SCHOOL OF ENGINEERING

VANDERBILT UNIVERSITY

CS 3250
Algorithms
Prim's Algorithm

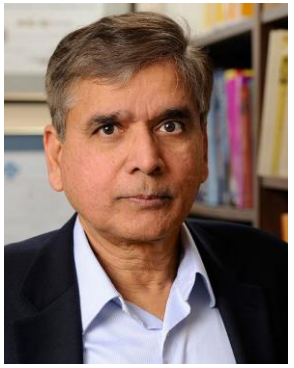
Announcements

- **HW3 will be split into (2) parts.**
  - One part will be due before the first exam.
  - One part will be due after the first exam.
- **HW3 Stop (HW3 – Part 1)**
  - You may work with a partner from this section.
  - You will take a quiz on Graphs in Brightspace. You can see the questions in advance, discuss them with your partner and go back and answer them.
  - You must list your partner's name on the quiz since only one of you "fills in" the quiz answers.
  - **Due date:** HW3 Part 1 (aka "STOP" HW) due by Wednesday, February 14th at 9AM.

VANDERBILT UNIVERSITY

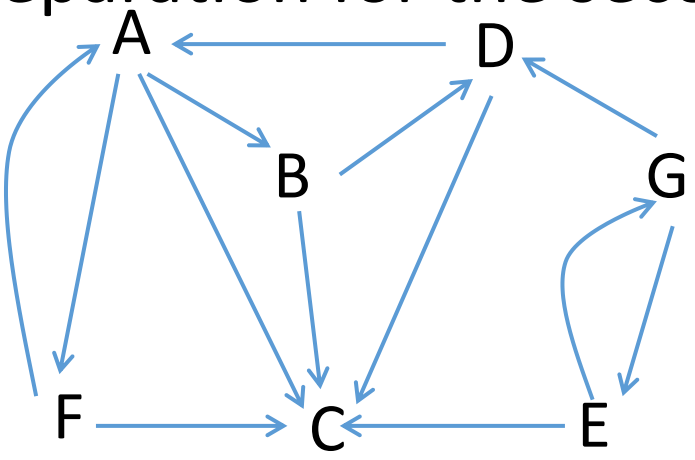
Recap: Directed Graphs and SCCs

- **Rao "Magic" Kosaraju.** Kosaraju's Algorithm is an amazingly subtle algorithm based on DFS that computes the SCCs in a directed graph G.
- Here's how it works…
  1. Run DFS on G.
  2. Compute the transpose or reverse of G, namely G^{rev}. This is the graph G with all arcs reversed.
  3. Run DFS on G^{rev}. That's it!
  4. Total = $\Theta(V+E)$



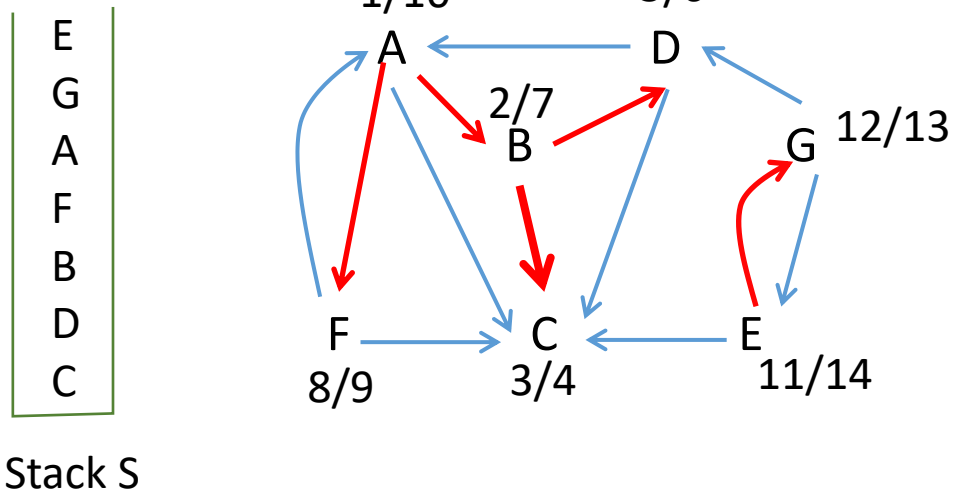
Recap: Kosaraju's Algorithm for SCCs

- Let's do a walkthrough of Kosaraju's algorithm on the DFS below.
- **Step 1:** Perform DFS starting at A. For practice, let's note the entry/exit time for each vertex using 1 as our start time. Use lexicographic ordering.
  1. When a vertex is **finished** processing push it onto a stack S in preparation for the second DFS pass.



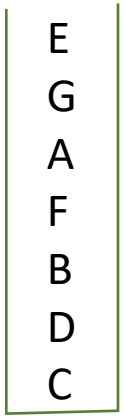
Recap: Kosaraju's Algorithm for SCCs

- After Step 1, we have the following (tree edges are denoted in red):

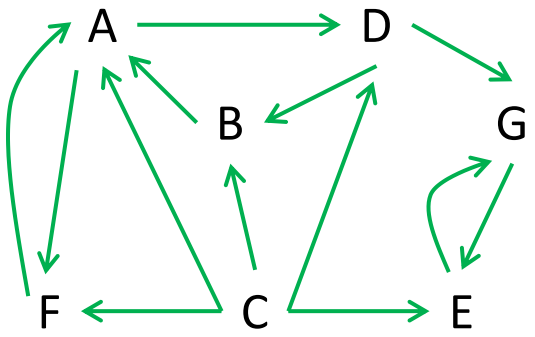


Recap: Kosaraju's Algorithm and SCCs

- **Step 2:**
  - Compute the transpose/reverse of the graph G, call it G^T or G^{rev}
  - I'll leave the details of "how" to do this to you. Reversing the pointers takes Θ(V+E) time. This part was a recent tech interview question for Yahoo.



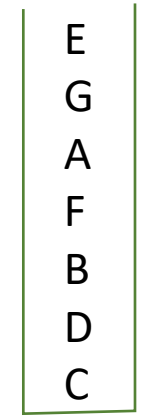
Stack S



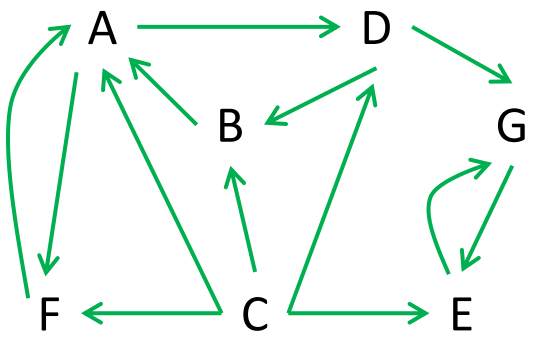
G^{rev}

Recap: Kosaraju's Algorithm and SCCs

- **Step 3:**
  - Run a new DFS on G^{rev} using any **unvisited** items on the stack to find the SCCs.
  - The vertices discovered during each DFS of a stack item belong to the same SCC.



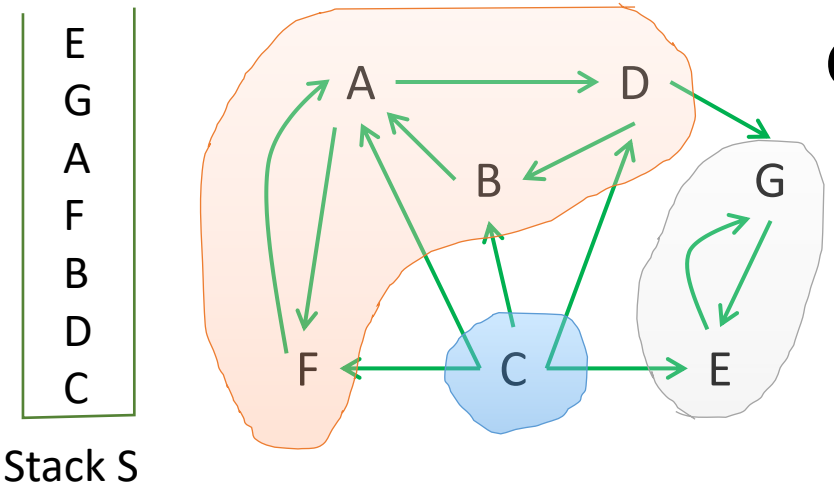
Stack S



G^{rev}

Recap: Kosaraju's Algorithm and SCCs

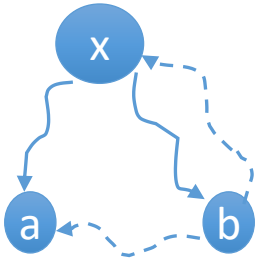
- After Step 3, we have the following:
  - The three SCCs have been correctly determined.



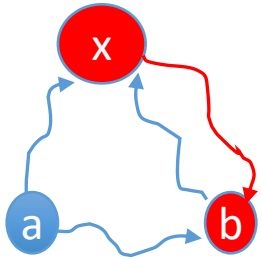
Kosaraju's Algorithm: Why Does it Work?

- **Why does Kosaraju's work?** The problem with determining the SCCs via straight DFS is that there is no control as to how we explore the graph.
- In a DFS forest of a graph G, several SCC's may be in one DFS tree.
- With the transpose G^{rev}, each tree in the DFS forest obtained from G^{rev} contains exactly one SCC.

Original graph G

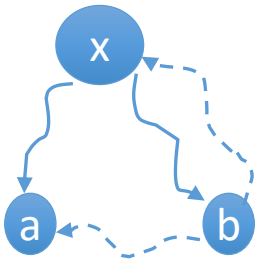


Transpose graph G_T

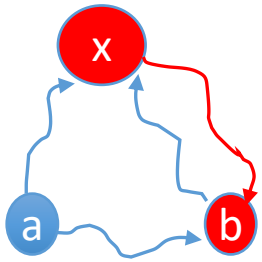


Kosaraju's Algorithm: Why Does it Work?

- In Kosaraju's, the order is enforced from our first pass using DFS. Then, each unvisited vertex that is popped off the stack becomes the root of an SCC.
  - Everything root x can reach in G^{rev} is in one SCC.
  - x was placed on the stack when finished in the original graph G.
  - In G, any vertex x can reach is below it on the stack because it finished before x.



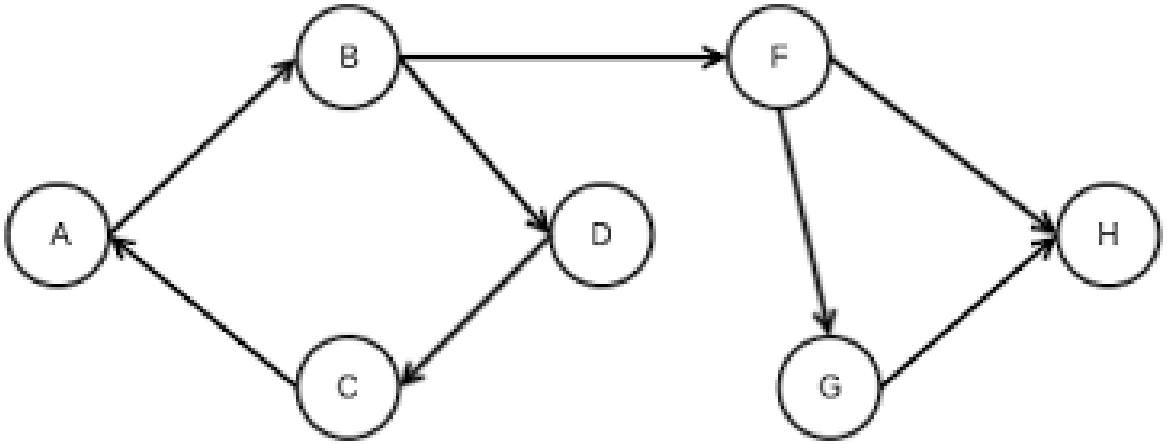
Original graph G



Transpose graph G_T

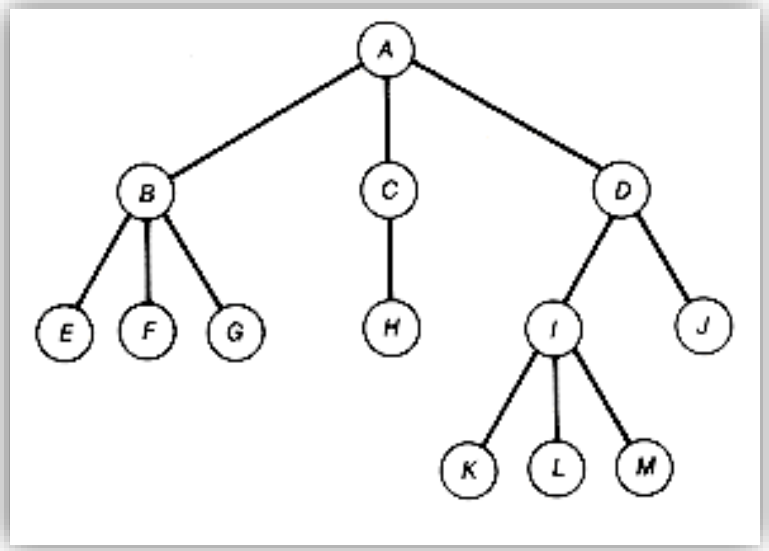
Practice: Kosaraju's Algorithm and SCCs

- **On Your Own:** Perform Kosaraju's Algorithm on the graph below and identify the SCCs of the graph. Be sure to indicate G, the stack items, G^{rev} and the SCCs.



Quick Review: Trees

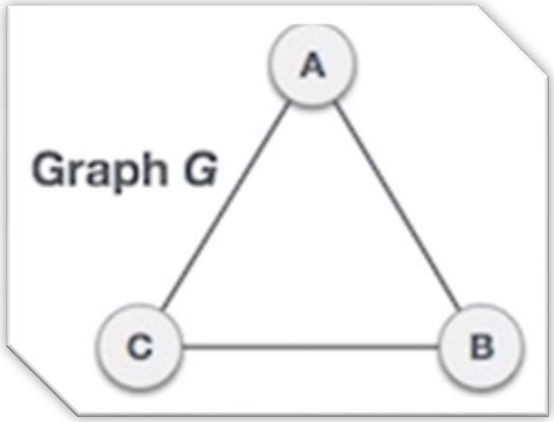
- Recall a **tree** is a connected graph without cycles.
- Graphs that are intended as trees are typically drawn like an upside-down tree.
- An **undirected** tree with n vertices or nodes will have exactly n-1 edges.
- Adding any new edge will create a cycle.



VANDERBILT UNIVERSITY

Spanning Trees

- A **spanning tree** is a subset of a graph G, which has **all** the vertices covered with a **minimum** number of edges.
- A **spanning tree** must be connected and acyclic.
  - Prof. Obvious says, yeah - if it's a tree, by definition, it must be acyclic.



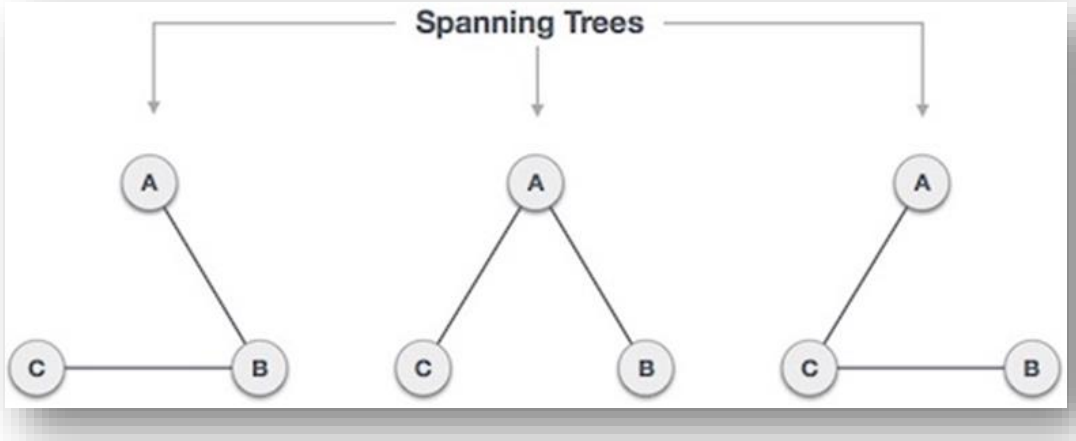
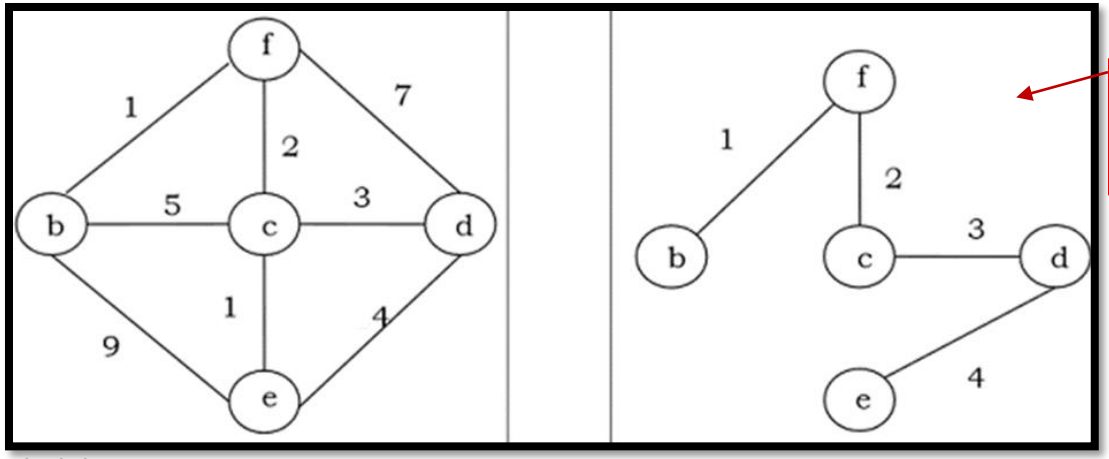


Image courtesy of TutorialsPoint.com

Minimal Spanning Tree (MST)

- A **minimal spanning tree (MST)** is a spanning tree that minimizes the sum of weights on the edges of all spanning trees.
- **Translation:** An MST connects all vertices together with the **minimal total weighting for its edges**.
- If the edge weights are unique, the MST will be unique. Otherwise, there can be more than one.



Not an MST
Why?

Image courtesy of TutorialsPoint.com

Minimal Spanning Tree (MST) in Action

- **Undirected** graph problem
- Suppose you want to supply a set of houses with:
  - Electric Power
  - Water
  - Telephone/Cable

- The goal would be to connect all houses with the needed utility for **minimum cost**.
- **Minimum spanning trees** can help with decision and cost analysis.

VANDERBILT UNIVERSITY

Minimal Spanning Tree (MST) in Action

Example: We need to connect all the homes in a newly constructed neighborhood with hi-speed cable access.

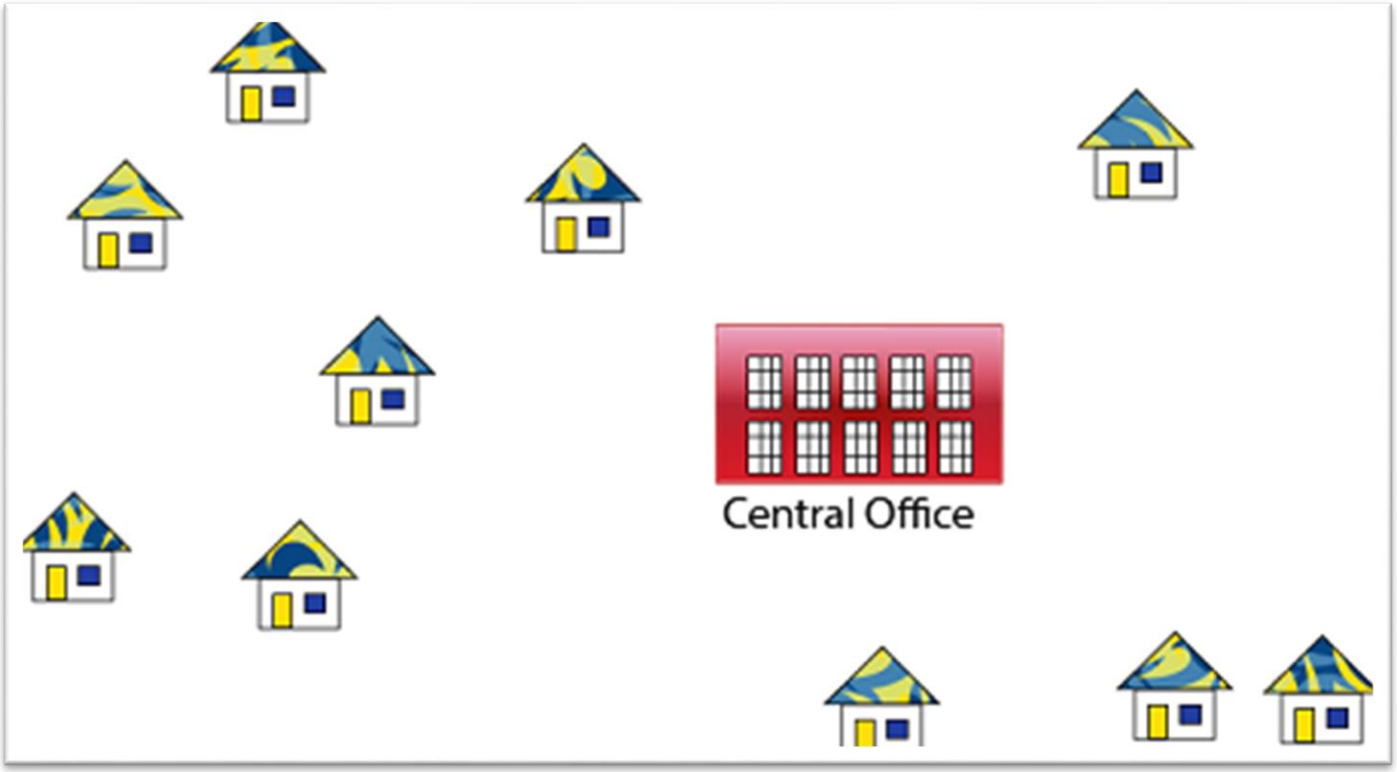


Image courtesy of JavaTPoint.com

Minimal Spanning Tree (MST) in Action

- **One solution:** A naïve approach is shown below. This is simple to design, but not necessarily the most cost effective

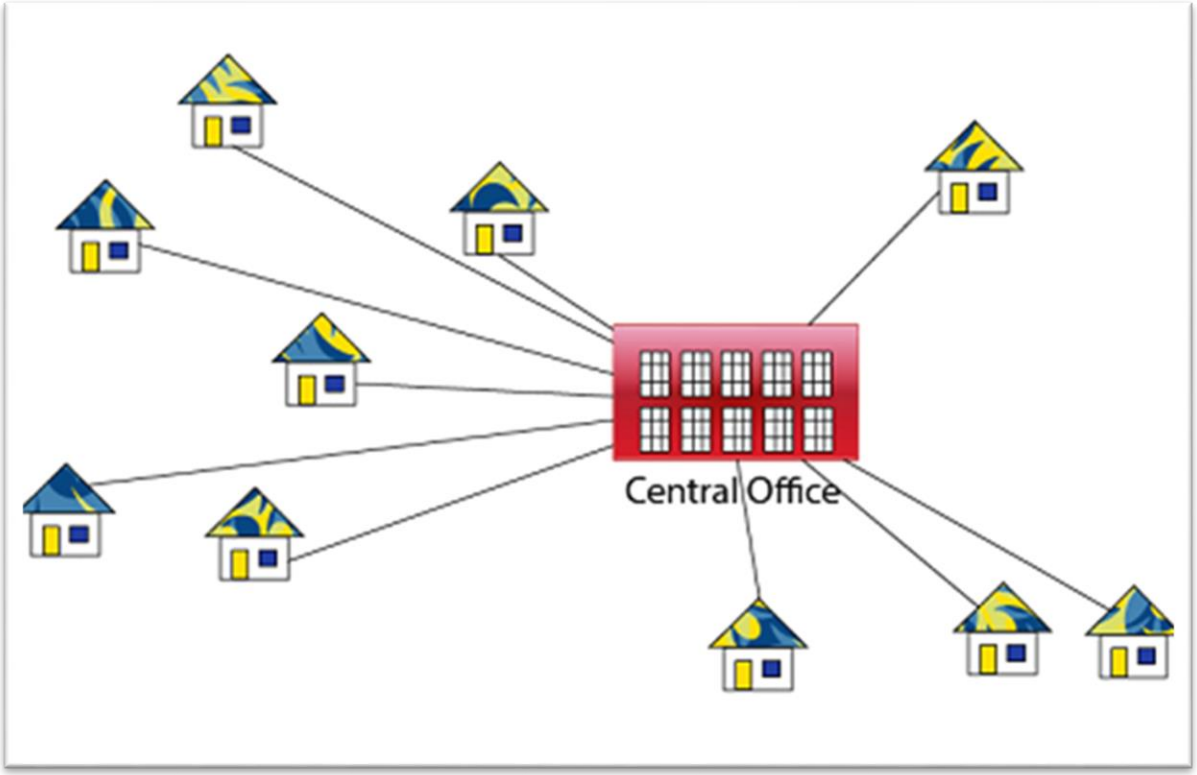


Image courtesy of JavaTPoint.com

Minimal Spanning Tree (MST) in Action

- The **MST** approach is more effective. We attempt to minimize the **total length of cable** connecting all the customers. That might look like this:

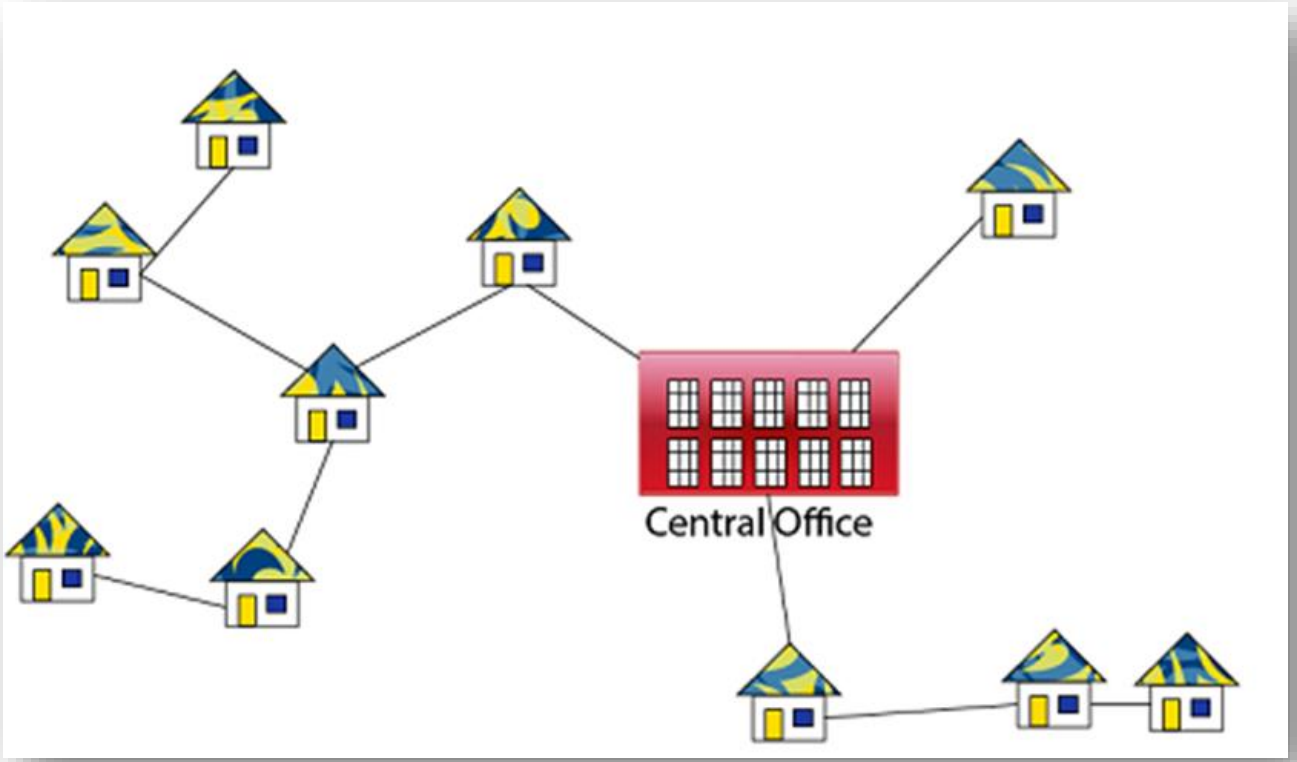


Image courtesy of JavaTPoint.com

MST Type Interview Questions…

- **Question:** You must build inter-city roads for a state. Building a road between two cities costs some amount of money. You must build roads such that the total amount spent on building connections between cities should be minimum.
- **Some clues this is an MST problem:**
  - Roads can be represented as edges
  - Cities can be represented as vertices
  - Need to connect all the cities (vertices)
  - Objective is to minimize total cost

A Proposed Algorithm for MST

- Given that MST is a problem worth solving, can we write an efficient algorithm to determine the MST?
- **Consider this algorithm:**
  1. Consider all possible spanning trees that can be created from the graph.
  2. Use a fast sort to sort the associated weights of the various spanning trees.
  3. Choose the smallest spanning tree. This is the MST.

VANDERBILT UNIVERSITY

A Proposed Algorithm for MST

- **Questions to ask:**
  - Is this a correct solution? (i.e., will it work?). **YES.**
  - Is this an efficient solution or not? How do you know?
    - Need to be able to analyze this solution and prove that we can (or can't) do any better.

A Proposed Algorithm for MST

- Whenever someone suggests a good way to solve a problem would be to consider all possible orderings that can be created from an input size of n.

VANDERBILT UNIVERSITY

A Proposed Algorithm for MST

- How much work does our proposed algorithm do?
- It's the sum of these three pieces of the algorithm.
  1. Consider all possible spanning trees: O(???)
  2. Use a smart sort to sort all spanning trees by total weight: O(nlogn)
  3. Select the smallest total from sorted list: O(1)

VANDERBILT UNIVERSITY

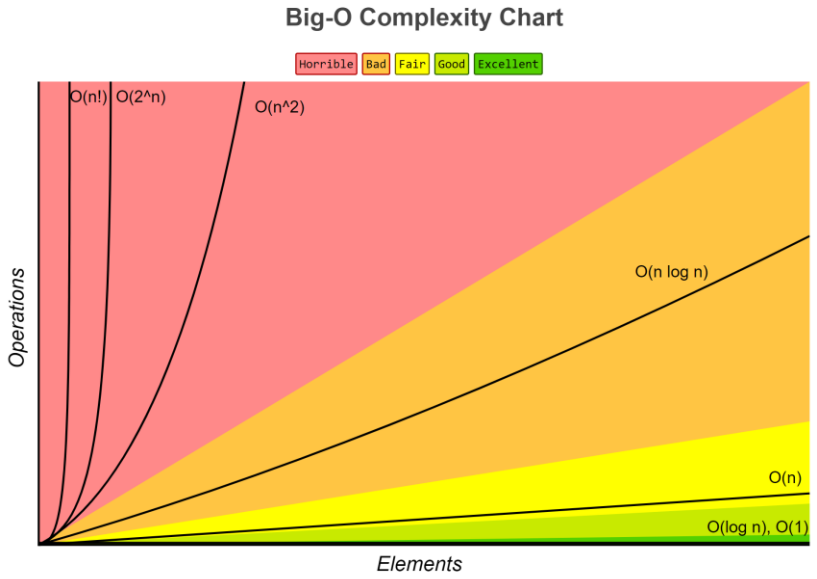
A Proposed Algorithm for MST

- **Question:** Bedha Skolar says…why are you sorting when you only need the minimum?
- Explain how you would answer Bedha.
- **Answer:**
- Since the bottleneck of the algorithm will be generating the possible trees (which is way more than nlogn, who really cares whether we spend O(n) vs. O(nlogn) to find the minimum?

VANDERBILT UNIVERSITY

A Proposed Algorithm for MST

- So, how much work to consider all spanning trees?
- Start drawing all possible spanning trees starting with a small number of vertices and edges. You will quickly realize the number of trees grows exponentially as number of edges increases.
- For a complete graph it's roughly: n^{n-2}
- $O(n^n)$ is horrible.



Prim's Algorithm for MST

- One well-known algorithm that tackles the MST problem is **Prim's algorithm.** You saw it in 2212.
  1. Pick any vertex to be the root of the tree.
  2. WHILE the tree does not contain all vertices
     - Find smallest edge in the graph for any vertex in the MST.
     - Add it to the MST.
  3. You have constructed the MST

Prim's Algorithm for MST

- **How it works:** Prim's builds a series of MSTs one vertex at a time from a graph G. The algorithm stops when all vertices are in the MST.
- A couple of assumptions to make our life easier:
  1. **The graph is connected**. If the graph is not connected, one can adapt the algorithm to compute the MSTs of each of its connected components (known as a minimum spanning **forest** – MSF).

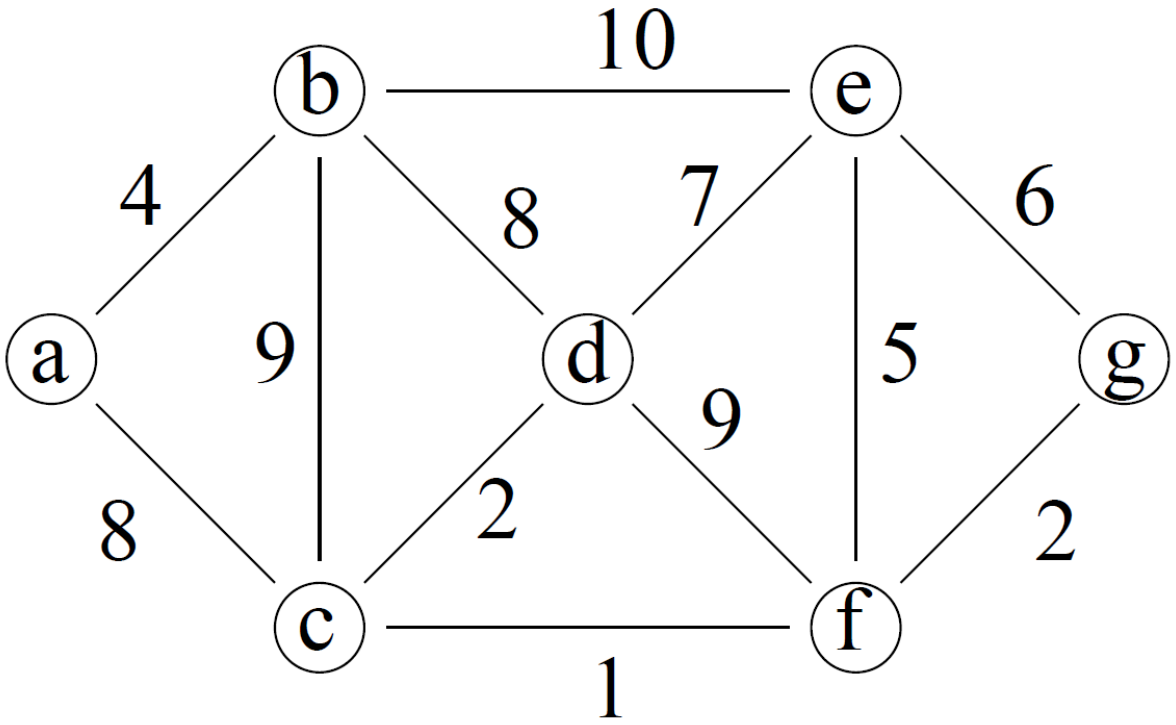
Prim's Algorithm for MST

- A couple of assumptions to make our life easier.
  2. **The edge weights are unique.** Prim's algorithm works even if the edges are not unique. But if the edges are unique, you are guaranteed a unique answer for the MST weight.

VANDERBILT UNIVERSITY

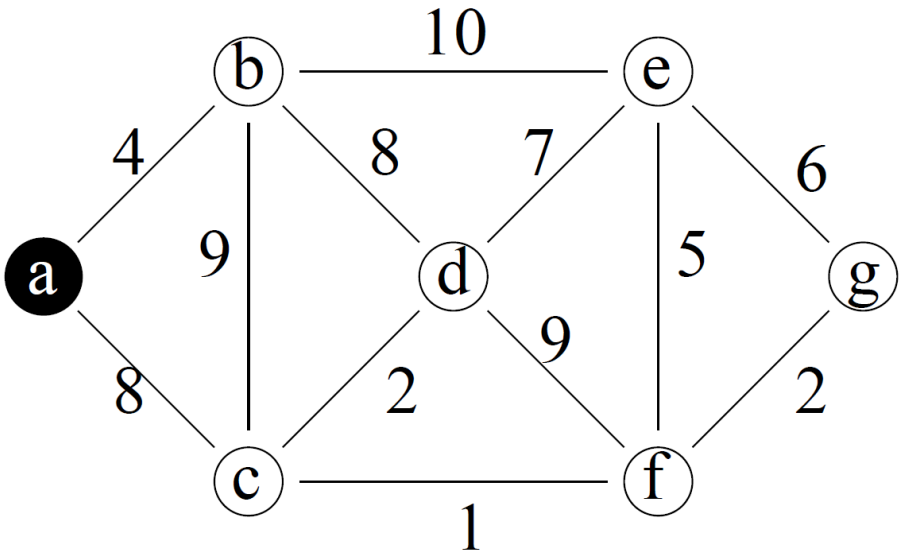
Prim's Algorithm for MST

- Let's follow Prim's algorithm to build an MST based on the graph below starting at vertex 'a' (the start choice of the starting vertex is arbitrary):



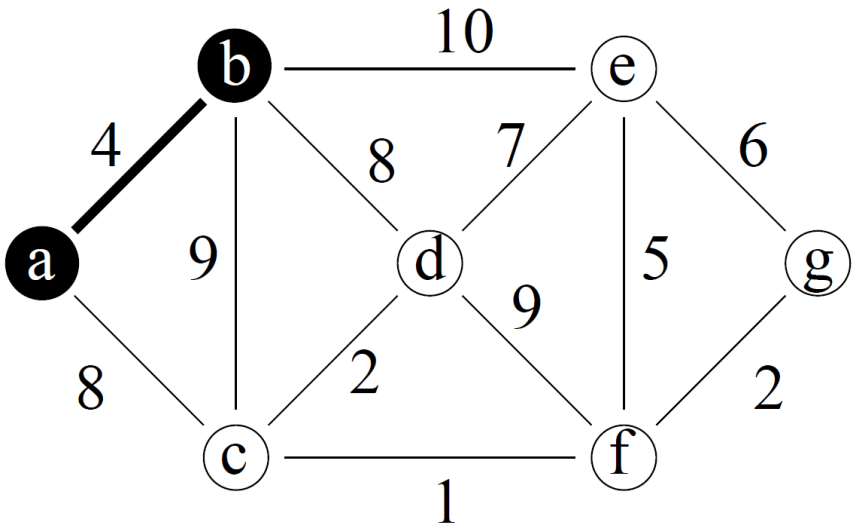
Prim's Algorithm for MST

Step 1: Choose "a" as the first vertex in our MST. Next, look around for the lightest neighbor edge of **any** vertex in the MST. This would be the edge (a, b). We choose this edge and the connecting vertex "b" to be added to the MST.



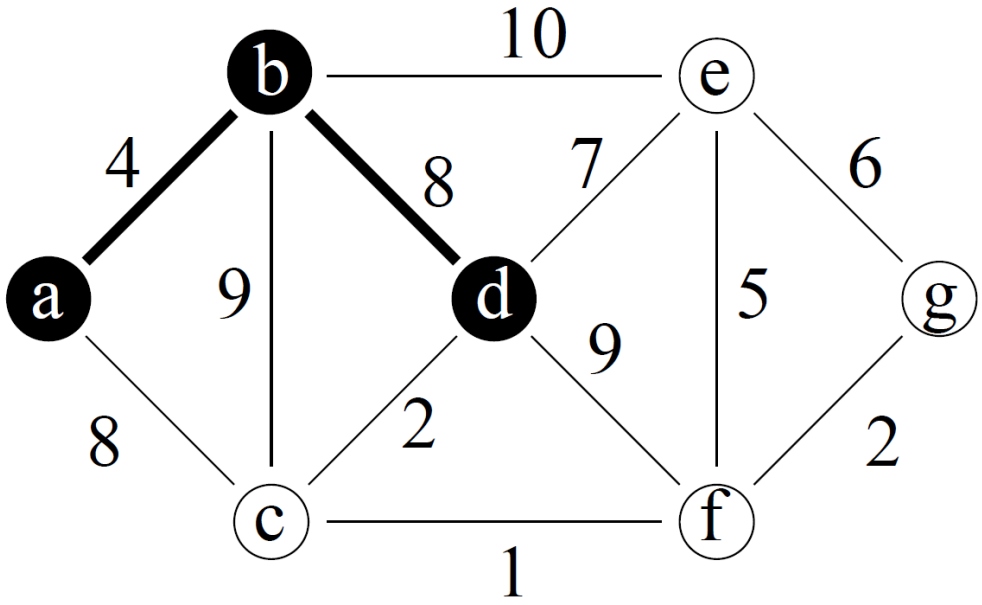
Prim's Algorithm for MST

Step 2: With "a" and "b" in our MST, we look around for the lightest neighbor edge from either vertex. Since note all edge weights are unique, this can be either (a, c) or (b, d). Let's choose (b, d) and the connecting vertex "d" to be in the MST.



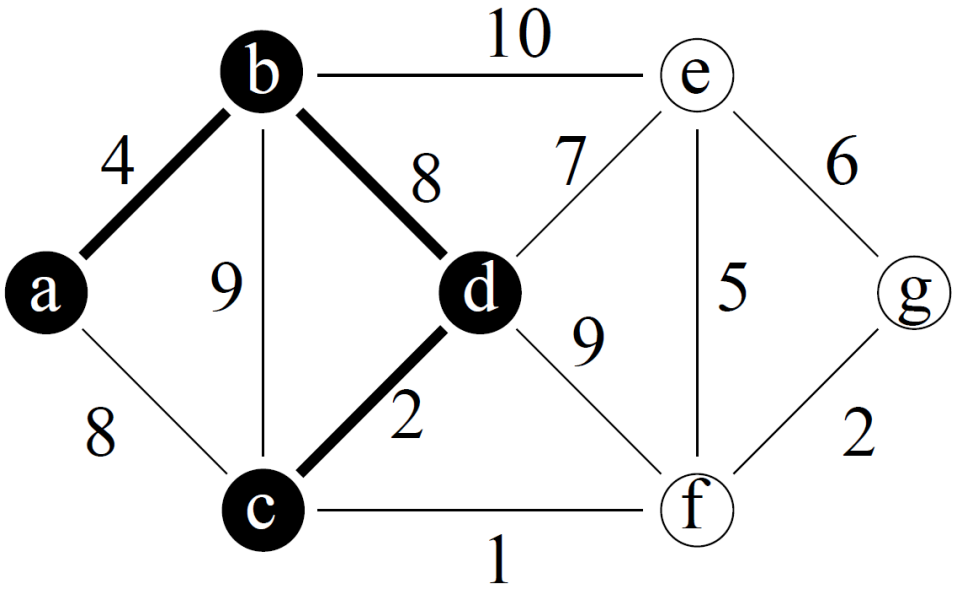
Prim's Algorithm for MST

Step 3: With vertices "a", "b" and "d" in our MST, we look around for the lightest neighbor edge from any of those vertices. This will be the edge (d, c). Let's choose (d, c) and the connecting vertex "c" to be in the MST.



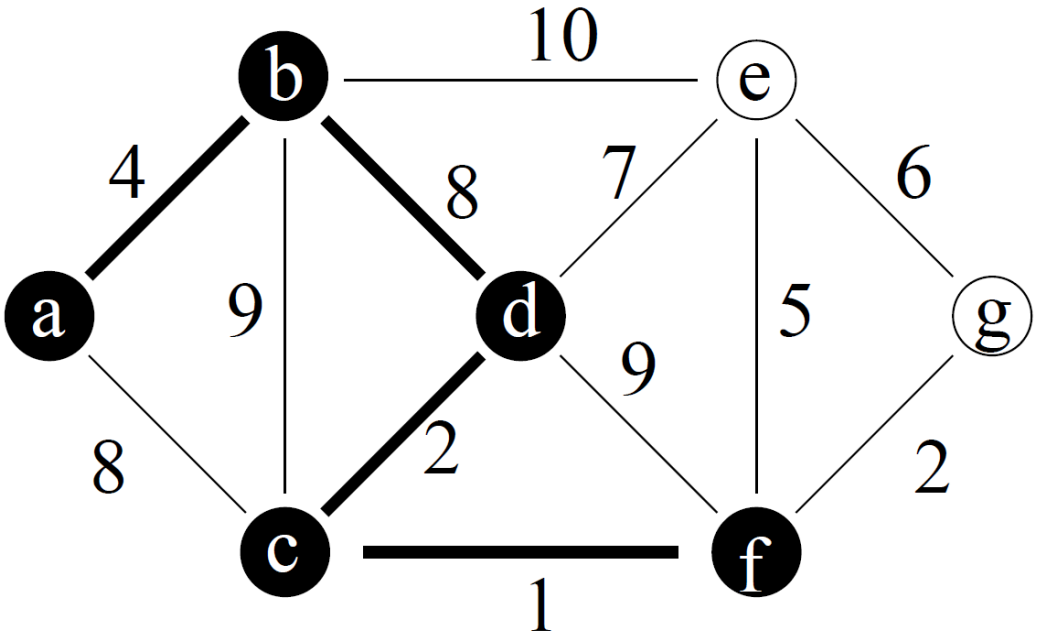
Prim's Algorithm for MST

Step 4: The algorithm continues, and we look around for the lightest neighbor edge from any vertex currently in the MST. This will be (c, f). We add it and the corresponding vertex "f" to the MST.



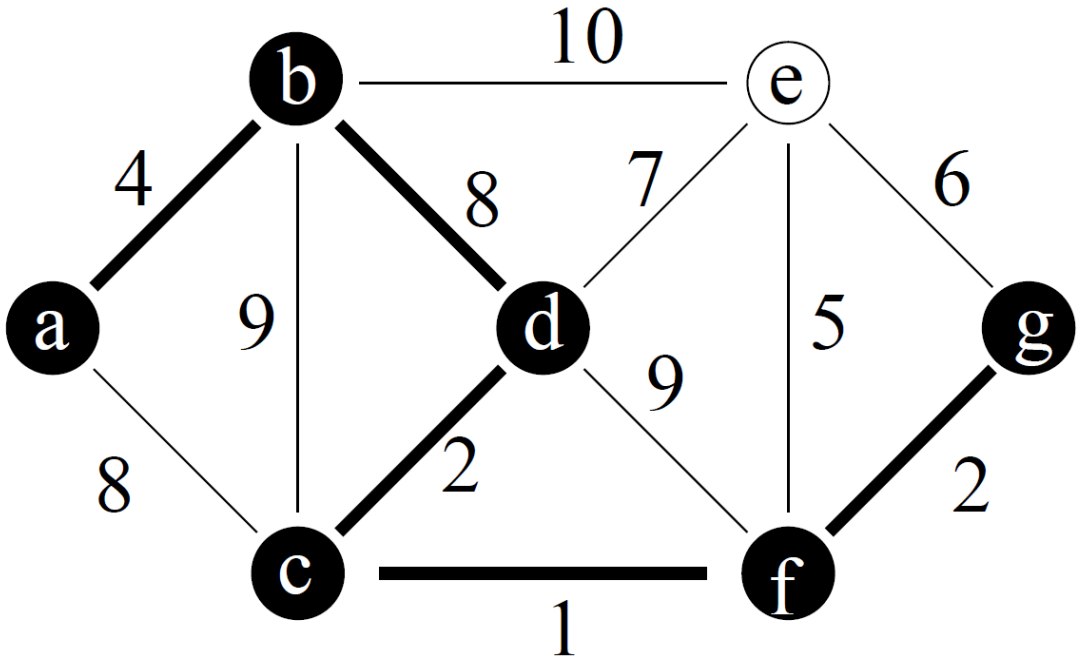
Prim's Algorithm for MST

Step 5: The algorithm continues, and we look around for the lightest neighbor edge from any vertex currently in the MST. This will be (f, g). We add it and the corresponding vertex "g" to the MST.



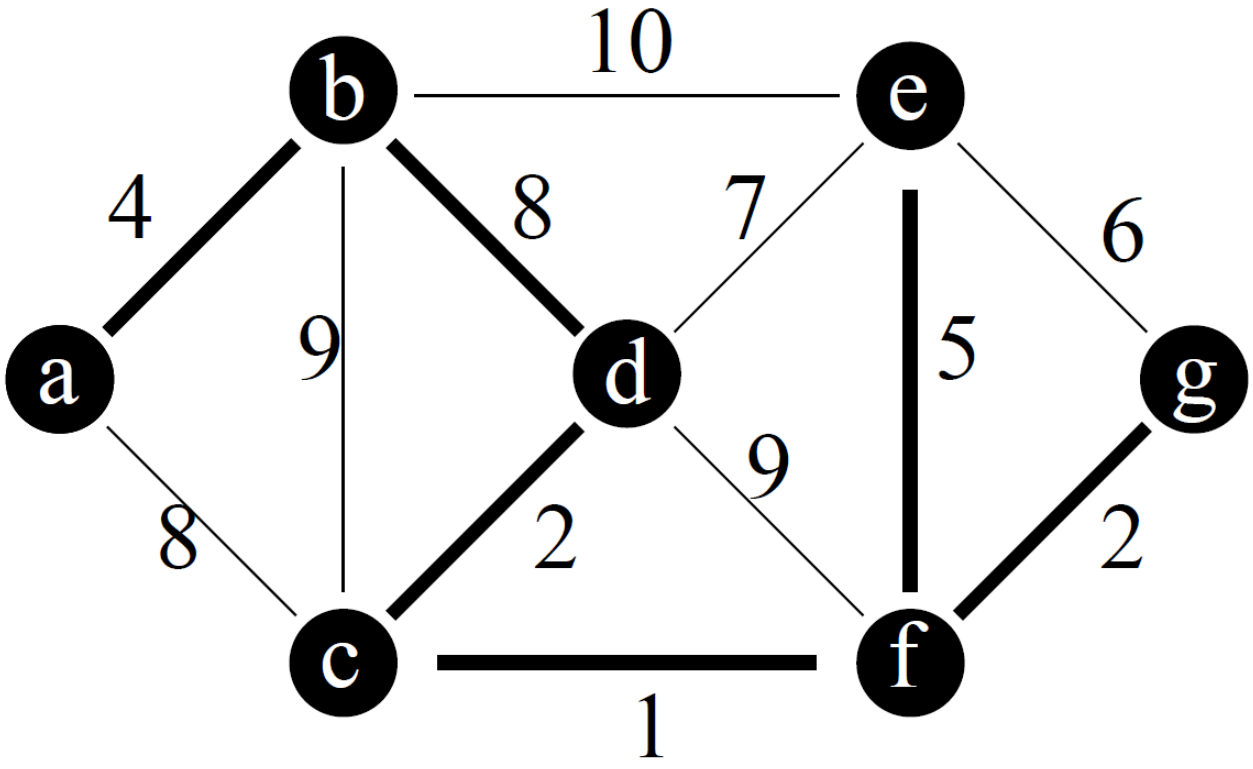
Prim's Algorithm for MST

Step 6: The algorithm continues, and we look around for the lightest neighbor edge from any vertex currently in the MST. edge which will (f, e). We add it and the corresponding vertex "e" to the MST.



Prim's Algorithm for MST

Since all vertices are now in the MST, Prim's is done.
Our MST is complete with a total weight of 22.

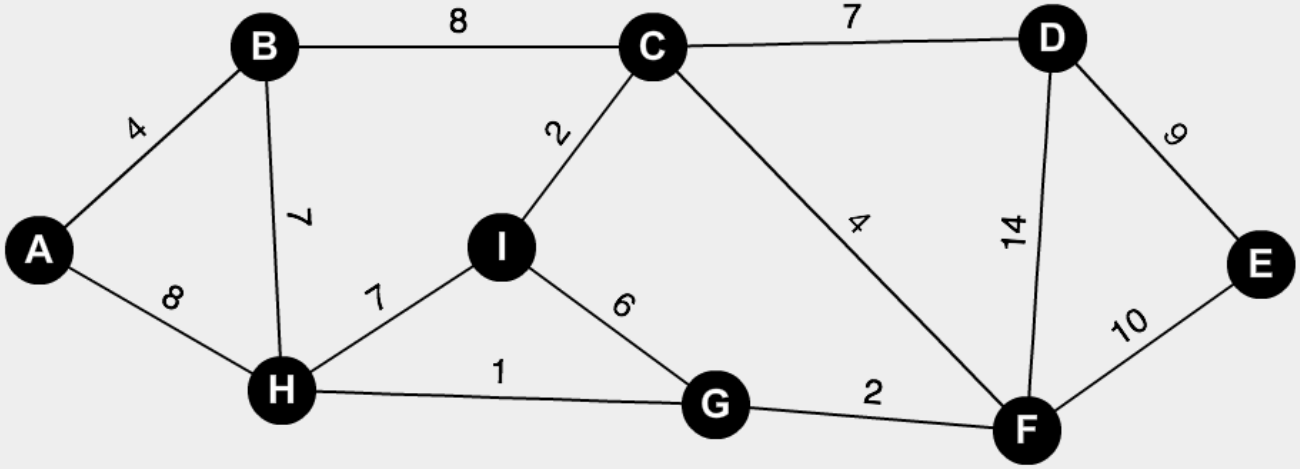


Reminder: VisualGo.Net – A Visual Tool

- **Observation:** Notice that Prim's always connects to a set of vertices already in the MST. There is always a connected tree in the MST we are building.
- **Watch it in action:** VisualGo.net features numerous sorting and graph algorithms where you can watch the algorithm in action.
- A great site if you're a visual learner. Check out Prim's, Kruskal's, DFS, BFS and more.
- https://visualgo.net/en

On Your Own: Prim's Algorithm for MST

- **Practice:** Starting at vertex A, use Prim's algorithm to determine the MST of the graph below.



PRIM'S MST Algorithm

Steps: Given a connected undirected graph G

1. Create a set mstSet that keeps track of vertices from G already included in the MST (this is initially empty).
2. Assign an initial key value to all vertices in the input graph. Initialize all key values as INFINITE since we will be seeking minimums (some people use null).
3. Assign key value as 0 to the first vertex to be included in the MST so it is initially picked in step #4.

PRIM'S MST Algorithm

Steps (con't):

4. WHILE mstSet doesn't include all vertices in G
   a) Pick the vertex **u** not in the mstSet that has the **minimum** key value.
   b) Add u to the mstSet.
   c) Update the key value of all vertices **adjacent** to u. To update the key values, iterate through all adjacent vertices to u. For each adjacent vertex v, if the weight of edge (u, v) is less than the previous key value of v, update key value of v to be weight of (u, v).

Greedy Algorithms

- A **greedy algorithm** is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.
- In other words, we build a solution piece by piece by always choosing the next one that offers the most obvious and immediate benefit (i.e., "greedy"). No regrets.
- Prim's is a greedy algorithm

VANDERBILT UNIVERSITY

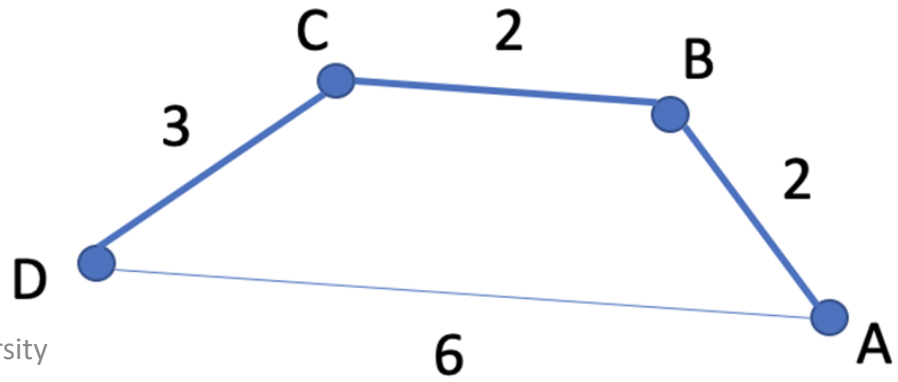
Prim's Algorithm: Do You Believe It?

- **Prove or disprove:** Since Prim's algorithm always chooses the minimum edge between vertices, it also finds the shortest path between any two vertices in the graph along the way.
  - If it's true, how would you prove it?
  - If it's not true, can you find a counter example?

VANDERBILT UNIVERSITY

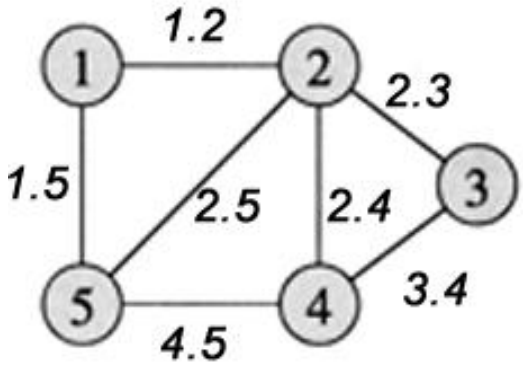
Prim's Algorithm: Do You Believe It?

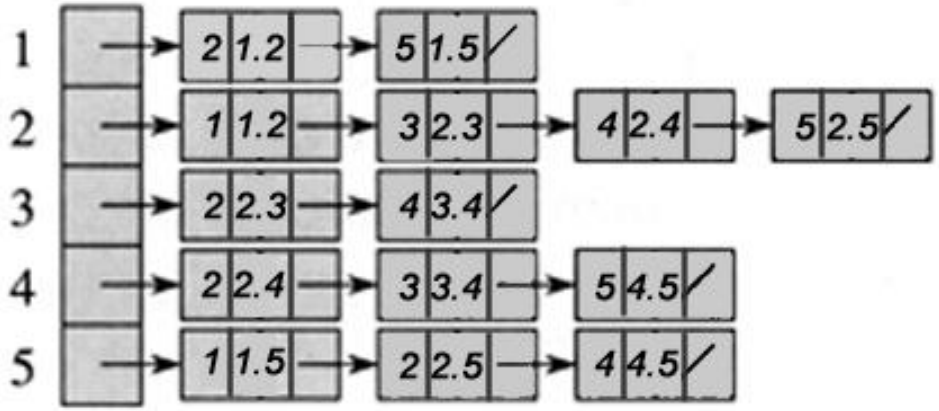
- **Prove or disprove:** Since Prim's algorithm always chooses the minimum edge between vertices, it also finds the shortest path between any two vertices in the graph along the way.
- **Answer:** Disprove. Given the graph below, Prim's creates the MST indicated by the bolded edges. The length of the path from D to A in the MST is 7. However, the shortest path from D to A is 6.



PRIM'S MST Algorithm: Data Structure

- Recall one way to represent the undirected weighted graph is using an adjacency list.
- Using an array of vertices where each vertex points to a linked list of neighbor nodes is ideal for Prim's.





VANDERBILT UNIVERSITY

Prim’s Algorithm: Analysis

- **Question:** What do you think is the running time of Prim's?
  A. O(V+E)
  B. O(VlogV)
  C. O(ElogV)
  D. O(V^2)
  E. None of the above

VANDERBILT UNIVERSITY

Prim’s Algorithm: Analysis

- **Question:** What do you think is the running time of Prim's?
  A. O(V+E)
  B. O(VlogV)
  C. O(ElogV)
  D. O(V^2)
  E. None of the above

VANDERBILT UNIVERSITY

PRIM'S MST Algorithm: Analysis

1. Create a set mstSet that keeps track of vertices from G already included in the MST (this is initially empty).
2. Assign an initial key value to all vertices in the input graph. Initialize all key values as INFINITE (some people use null).
3. Assign key value as 0 to the first vertex to be included in the MST so it is picked in the step #4.
4. While mstSet doesn't include all the vertices in G
   a) Pick a vertex **u** not in the mstSet that has minimum key value.
   b) Add u to the mstSet.
   c) Update the key value of all vertices adjacent to u.
      - To update the key values, iterate through all adjacent vertices to u. For each adjacent vertex v, if the weight of edge (u, v) is less than the previous key value of v, update the key value of v to be weight of (u, v).

O(V)

O(1)

O(V)

O(E) total

V times

Where's the bottle neck?

PRIM'S MST Algorithm: Analysis

1. Create a set mstSet that keeps track of vertices from G already included in the MST (this is initially empty).
2. Assign an initial key value to all vertices in the input graph. Initialize all key values as INFINITE (some people use null).
3. Assign key value as 0 to the first vertex to be included in the MST so it is picked in the step #4.
4. While mstSet doesn't include all the vertices in G
   a) **Pick a vertex u not in the mstSet that has minimum key value.**
   b) Add u to the mstSet.
   c) Update the key value of all vertices adjacent to u.
   To update the key values, iterate through all adjacent vertices to u. For each adjacent vertex v, if the weight of edge (u, v) is less than the previous key value of v, update the key value of v to be weight of (u, v).

O(V)

O(1)

O(V) each time

V times

O(E) total

Here's the bottle neck?

PRIM'S MST Algorithm: Analysis

- In a nutshell:
  1. Look though vertices to find cheapest to add.
  2. Put that one in the MST.
  3. Repeat until all vertices are in the MST.
- Hmmm…Prim's algorithm sounds a lot like selection sort. That's how the analysis looks as well. In the worst case, we consider V-1 vertices, then V-2 vertices, etc.

$$\sum_{i=1}^{V-1} i = \frac{V(V-1)}{2} = O(V^2)$$

PRIM'S MST Algorithm: Analysis

- **Initialization:** $c*V + d_0$ = **O(V)**
- **Find the next minimum vertex, "c" to add to the MST:**
  - Finding the minimum = **O(V)**
  - We do this for every vertex. So, the total time for our bottleneck step = **$O(V^2)$**
- **Update the cost to each of c's neighbors:**
  - This is similar to processing each edge in the graph
  - That is = **O(E) (which will be at most ~= V^2)**
- **Conclusion:**
  - Our current implementation of Prim's is then **$O(V^2)$**
  - Stay tuned. We'll be able to make Prim's more efficient later in the semester after we learn some new tricks.

Review: PRIM'S MST Algorithm

- Some observations about Prim's:
  - Prim's is an $O(V^2)$ algorithm.
  - Many of other graph algorithms are O(V+E).
  - It would be nice if we had an MST algorithm whose efficiency was not quadratic.
  - We'll get smarter later in the semester…stay tuned.

VANDERBILT UNIVERSITY

That's All For Now…

- Coming to a Slideshow Near You Soon…
  1. Dijkstra's Algorithm
  2. Kruskal's Algorithm
  3. Union-Find

That's All For Now