SCHOOL OF ENGINEERING

VANDERBILT UNIVERSITY

CS 3250
Algorithms
Single Source Shortest Path (SSSP)

Dijkstra'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 Friday, February 16th at 9AM.

VANDERBILT UNIVERSITY

Announcements

- **HW3 GO (HW3 – Part 2)**
  - You will work with you same partner Part 1 STOP (unless you are working by yourself).
  - Create a SlideDoc for one of the problems you answered on the quiz (choose one you got correct).
  - Follow the template for the SlideDoc and look at the examples on Brightspace.
  - **Due date:** HW3 Part 2 (aka "GO" HW) due by Wednesday, February 28th at 9AM.
- **Exam Review Friday**

Announcements

- **Exam #1 – Wednesday, February 21st**
  - Arrive early. Closed book. No notes or electronics.
  - Multiple choice, algorithm comprehension, written response, algorithm design. No hashing questions.
  - Covers through today's lecture.
  - Exam review on Friday.
- **How I would study for the exam…**
  - Review lectures on TopHat and re-do examples (remember the slides on Brightspace are **not** current).
  - Review TopHat questions.
  - Reinforce weak area (via videos, books, etc.). Look at technical interview questions on related topics.

Greedy Algorithms and Optimal Substructure

- **Question:** Suppose we have an undirected weighted graph where w is on the **shortest path** from u to v. Is the path from u to w (on the way to v) guaranteed to be the shortest path from u to w? Explain.

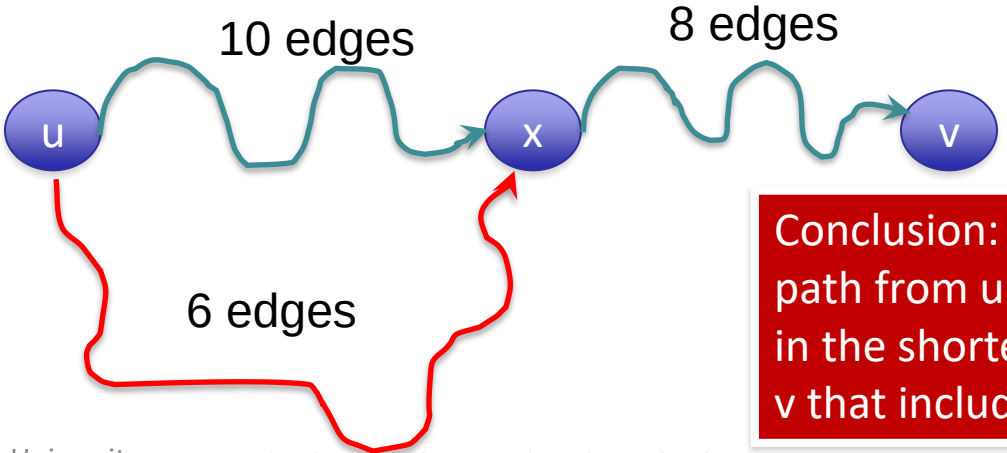
Greedy Algorithms and Optimal Substructure

- **Question:** Suppose we have an undirected weighted graph where w is on the **shortest path** from u to v. Is the path from u to w (on the way to v) the shortest path from u to w? Explain.
- **Answer: Yes.** If there were a shorter path from u to w on the way to v, we would use that path to get from u to w on the way to v instead of the longer path. This concept is referred to as **optimal substructure**. More on this later.

VANDERBILT UNIVERSITY

Greedy Algorithms and Optimal Substructure

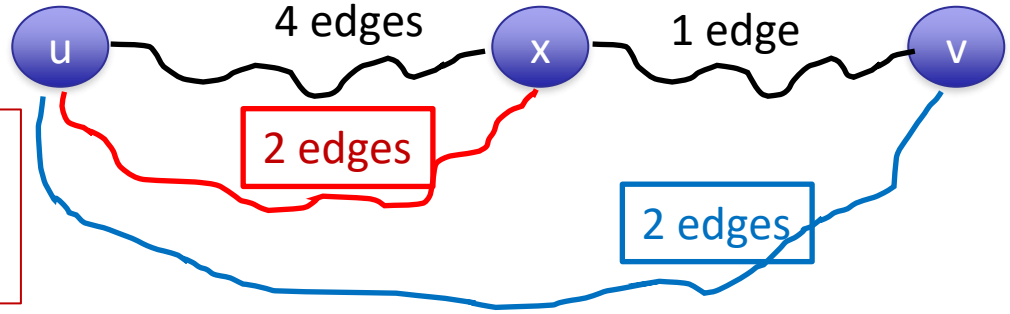
- **Shortest path** – Consider the general scenario depicted below. Is it possible that the shortest path from u to v that includes x does not house the shortest path from u to x?
- No. If there was a shorter path from u to x, we would use it to shorten the path from u to v.



Conclusion: The shortest path from u to x is contained in the shortest path from u to v that includes x

Greedy Algorithms and Optimal Substructure

- **Let's take a closer look…**
- Consider the scenarios below on an unweighted, undirected graph.



This path can't exist. If it existed, we would have used the red path instead.

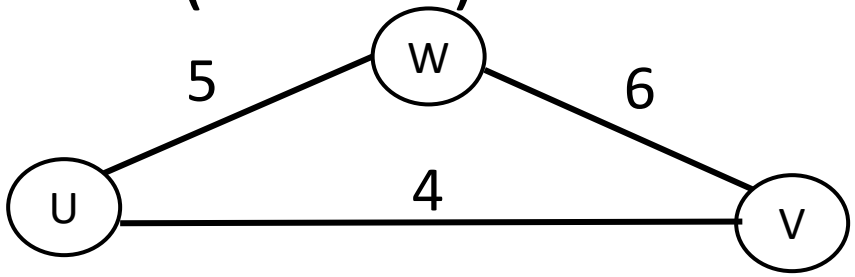
This path can't exist or the shortest path from u to v wouldn't include x

Greedy Algorithms and Optimal Substructure

- **Question:** Suppose w is on the **longest path** from u to v in an undirected, weighted graph. Is the path from u to w (on the way to v) the longest path from u to w? Explain.

Greedy Algorithms and Optimal Substructure

- **Question:** Suppose w is on the **longest path** from u to v in an undirected, weighted graph. Is the path from u to w (on the way to v) the longest path from u to w? Explain..
- **Answer:** No. The longest path from u to v is the path from u --> w -> v (cost 11). Within that path, the cost to get to w is u -> w or (cost 5). However, there is a longer path to get to w, that is the path from u -> v -> w (cost 10).

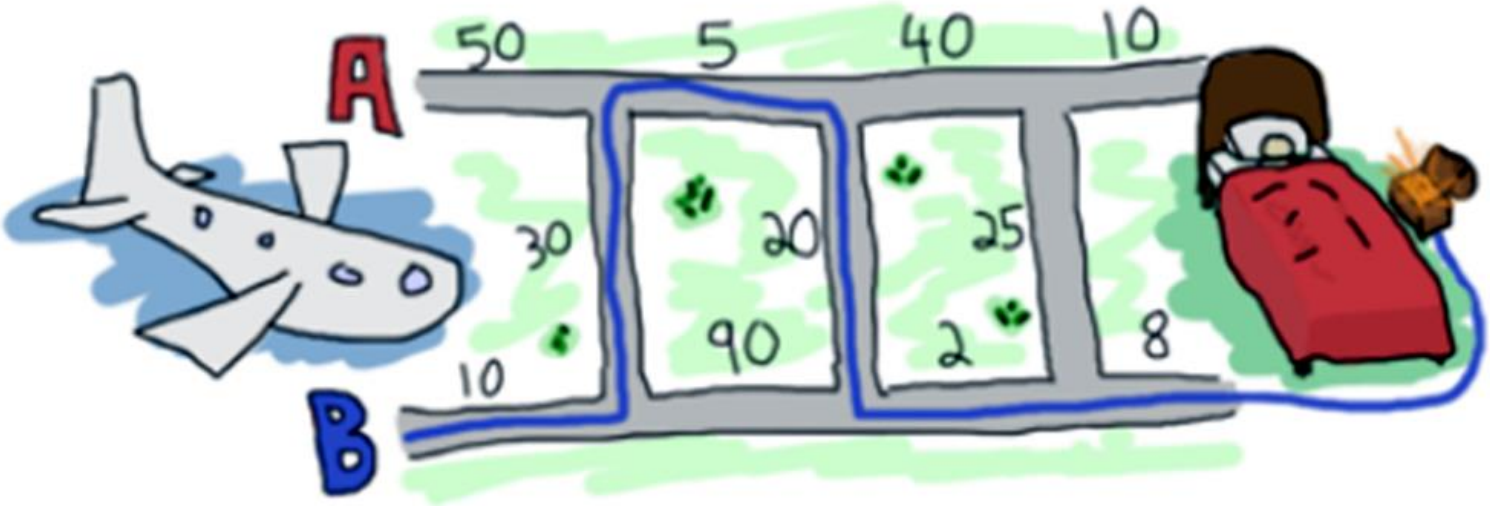


Graphs: Single Source Shortest Path

- One common problem often solved using graphs is the **Single Source Shortest Path** problem (SSSP).
- **SSSP** - Starting at some vertex u, find the shortest path to all other vertices in the graph.

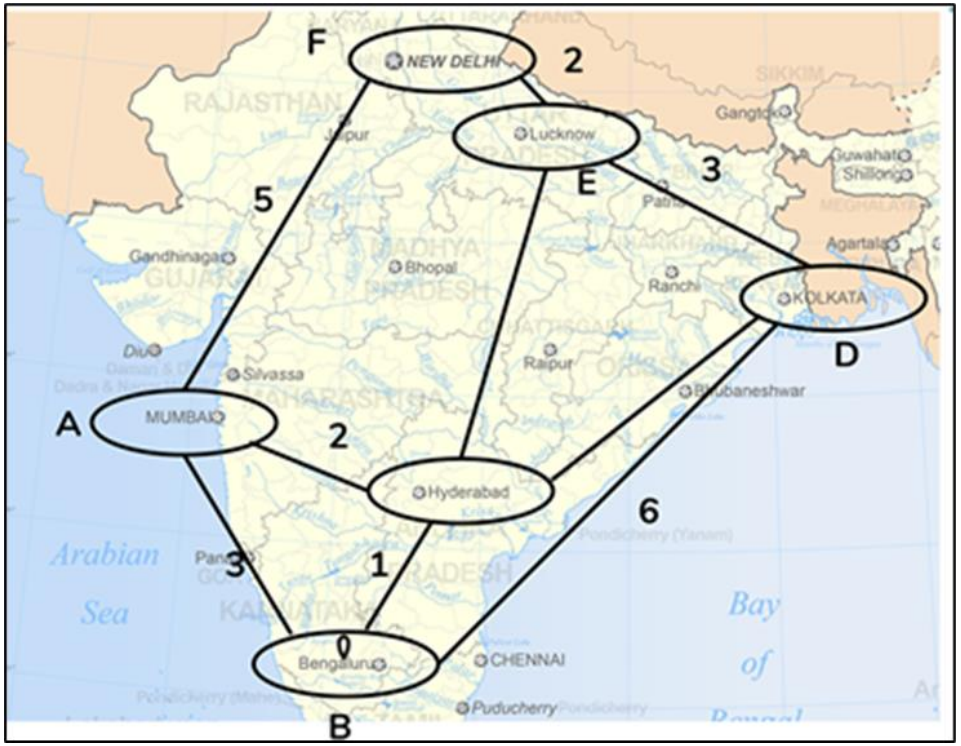
Graphs: Single Source Shortest Path

- **Transportation** (road maps, airlines) – Cheapest or quickest way to get from one place to another.
- **Communication** – Quickest way to get a message from one place to another.
- **Social Networks** – Shortest number of connections from one person to another.



Graphs: Single Source Shortest Path

- **Technical Interview Question:** Consider the map below. We need to travel from Bengaluru to all other destination cities indicated. Identify the paths with minimal cost originating from Bengaluru.



Graphs: Single Source Shortest Path

- Nadha Skolar claims BFS already locates the shortest path. We should be able to simply run BFS starting at vertex u and when we get to vertex v, we'll know the shortest path from u to v.
- **Question:** Will our BFS algorithm locate the shortest path?

VANDERBILT UNIVERSITY

Graphs: Single Source Shortest Path

- **Question:** Will our BFS algorithm locate the shortest path?
- **Answer:** BFS only works for shortest path in an *unweighted* graph (or a graph where all edge weights are equal). Provided the "shortest" path from u to v is the path that has the fewest edges from u to v, the BFS approach will work.
- Otherwise, we need something else.

VANDERBILT UNIVERSITY

Graphs: Single Source Shortest Path

- Bedha Skolar claims the shortest path is the same problem as MST. We can use Prim's algorithm, and it will correctly locate the shortest path from a source vertex u to another vertex v.
- **Question:** Is Bedha right? Explain.

VANDERBILT UNIVERSITY

Graphs: Single Source Shortest Path

- Bedha Skolar claims the shortest path is the same problem as MST. We can use Prim's algorithm, and it will correctly locate the shortest path from a source vertex u to another vertex v.
- **Question:** Is Bedha right? Explain.
- **Answer:** No. When looking for the shortest path, we do not necessarily need all vertices in the graph to be connected in a tree. Prim's chooses the minimal cost to connect **all** vertices, not necessarily the minimal cost for the path from u to v.

Graphs: Single Source Shortest Path

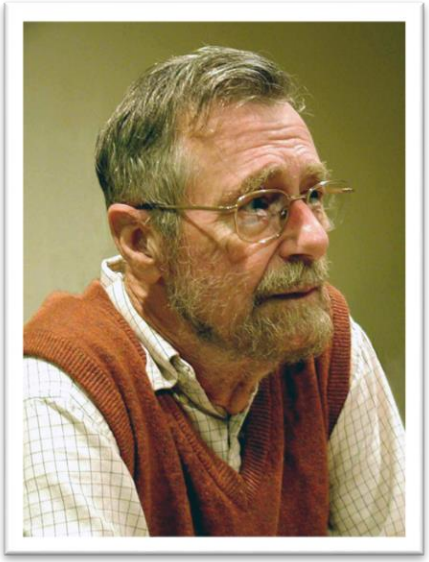
- **Some assumptions –** To make our life a little easier, as always, we will assume the following:
  1. **The graph is connected** – If this isn't the case, (see G1), we can easily code around this using DFS or BFS. But to simplify coding, let's assume we're looking at a connected graph (see G2).

Graphs: Single Source Shortest Path

- **Some observations –** To make our life a little easier, we will assume the following:
  2. **The edge weights are positive** – It's not very common to need to handle negative edge weights. For this reason, we'll assume the graph does not have negative edge weights.
     - We'll learn other algorithms we can use should we need to handle the rare care of negative edge weights in a shortest path.

Graphs: Single Source Shortest Path

- Our regular BFS doesn't work and our regular MST using Prim's doesn't work to solve shortest path.
- There are aspects to those graph algorithms that are close to what we need, but don't exactly fit.
- This is precisely the time to look for a well-known algorithm and then modify it for our needs.
- **Edsger Dijkstra** to the rescue.



VANDERBILT UNIVERSITY

Dijkstra’s Algorithm for Shortest Path

- In 1956, **Edsger Dijkstra** realized that a simple modification to Prim’s algorithm does the job.
- This modification, known as **Dijkstra's algorithm** is widely used today to provide directions between physical locations on Google Maps, Waze, Siri, etc.

Review: **PRIM'S** MST Algorithm

1. Initialize all key values to ∞
2. Assign key value 0 to first vertex added to the MST.
3. While MST doesn't include all the vertices in G
   a) Pick a vertex u not in the MST with minimum key value.
   b) Add u to the MST.
   c) Update the key value of all vertices adjacent 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).

Dijkstra's Algorithm for Shortest Path

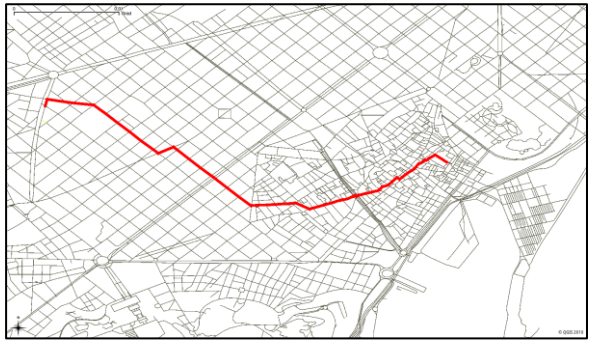
- While Prim's picks the smallest edge weight next, Dijkstra's picks the edge that **minimizes the total**.

1. Initialize the cost of each vertex to ∞
2. Initialize the cost of the source vertex to 0.
3. WHILE undiscovered vertices are left in the graph
4. Select an undiscovered vertex u with lowest cost
5. Mark u as discovered
6. FOR each vertex v adjacent to u
7. v's cost=min(v's old cost, u's cost + cost of (u, v))
8. Mark u as processed
9. END-WHILE

Dijkstra's Algorithm for Shortest Path

Under the hood:

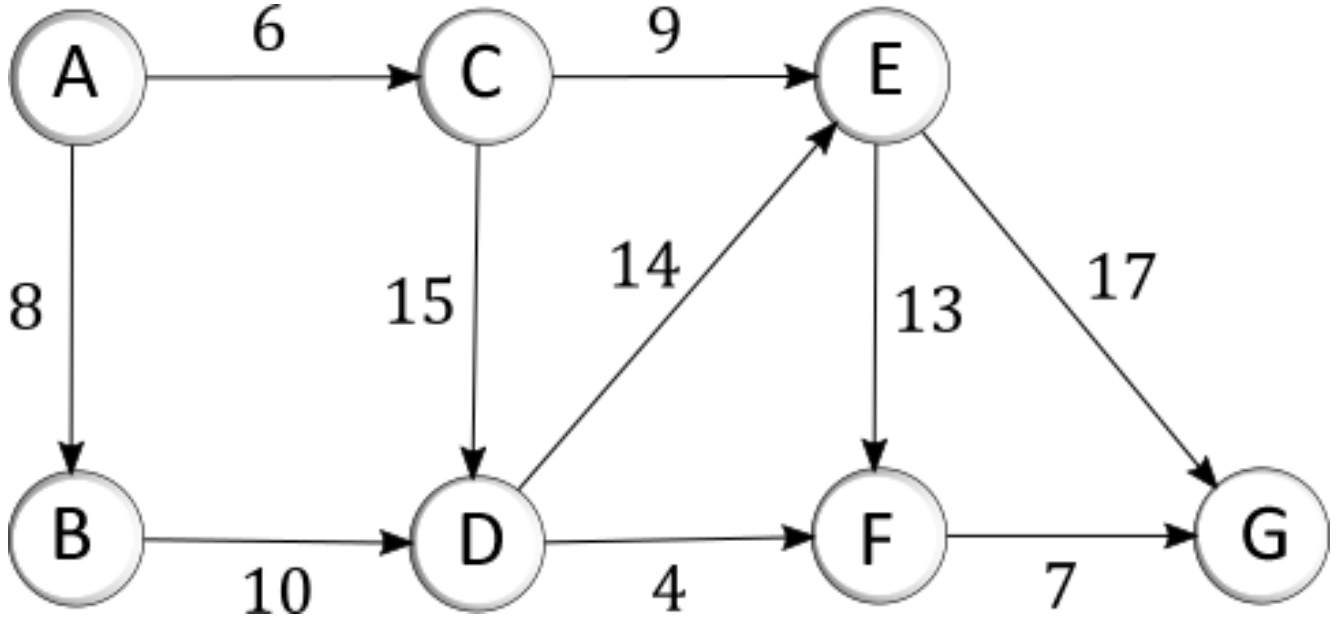
- Some programmers maintain a "cost" field in each vertex to hold the minimum cost to reach that vertex from the source vertex.
- Others maintain a separate array of costs.
- As always, the details of how this is done under the hood don't matter, provided they do not affect the overall running time of the algorithm.



VANDERBILT UNIVERSITY

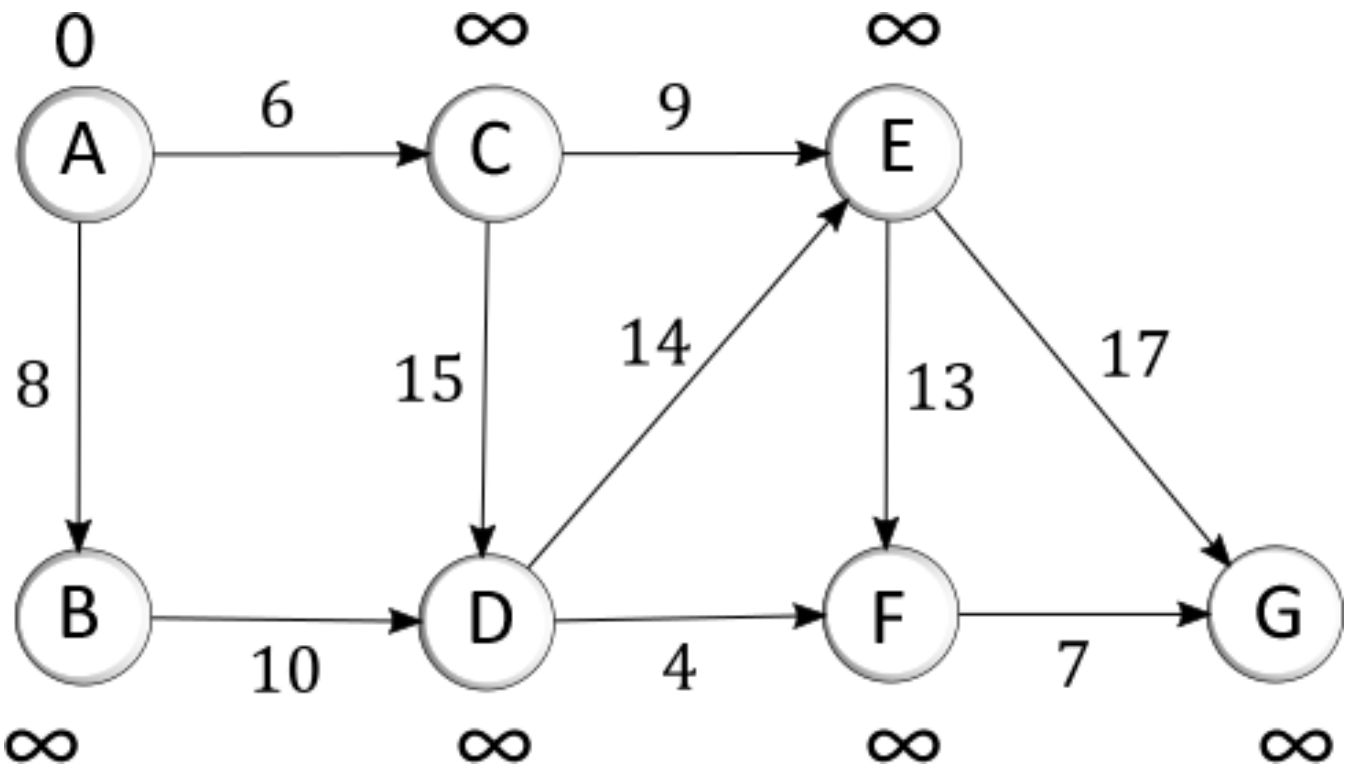
Dijkstra's Algorithm for Shortest Path

- Let's try an example using vertex A as our source and try to locate the shortest path from vertex A to vertex G.



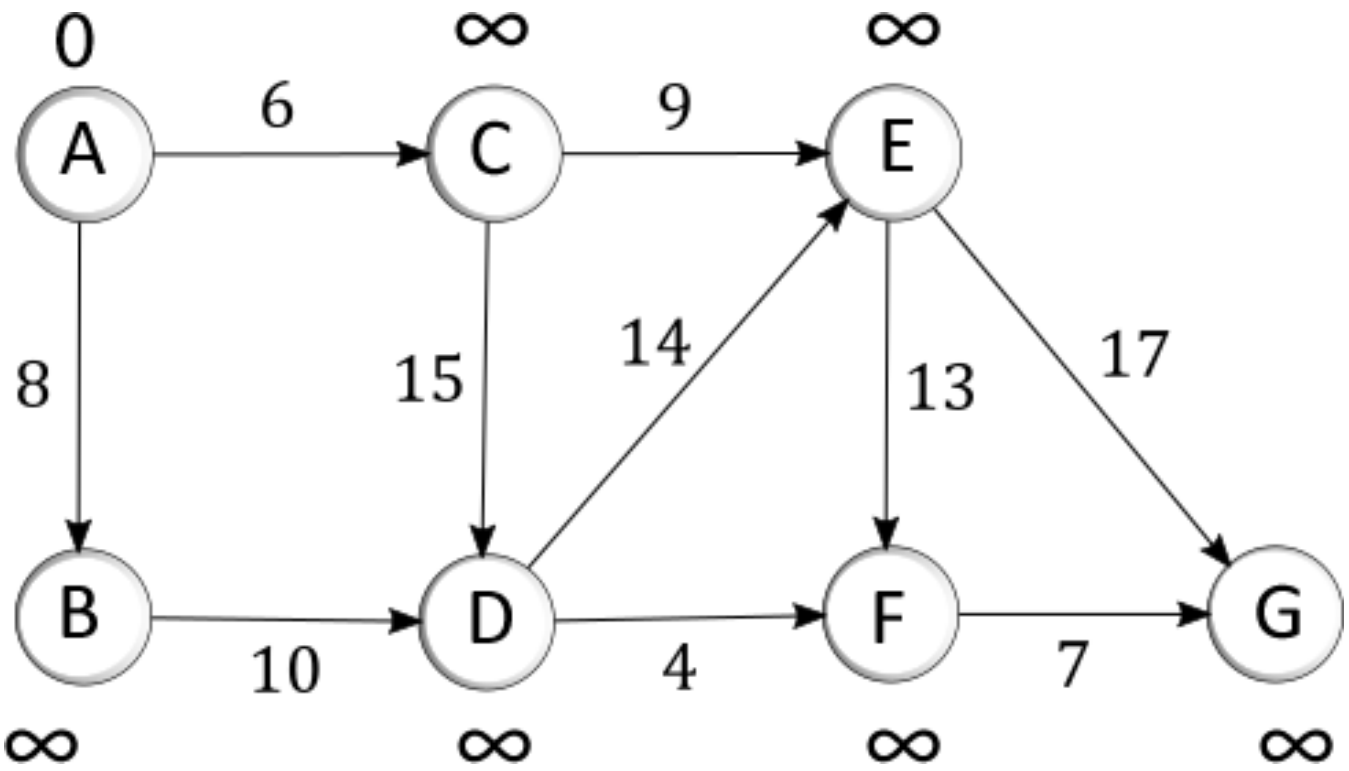
Dijkstra's Algorithm for Shortest Path

- **Initially:** All vertices are tagged with infinity as the cost to reach them. The source vertex A is tagged with cost 0 (yep, sounds a lot like Prim's).



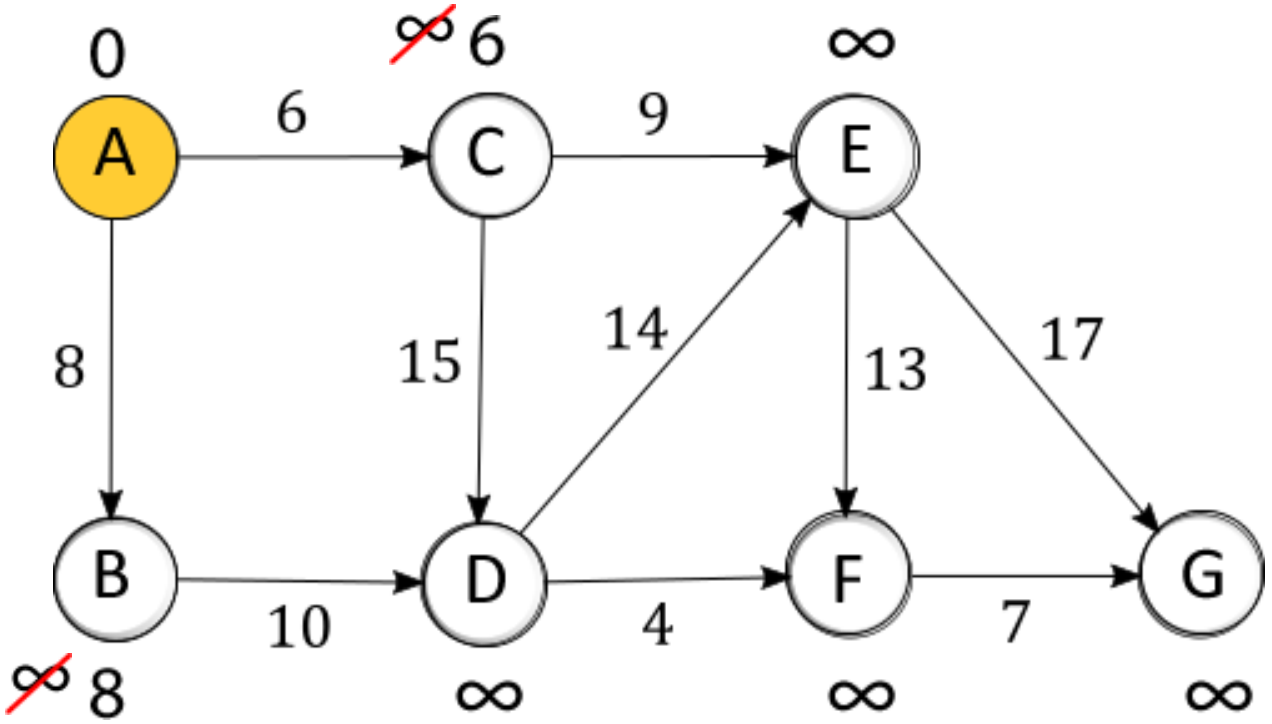
Dijkstra's Algorithm for Shortest Path

- **Step 1:** A is the lowest cost vertex available, so we choose it and consider its adjacent vertices (again, sounds like Prim's).



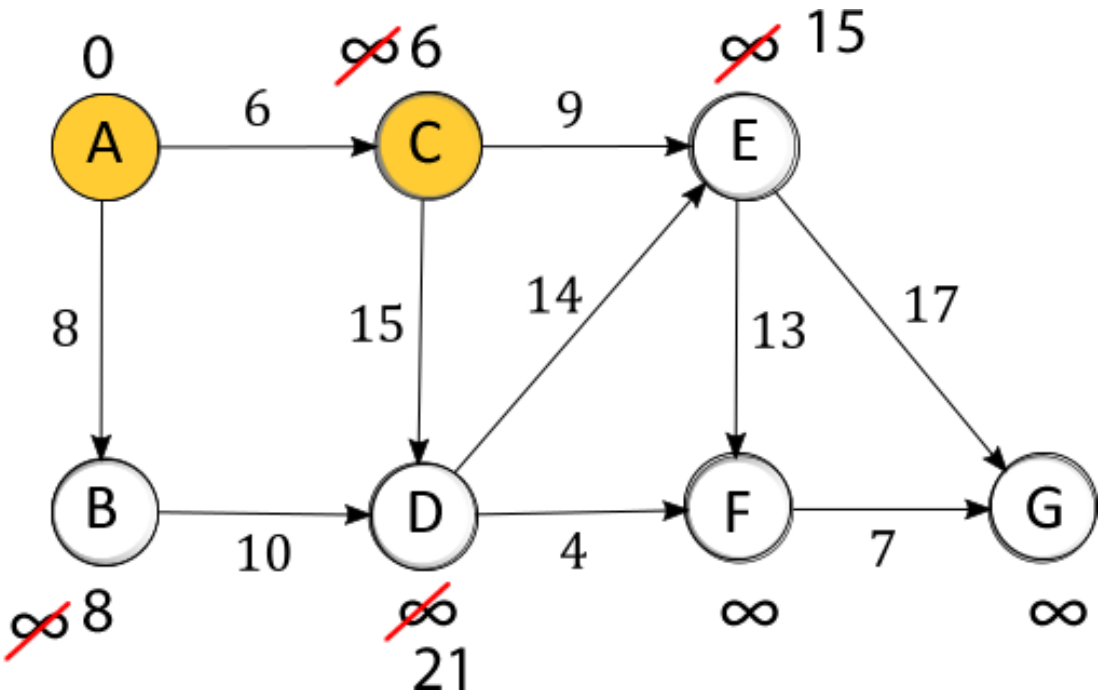
Dijkstra's Algorithm for Shortest Path

- **Step 2:** We update the vertices adjacent to A with the current minimal cost to reach them. We can now mark A as finished. This means we know for certain the minimum cost to reach A from A is 0.



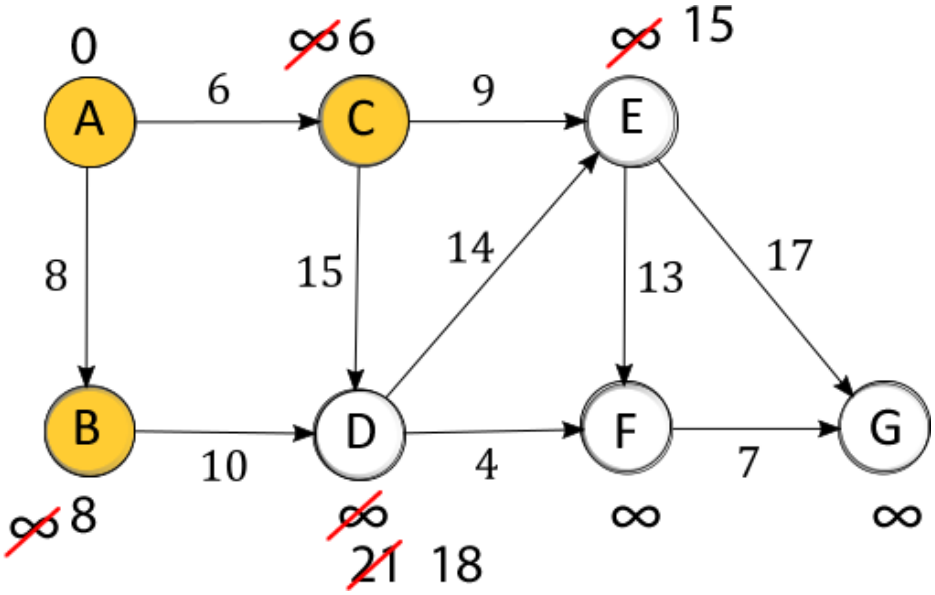
Dijkstra's Algorithm for Shortest Path

- **Step 3:** We pick C next as it is the minimum cost vertex. We update the cost of the vertices adjacent to C with the current minimum cost to reach them from A. We mark C finished. We know for certain the minimum cost to reach C from A is 6.



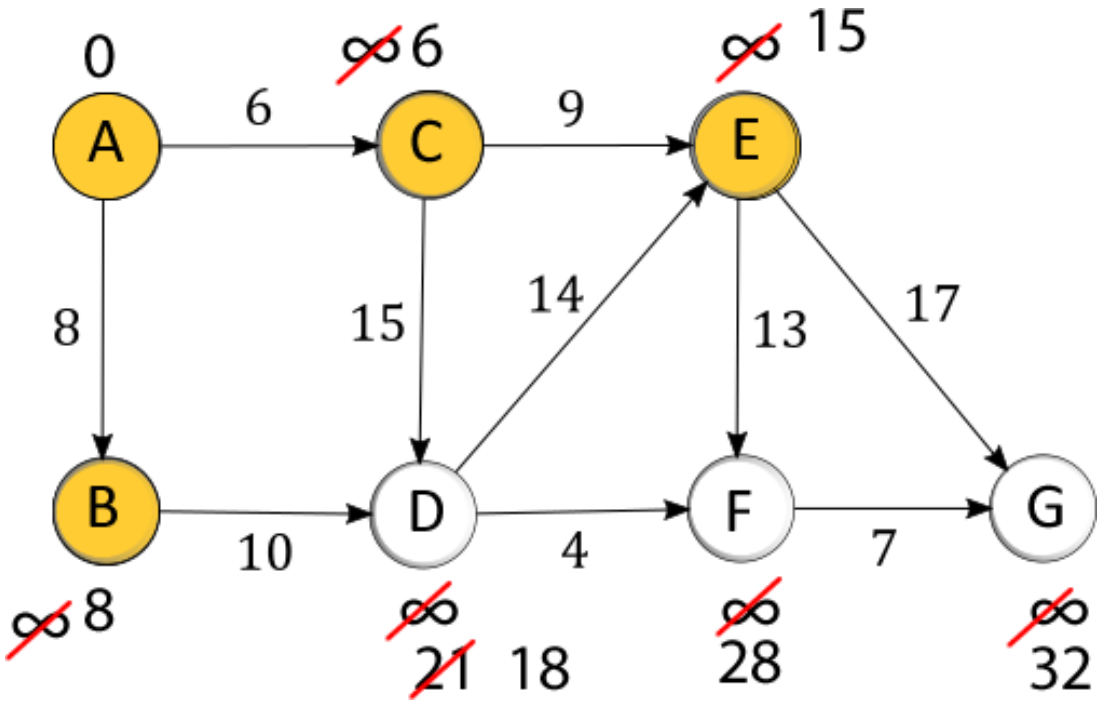
Dijkstra's Algorithm for Shortest Path

- **Step 4:** We pick B to explore next since that is the minimum cost vertex. We update the cost of the vertices adjacent to B with the current minimum cost to reach them (notice how we re-update D). We mark B finished and know the minimum cost to reach B from A is 8.



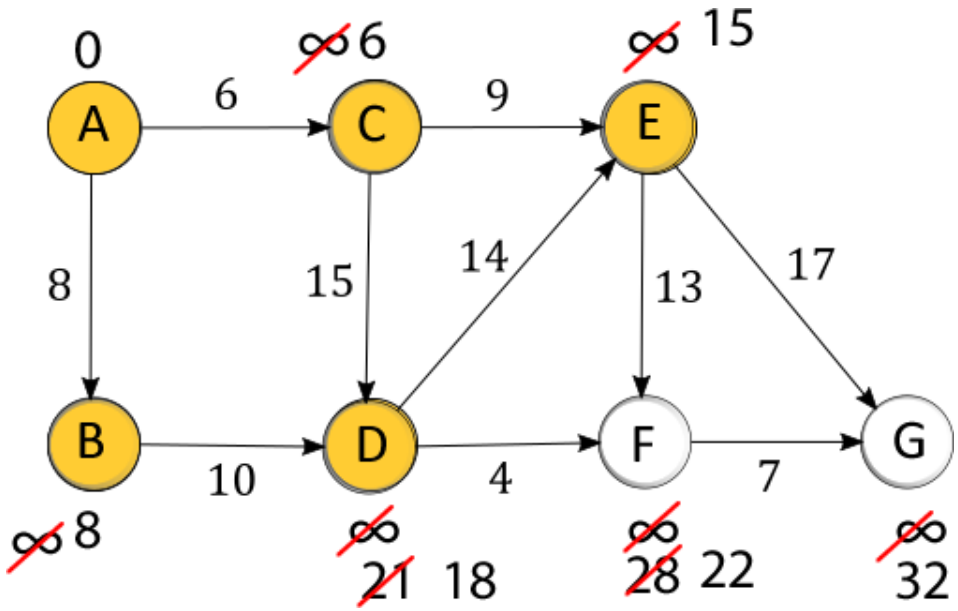
Dijkstra's Algorithm for Shortest Path

- **Step 5:** We pick E to explore next since as it is the minimum cost vertex. We update the cost of the vertices adjacent to E with the current minimum cost to reach them. We mark E finished and know the minimum cost to reach E from A is 15.



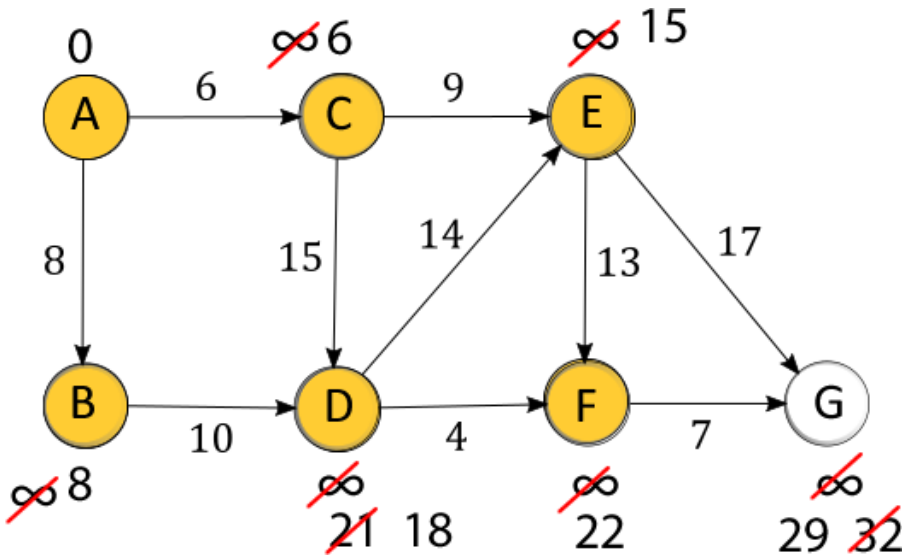
Dijkstra's Algorithm for Shortest Path

- **Step 6:** We pick D to explore next as it is the minimum cost vertex. We update the cost of the vertices adjacent to D with the current minimum cost to reach them (notice F gets re-updated). We mark D finished and know the minimum cost to reach D from A is 18.



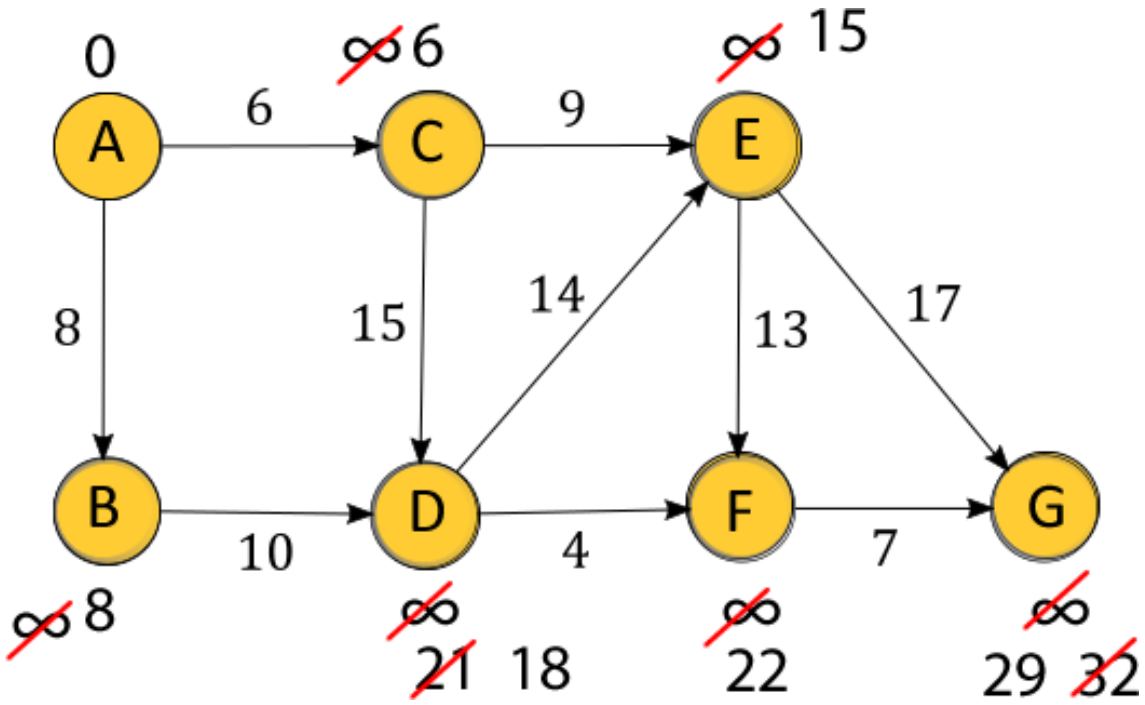
Dijkstra's Algorithm for Shortest Path

- **Step 7:** We pick F to explore next since as it is the minimum cost vertex. We update the cost of the vertices adjacent to F with the current minimum cost to reach them (notice G gets re-updated). We mark F finished and know the minimum cost to reach F from A is 22.



Dijkstra's Algorithm for Shortest Path

- **Step 8:** We pick G to explore next. We have reached our destination so regardless of whether G has any outgoing edges, we are done. We mark G as finished and know for certain the **minimum cost to reach G from A is 29.**



Graphs: Single Source Shortest Path

- We know for MSTs it doesn't matter whether the edge weights are positive or negative. Prim's correctly handles negative edge weights and still arrives at the correct answer.
- All the weights were positive edge weights in our graph. Would negative edge weights present an issue for us in single-source shortest path?

Graphs: Single Source Shortest Path

- **Question:** As with MST, the shortest path problem works fine when edges have negative weights.

A. Yes, of course.

B. No, negative edge weights make it impossible to find the shortest path.

C. Maybe. It just depends on the graph.

VANDERBILT UNIVERSITY

Graphs: Single Source Shortest Path

- **Question:** Just like MST, the shortest path problem works fine when edges have negative weights.

A. Yes, of course.

B. No, negative edge weights make it impossible to find the shortest path.

C. Maybe. It just depends on the graph.

VANDERBILT UNIVERSITY

Dijkstra's Algorithm: Analysis

- Since Dijkstra's relies on Prim's algorithm as its backbone, we can guess that Dijkstra's running time will be the same as Prim's, currently $O(V^2)$
- Let's walk through the analysis.
- We'll be able to improve this running time when we learn some new tricks later in the semester.

VANDERBILT UNIVERSITY

Dijkstra's Algorithm: Analysis

- **Initialization:** $c*V + d_0$ = **O(V)**
- **Find the next minimum cost vertex to explore:**
  - Finding the cheapest = **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 u's neighbors:**
  - This is similar to processing each edge in the graph
  - So that = **O(E)**
- **Conclusion:**
  - Our current implementation of Dijkstra's is **$O(V^2)$**
  - Stay tuned. We'll be able to make Dijkstra's more efficient later in the semester after we learn a few new tricks.

Dijkstra's Algorithm Summary

- **When to use Dijkstra's:** Dijsktra's is a proven algorithm to find the shortest path from a vertex to any other vertex. Assumes non-negative weights.
- **Where Dijkstra's Shines:** GPS navigation, network connectivity.
- **Running Time:** $O(V^2)$ for now which is the same as Prim's. Stay tuned for an improvement later in the semester.

Dijkstra's vs Prim's: Some Key Differences

Dijkstra's Algorithm	Prim's Algorithm
Finds the **shortest path**	Finds the **MST**
Works on **undirected or directed**	Works only on **undirected**
No negative edge weights	**Handles negative** weights

VANDERBILT UNIVERSITY

That's All For Now…

- Coming to a Slideshow Near You Soon…
  1. Longest Path
  2. Greedy Algorithms

That's All For Now