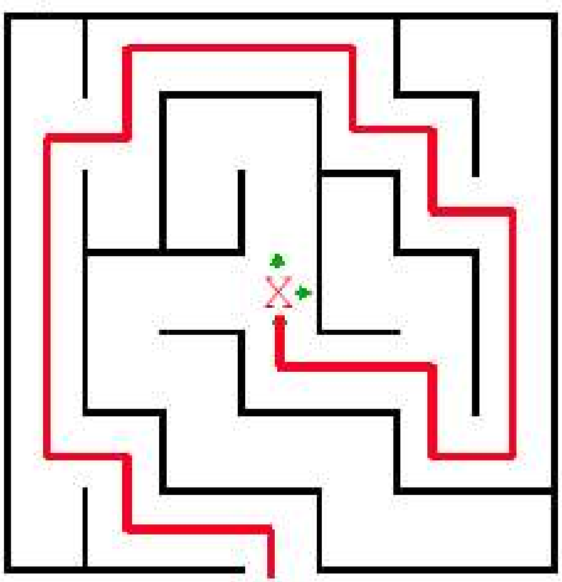


The recursive calls end and return until we find ourselves back here.



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: