

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.

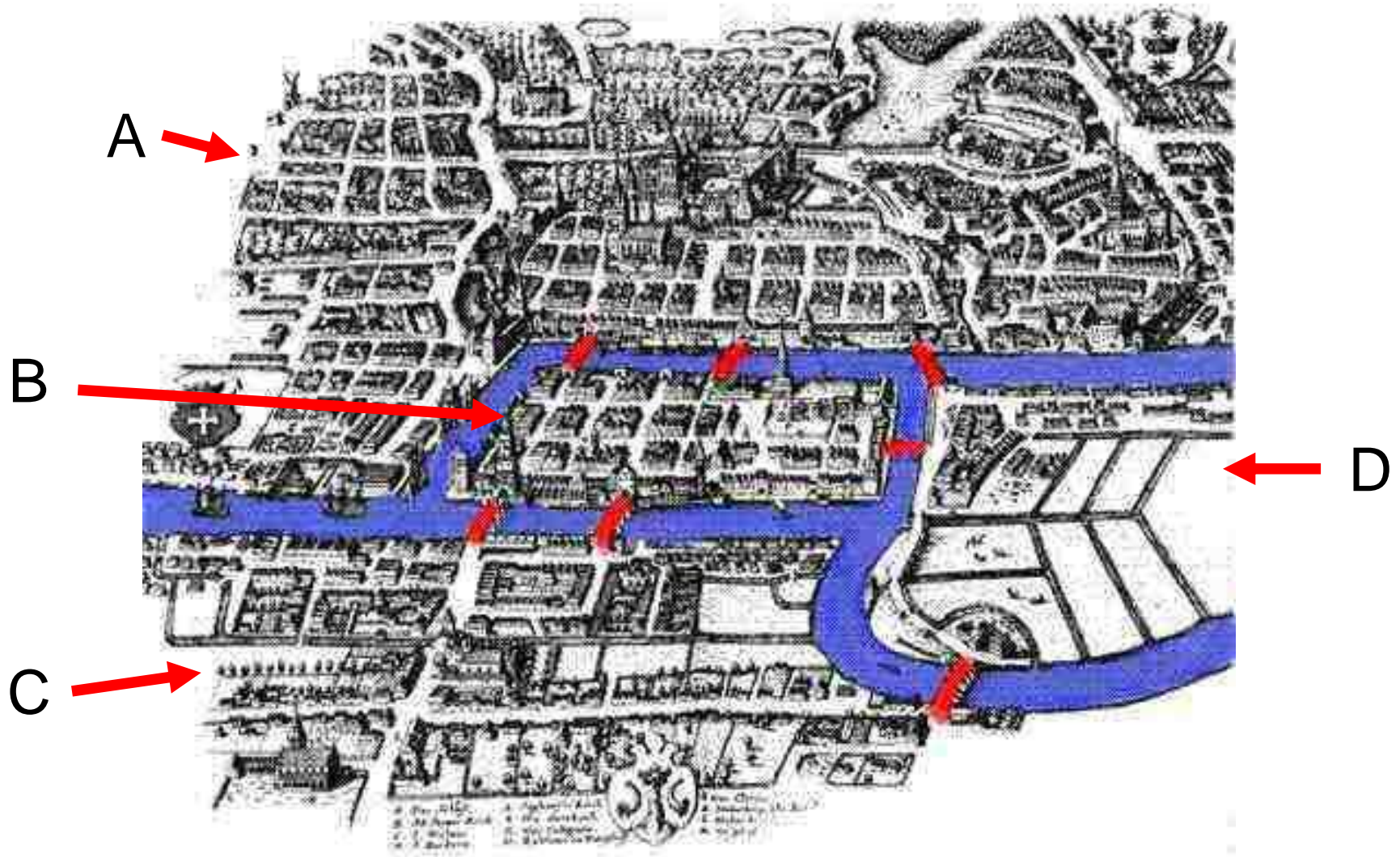
In 2013 Harvey Keitel has become the center of the Hollywood Universe. Connery is 136th.

Bacon has moved up to 370th.

An Early Problem in Graph Theory

- ▶ Leonhard Euler (1707 - 1783)
 - One of the first mathematicians to study graphs
- ▶ The Seven Bridges of Königsberg Problem
 - Königsberg 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*

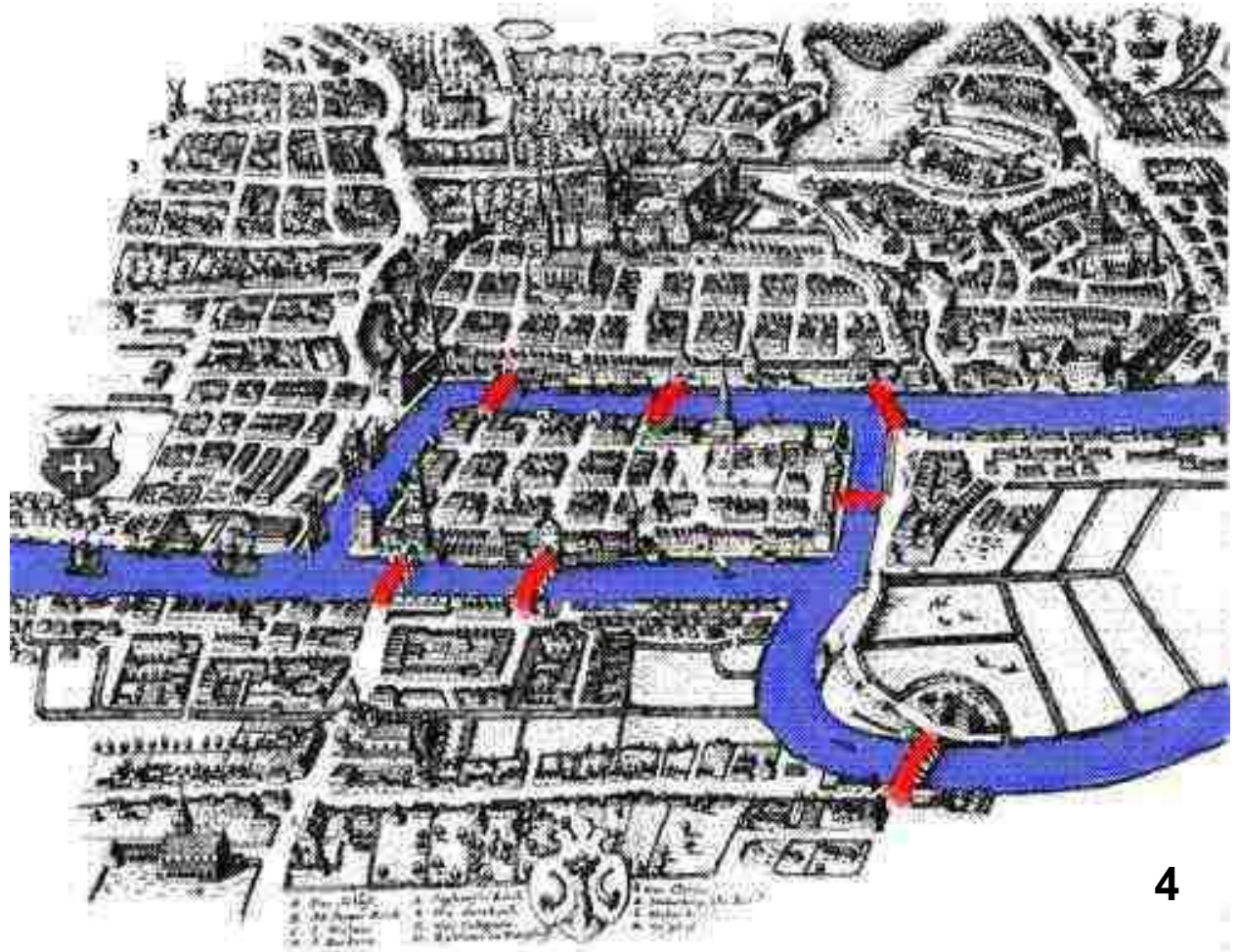
Konigsberg and the River Pregel



Clicker 1

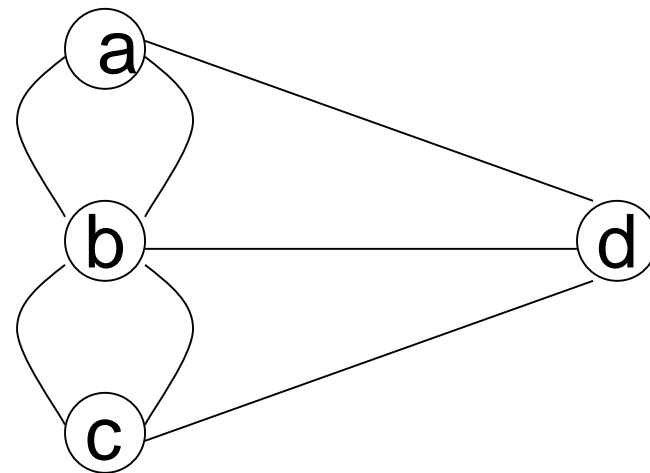
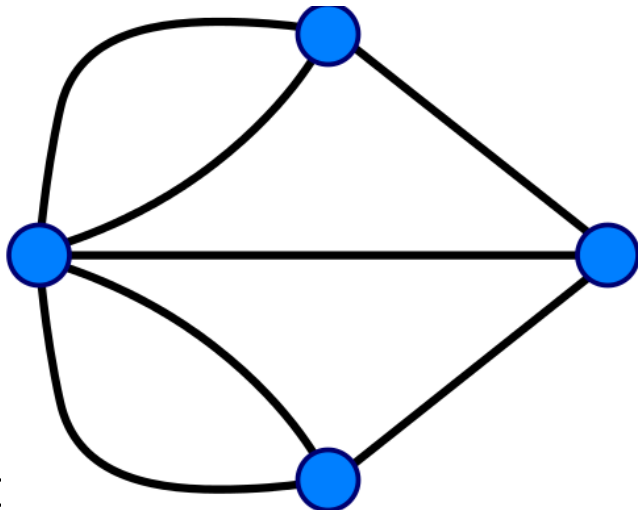
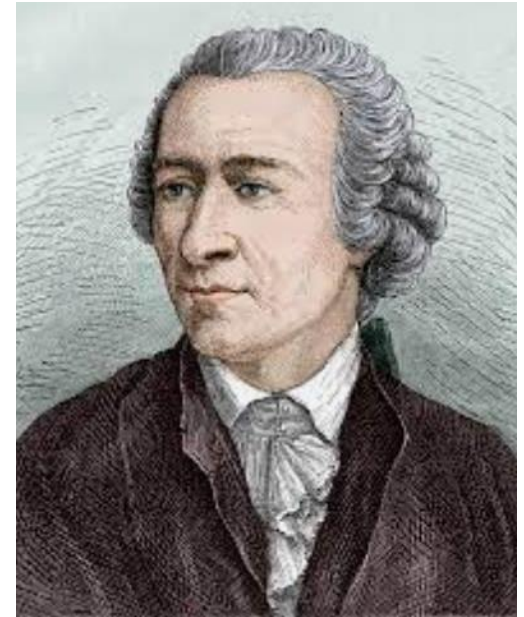
► How many solutions does the Seven Bridges of Königsberg Problem have?

- 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

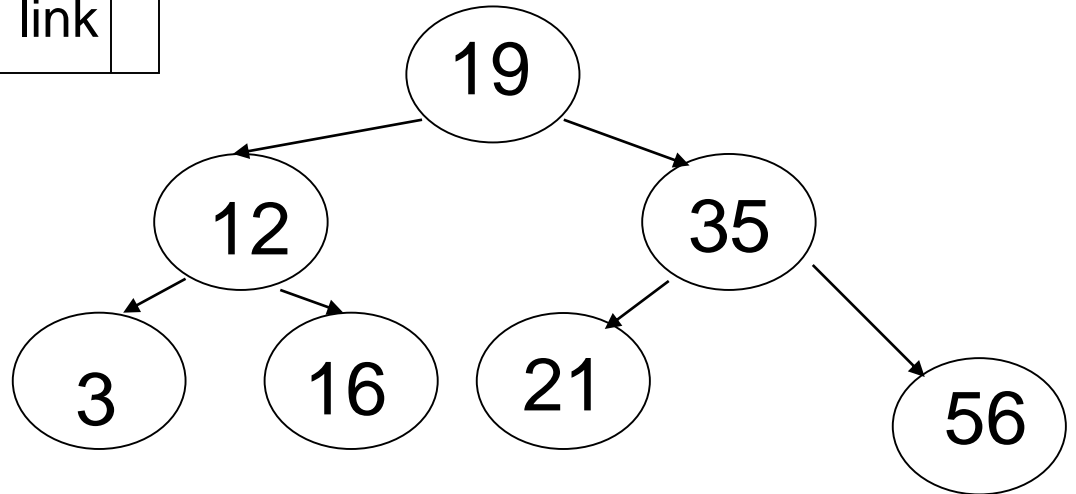
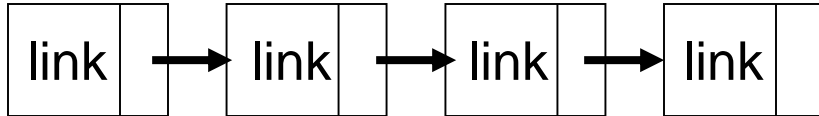
Graph Definitions

- ▶ 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

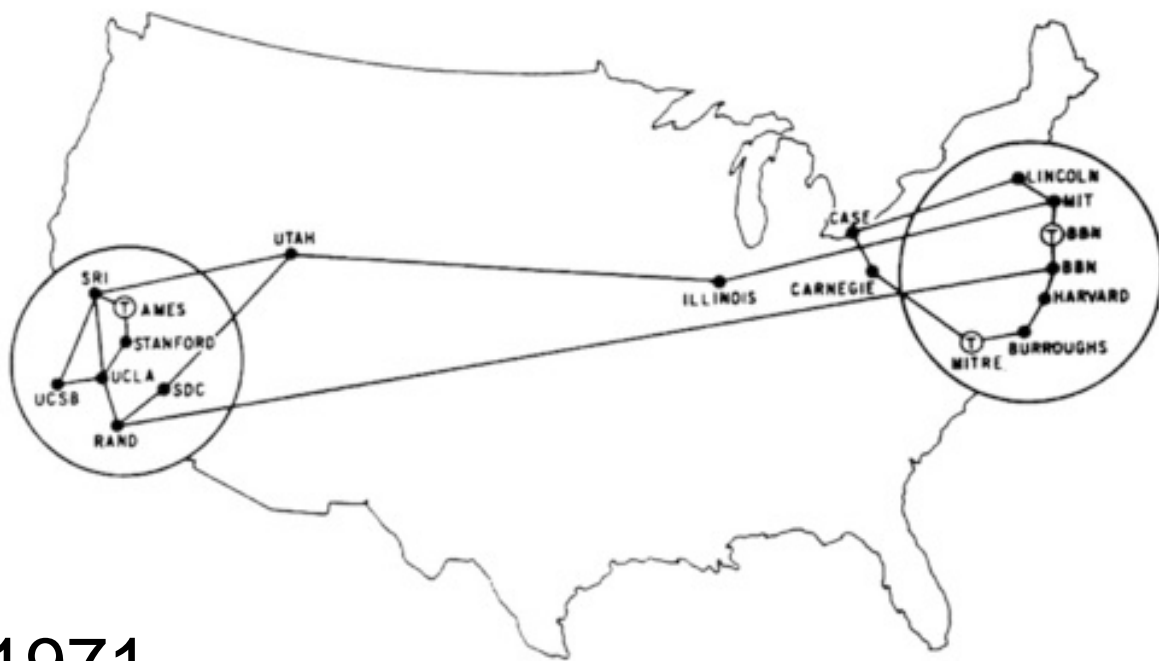
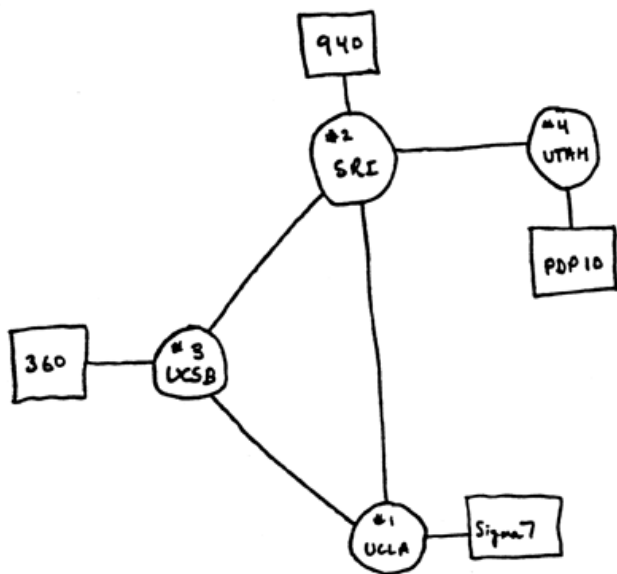
- ▶ In a *weighted* graph the edge has cost or weight that measures the cost of traveling along the edge
- ▶ A *path* is a sequence of vertices connected by edges
 - The *path length* is the number of edges
 - The *weighted path length* is the sum of the cost of the edges in a path
- ▶ A *cycle* is a 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

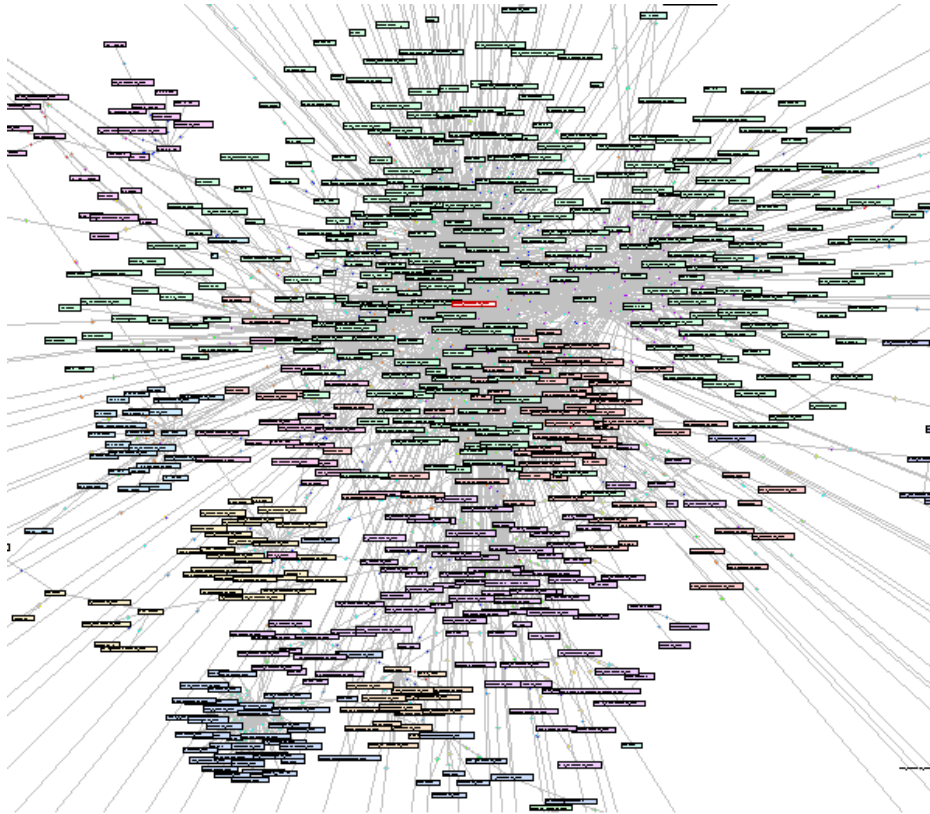
Roman
Transportation
Network



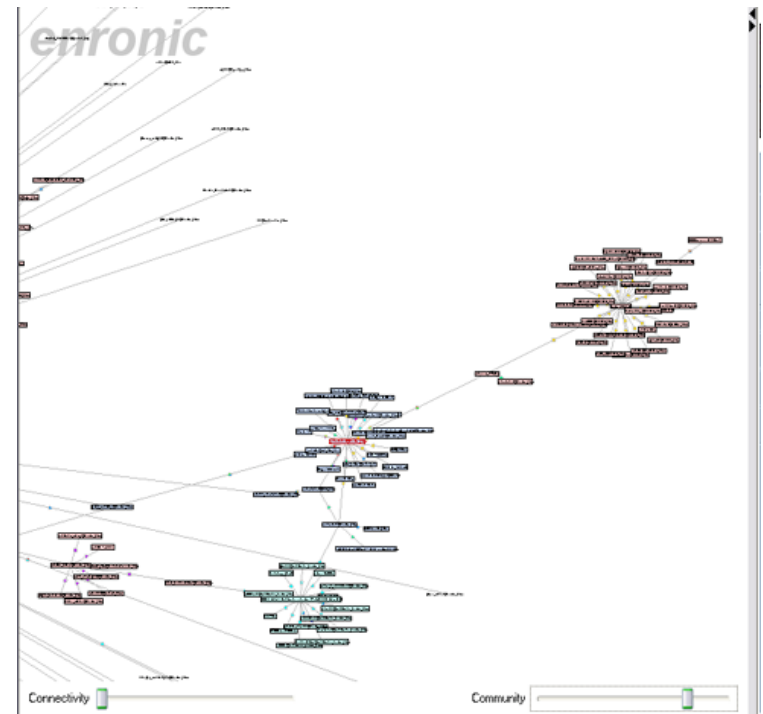
Roman Transportation Network



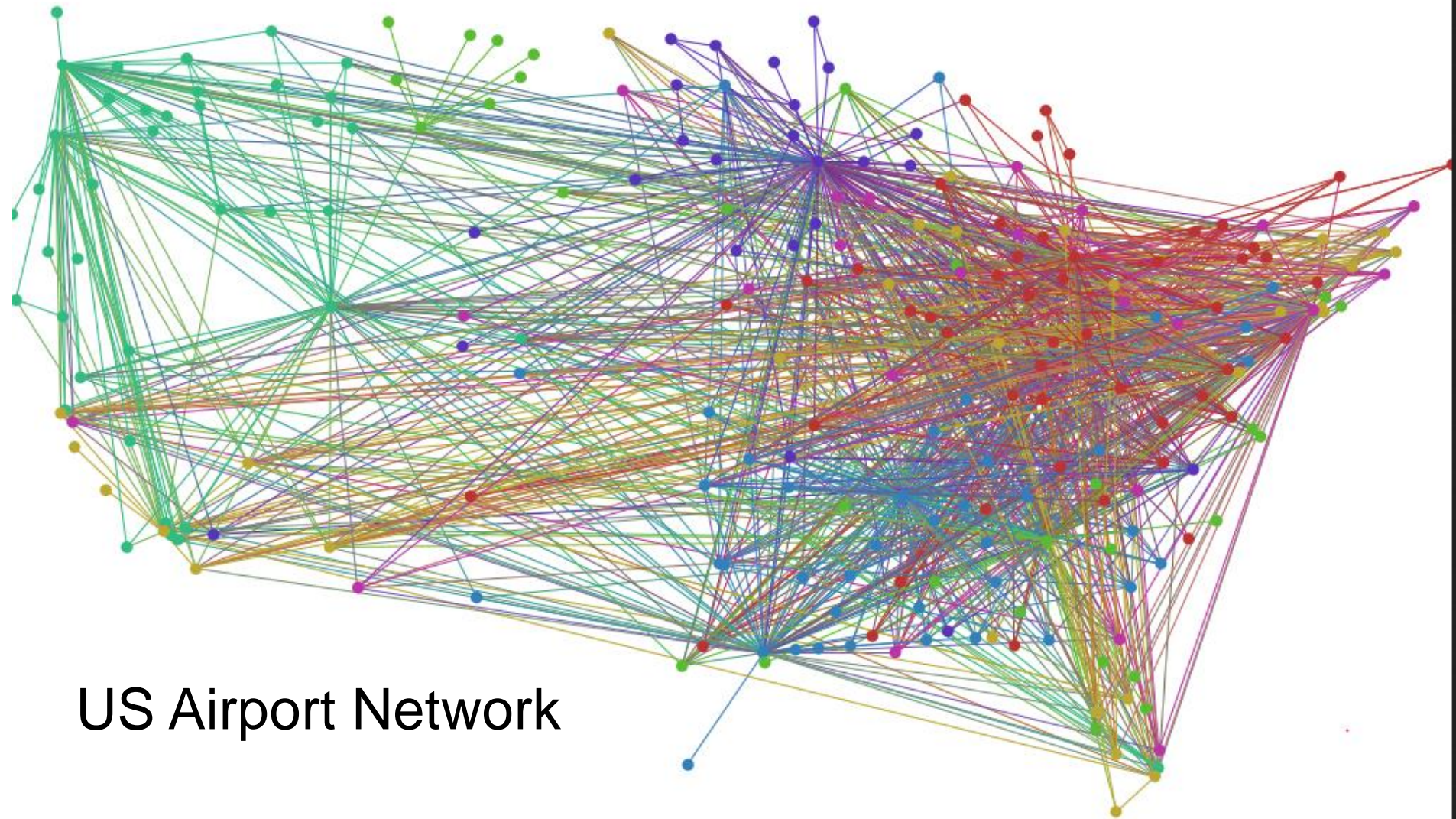
Example Graph



Enron emails 2001



Example Graph



US Airport Network

Example Graph

facebook DEVELOPERS

DocumentationSupportBlogApps

Search Documentation / Apps

Getting Started

Core Concepts >

Social Design

Social Plugins

Open Graph protocol

Social Channels

Authentication

Graph API

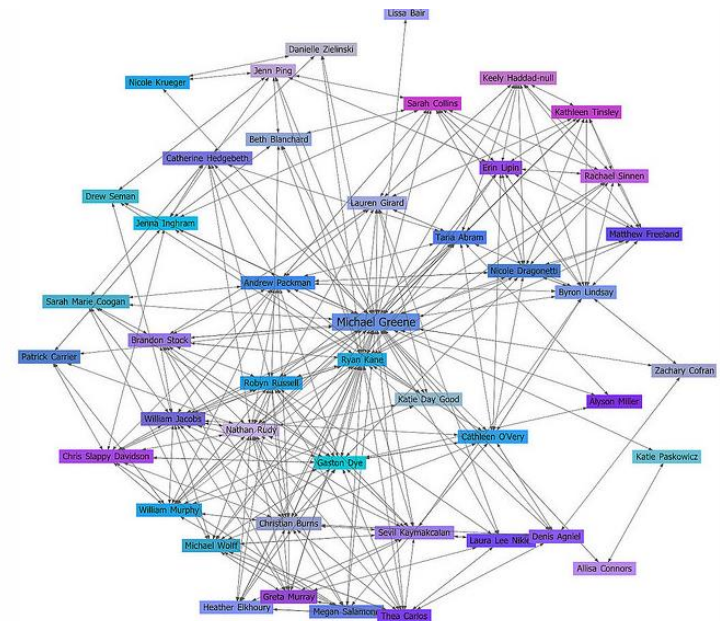
Advanced Topics

Graph API

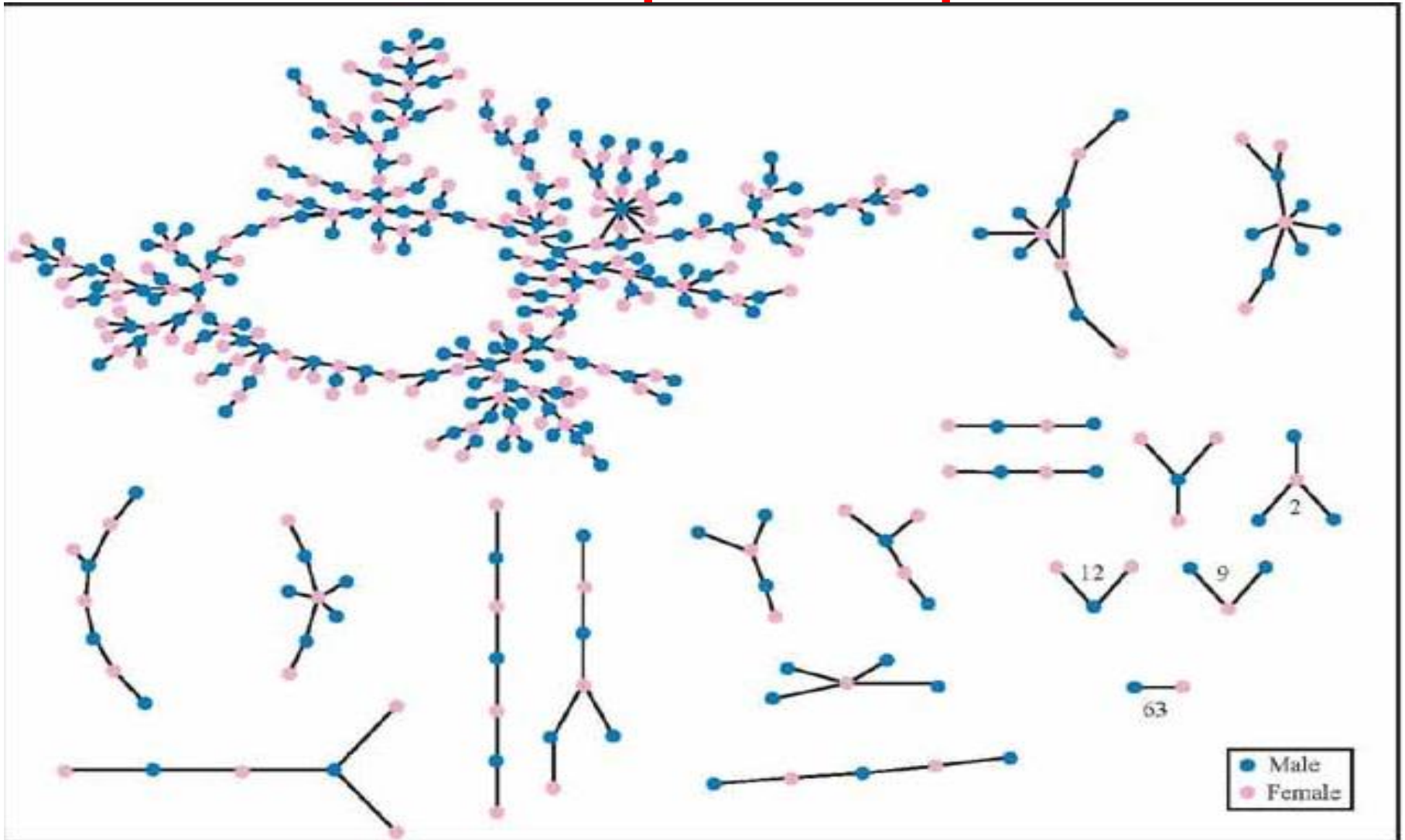
Core Concepts > Graph API

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
and Sexual Networks, 2005,

Chains of Affection: The Structure of Adolescent Romantic

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

- ▶ Use a ragged 2d array to save space

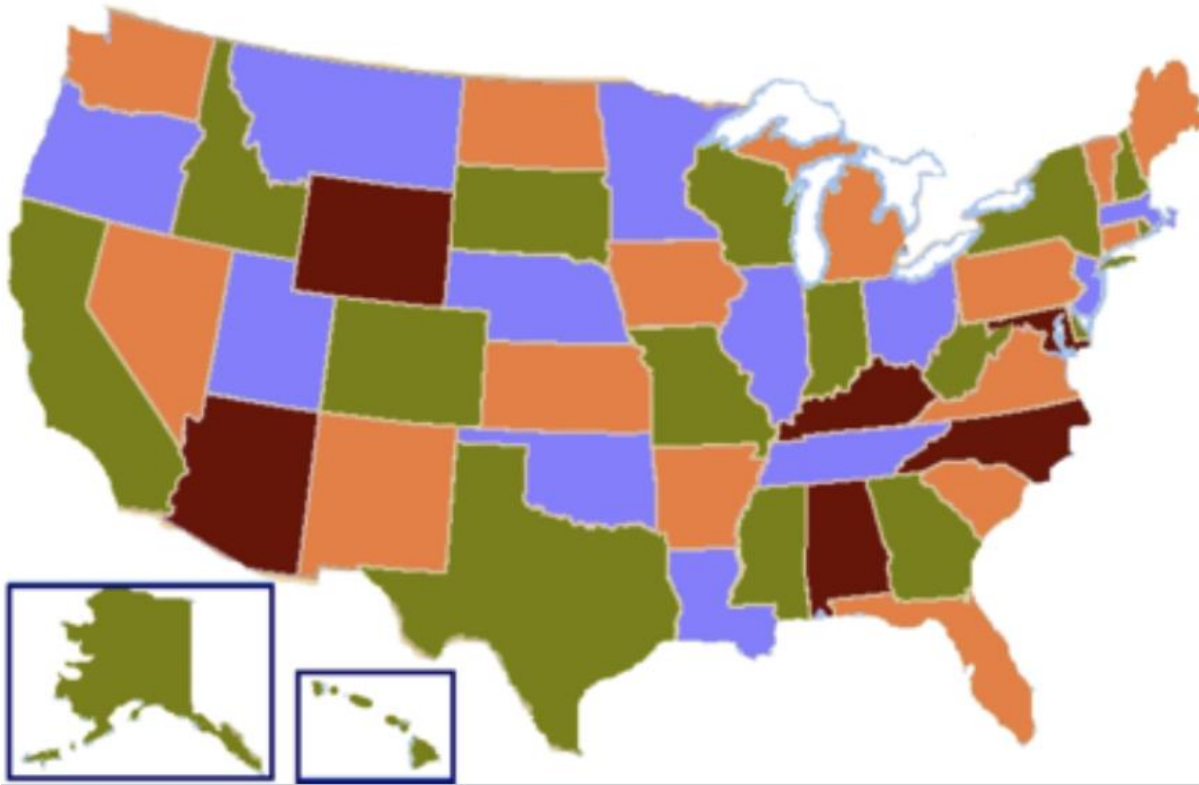
NORTH ISLAND DISTANCE CHART

[illegible]

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).

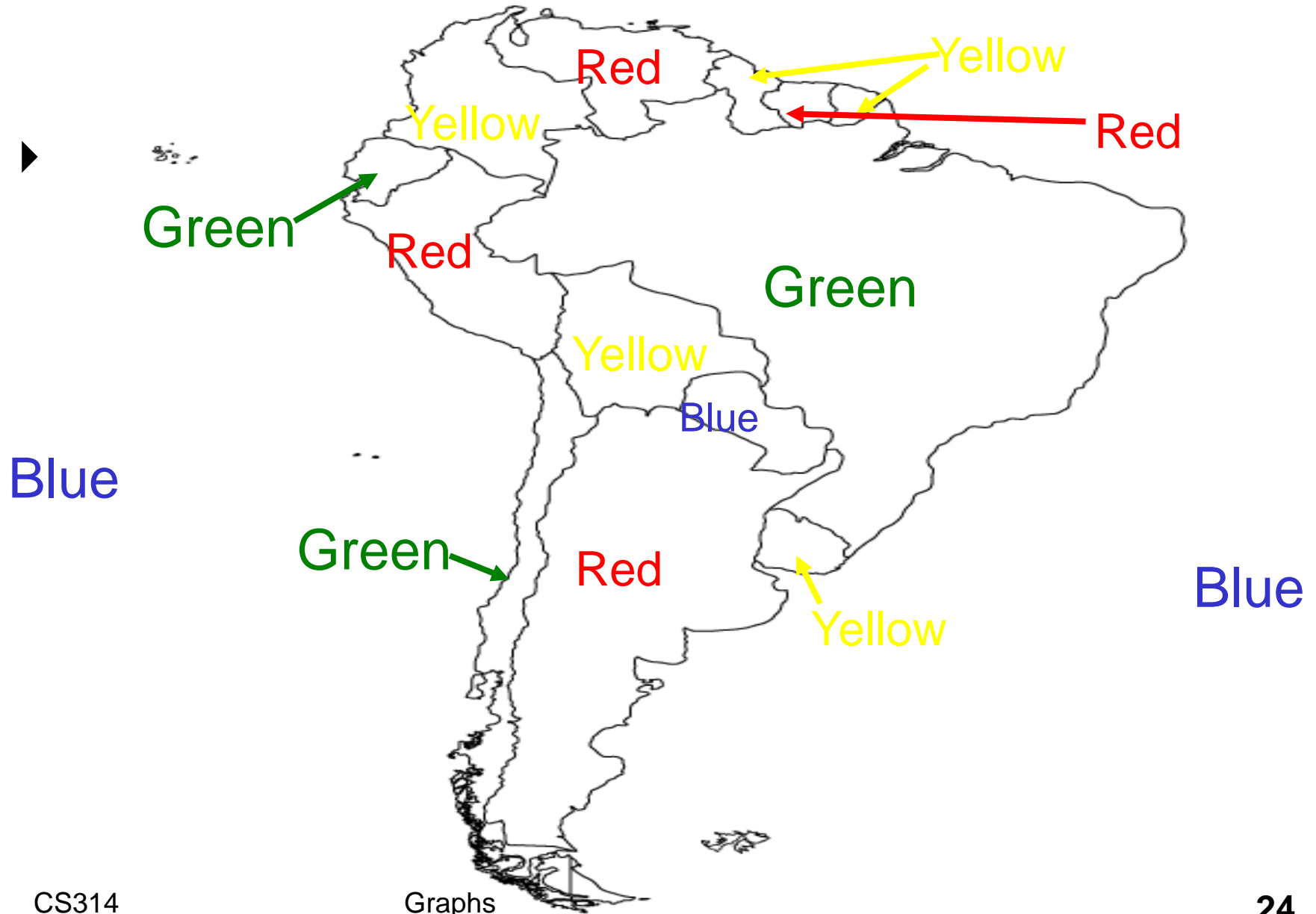
A Solution



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()
```

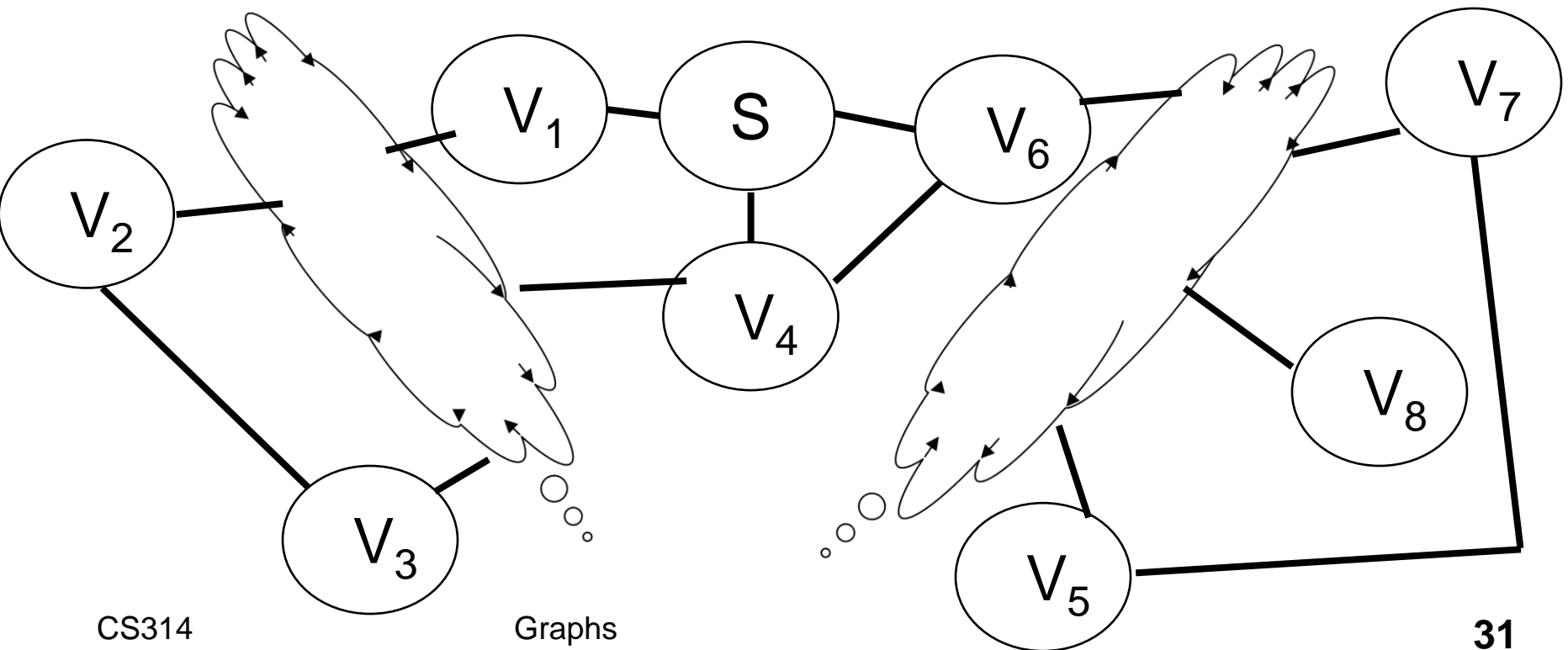
Edge Class (nested in Graph)

```
private static class Edge
    private Vertex dest;
    private double cost;

    private Edge(Vertex d, double c)
```

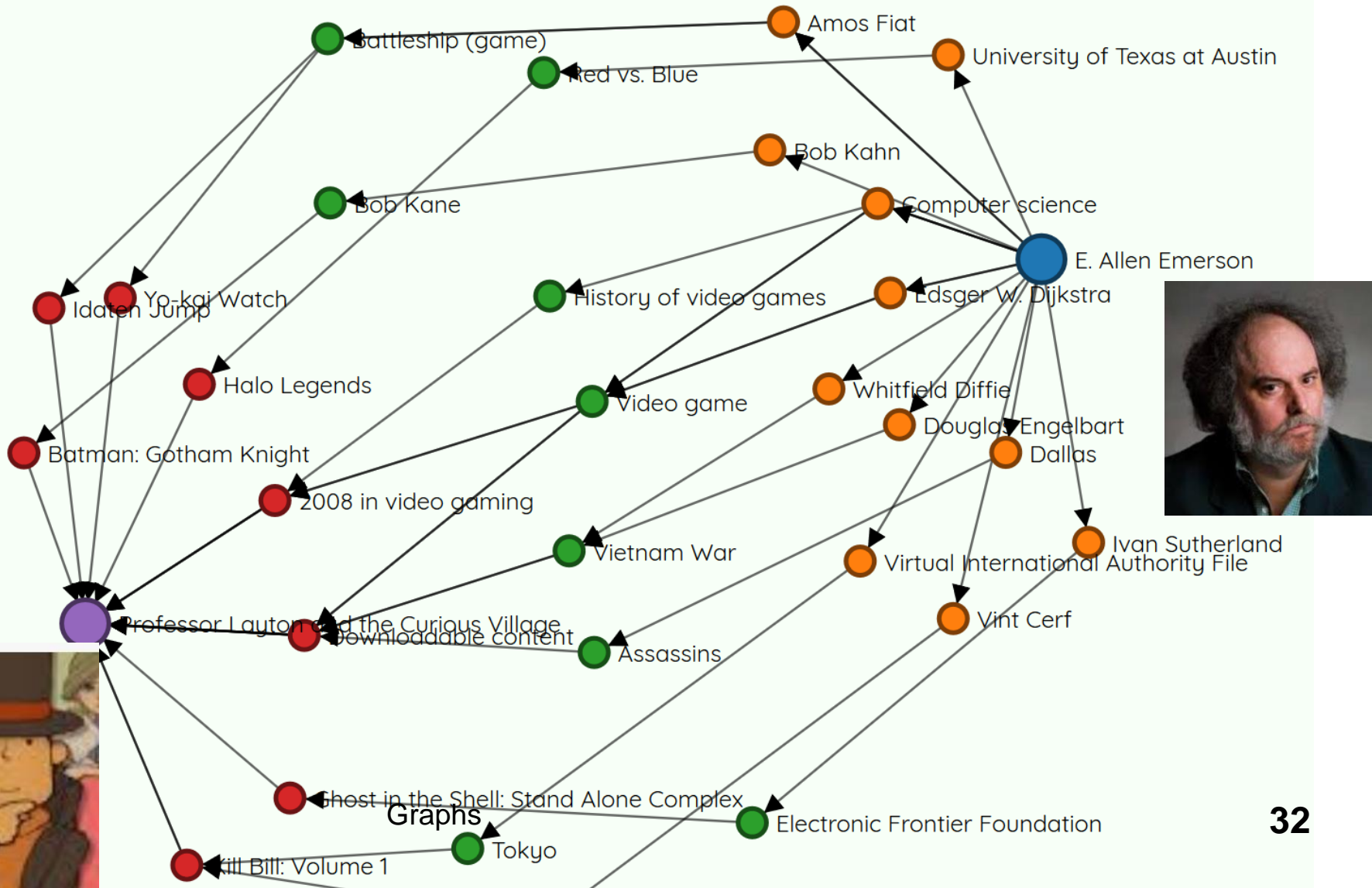
Unweighted Shortest Path

- ▶ 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

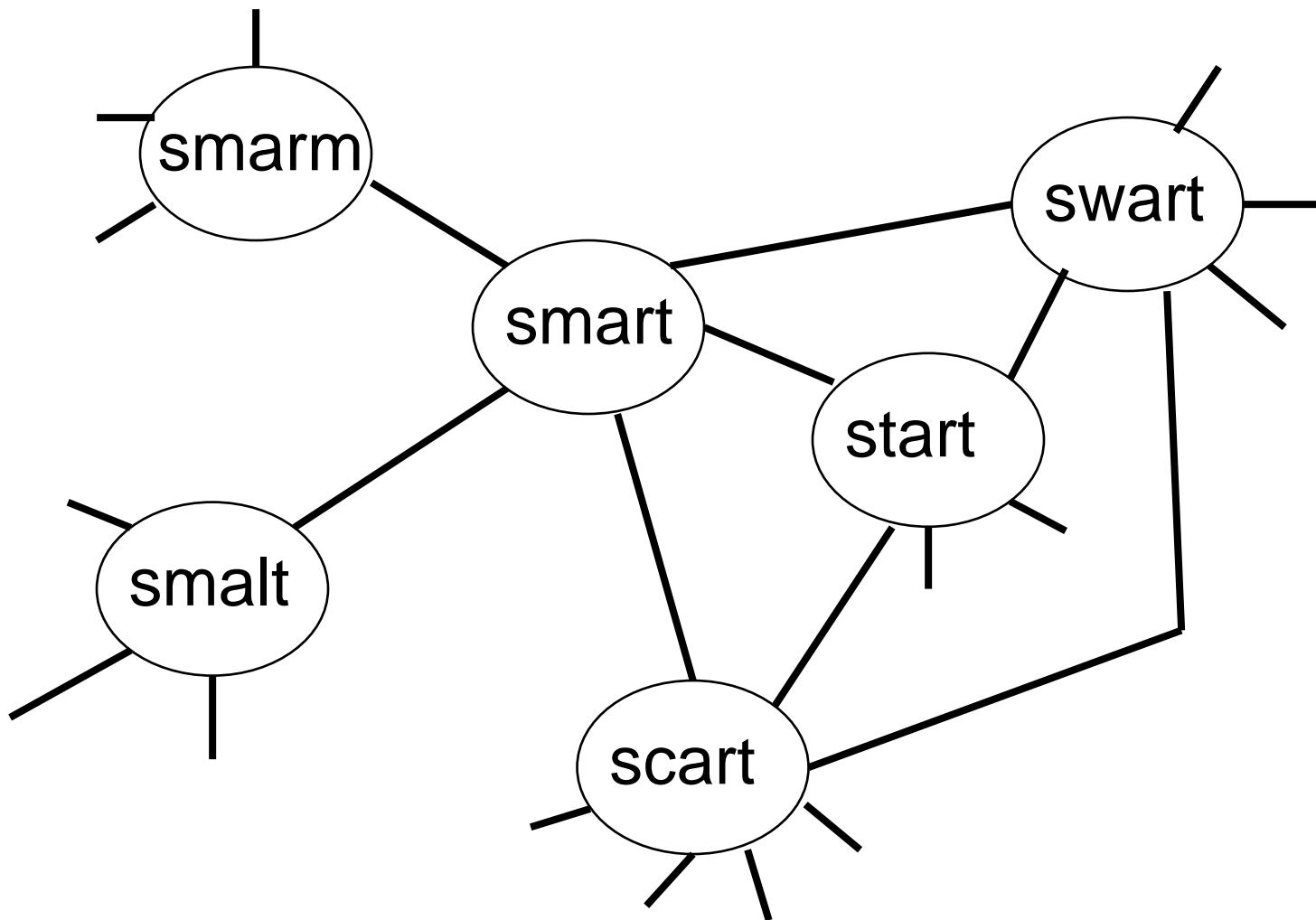
Words that share one or more letters

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

$$\begin{aligned}\text{Max number of edges} &= \\ &N * (N - 1) / 2 \\ &37,493,470\end{aligned}$$

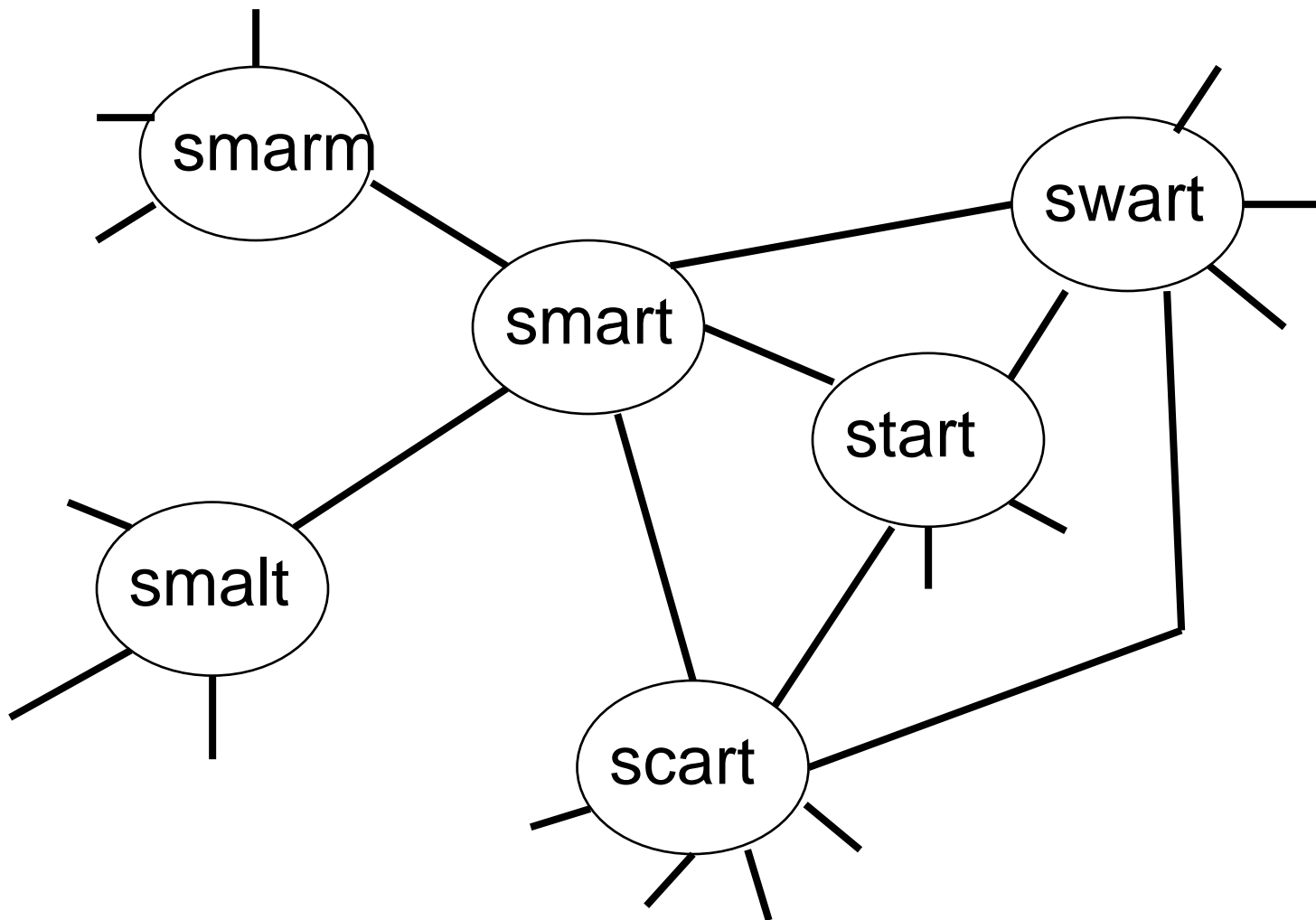
Clicker 5 - Unweighted Shortest Path Algorithm

- ▶ Problem: Find the **shortest word ladder** between two words if one exists
- ▶ What kind of search should we use?

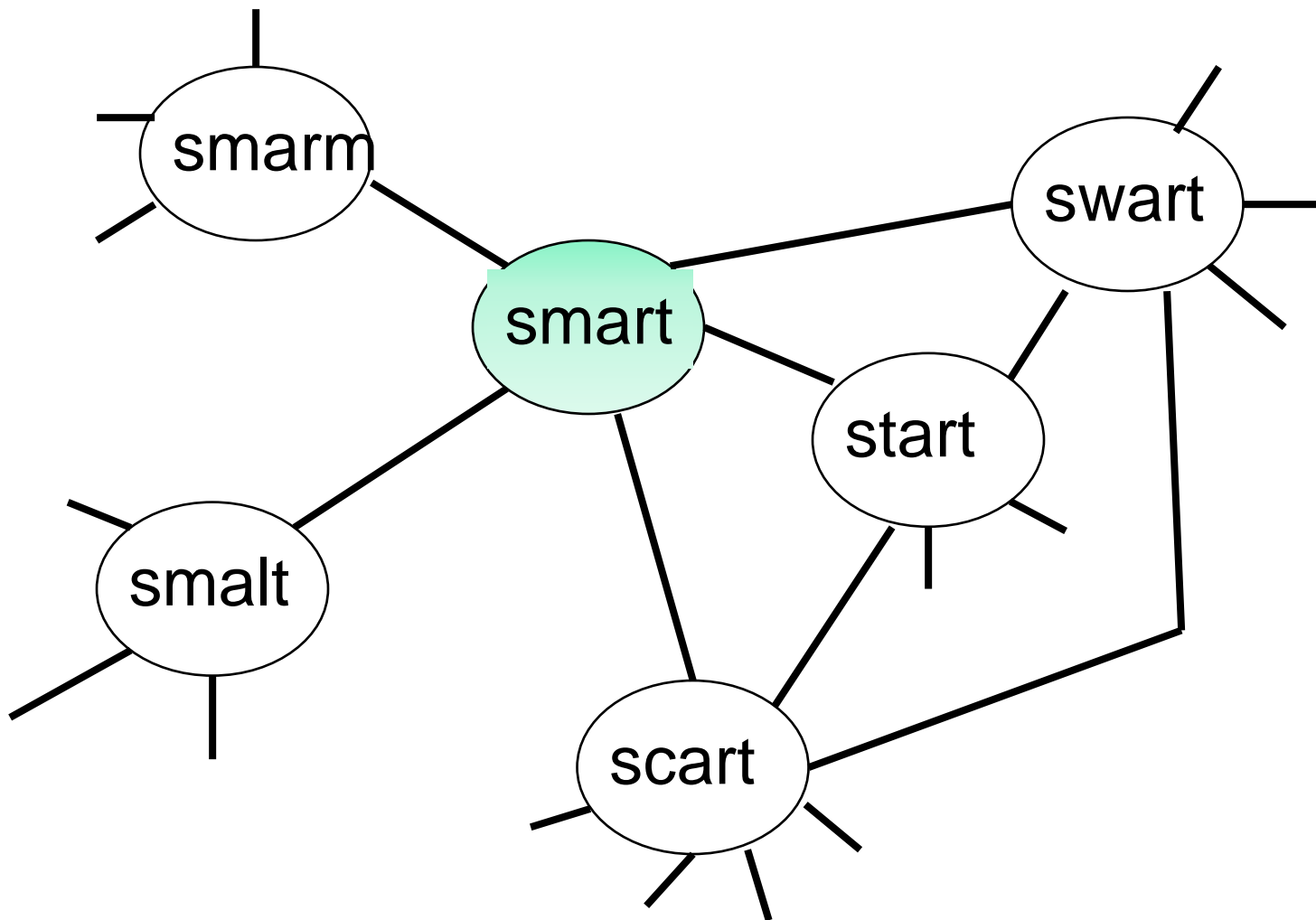
- A. Breadth First Search
- B. Depth First Search
- C. Either one

Unweighted Shortest Path Algorithm

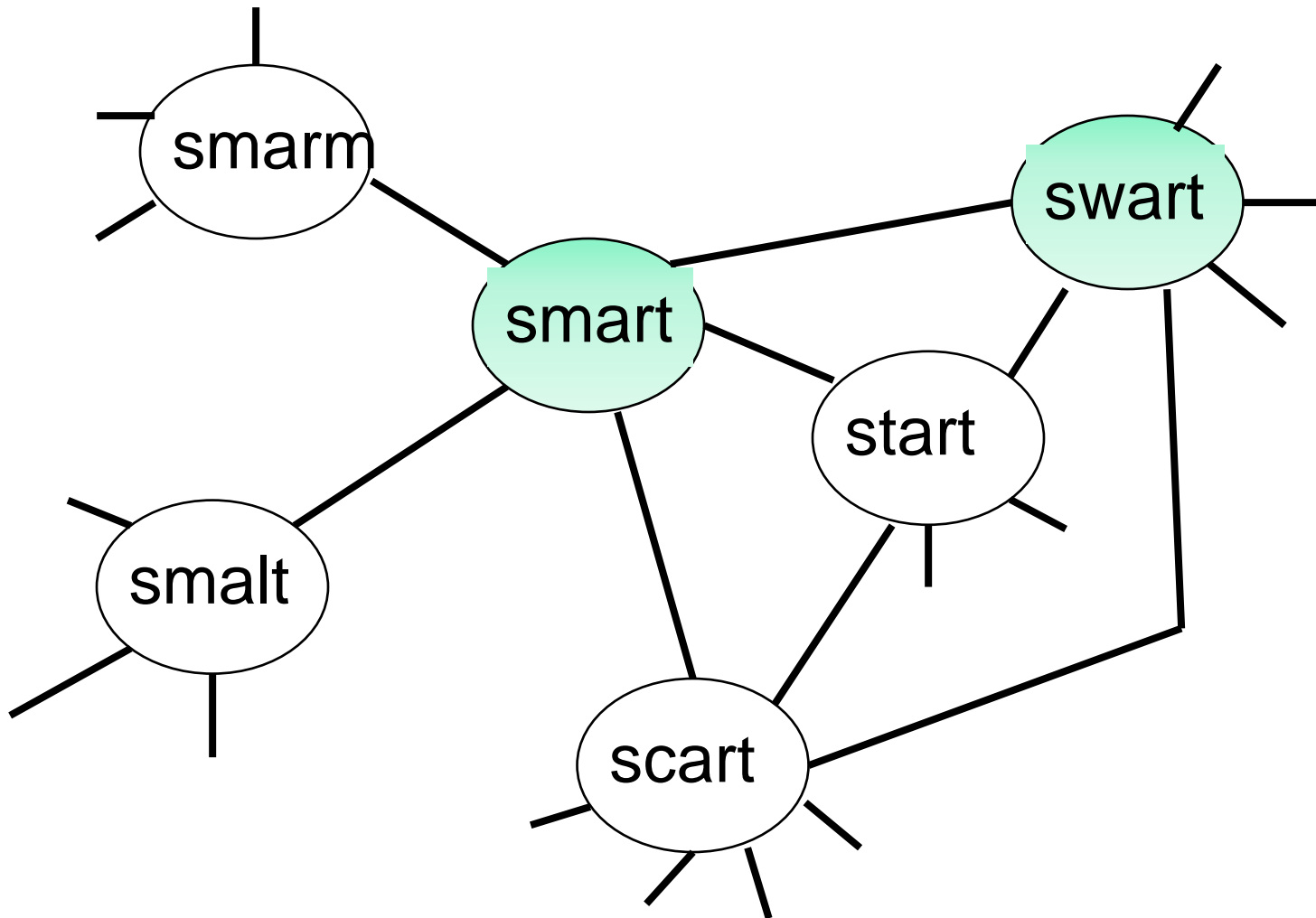
- ▶ Set distance of start to itself to 0
- ▶ Create a queue and add the start vertex
- ▶ while the queue is not empty
 - remove front
 - loop through all edges of current vertex
 - get vertex edge connects to
 - if this vertex has not been visited (have not found path to the destination of the edge)
 - sets its distance to current distance + 1
 - sets its previous vertex to current vertex
 - add new vertex to queue



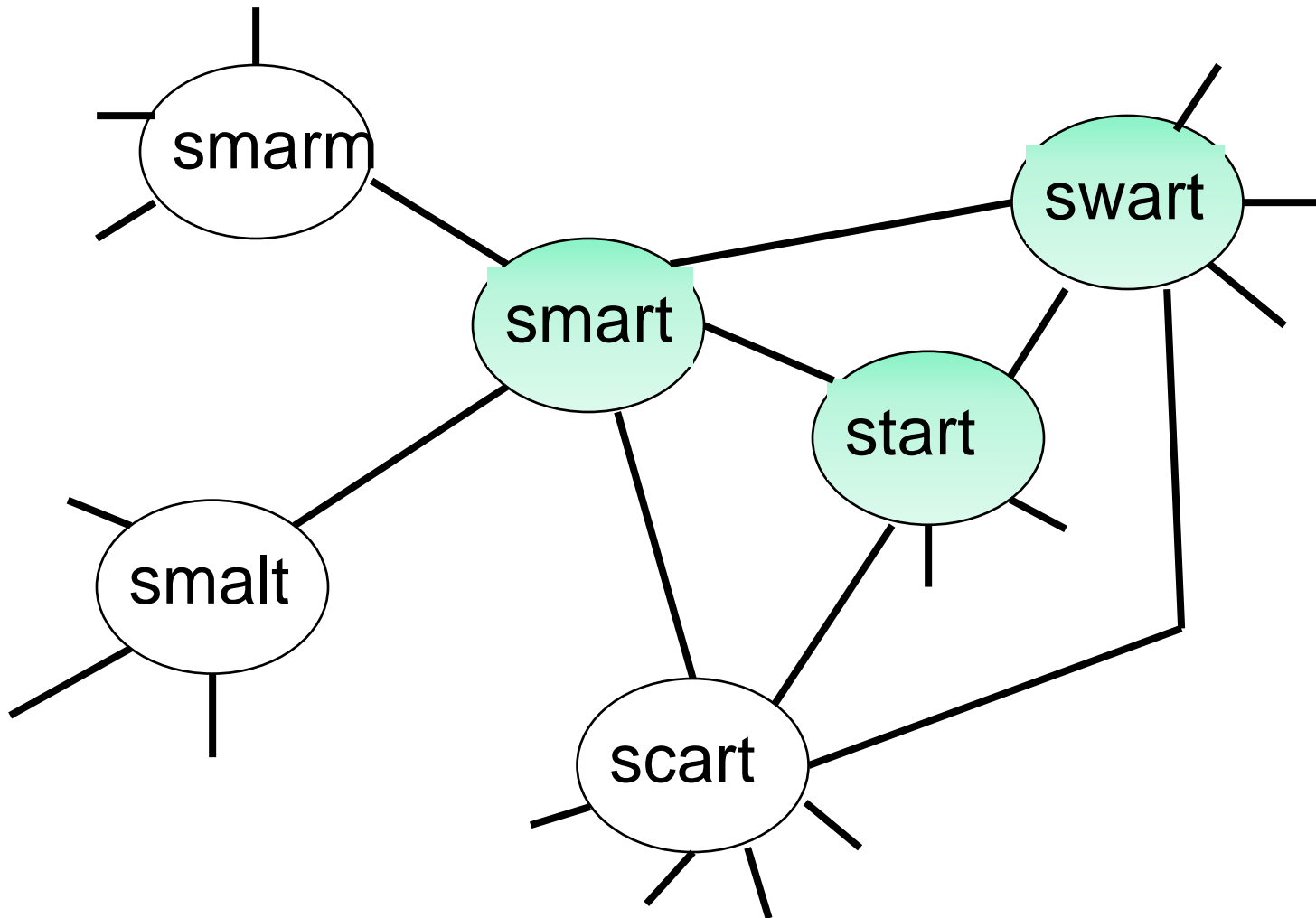
Portion of Graph



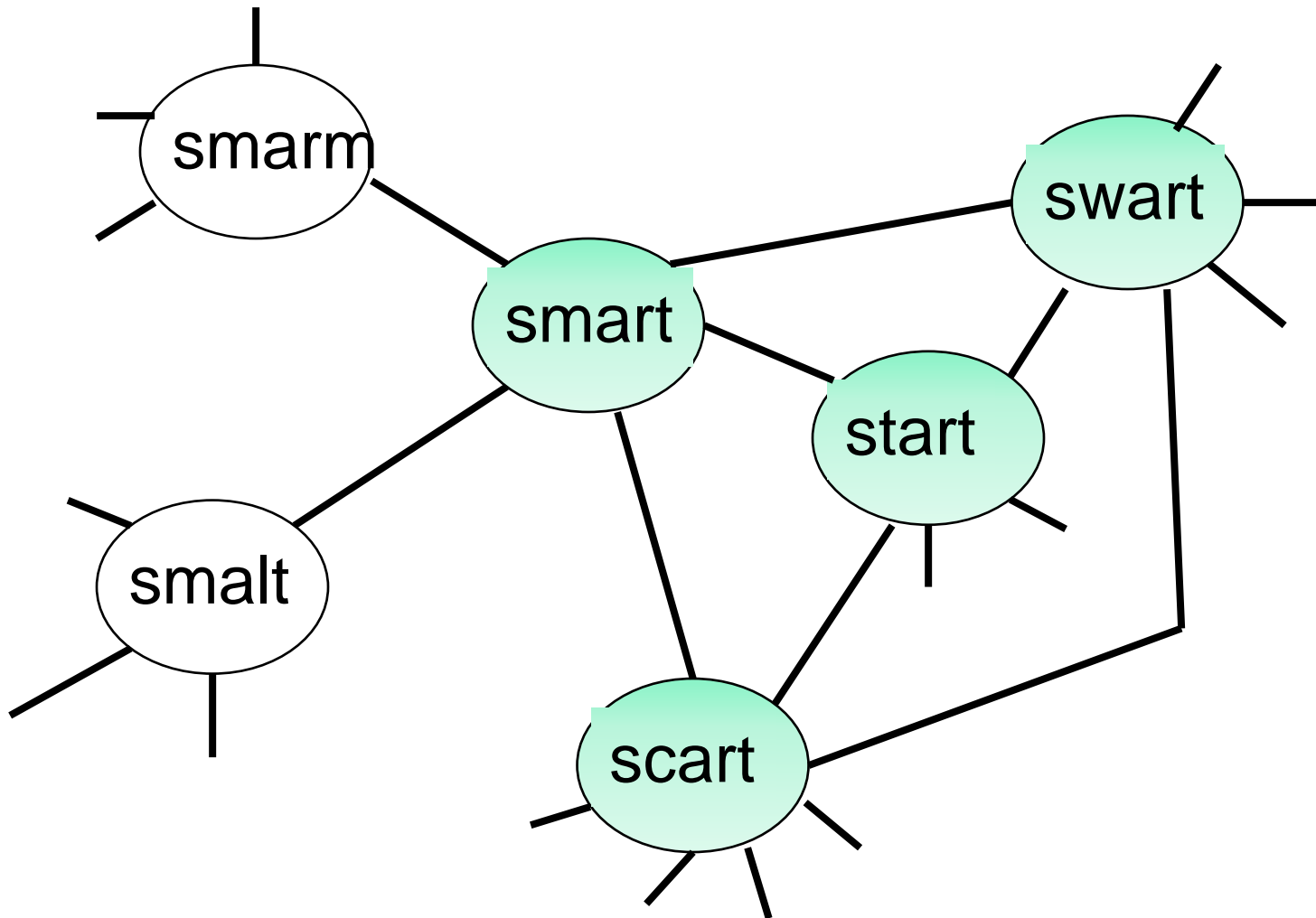
Start at "smart" and enqueue it
[smart]



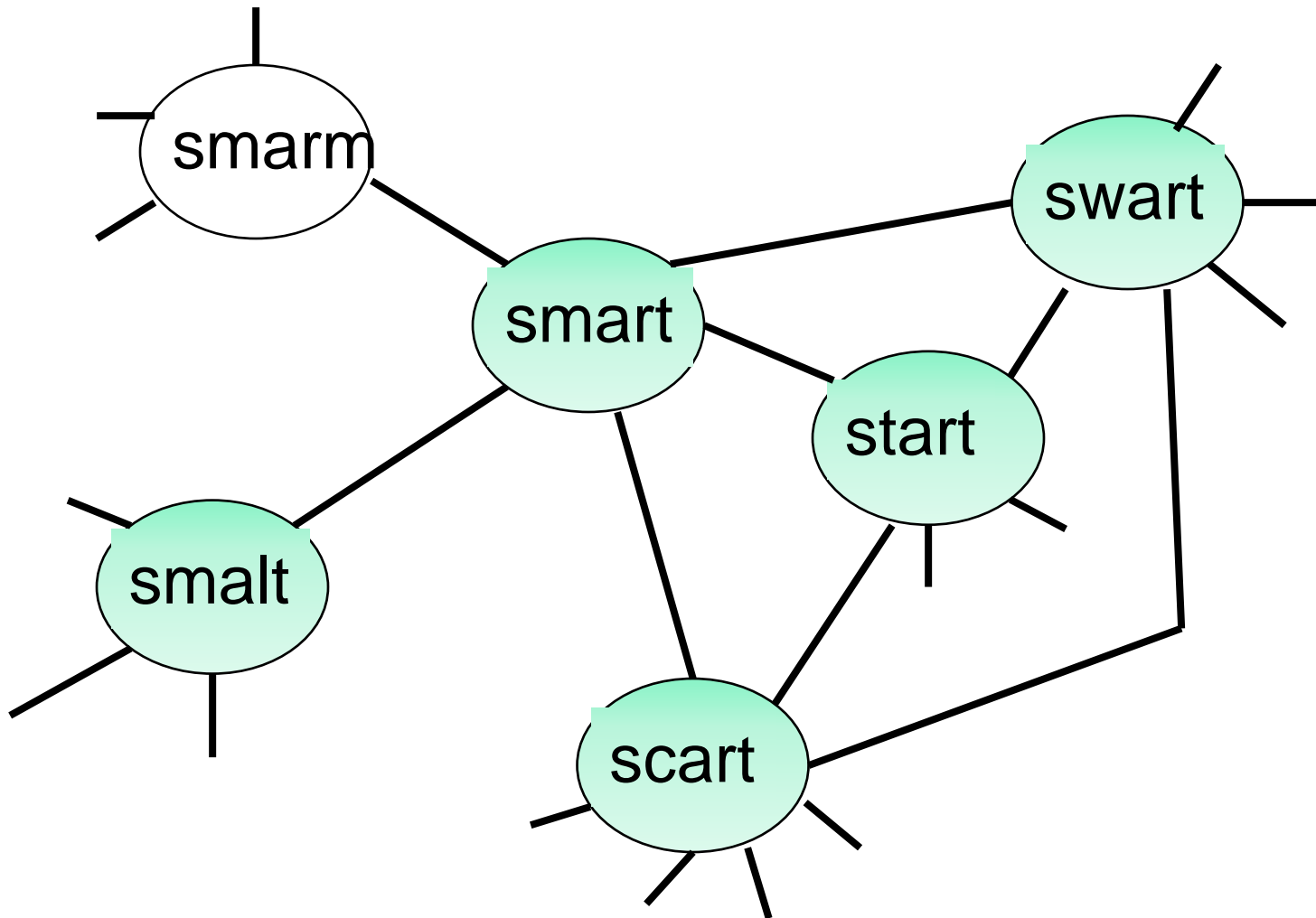
Dequeue (smart), loop through edges
[swart]



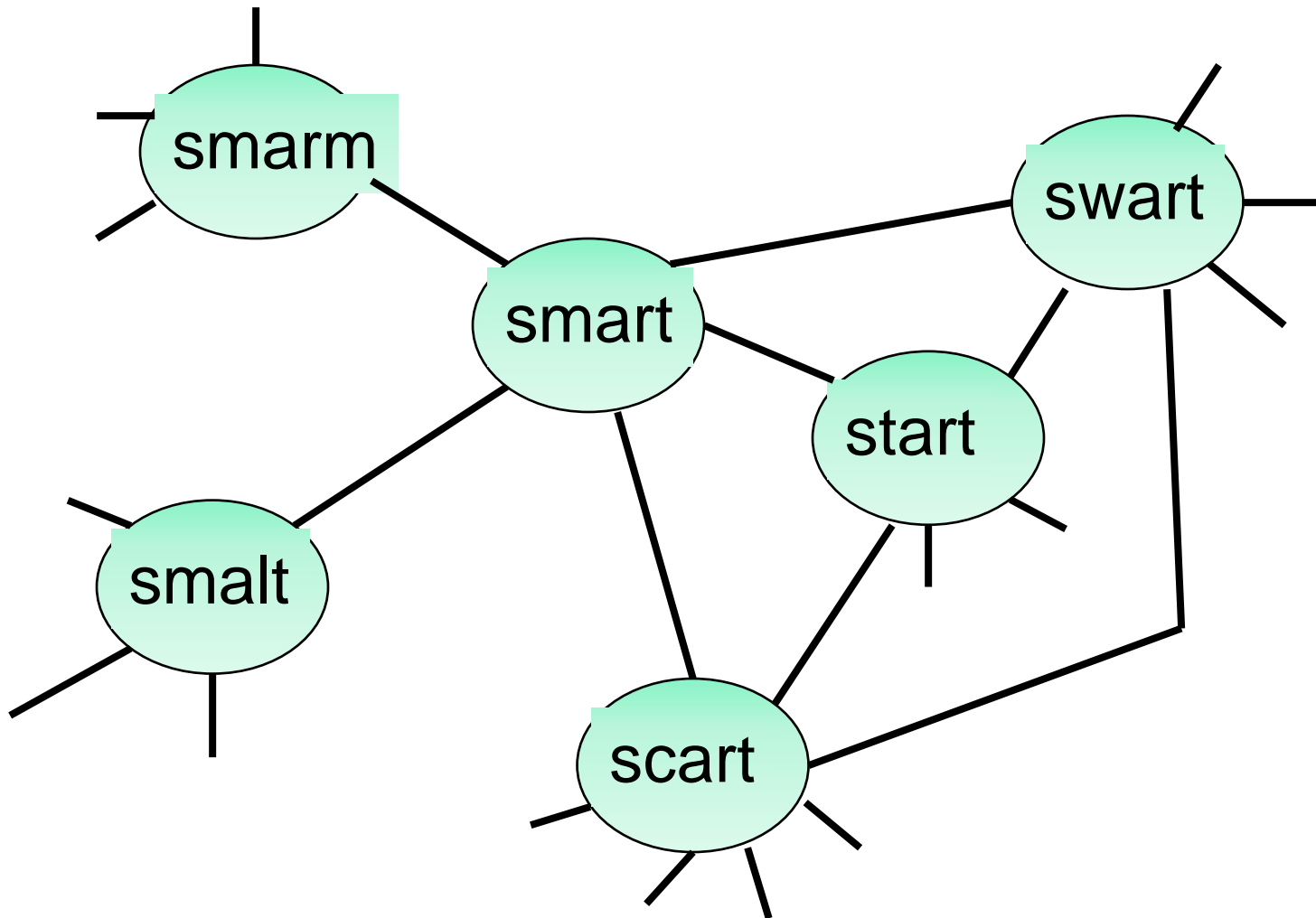
Dequeue (smart), loop through edges
[swart, start]



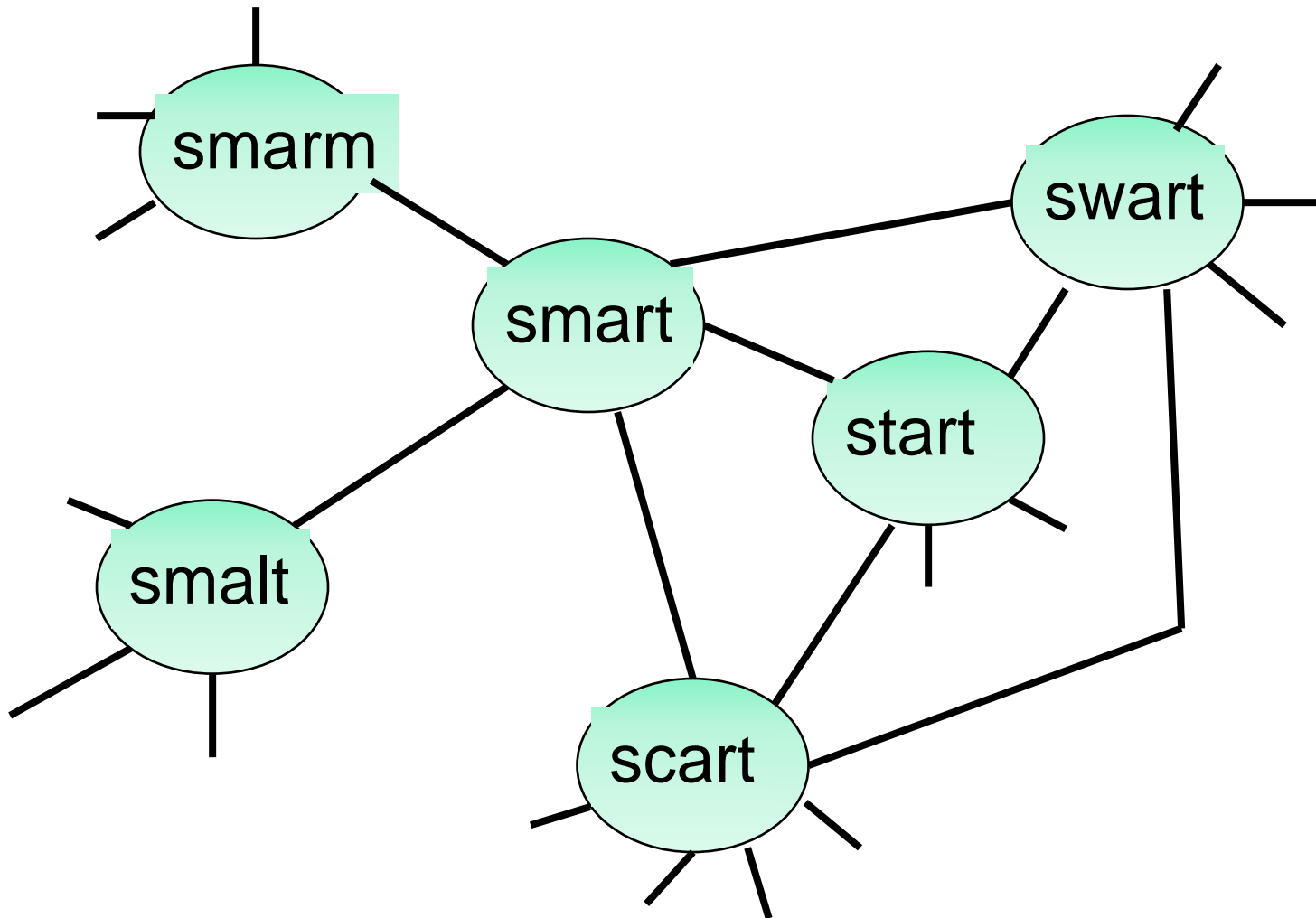
Dequeue (smart), loop through edges
[swart, start, scart]



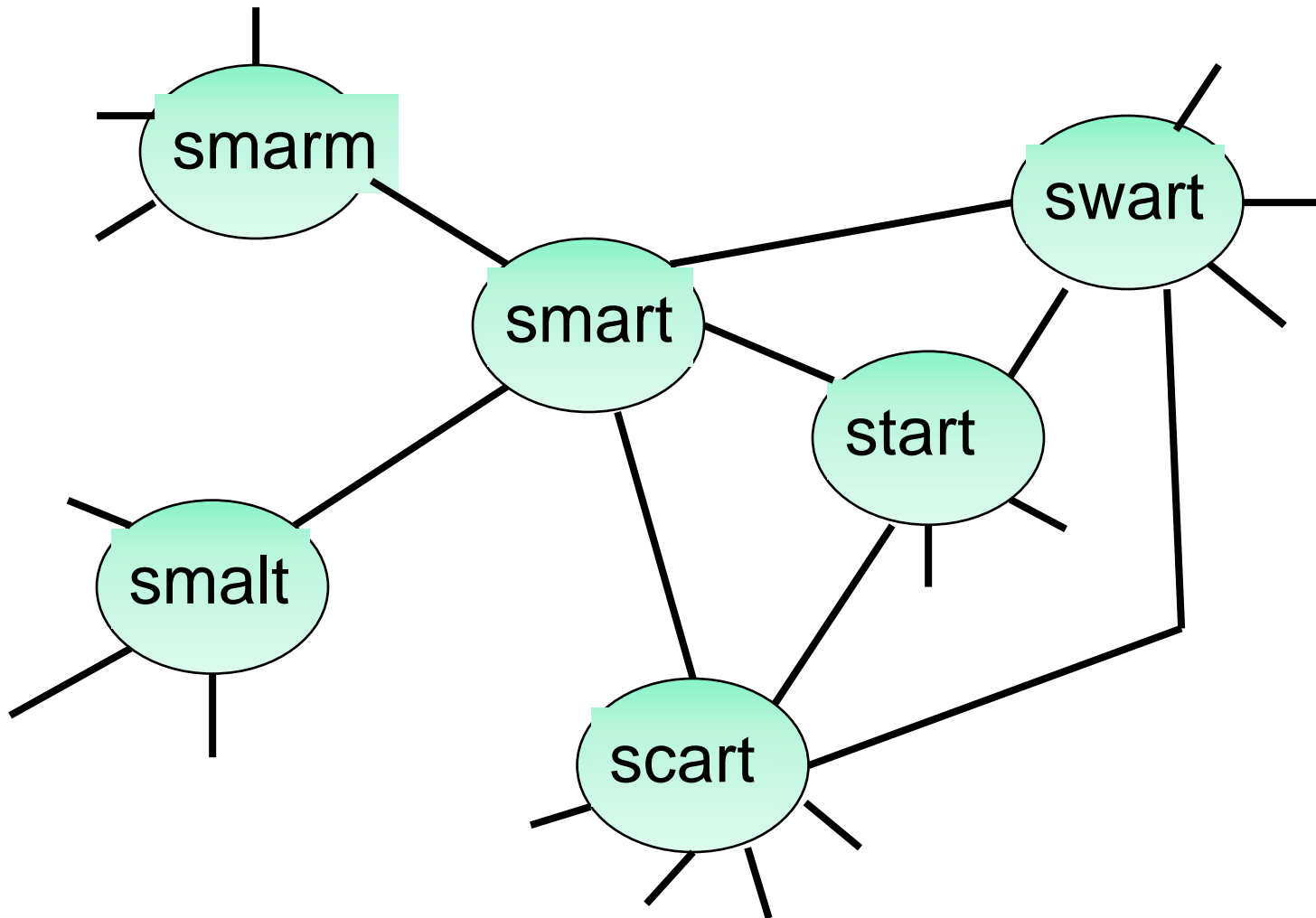
Dequeue (smart), loop through edges
[swart, start, scart, smalt]



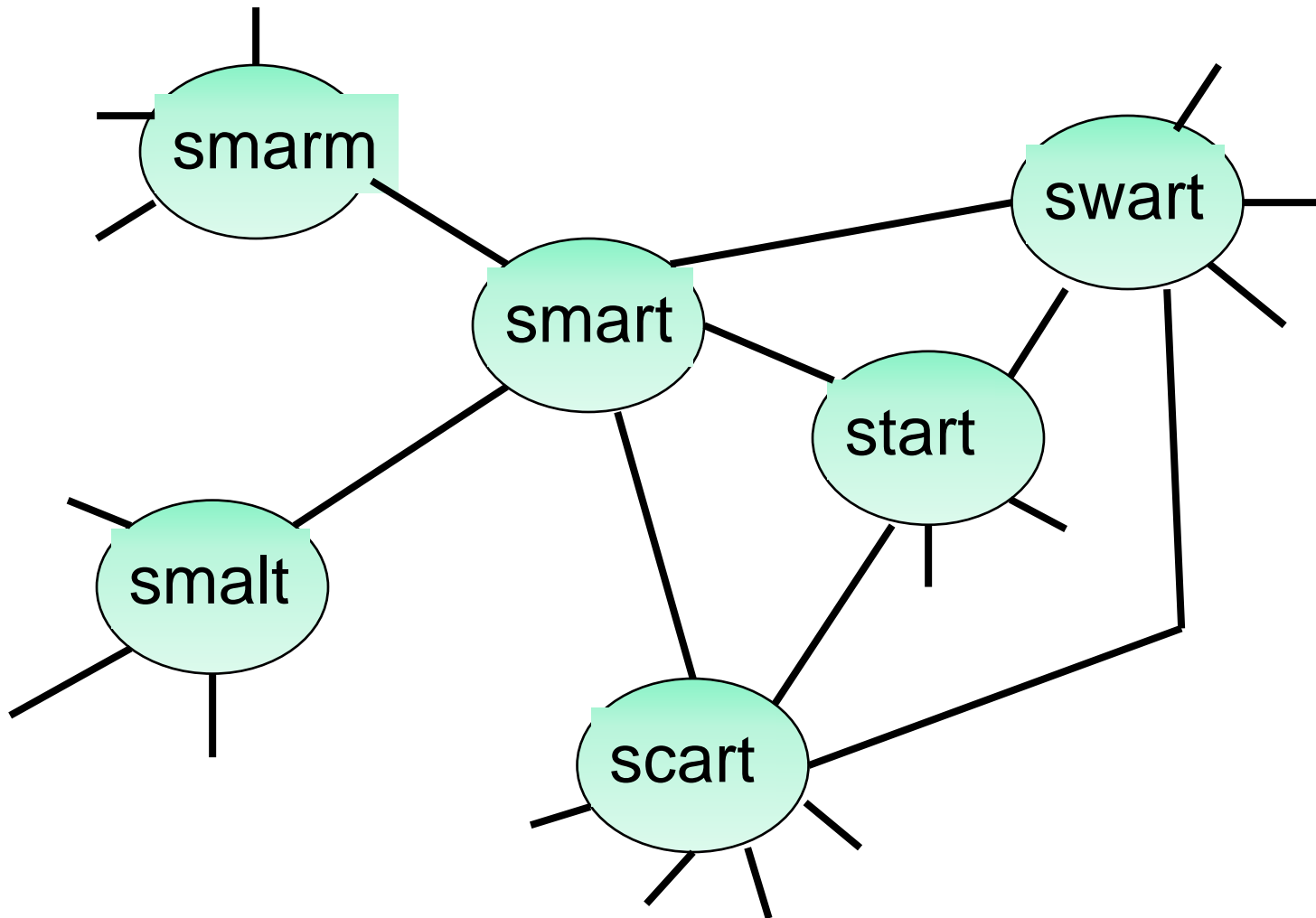
Dequeue (smart), loop through edges
[swart, start, scart, smalt, smarm]



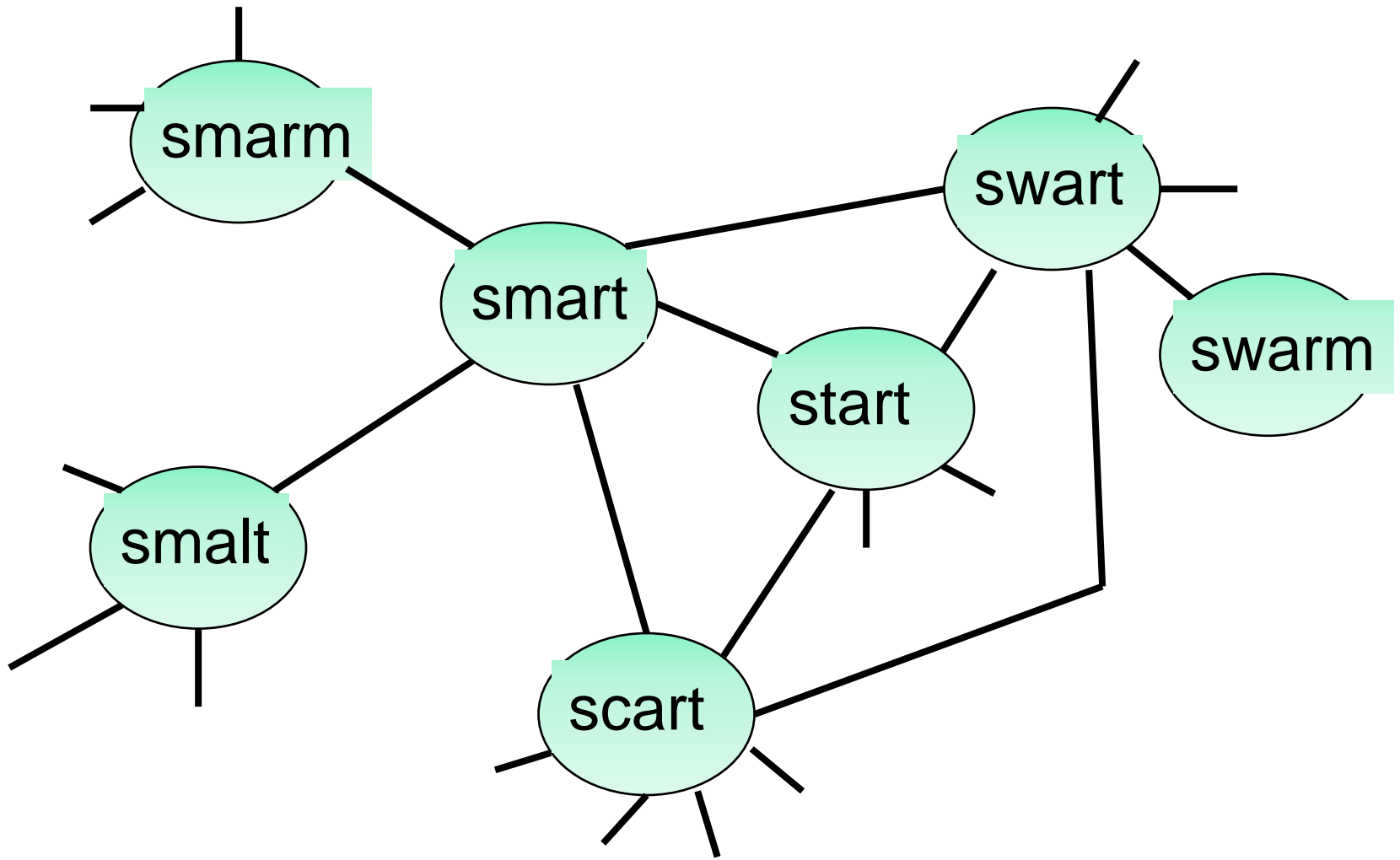
Done with smart, dequeue (swart)
[start, scart, smalt, smarm]



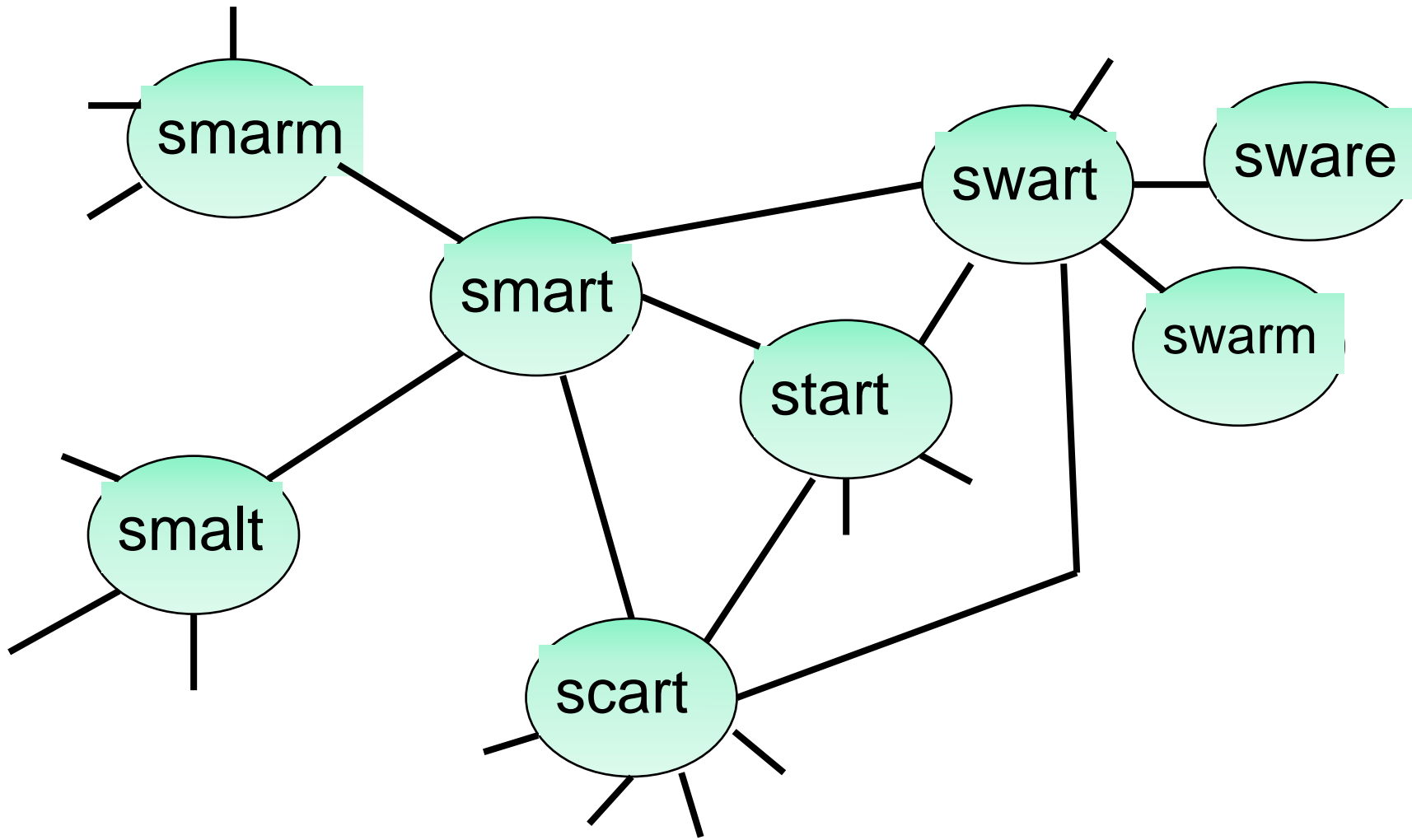
loop through edges of swart (start already present)
[start, scart, smalt, smarm]



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 path 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;

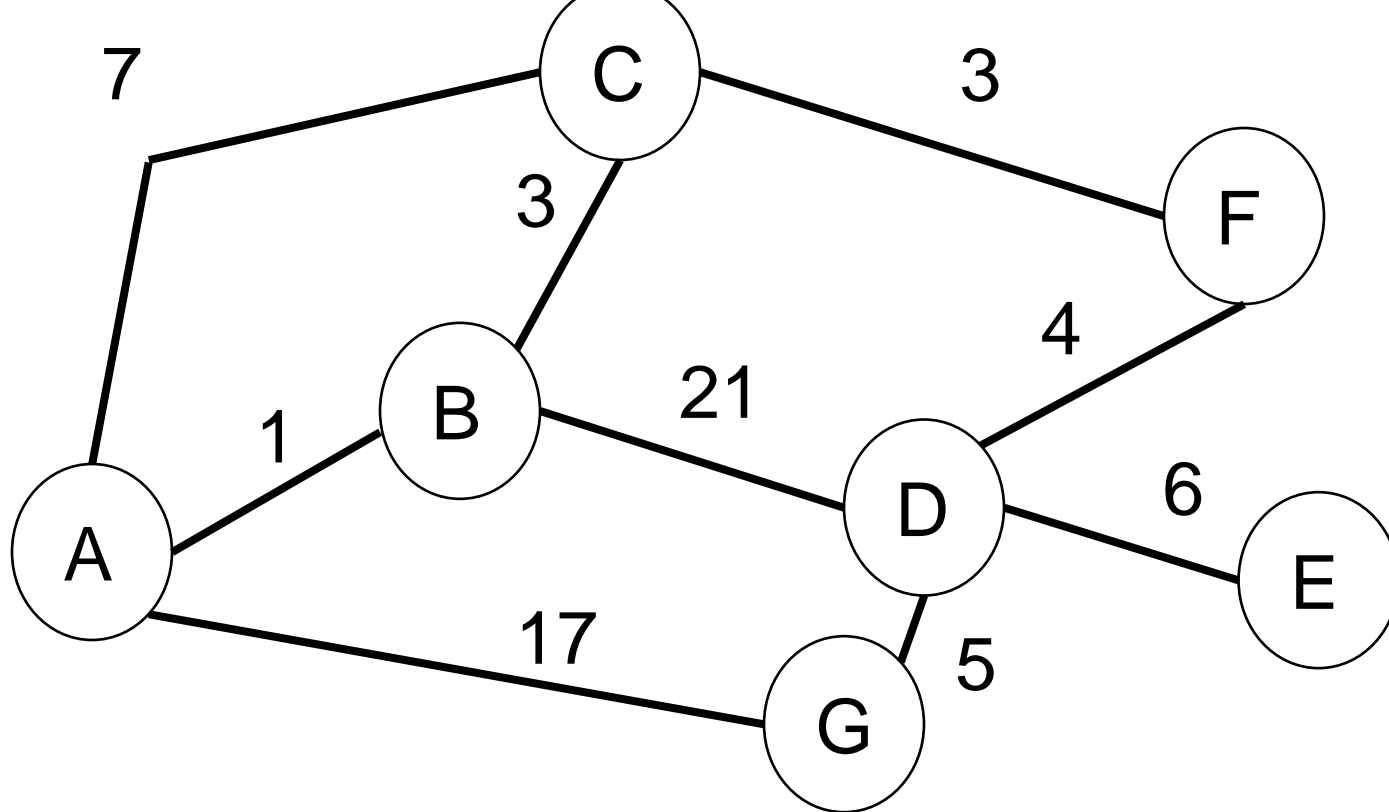
        // call before finding new paths
        public void reset()
```

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**

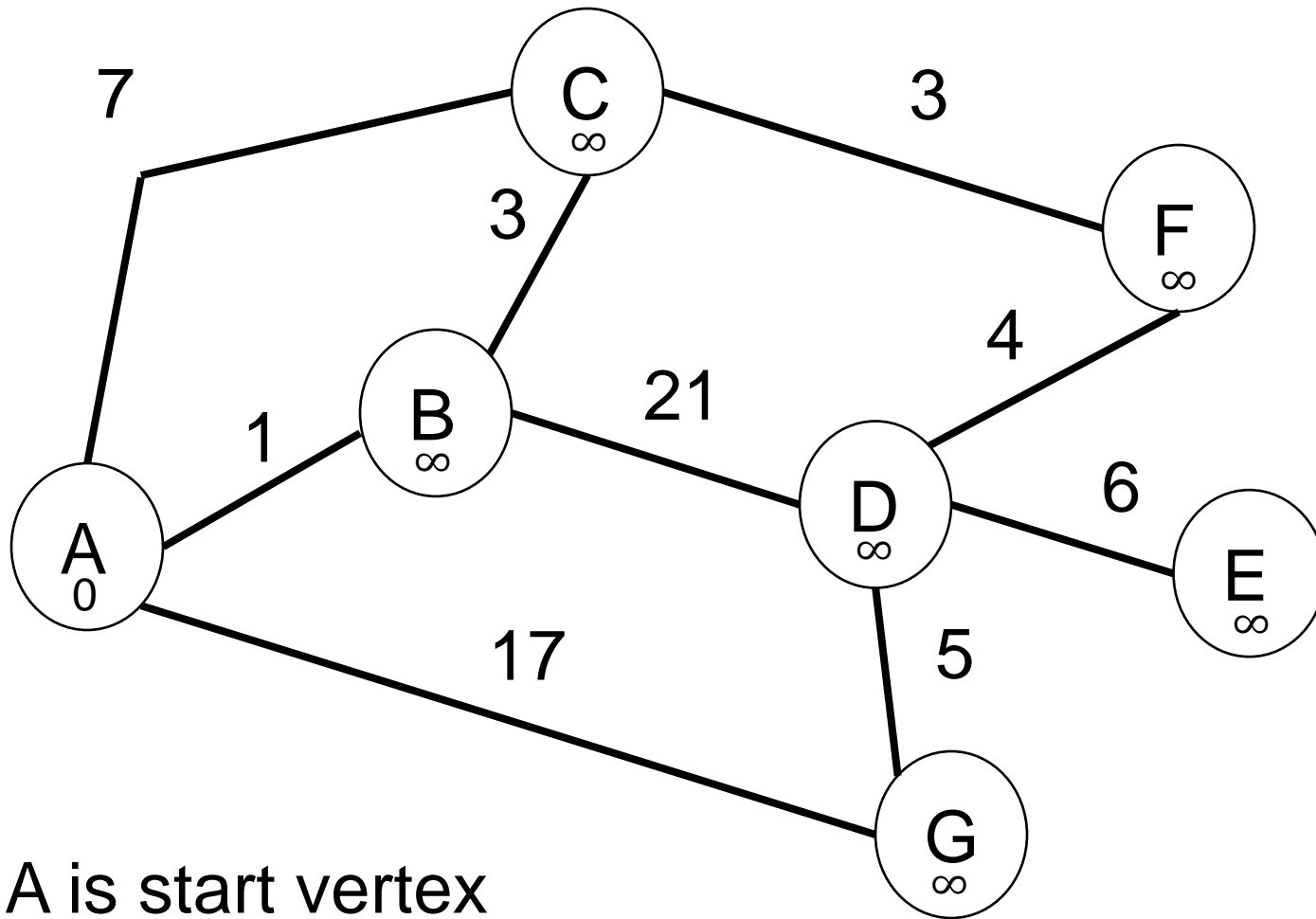
Dijkstra's Algorithm

- ▶ 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?

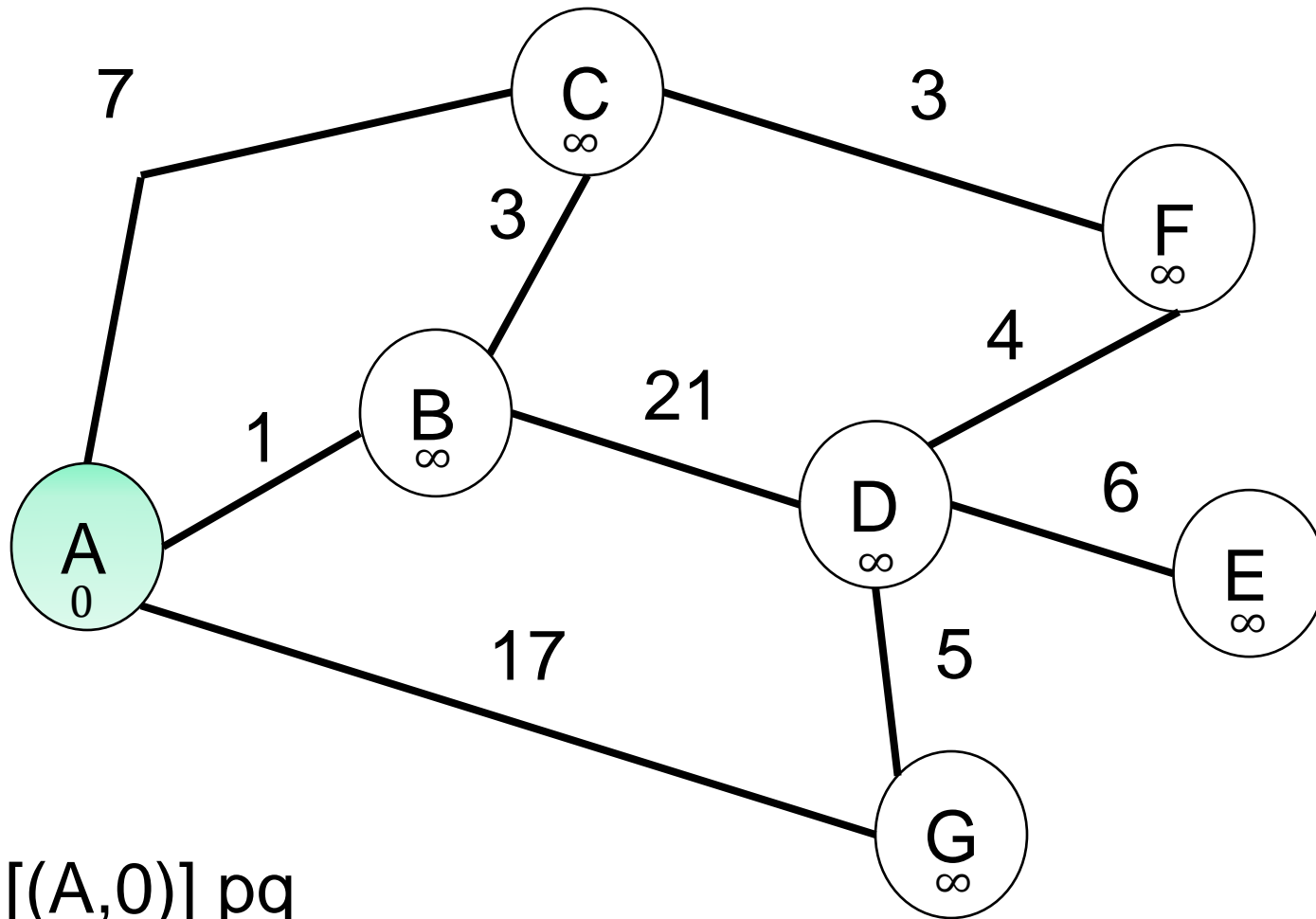
- A. 5
- B. 17
- C. 20
- D. 28
- E. 37



A is start vertex

Set cost of A to 0, all others to INFINITY

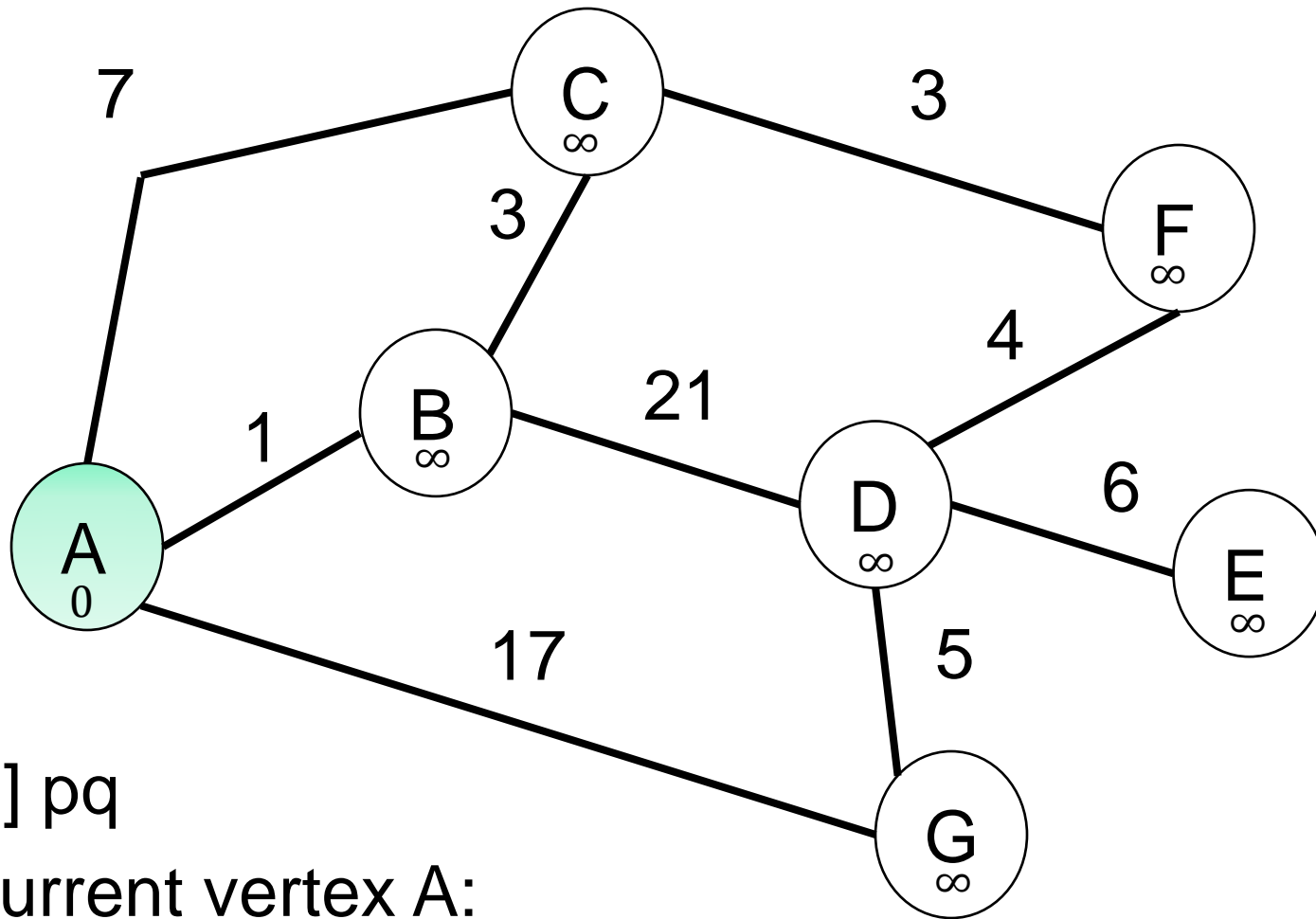
Place A in a priority queue



$[(A,0)]$ pq

dequeue $(A,0)$

Mark A as visited



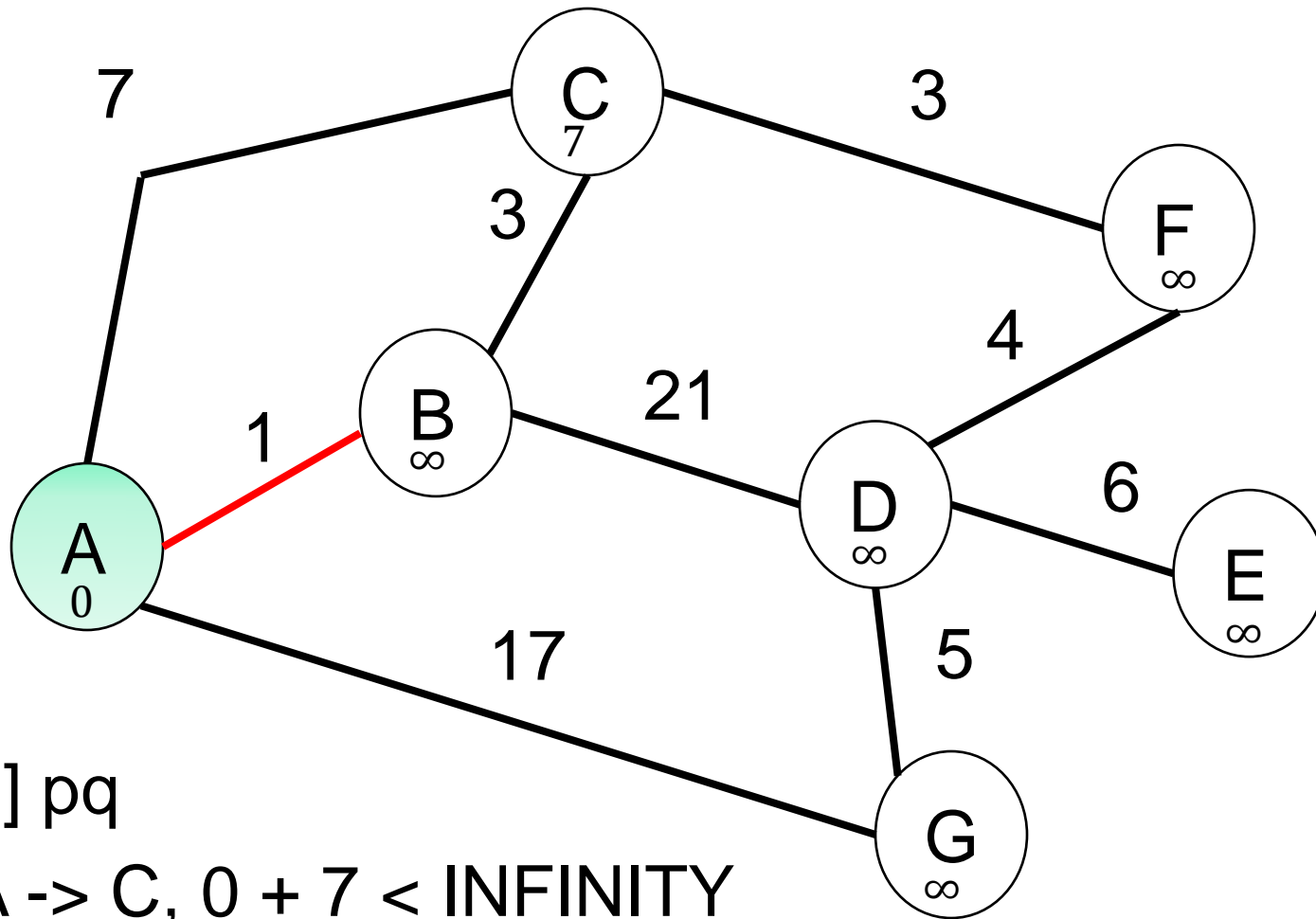
[] pq

current vertex A:

loop through A's edges

if sum of cost from A to dest is less than current cost

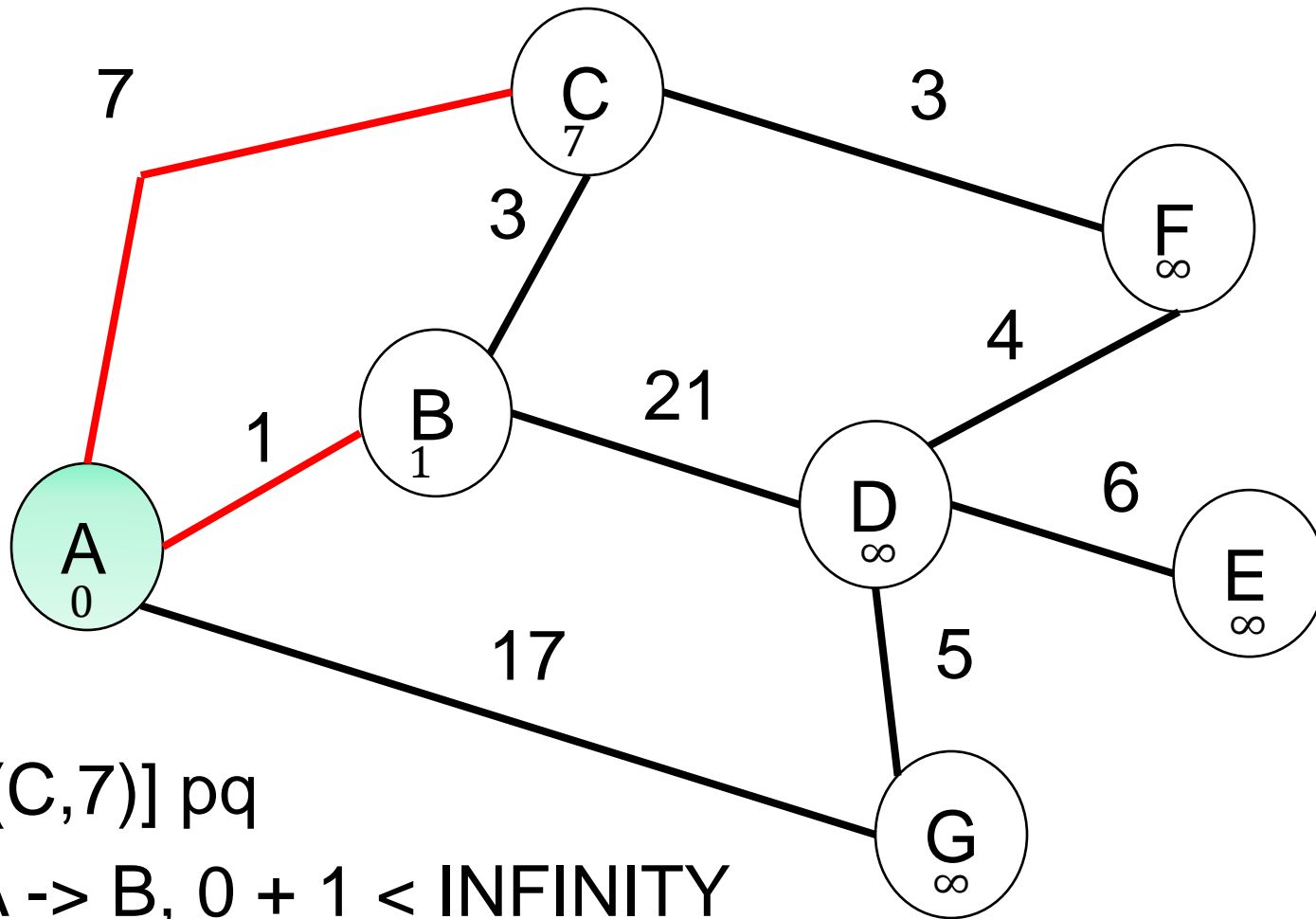
update cost and prev



[] pq

A -> C, $0 + 7 < \text{INFINITY}$

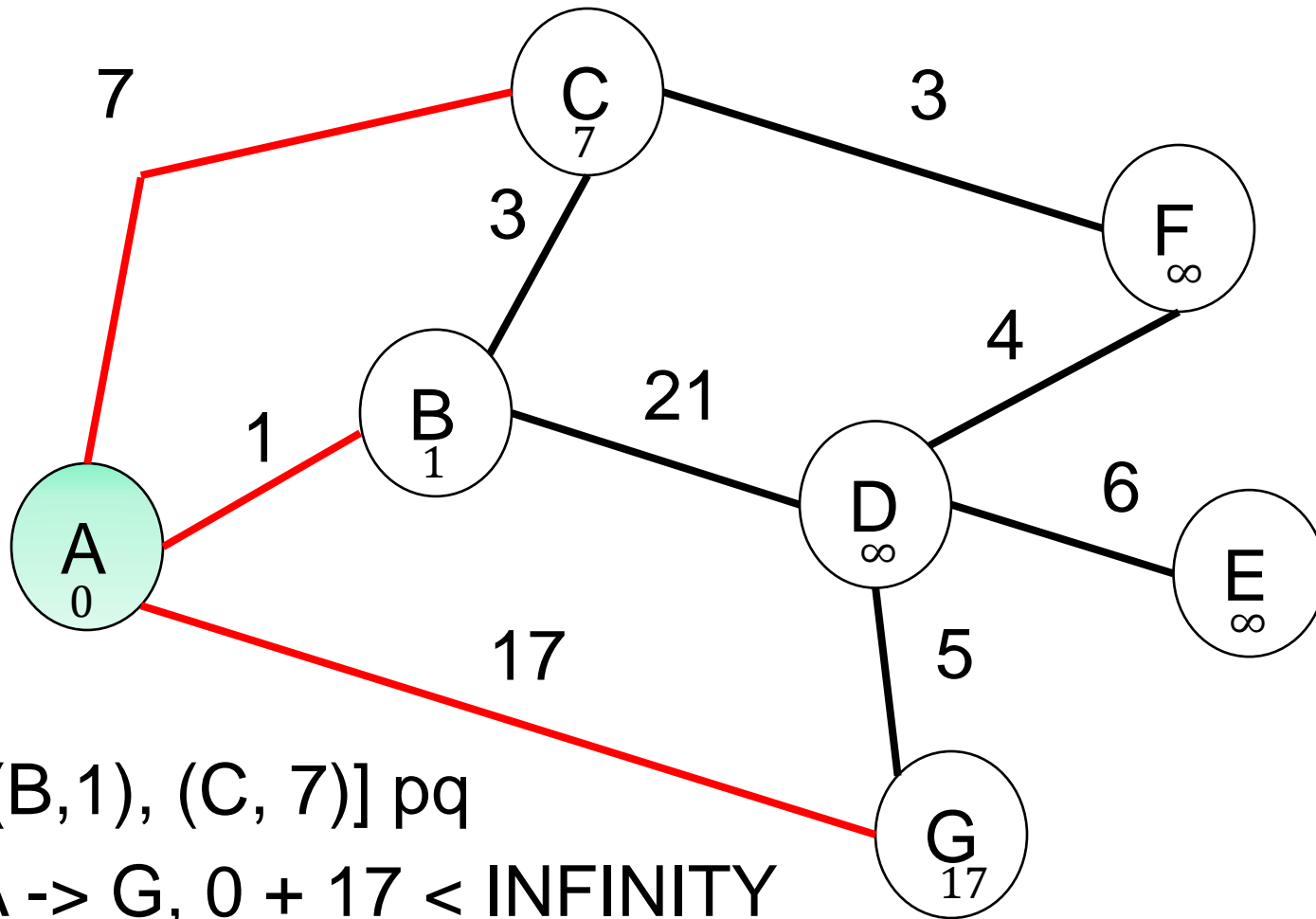
[(C,7)] pq



$[(C, 7)]$ pq

$A \rightarrow B, 0 + 1 < \text{INFINITY}$

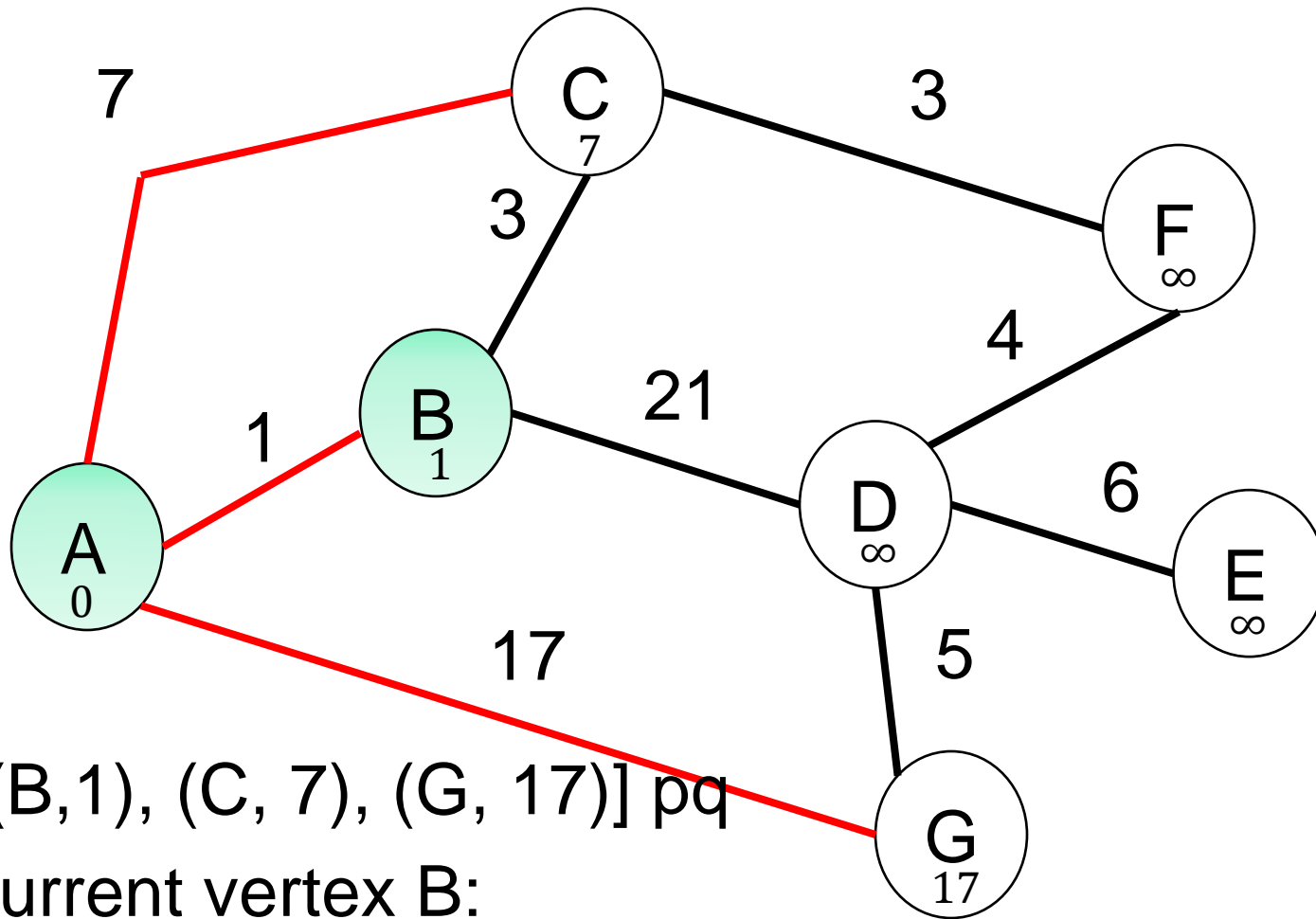
$[(B, 1), (C, 7)]$ pq (Note, the (B, 1) jumps in front of (C, 7))

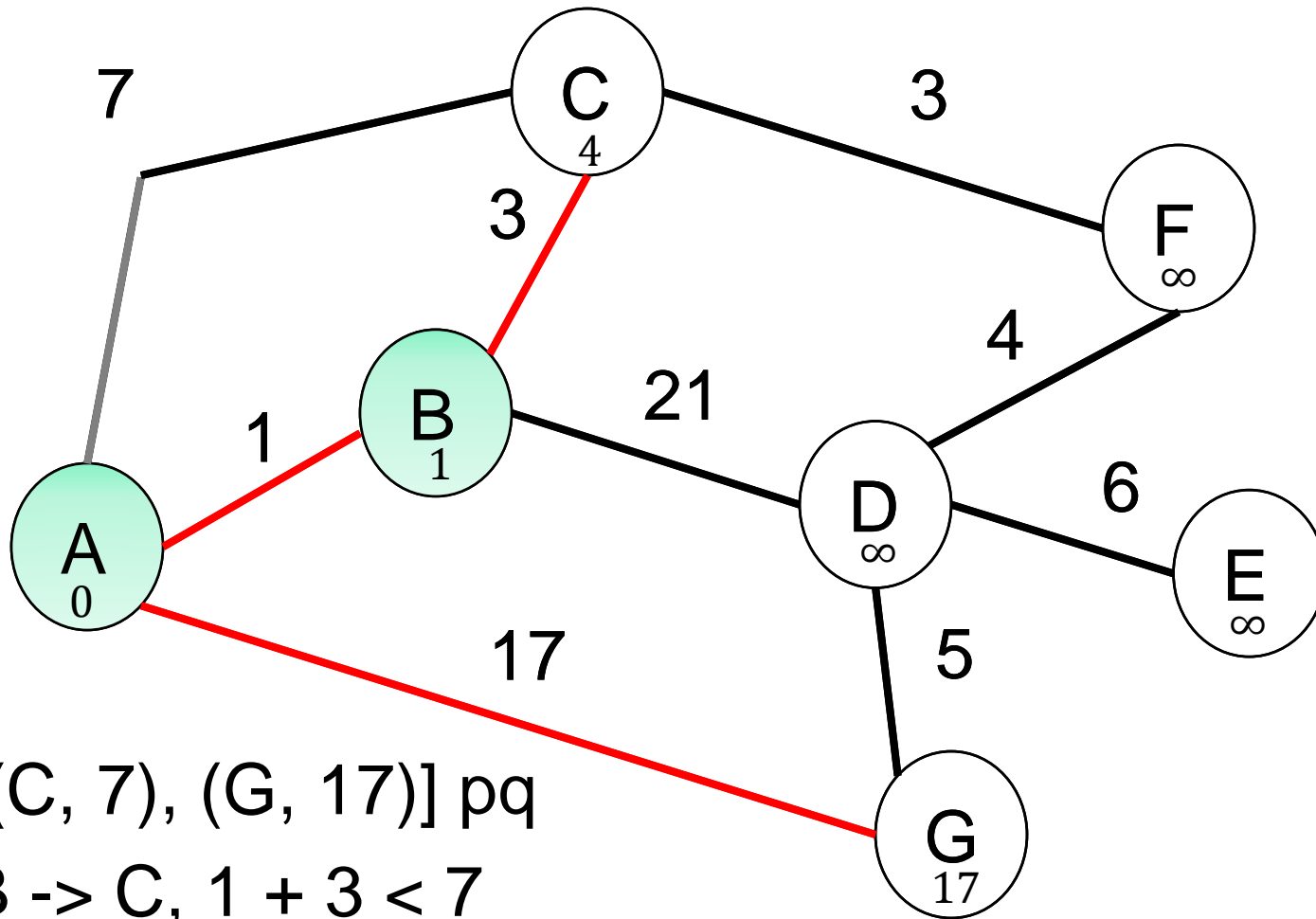


$[(B, 1), (C, 7)]$ pq

$A \rightarrow G, 0 + 17 < \text{INFINITY}$

$[(B, 1), (C, 7), (G, 17)]$ pq



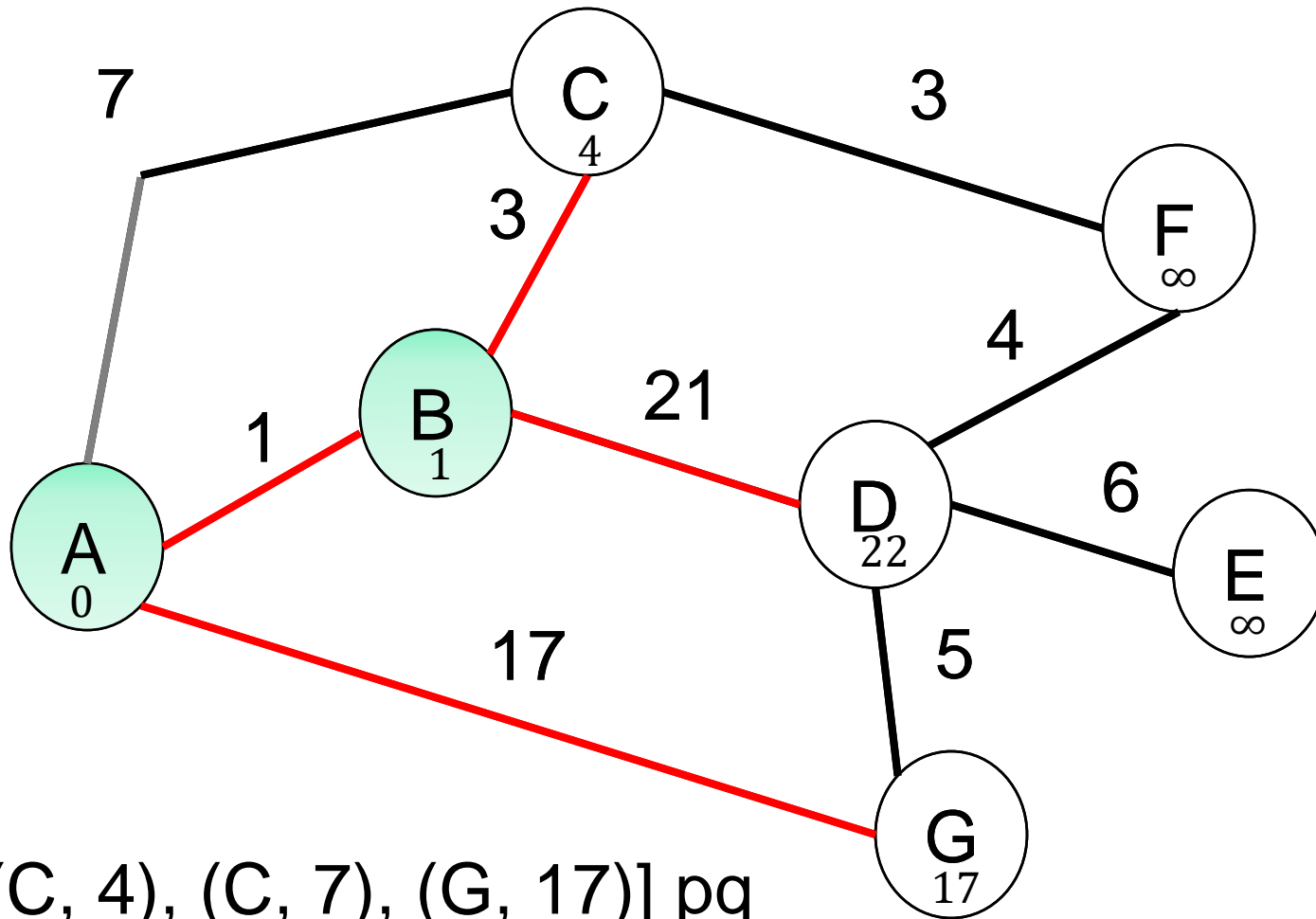


$[(C, 7), (G, 17)]$ pq

$B \rightarrow C, 1 + 3 < 7$

update C's cost and previous

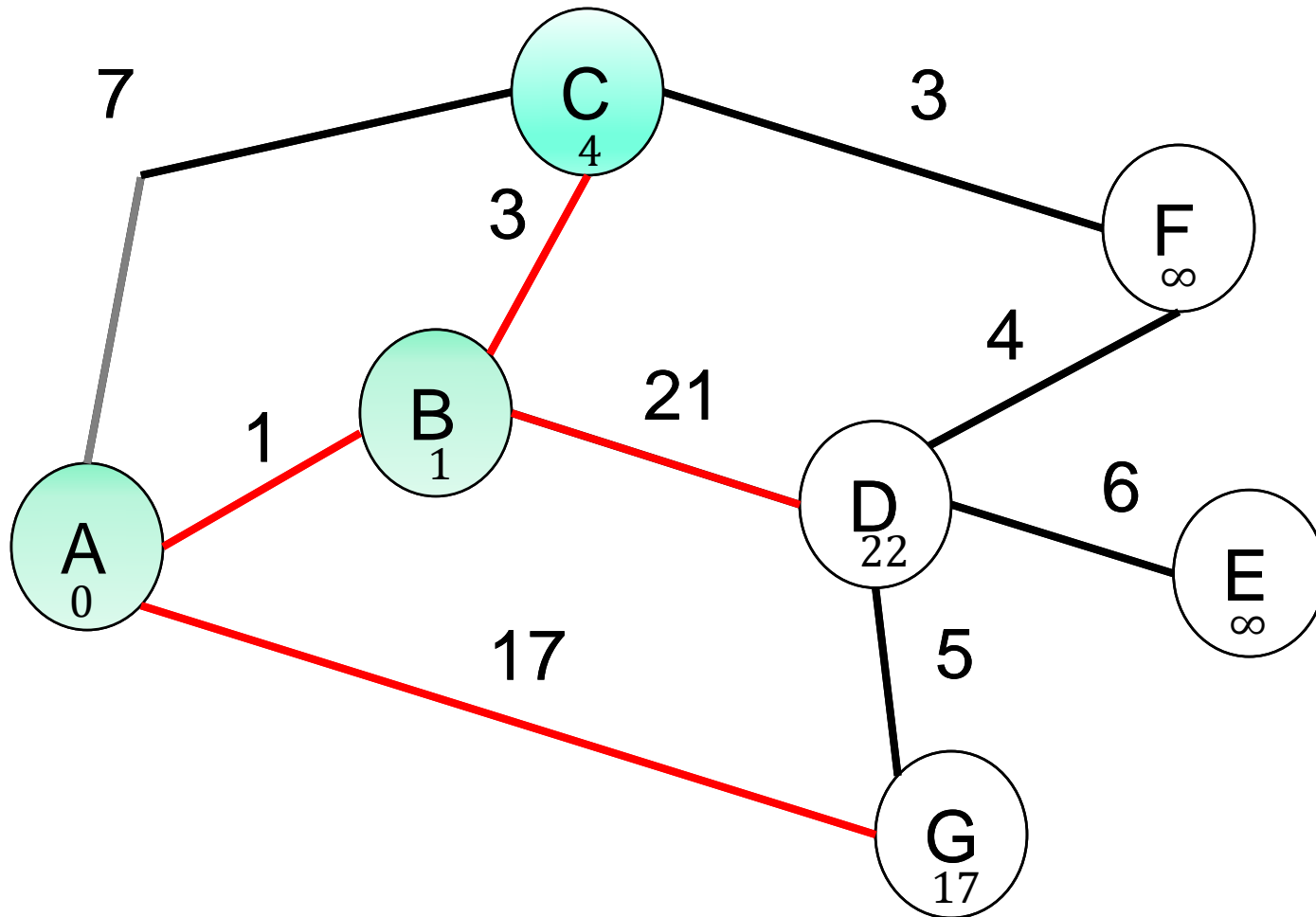
$[(C, 4), (C, 7), (G, 17)]$ pq



$[(C, 4), (C, 7), (G, 17)]$ pq

$B \rightarrow D, 1 + 21 < \text{INFINITY}$

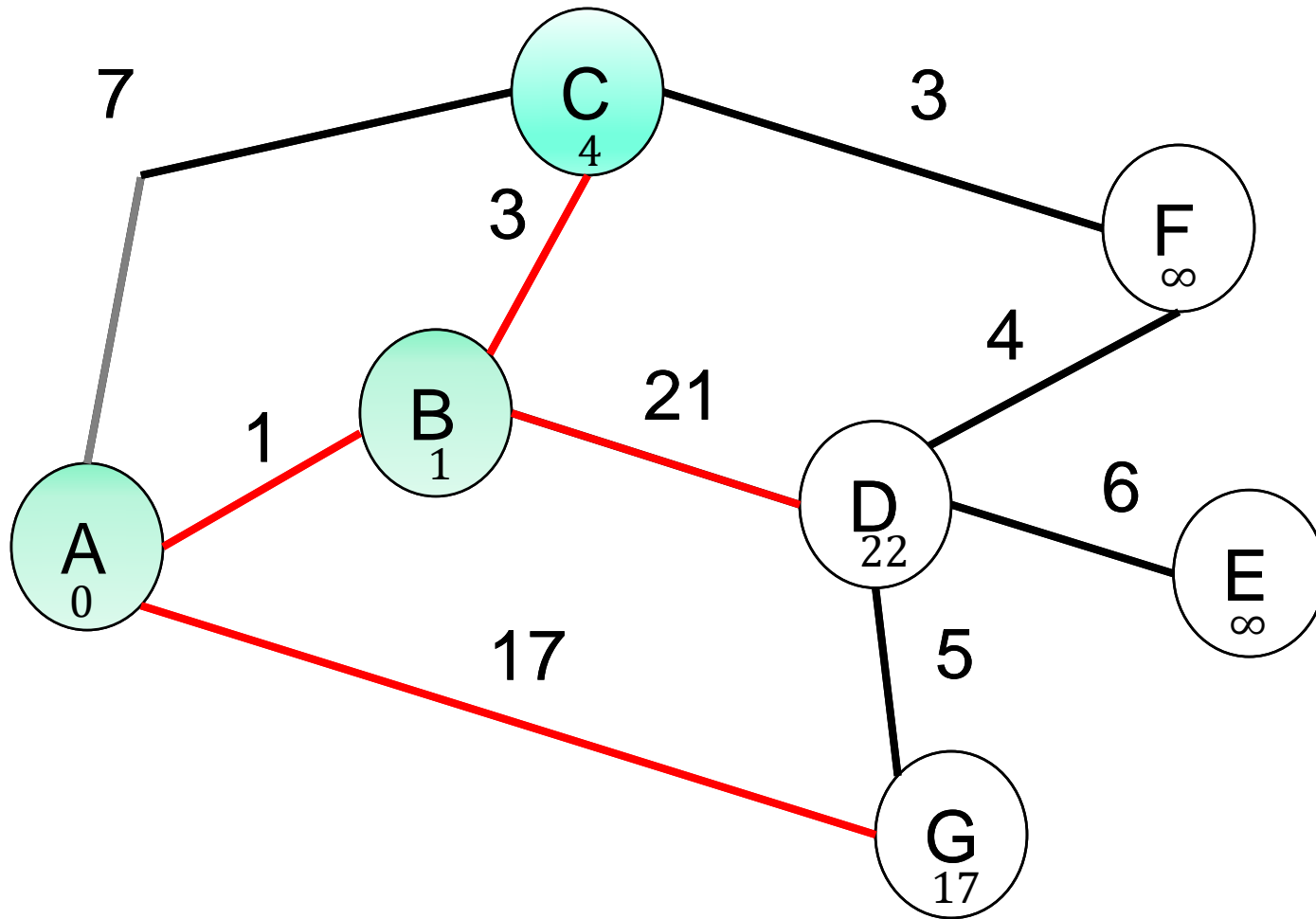
$[(C, 4), (C, 7), (G, 17), (D, 22)]$ pq



$[(C, 4), (C, 7), (G, 17), (D, 22)]$ pq

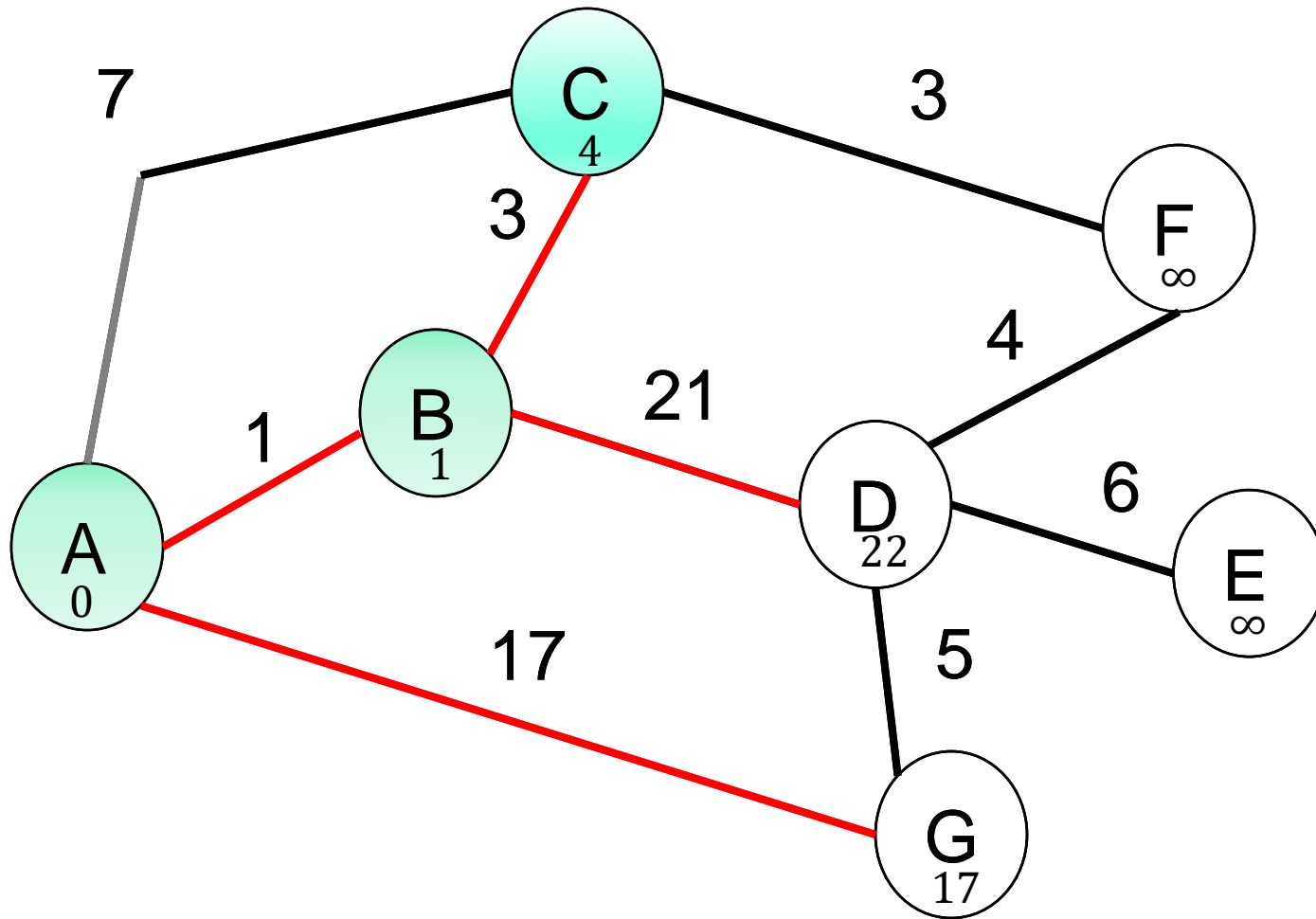
current vertex is C, cost 4

loop through C's edges



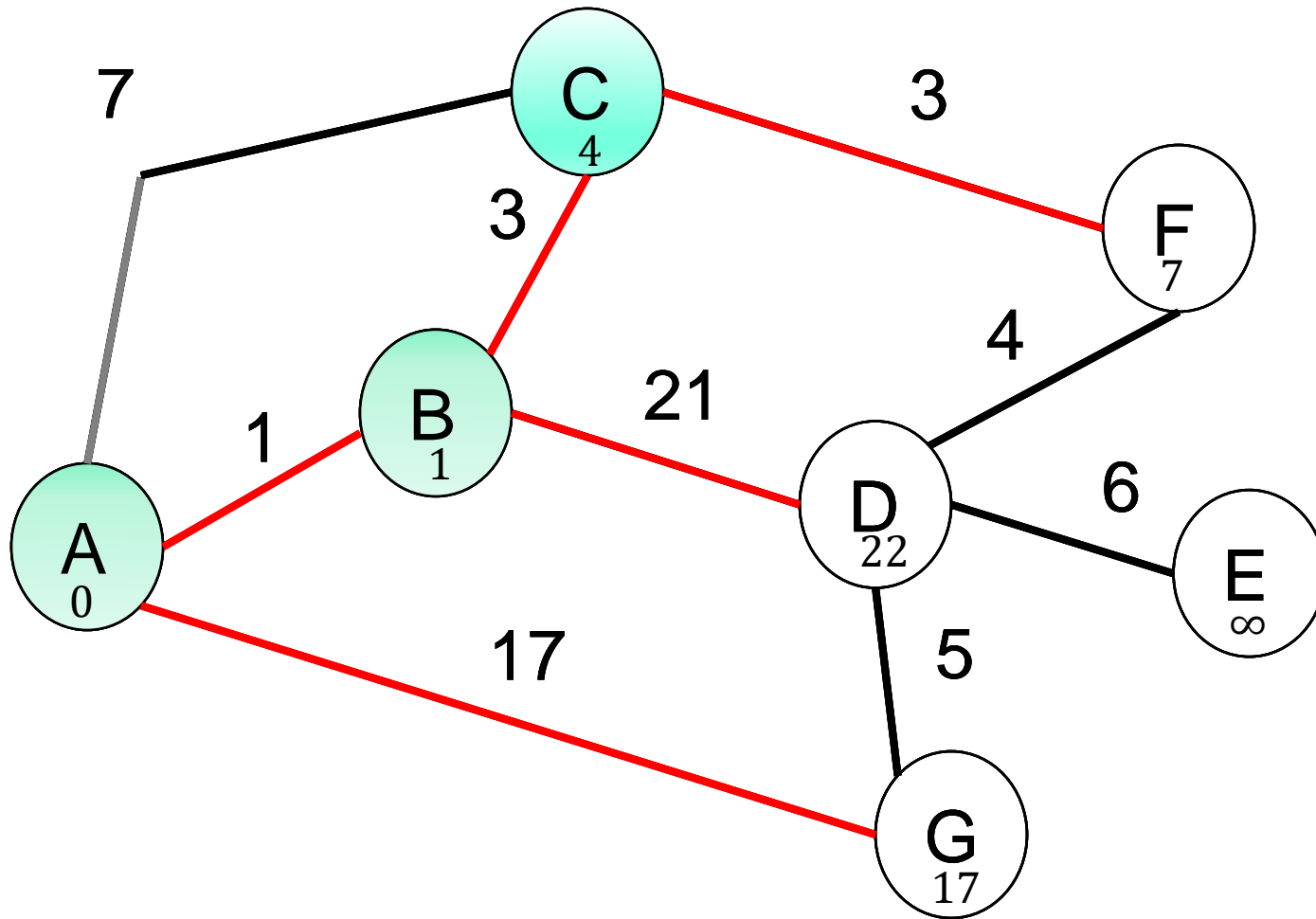
$[(C, 7), (G, 17), (D, 22)]$ pq

$C \rightarrow A, 7 + 4 \not< 0$, skip



$[(C, 7), (G, 17), (D, 22)]$ pq

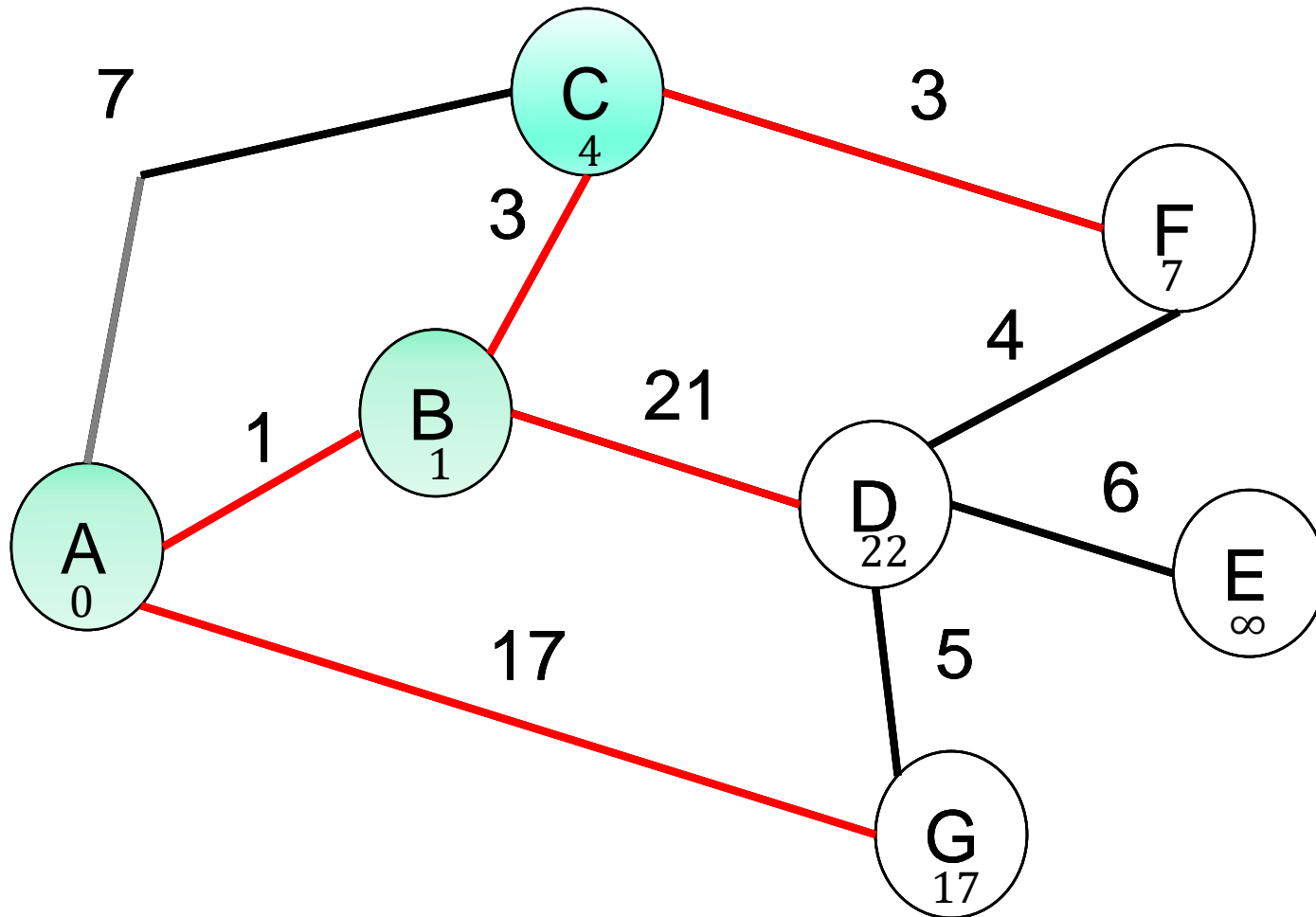
$C \rightarrow B, 4 + 3 \not< 1$, skip



$[(C, 7), (G, 17), (D, 22)]$ pq

$C \rightarrow F, 4 + 3 < \text{INFINITY}$

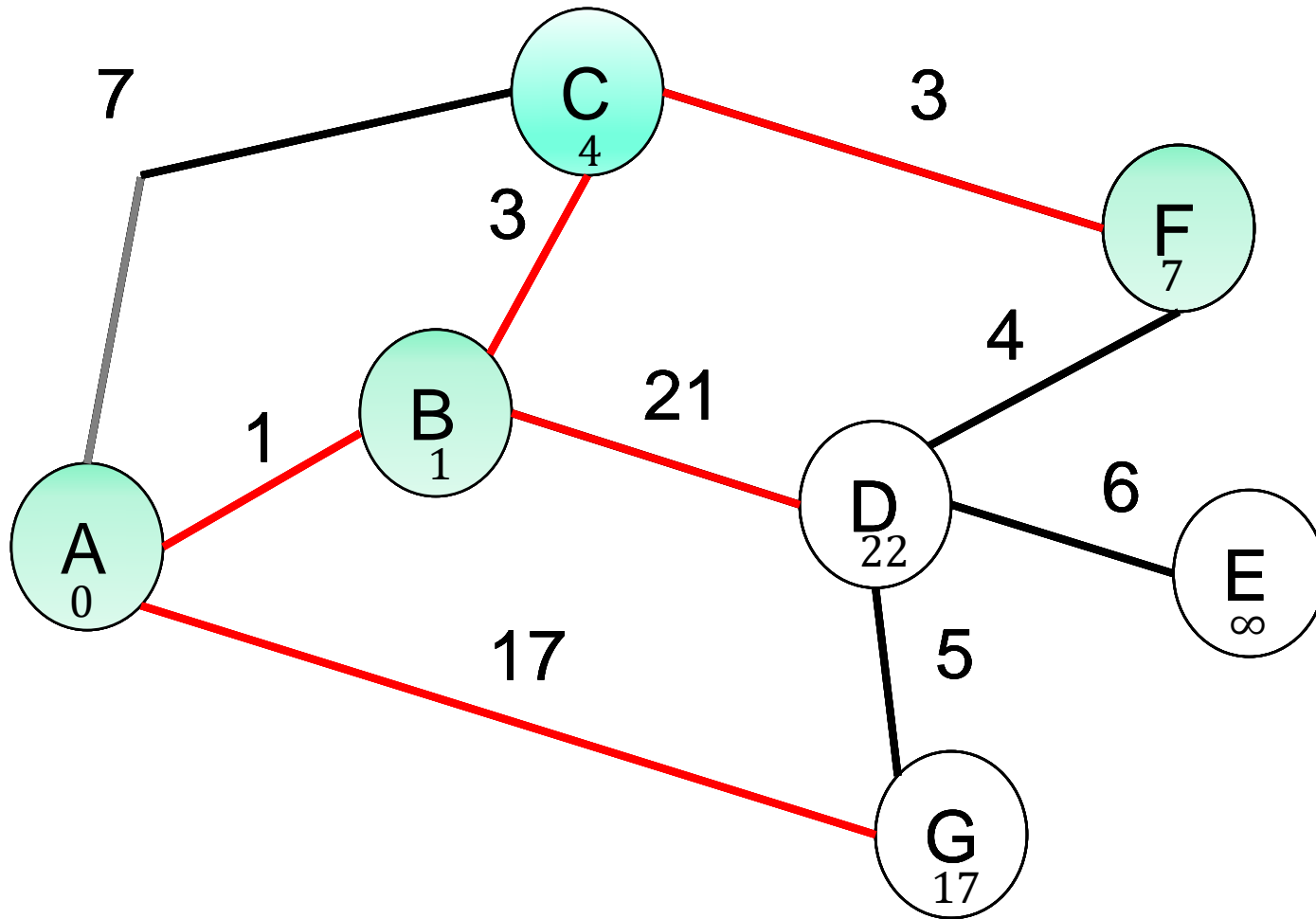
$[(C, 7), (F, 7), (G, 17), (D, 22)]$ pq



$[(C, 7), (F, 7), (G, 17), (D, 22)]$ pq

current vertex is C

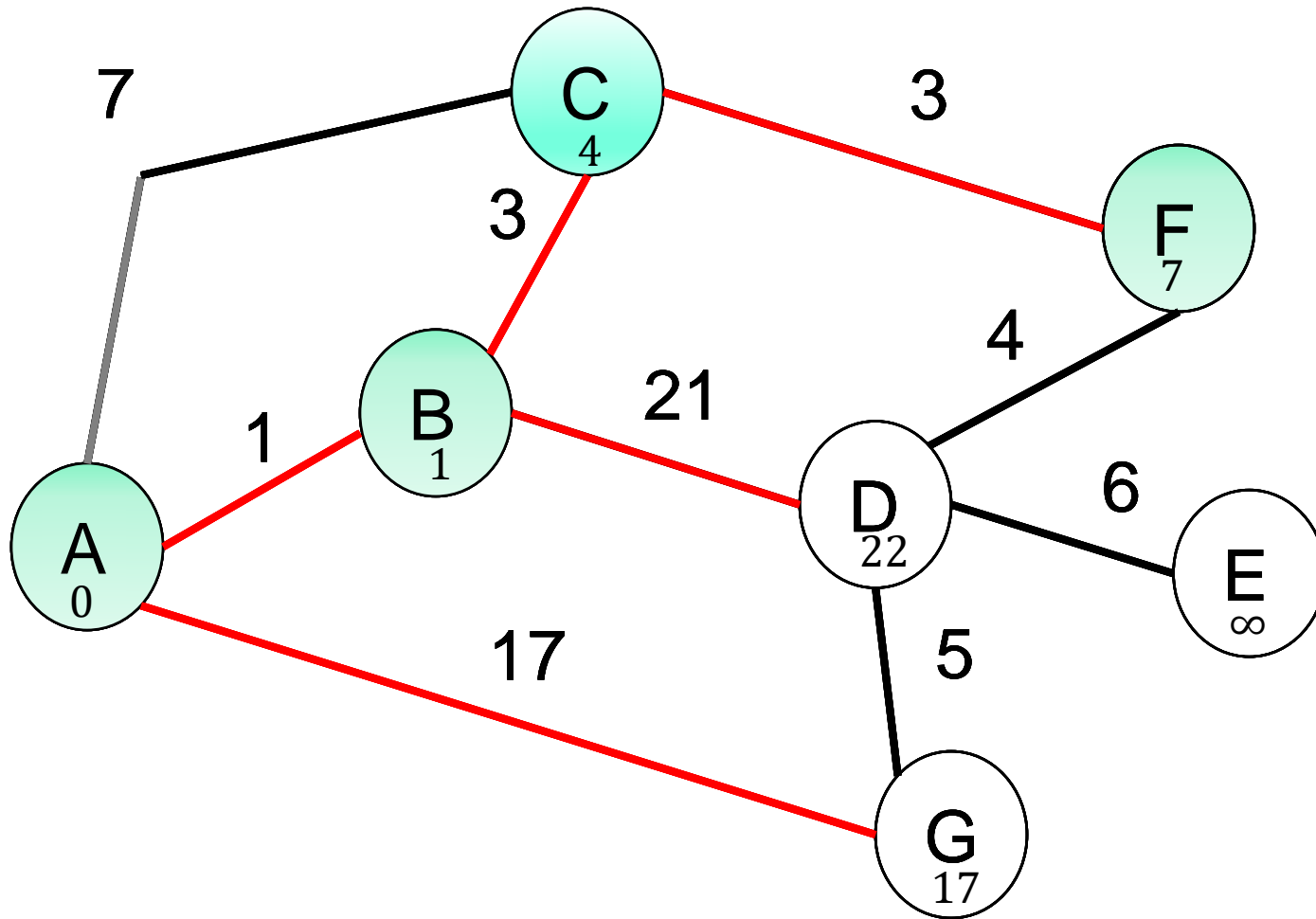
Already visited so skip



[(F, 7), (G, 17), (D, 22)] pq

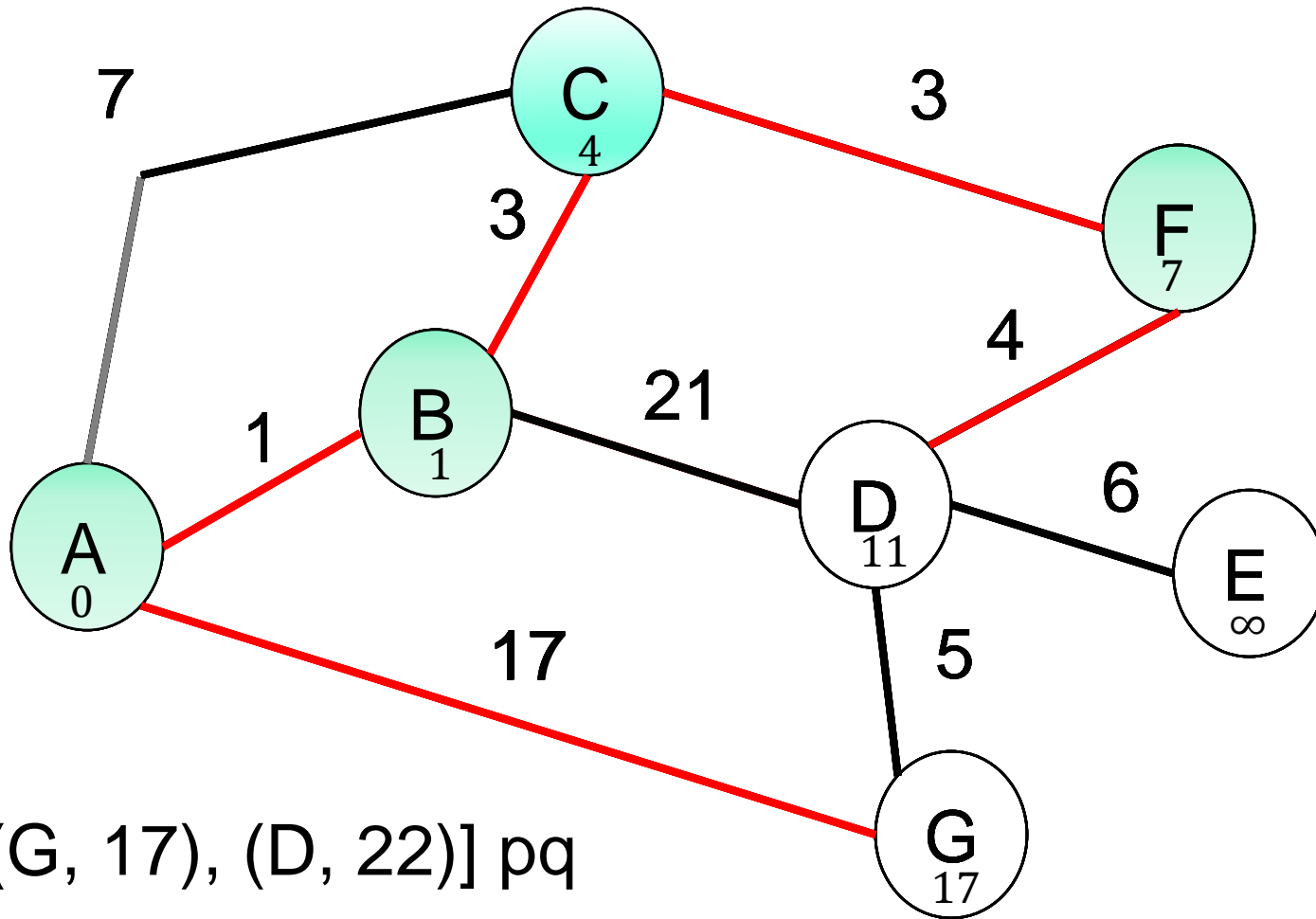
current vertex is F

loop through F's edges



$[(G, 17), (D, 22)]$ pq

$F \rightarrow C, 7 + 3 \not< 4$, so skip



$[(G, 17), (D, 22)]$ pq

$F \rightarrow D, 7 + 4 < 22$

update D's cost and previous

$[(D, 11), (G, 17), (D, 22)]$ pq

Aside - Implementing Dijkstra's

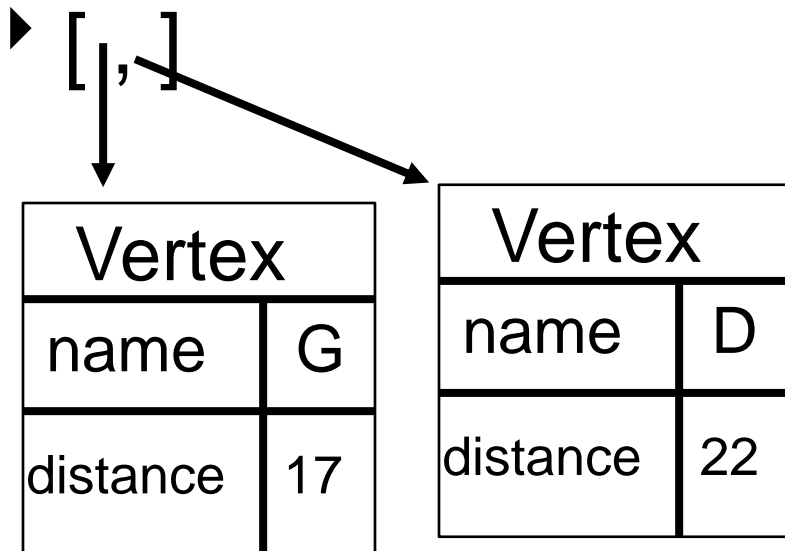
- ▶ Create a Path class to allow for multiple paths and distances (costs) to a given vertex

```
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

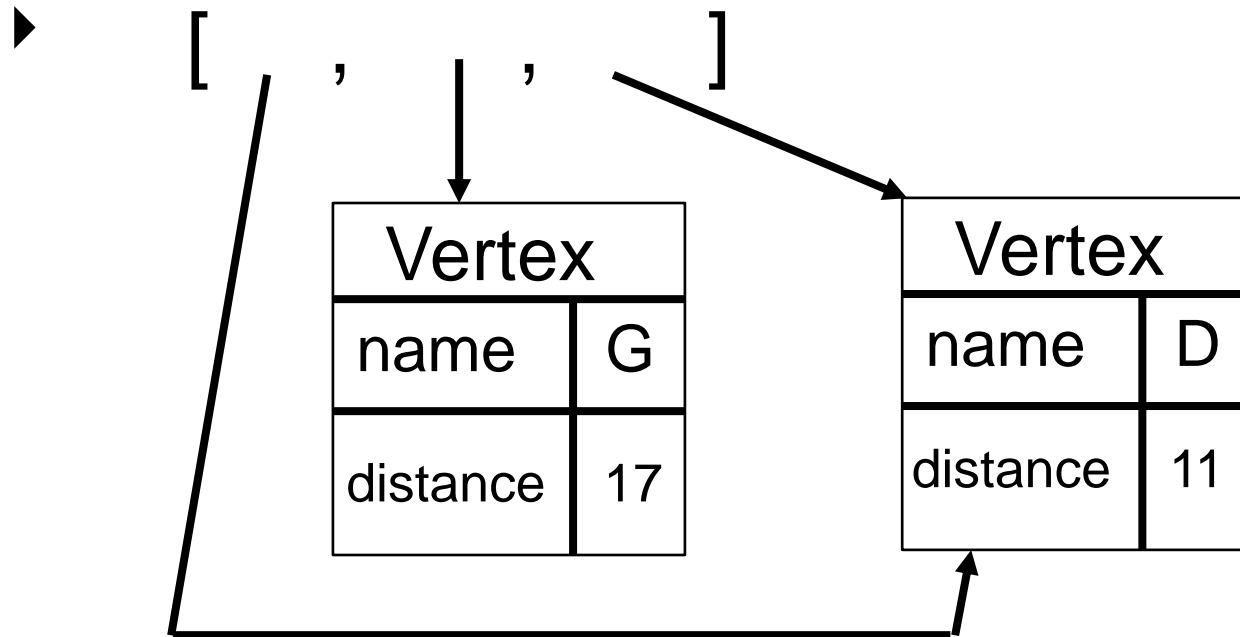
Why? References!!!

- ▶ 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***

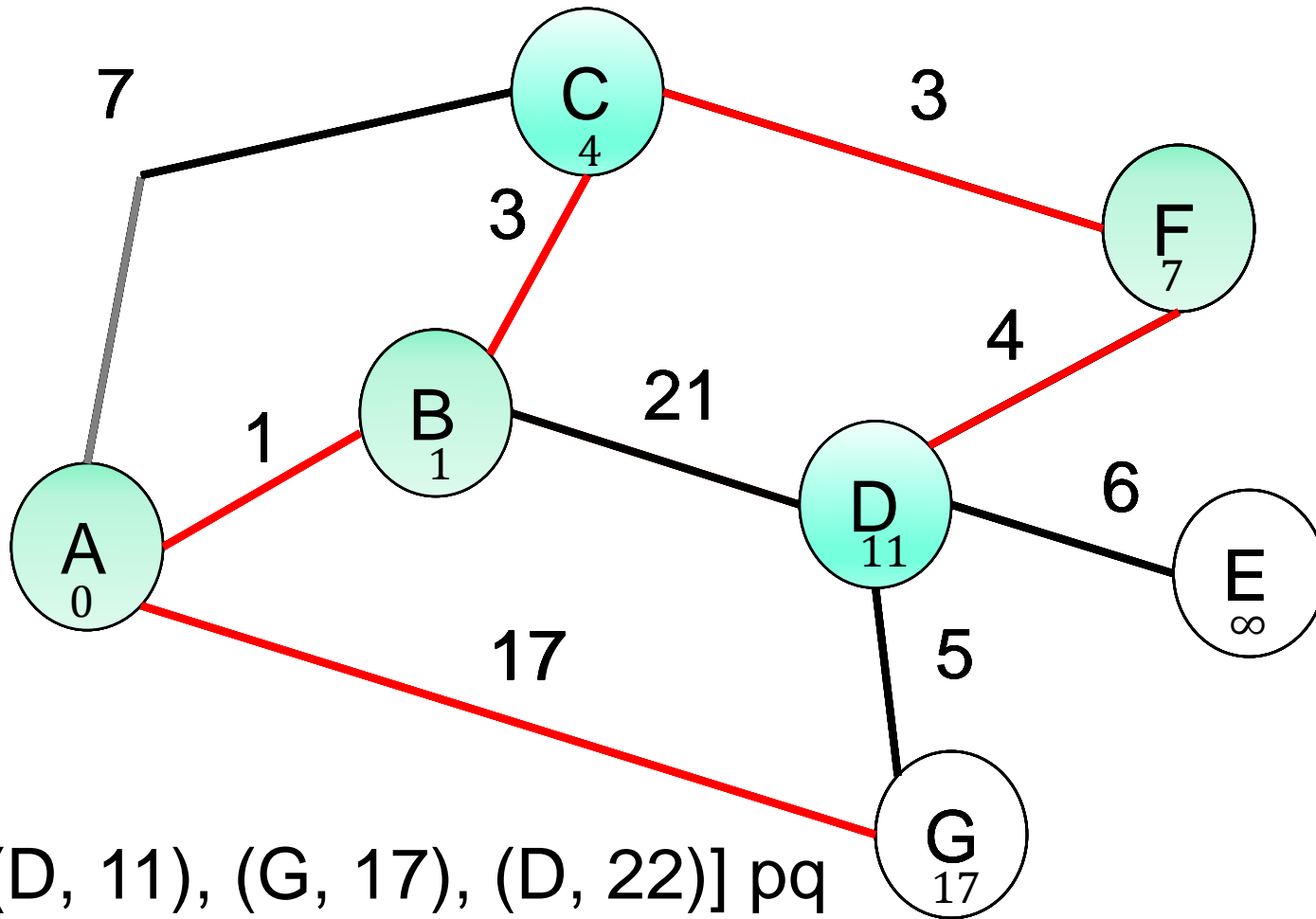


Lower Cost Path to D

- ▶ New, lower cost path to D. Alter Vertex D's distance to 11 and add to priority queue



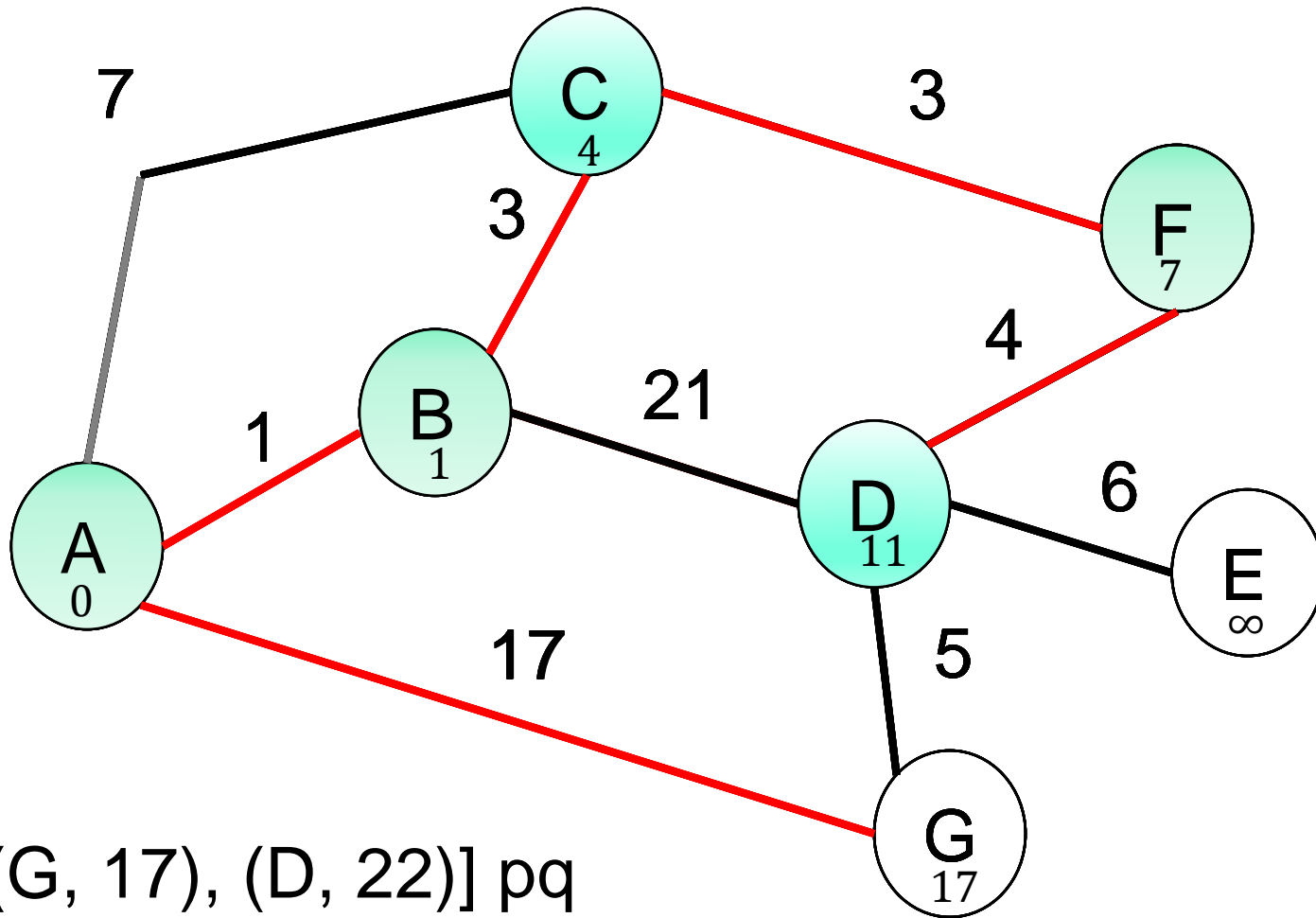
- ▶ PROBLEMS?????
- ▶ Abstractly [(D, 11), (G, 17), (D, 11)]



$[(D, 11), (G, 17), (D, 22)]$ pq

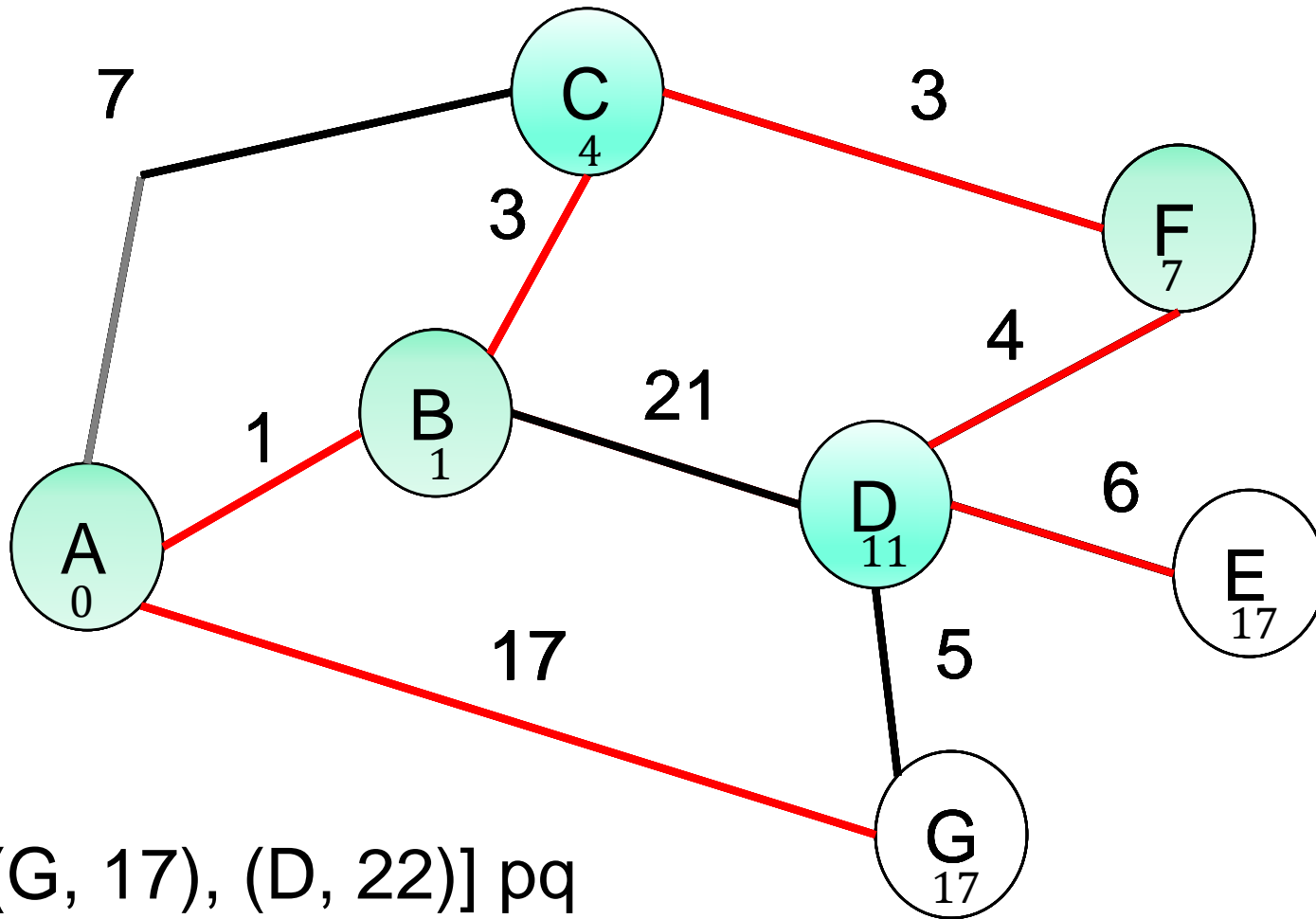
current vertex is D

loop through D's edges



$[(G, 17), (D, 22)]$ pq

D \rightarrow B, $11 + 21 \nless 1$, so skip

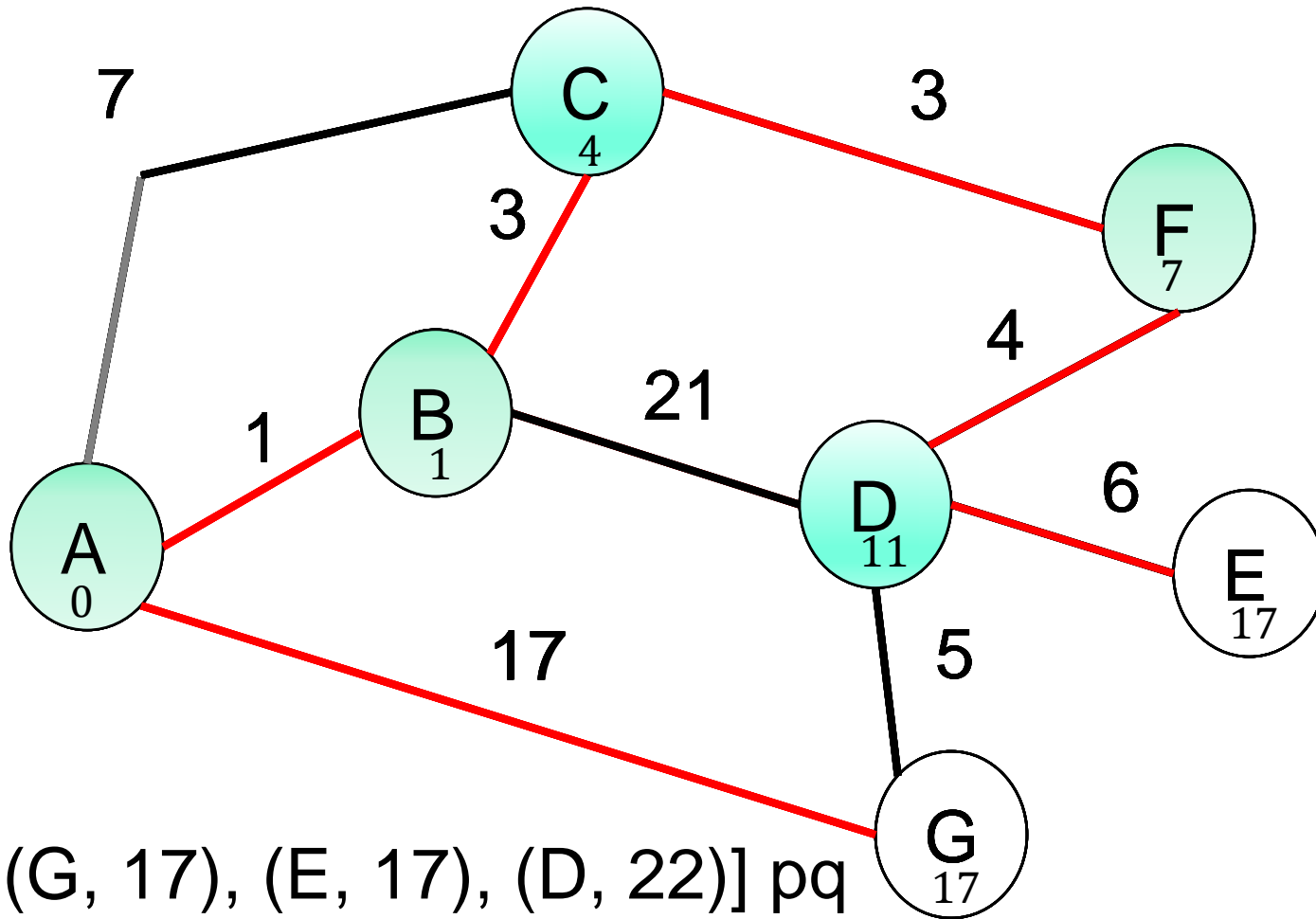


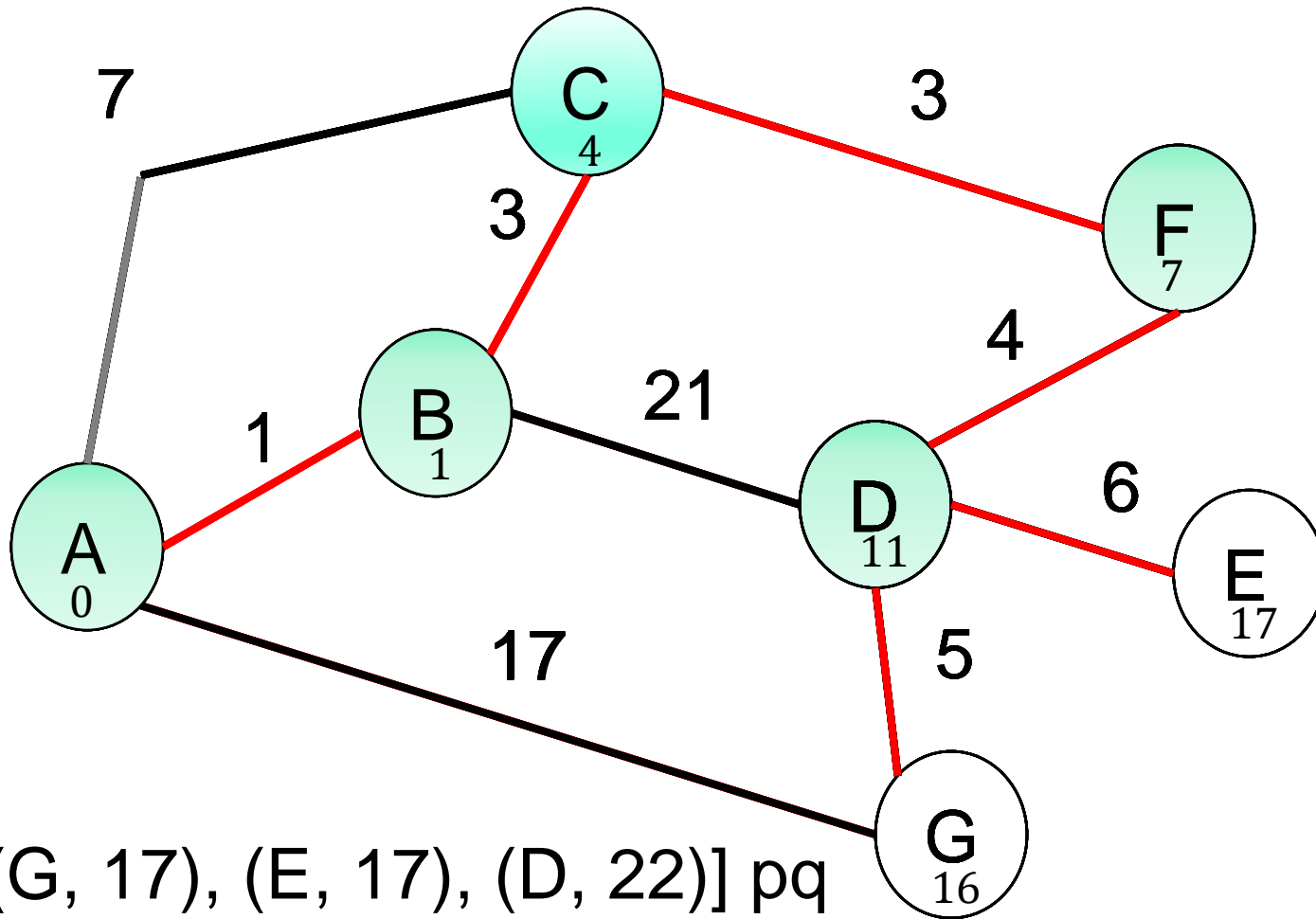
$[(G, 17), (D, 22)]$ pq

$D \rightarrow E, 11 + 6 < \text{INFINITY}$

update E's cost and previous

$[(G, 17), (E, 17), (D, 22)]$ pq



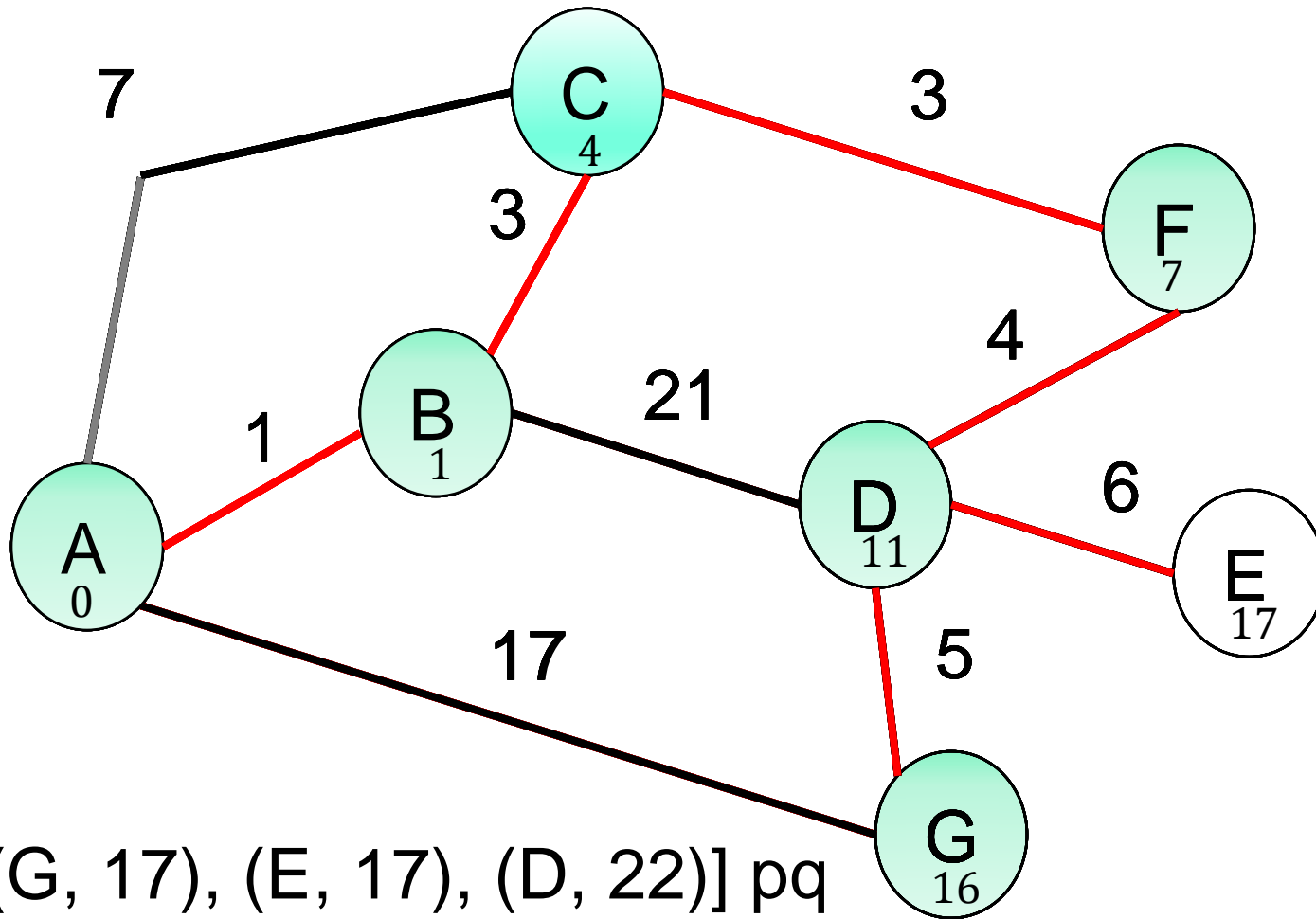


$[(G, 17), (E, 17), (D, 22)]$ pq

$D \rightarrow G, 11 + 5 < 17$

update G's cost and previous

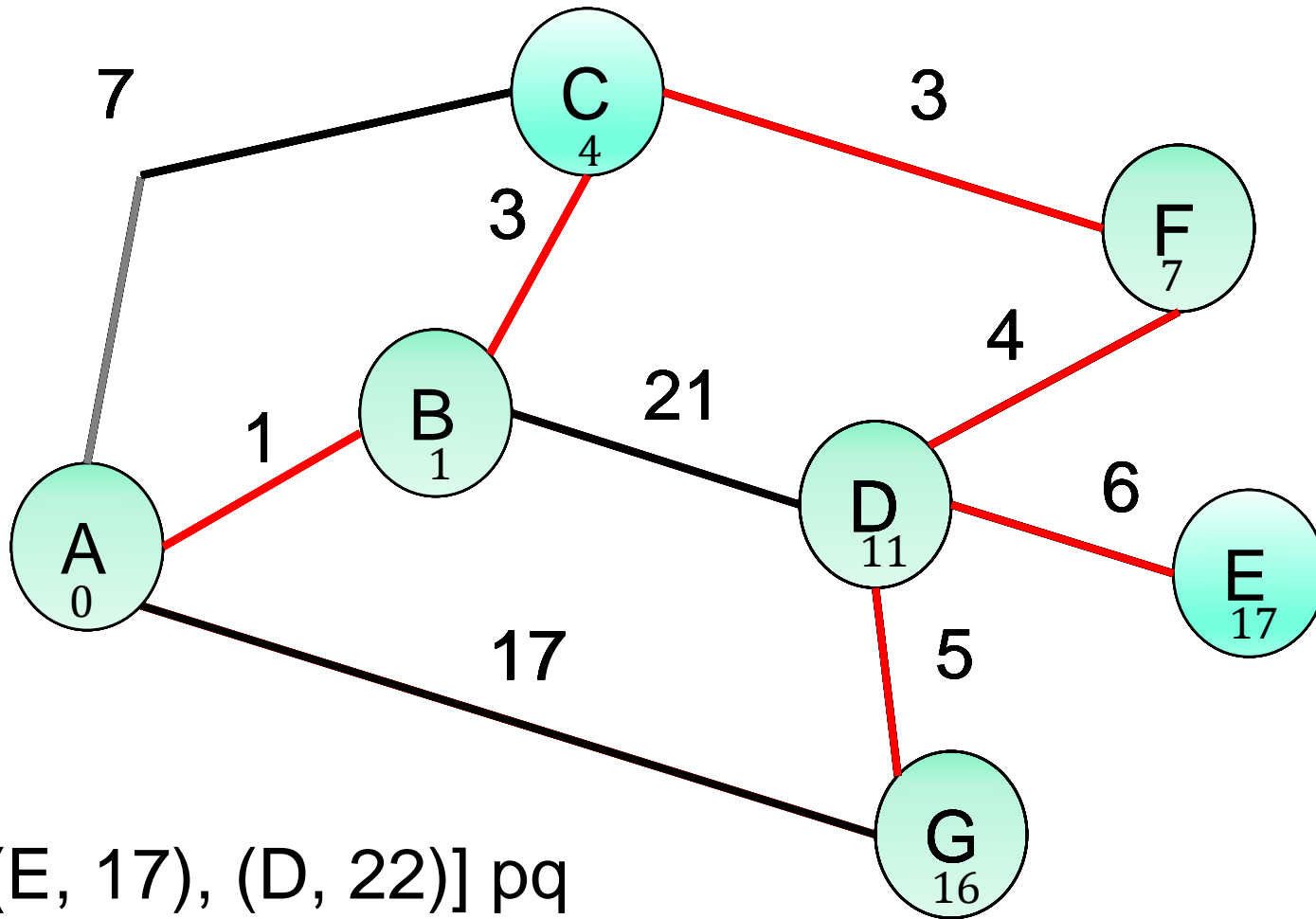
$[(G, 16), (G, 17), (E, 17), (D, 22)]$ pq



$[(G, 17), (E, 17), (D, 22)]$ pq

current vertex is G

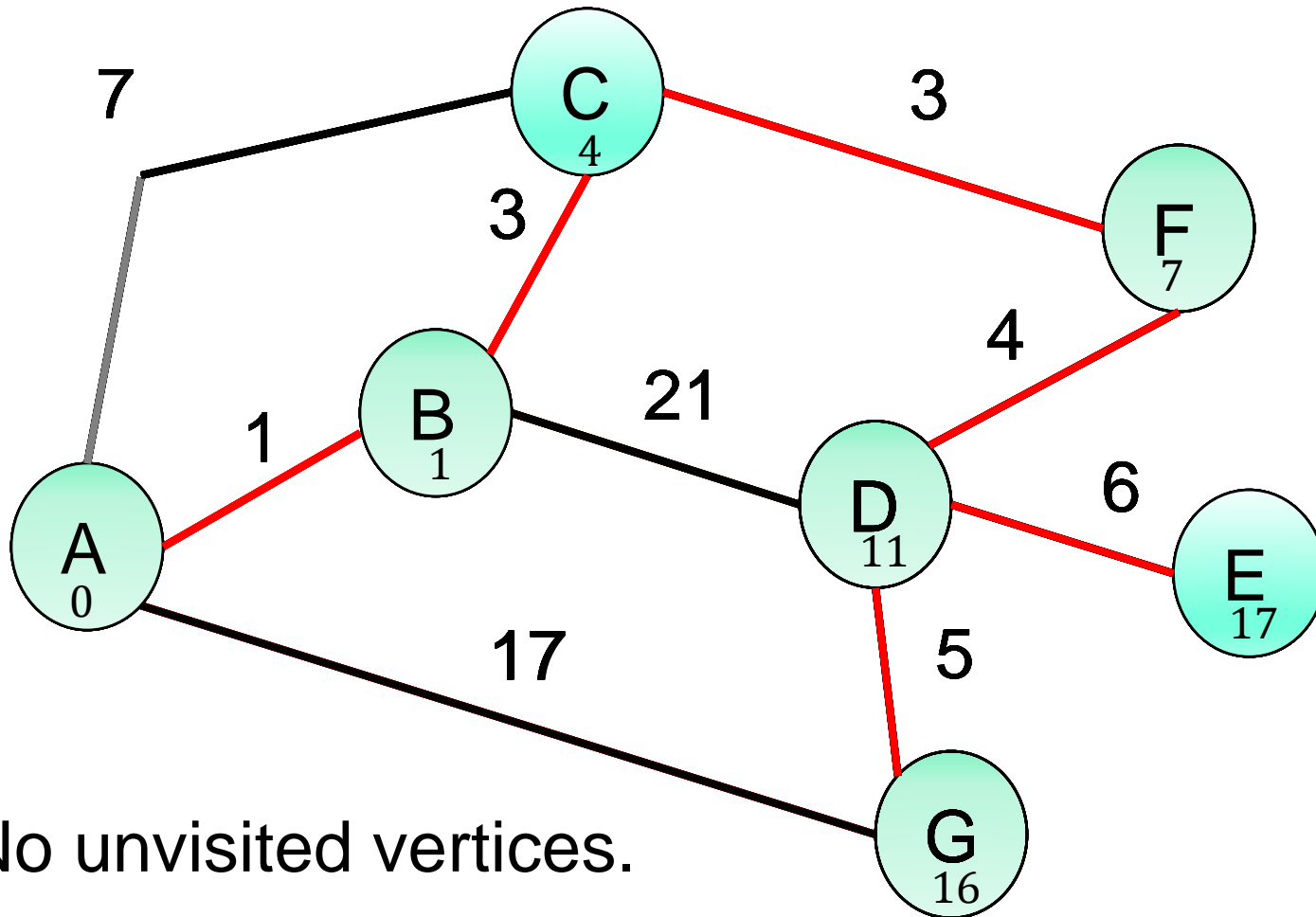
loop through edges, already visited all neighbors



$[(E, 17), (D, 22)]$ pq

current vertex is E

loop through edges, already visited all neighbors



No unvisited vertices.

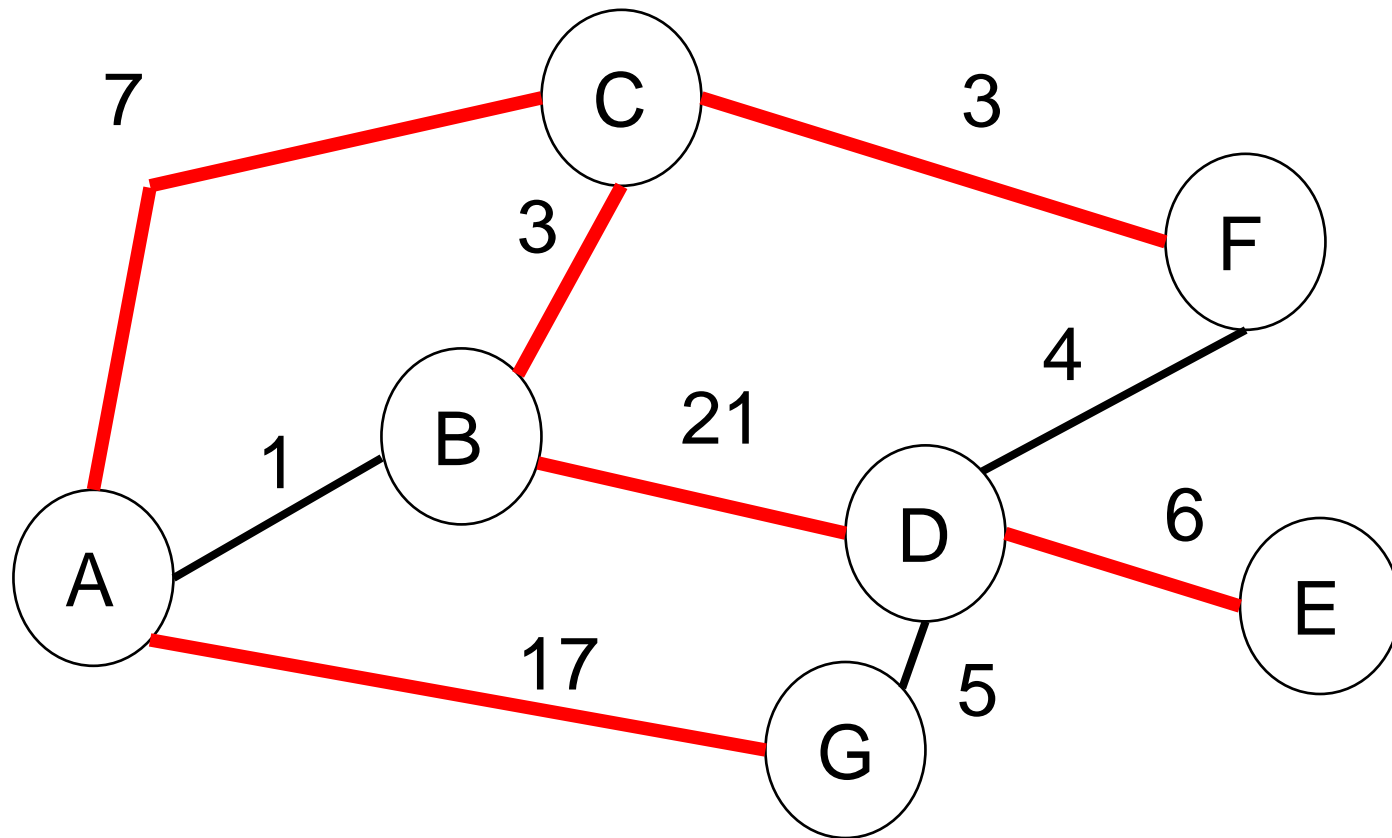
Each Vertex stores cost (distance) of lowest cost path from start Vertex to itself and previous vertex in path from start vertex to itself.

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

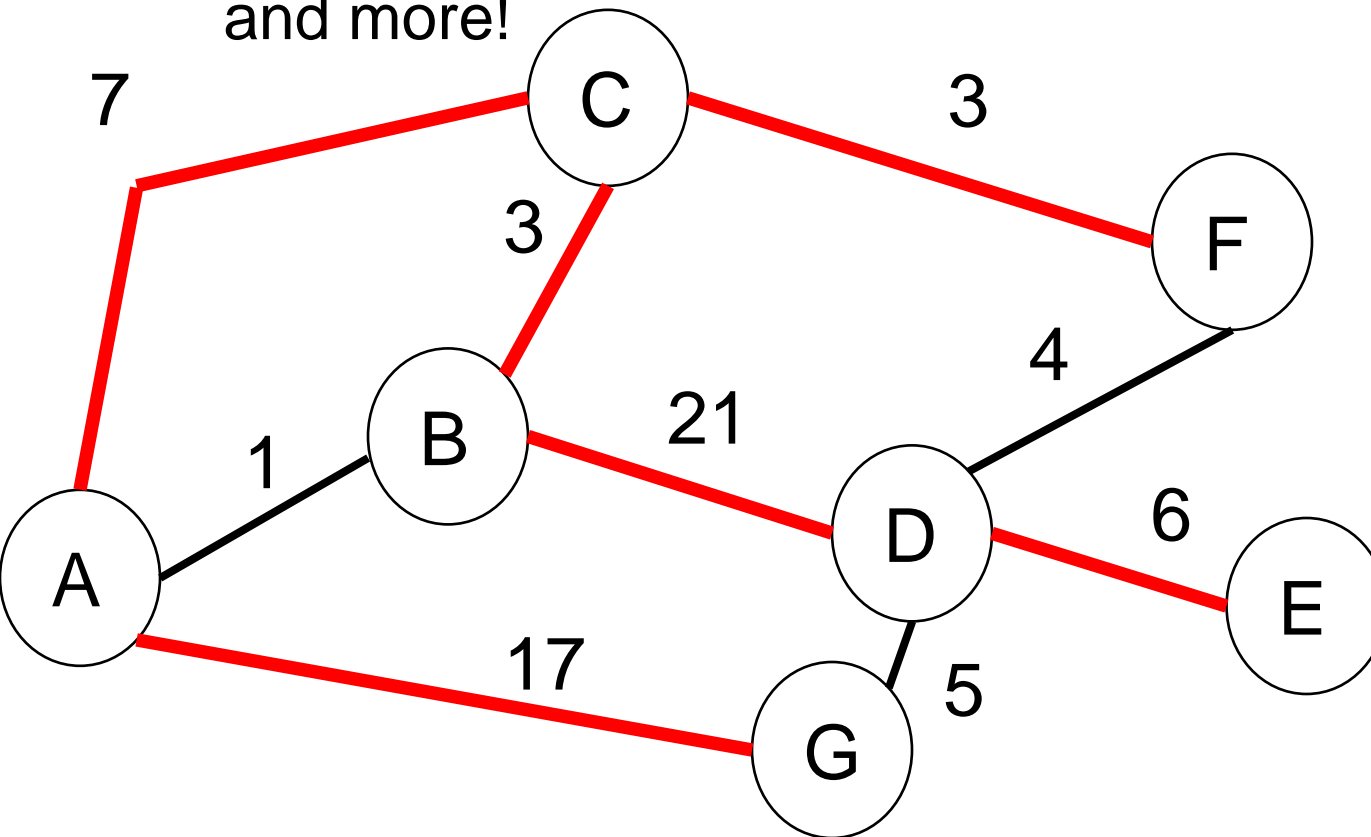
- ▶ *Spanning Tree*: A tree of edges that connects all the vertices in a graph



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?

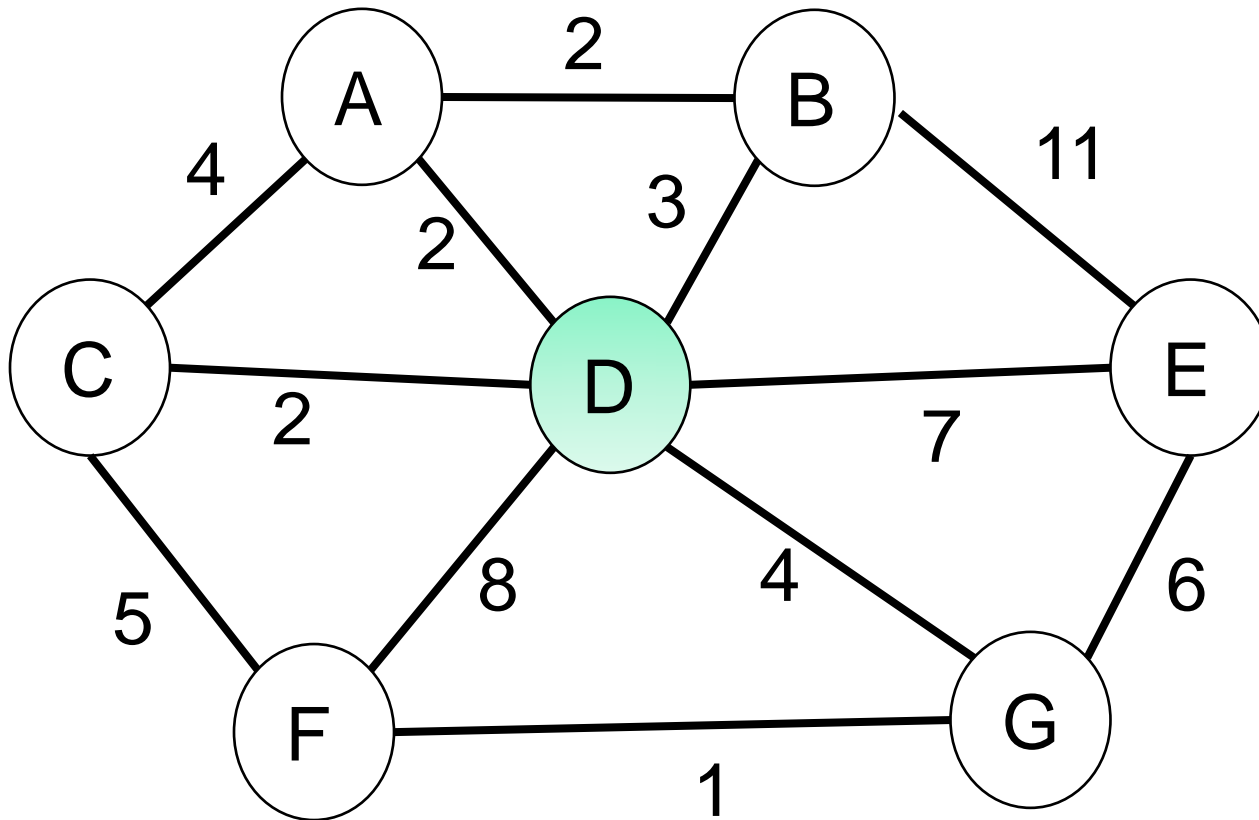
- A. 6
- B. 7
- C. 29
- D. 61
- E. None of These

Is it a MST for the graph?

Prim's Algorithm

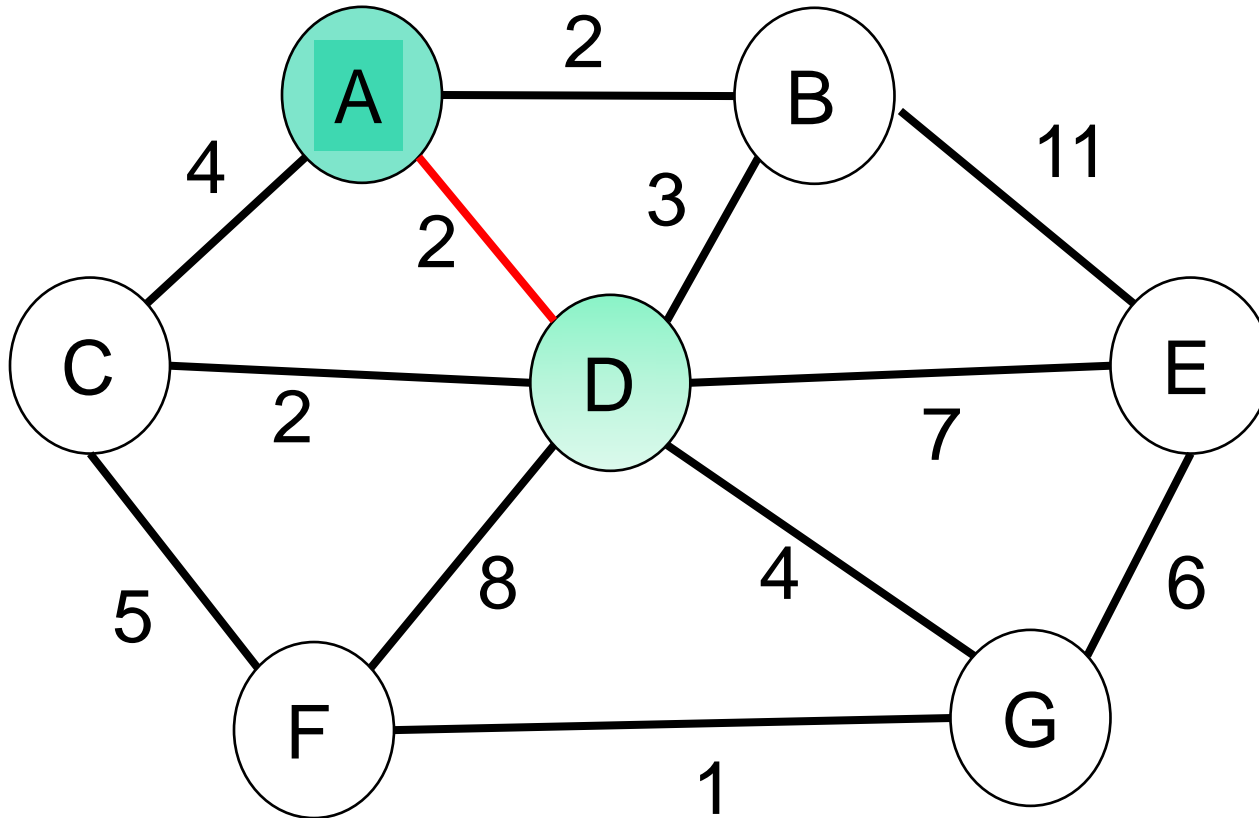
- ▶ Initially created by Vojtěch Jarník
- ▶ Rediscovered by Prim (of Sweetwater, TX) and Dijkstra
- ▶ Pick a vertex arbitrarily from graph
 - In other words, it doesn't matter which one
- ▶ Add lowest cost edge between the tree and a vertex that is not part of the tree UNTIL every vertex is part of the tree
- ▶ Greedy Algorithm, very similar to Dijkstra's

Prim's Algorithm



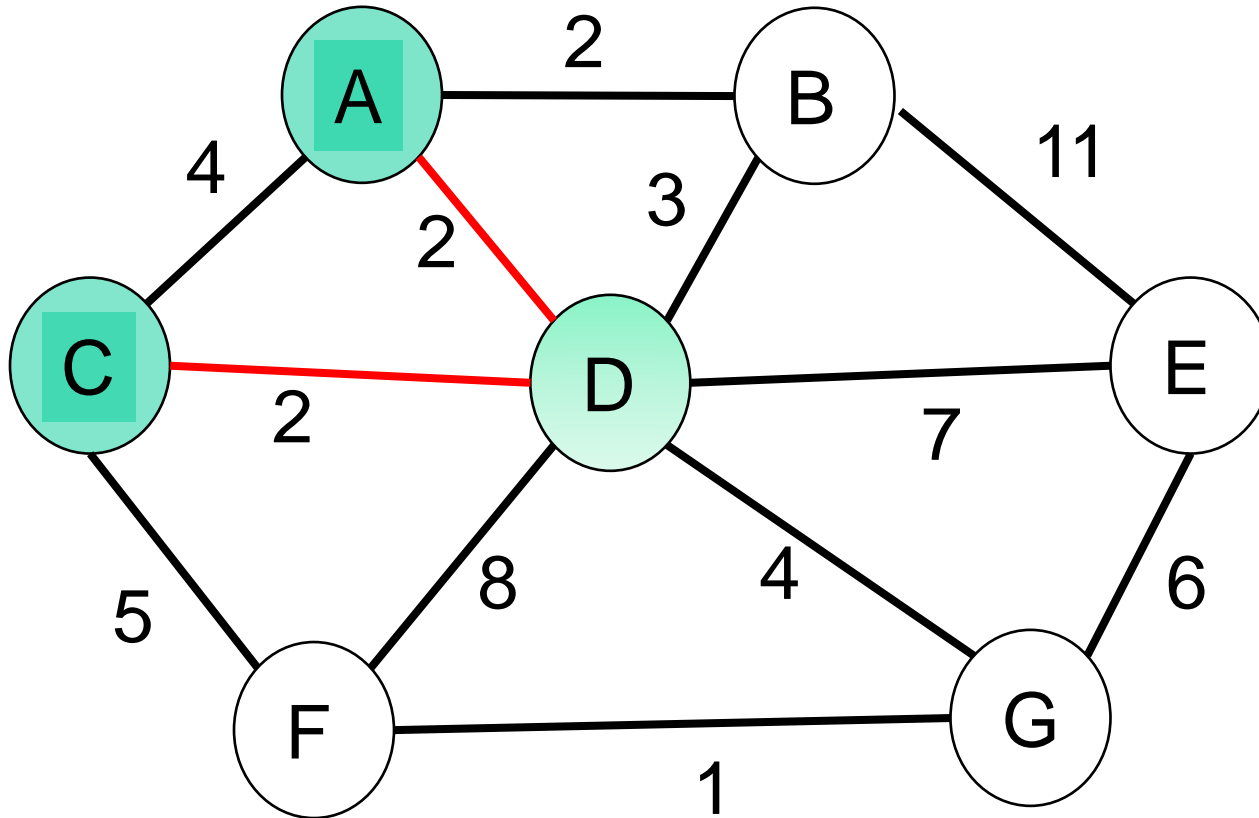
Pick D as root

Prim's Algorithm



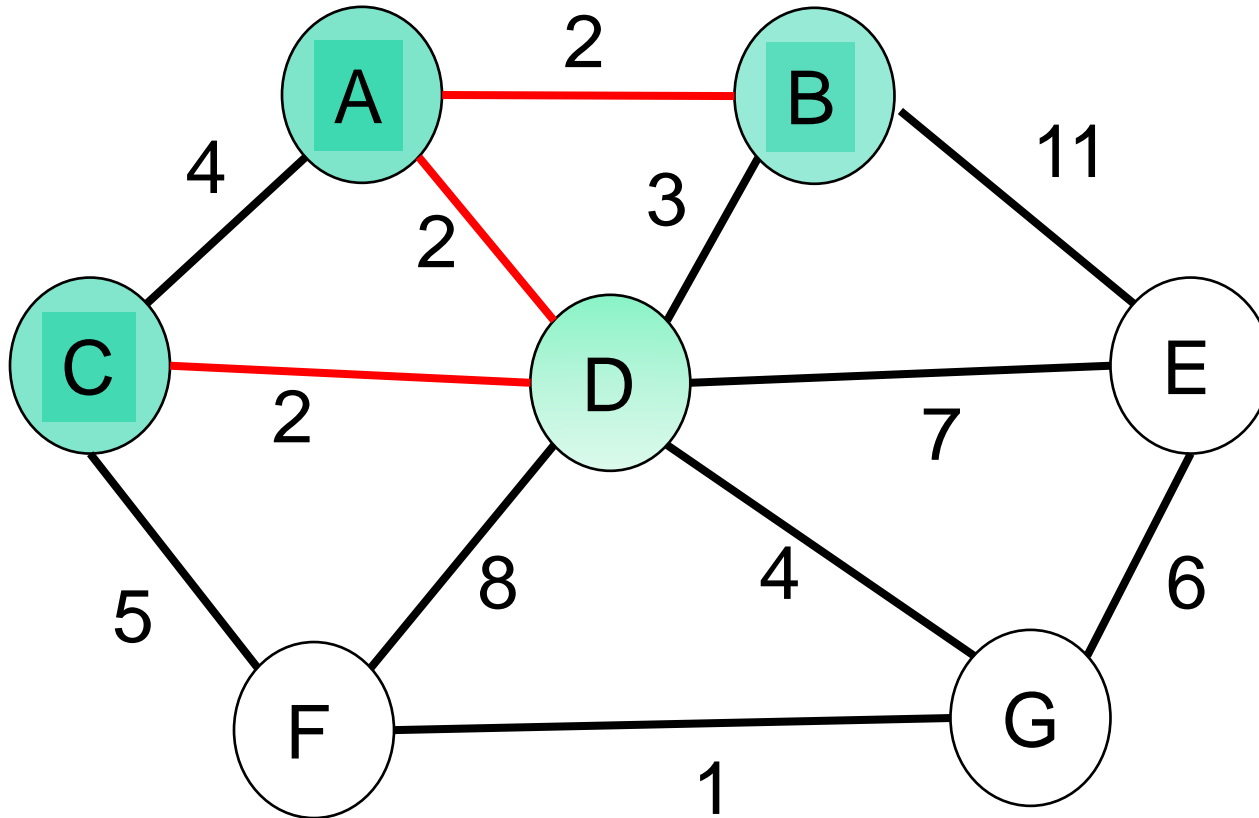
Lowest cost edge from tree to vertex not in Tree?
2 from D to A (or C)

Prim's Algorithm



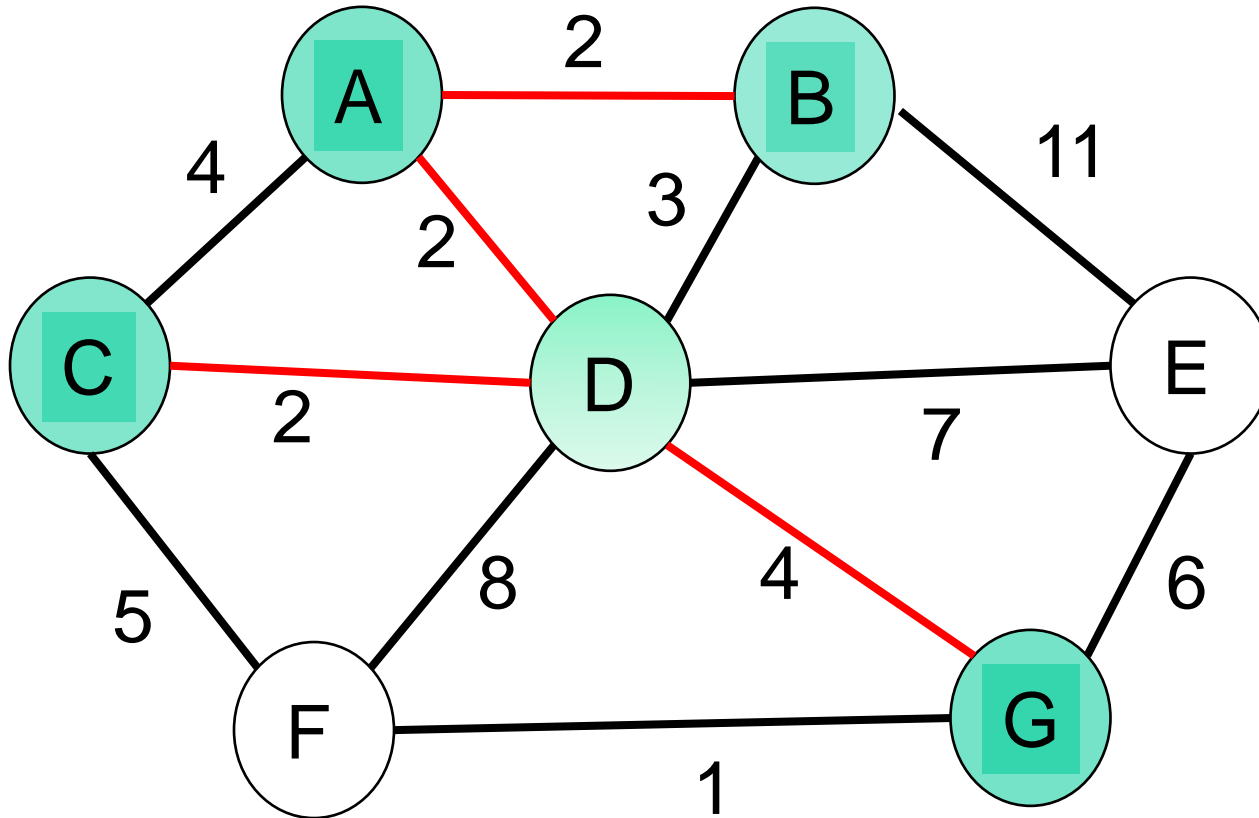
Lowest cost edge from tree to vertex not in Tree?
2 from D to C (OR from A to B)

Prim's Algorithm



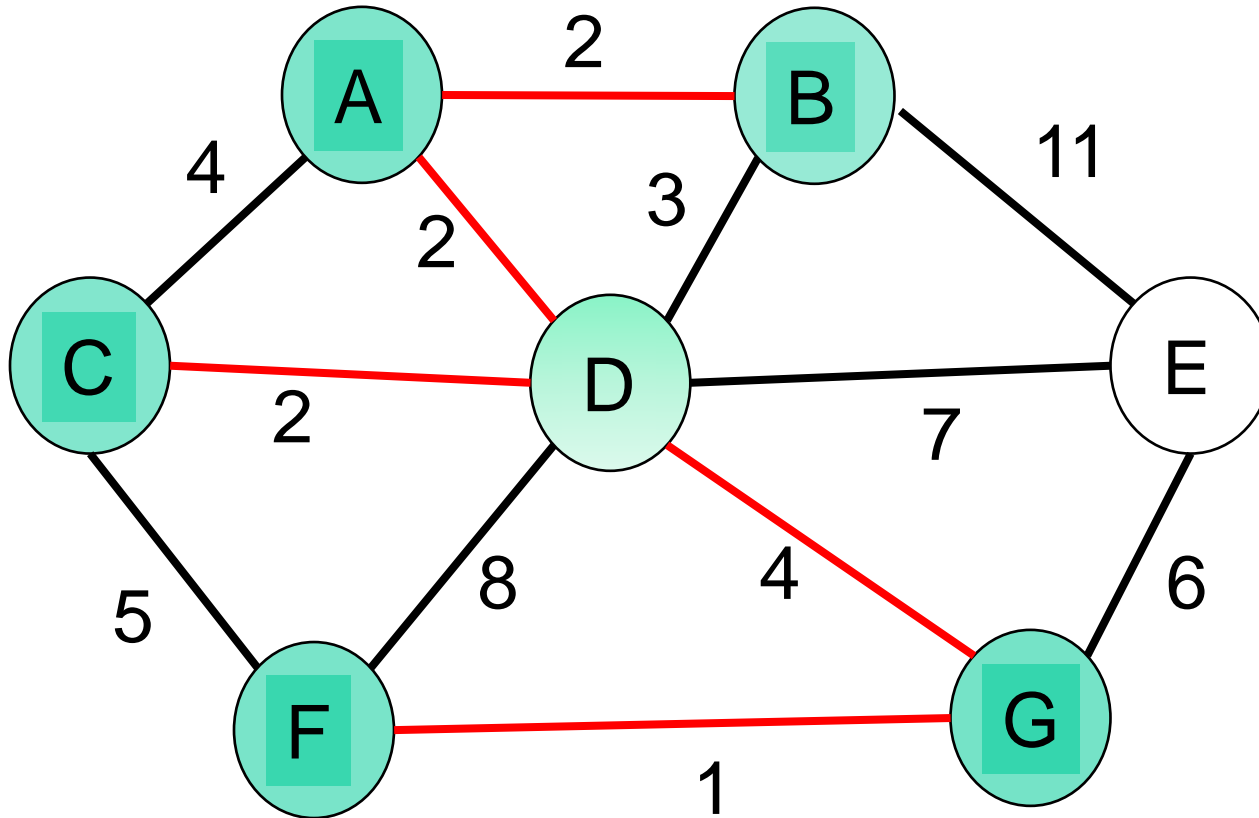
Lowest cost edge from tree to vertex not in Tree?
2 from A to B

Prim's Algorithm



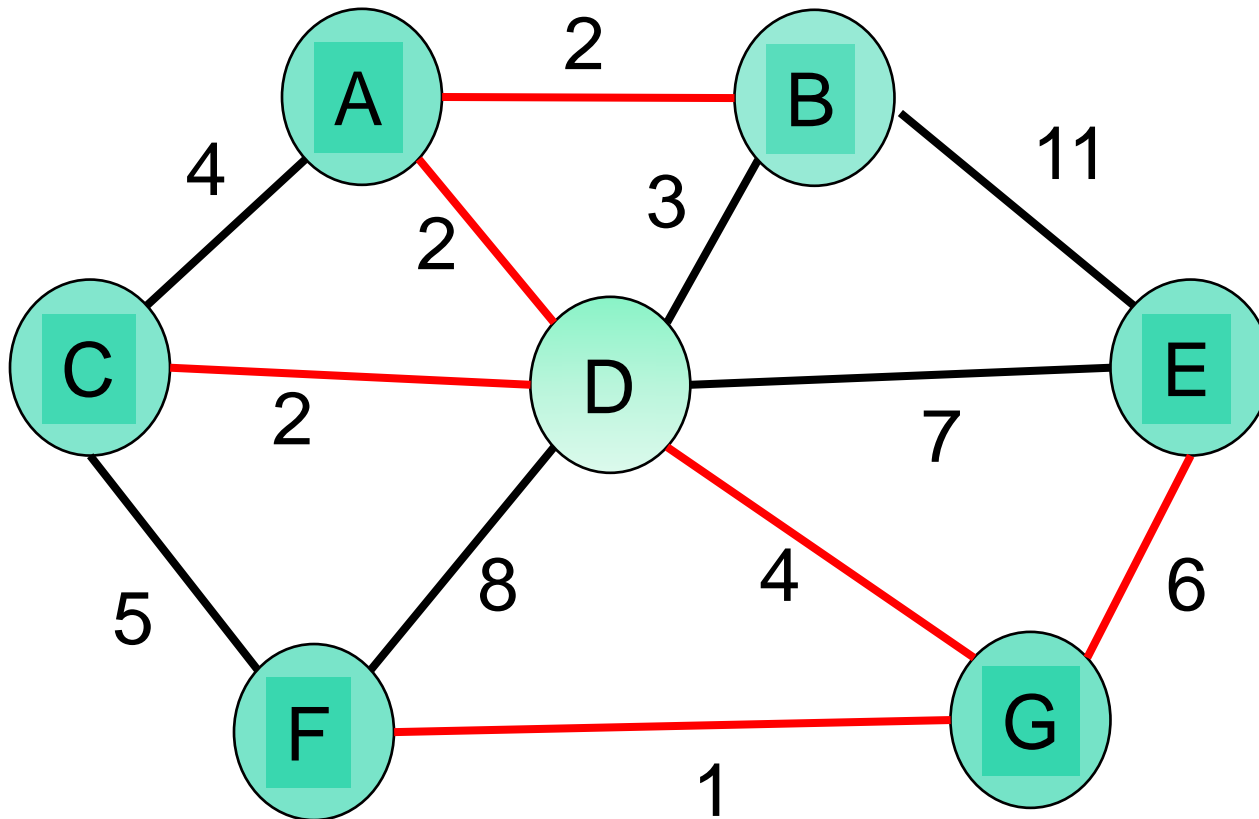
Lowest cost edge from tree to vertex not in Tree?
5 from D to G

Prim's Algorithm



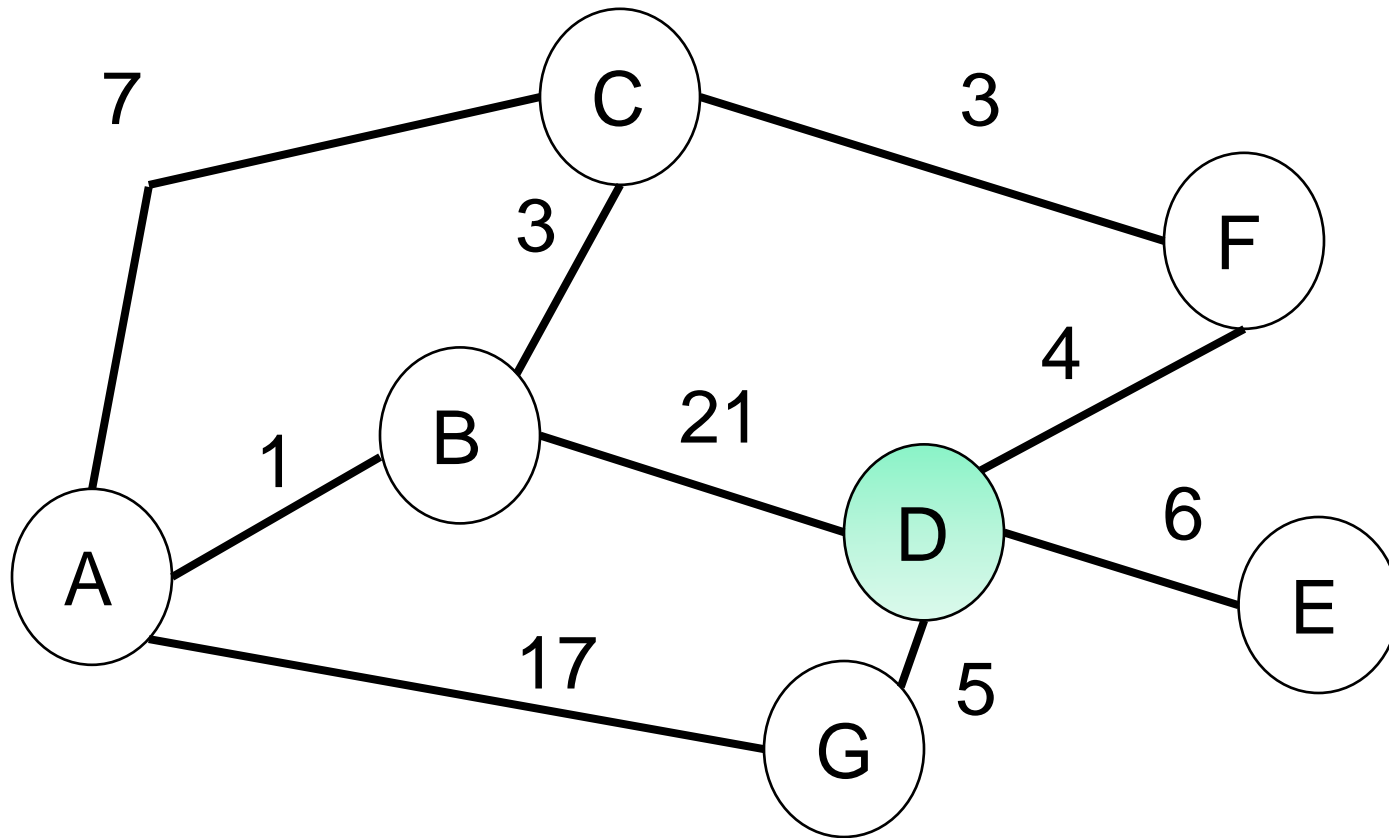
Lowest cost edge from tree to vertex not in Tree?
1 from G to F

Prim's Algorithm



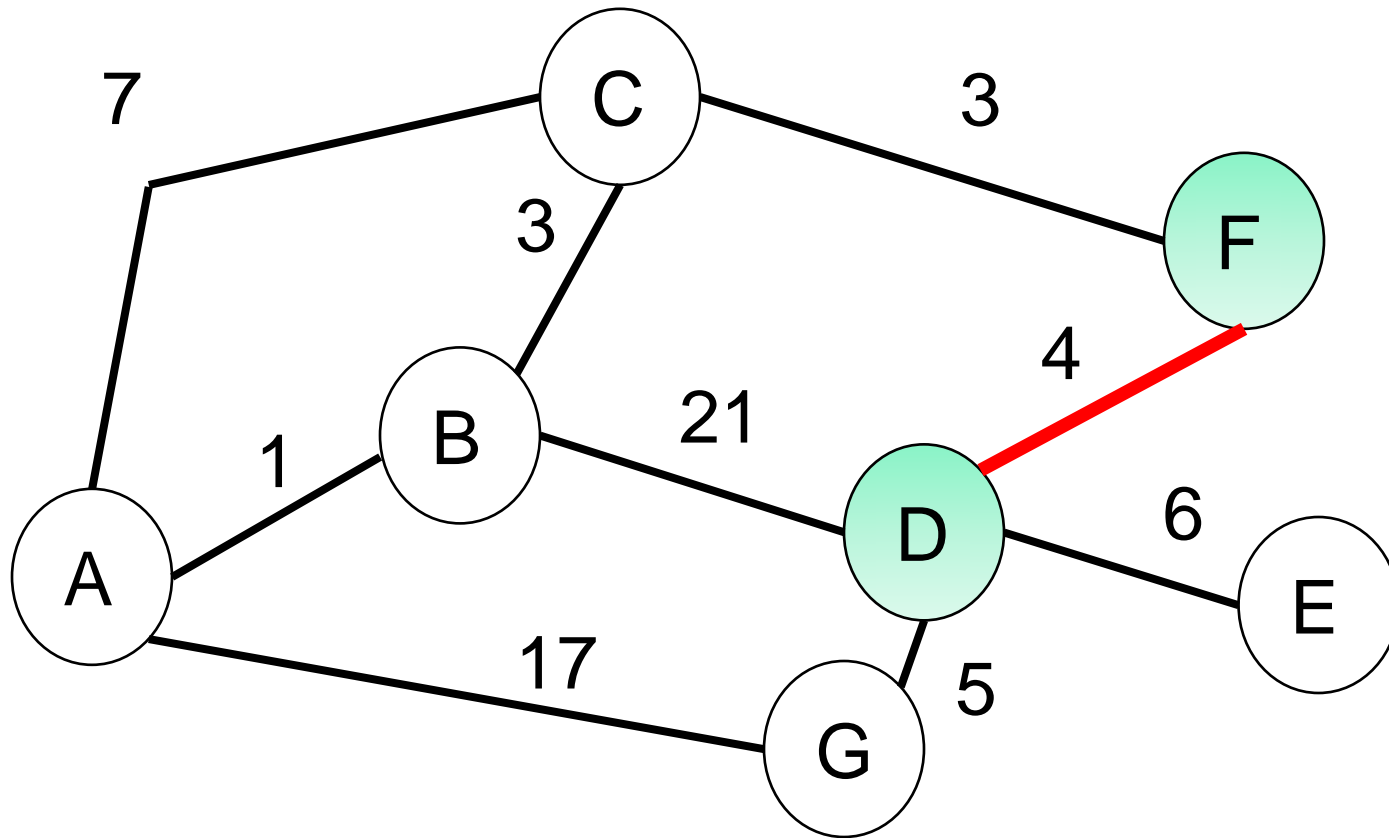
Lowest cost edge from tree to vertex not in Tree?
6 from G to E

Prim's Algorithm



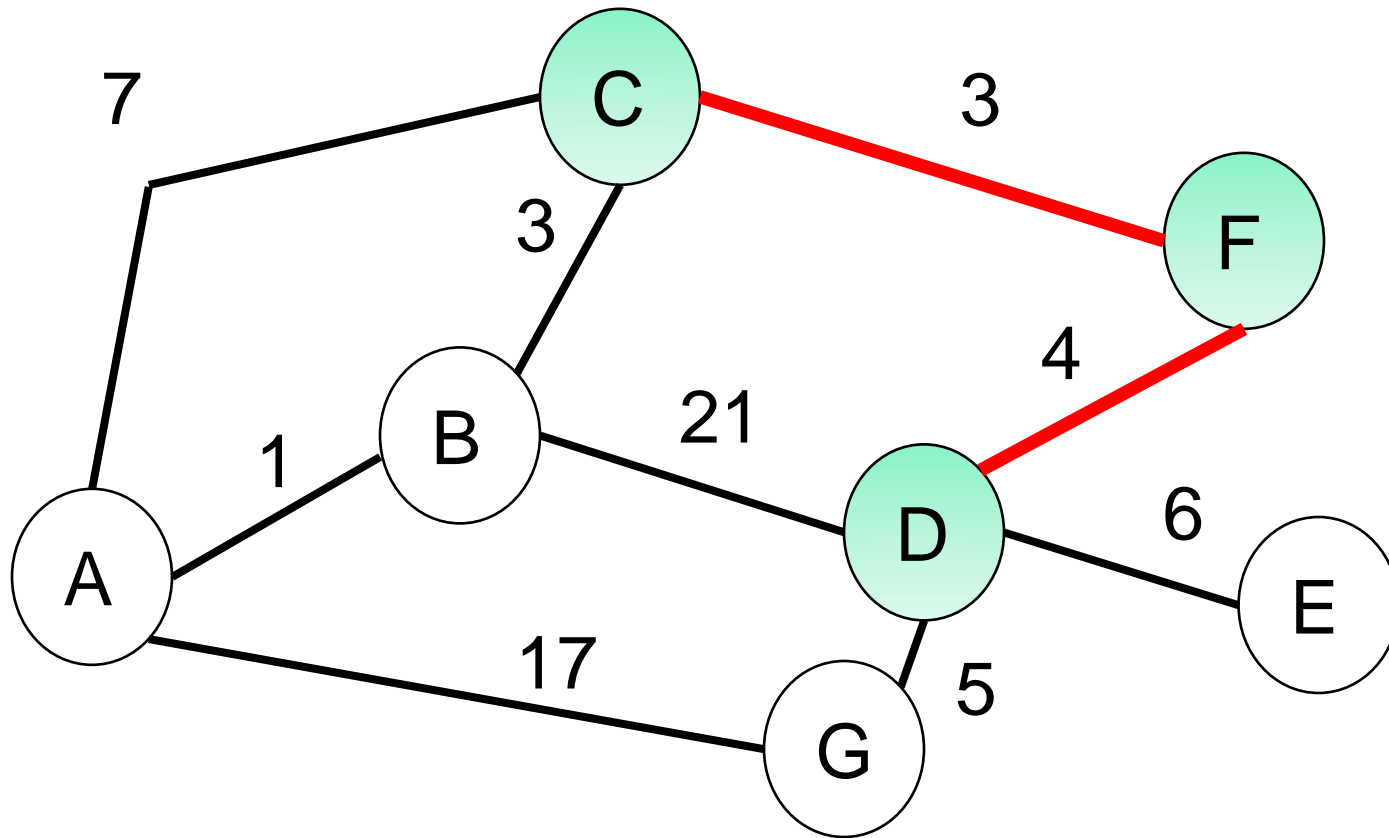
Pick D as root

Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?
4 from D to F

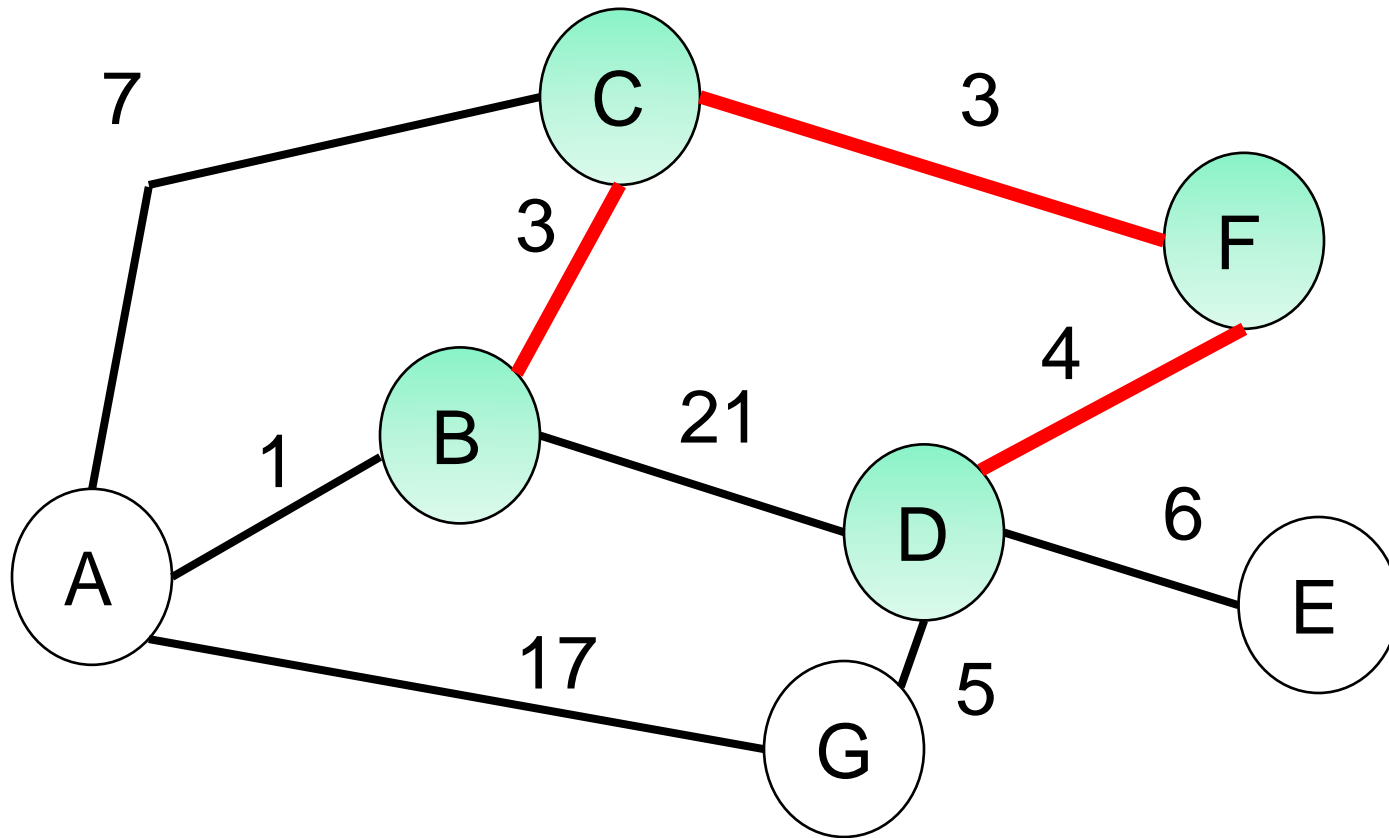
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

3 from F to C

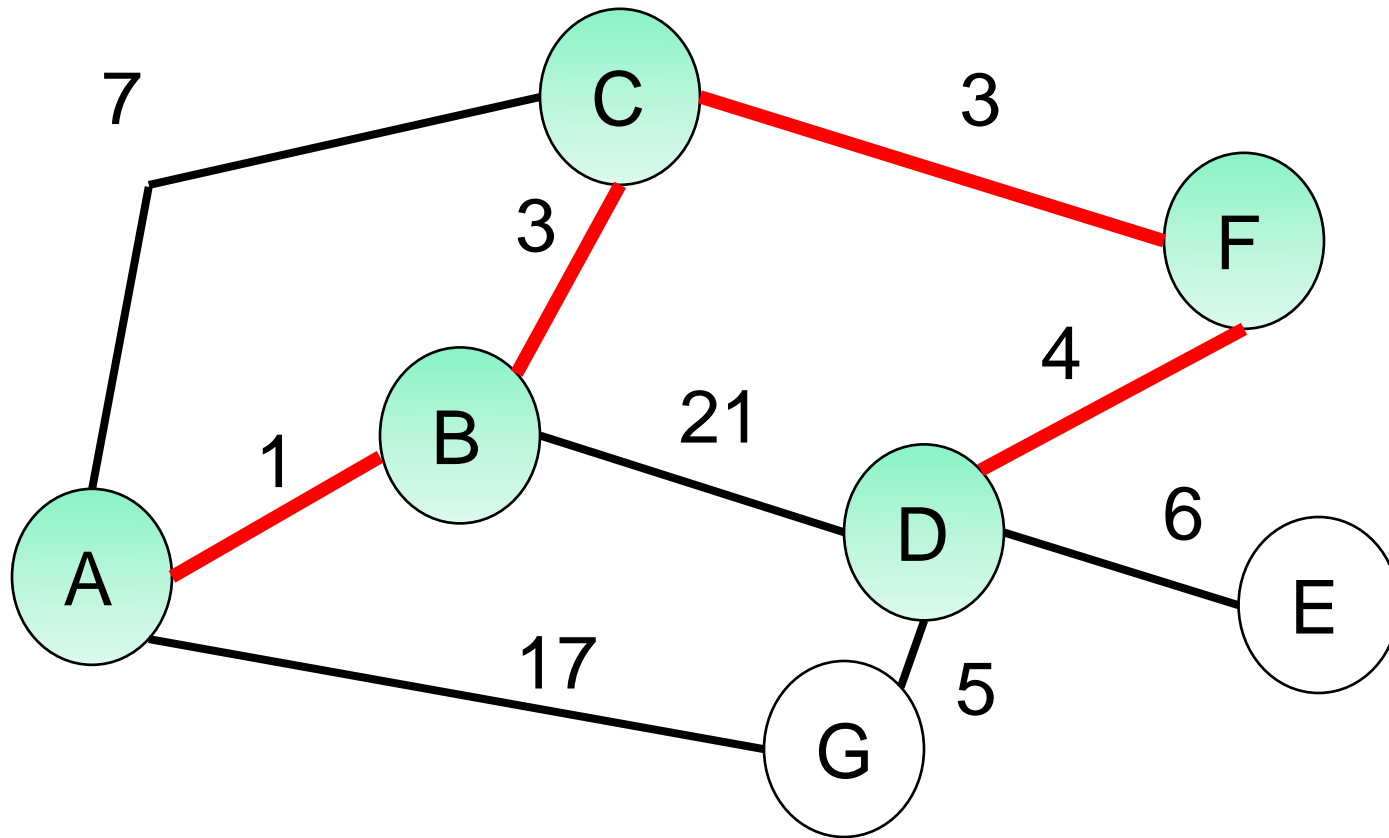
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

3 from C to B

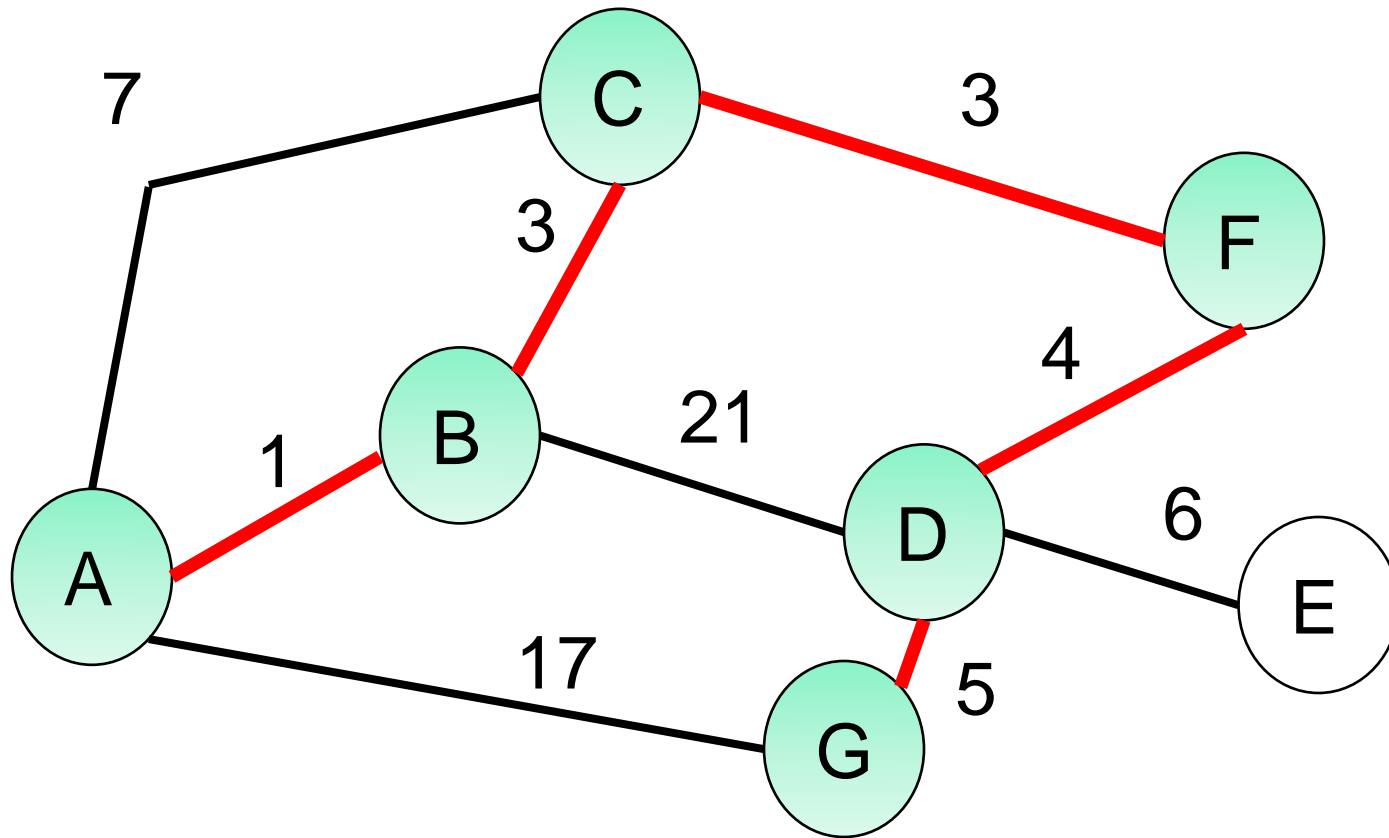
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

1 from B to A

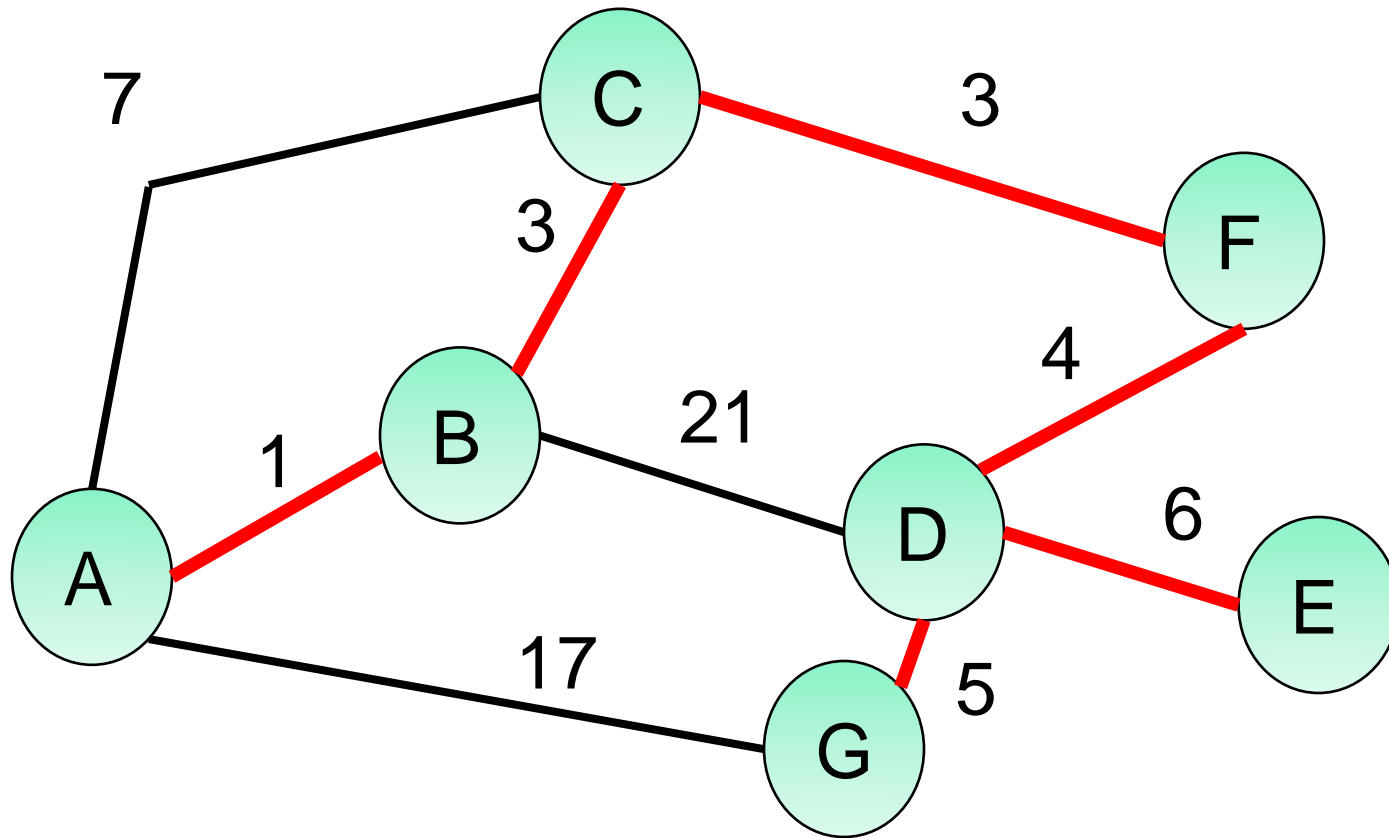
Prim's Algorithm



Lowest cost edge from tree to vertex not in Tree?

5 from D to G

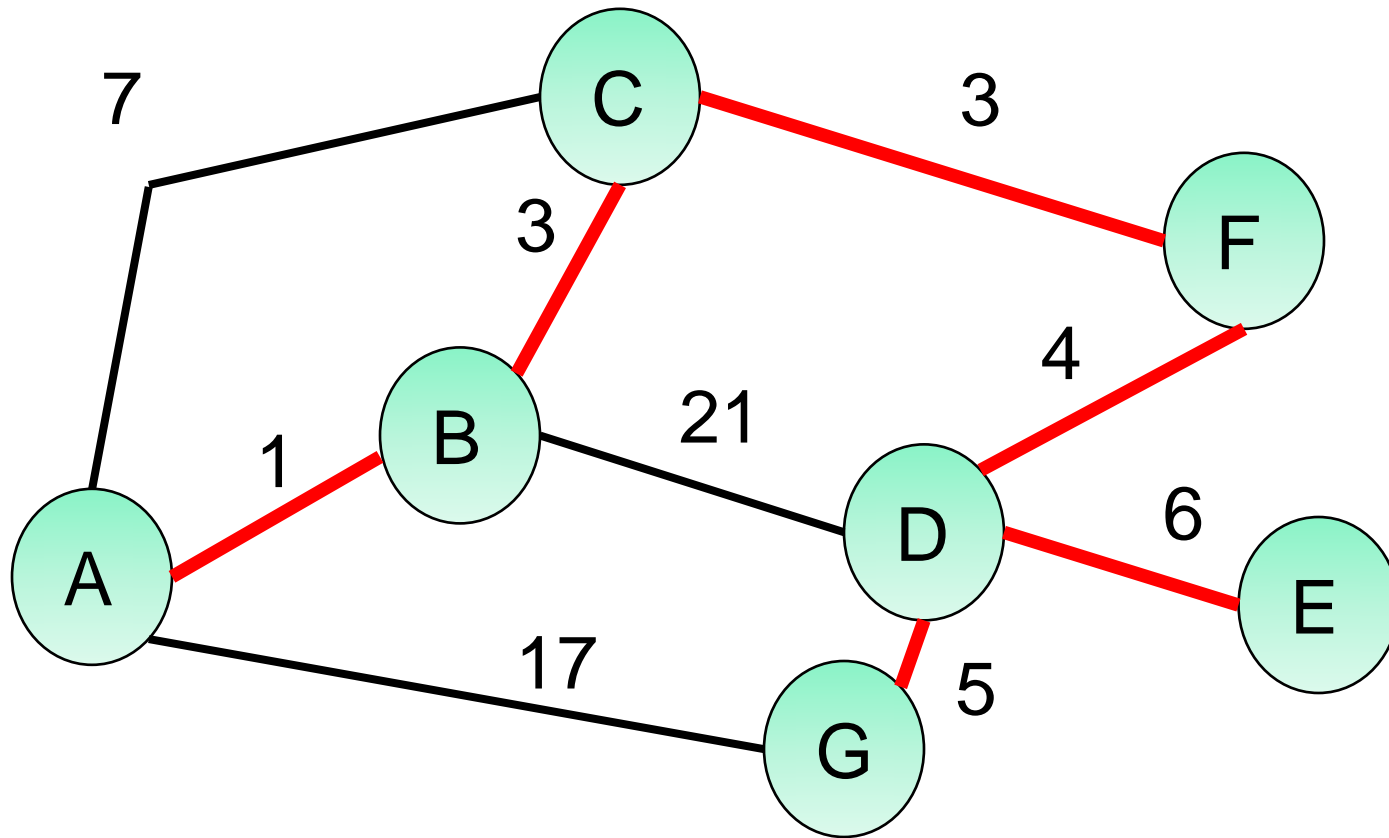
Prim's Algorithm



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