Discussion 11

Graph Algorithms

Objectives

- Shortest path algorithms
  - Dijkstra’s Algorithm
  - Bellman-Ford Algorithm
- Minimum spanning tree
  - Prim’s Algorithm
  - Kruskal’s Algorithm

Shortest Path Algorithms

Overview of Shortest Path Algorithms

1. **Single Source Shortest Path Problem**
   a. Minimum weight path from a single node to any other node

2. **All Pairs Shortest Path**
   a. Minimum weight path from **any** node to any other node
   b. Floyd-Warshall algorithm: O(|V|^3)
      i. can have negative weight edges
      ii. can detect negative weight cycles

Single Source Shortest Path Algorithms

Shortest path from a single source to all vertices

1. BFS: $O(|V| + |E|)$ <- shortest path in steps
   a. unweighted graph
2. Dijkstra's algorithm: $O(|V|^2)$, or if implemented by heap $O((|V| + |E|) * \log(|V|))$
   a. weighted graph
   b. no negative weight edge

   <- shortest path in weight
3. Bellman-Ford algorithm: $O(|V|*|E|)$
   a. can have negative weight edges
   b. no negative cycles (detects them)

   <- shortest path in weight

Dijkstra's algorithm

- Determines shortest path to each vertex from a single source
- Works on:
  - Weighted graphs
  - **That DON'T have negative weights**
  - Directed graphs OR undirected graphs
- A greedy algorithm
  - For each iteration, chooses the next vertex to visit by:
    - which unvisited vertex has the minimum distance to the source vertex.
- For each iteration:
  - Relaxation principle applied to all the edges that originate from the unvisited vertex that has the minimum distance to the source vertex.
- Time complexity
  - $O(|V|^2)$, or if implemented by heap and adjacency list $O((|V| + |E|) * \log(|V|))$

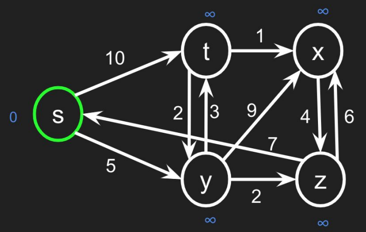
Edge Relaxation Principle

- Considers an edge and whether the path through vertex u to vertex v is better than any previously found path to vertex v.
- If so, it updates a distance map for key v to be the distance from the source to u plus the distance from u to v.
- Boils down to:

  For the edge from the vertex u to the vertex v, if $d[u]+w(u,v)<d[v]$ is satisfied, update $d[v]$ to $d[u]+w(u,v)$

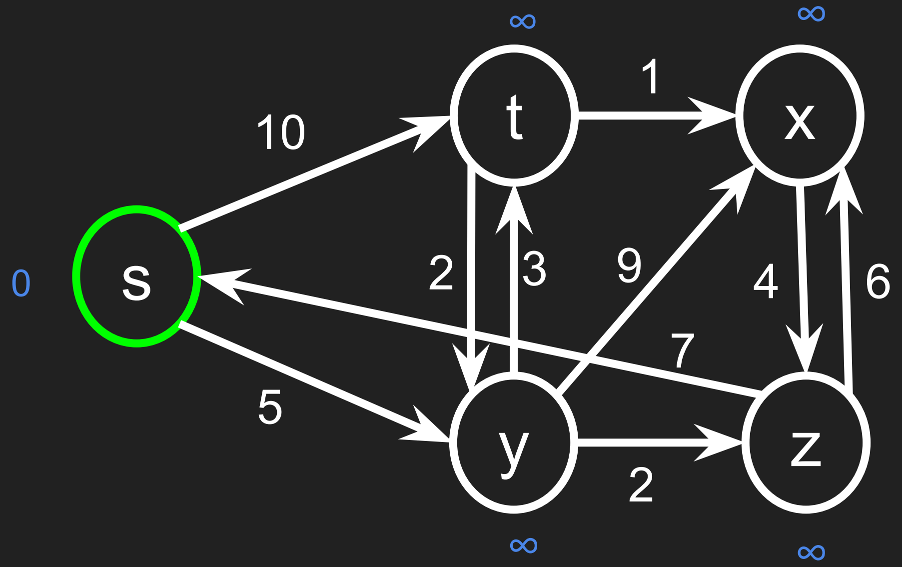
- The algorithms for the shortest paths problem solve the problem by repeatedly using the edge relaxation in a certain order of relaxation.

Dijkstra's algorithm



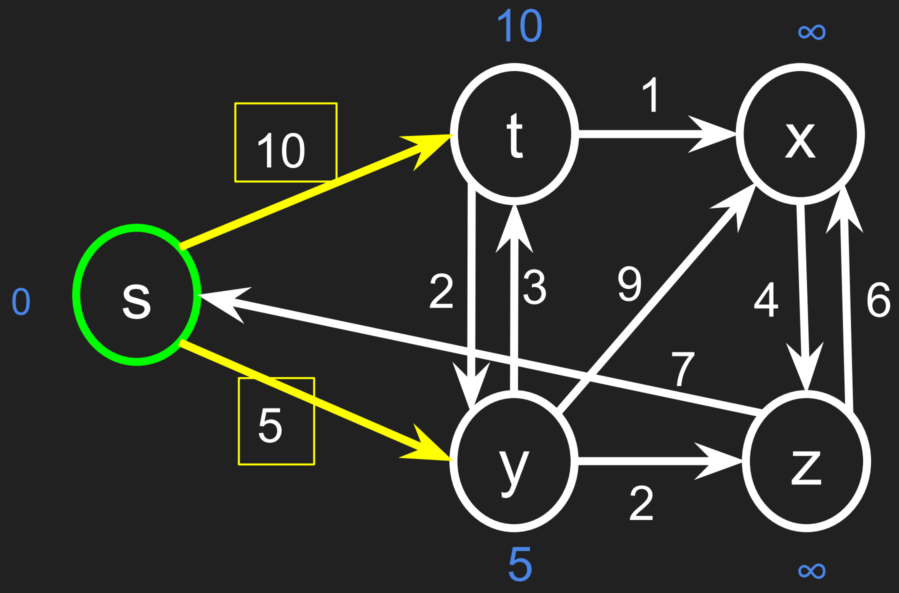
- We will cover the Dijkstra’s algorithm variant that maintains three maps. For all of these maps, each vertex is a key.
  - V: A map that tracks whether a vertex has been visited
    - A boolean value is used to indicate whether a vertex has been visited
  - dist: A map that stores all the distances from the source node
    - For all but source, distance = infinity to start
    - Source vertex distance = 0 (selected first)
  - P: A predecessor map
    - The value is the last vertex passed through to arrive at the key vertex
    - All values = null to start
    - Used to determine the shortest paths
    - *Ignore if you only want the shortest distance*

Dijkstra's algorithm: Initialization



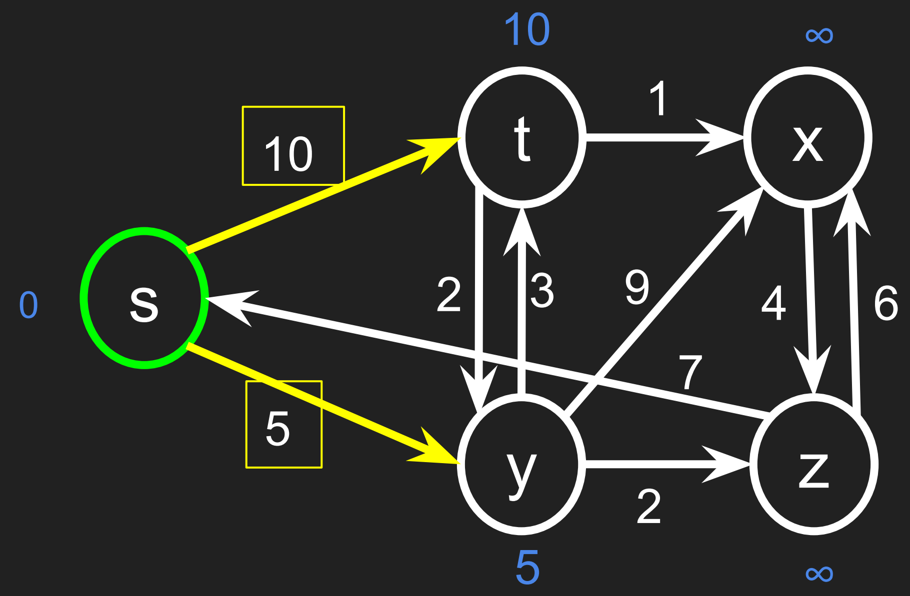
Key (k)	V[k]	dist[k]	p[k]
s	F	0	null
t	F	∞	null
y	F	∞	null
x	F	∞	null
z	F	∞	null

1st Iteration: s is selected



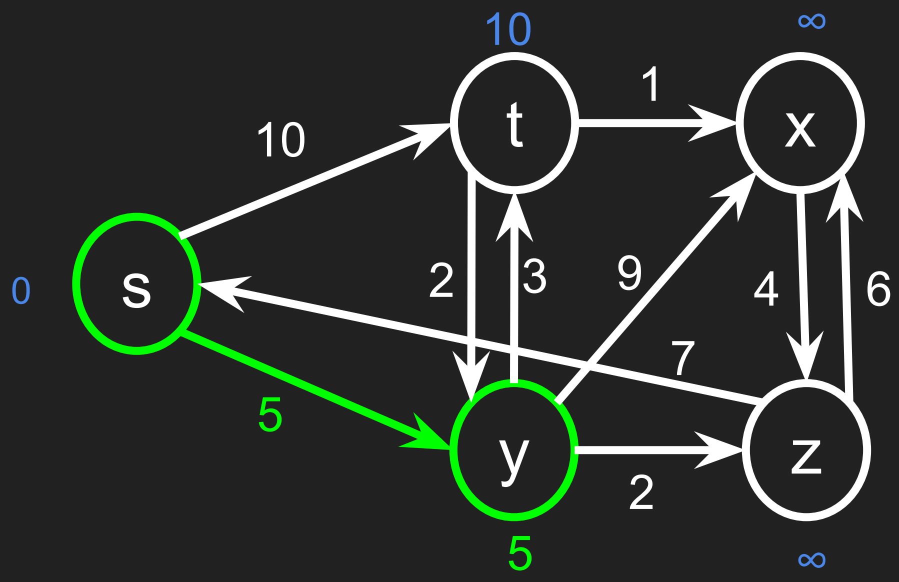
Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	F	∞	null
y	F	∞	null
x	F	∞	null
z	F	∞	null

1st Iteration: the distances to t and y are relaxed



Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	F	10	s
y	F	5	s
x	F	∞	null
z	F	∞	null

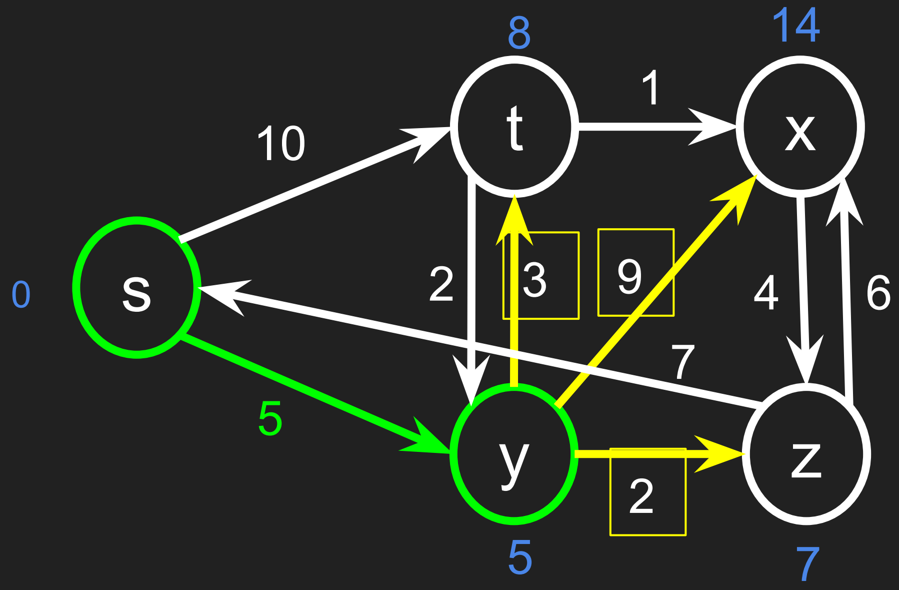
2nd iteration: y is visited



Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	F	10	s
y	T	5	s
x	F	∞	null
z	F	∞	null

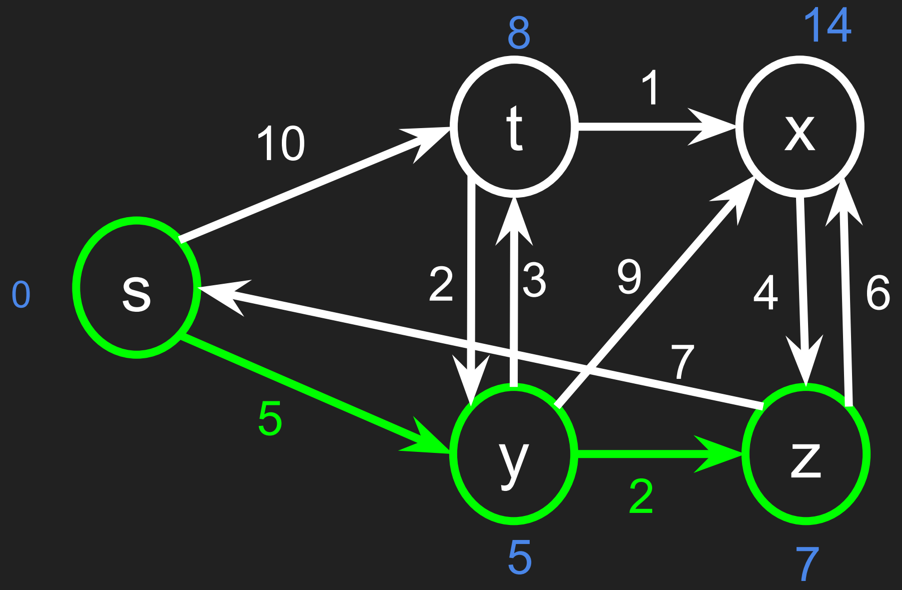
Note: Whenever we select a node N to visit, the current distance to N will already be minimized (this is why the algorithm can go node by node and why it doesn’t work for negative weights).
Notice that any path to y has to either go through y or t, so 5 is the shortest possible path to y.

2nd iteration: the distances to t, x, and z are relaxed



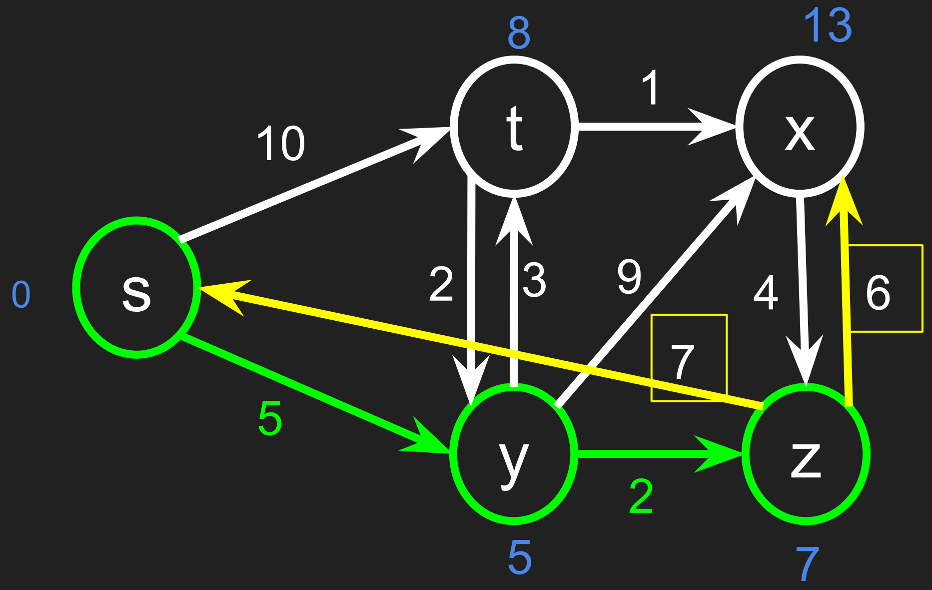
Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	F	8	y
y	T	5	s
x	F	14	y
z	F	7	y

3rd iteration: z is visited



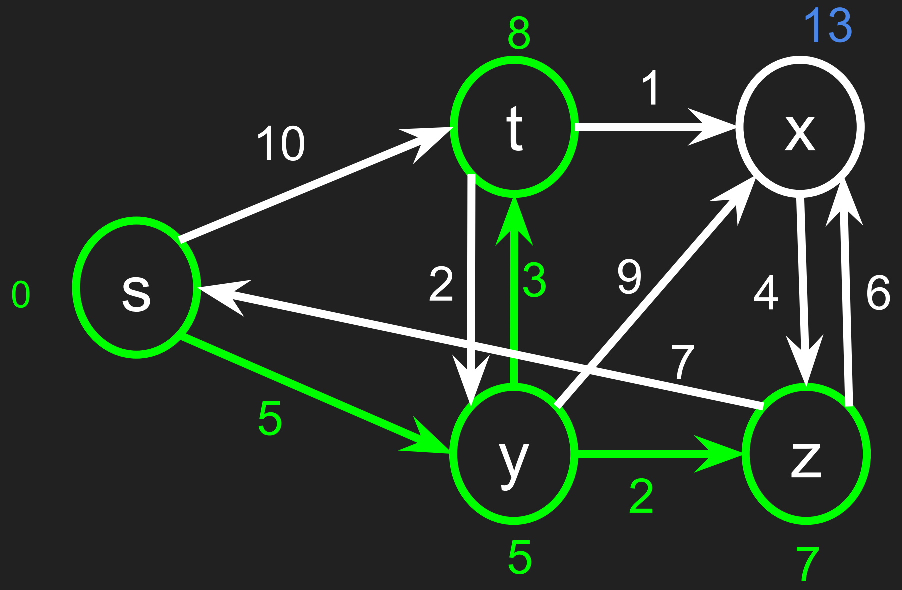
Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	F	8	y
y	T	5	s
x	F	14	y
z	T	7	y

3rd iteration: the distance to x is relaxed



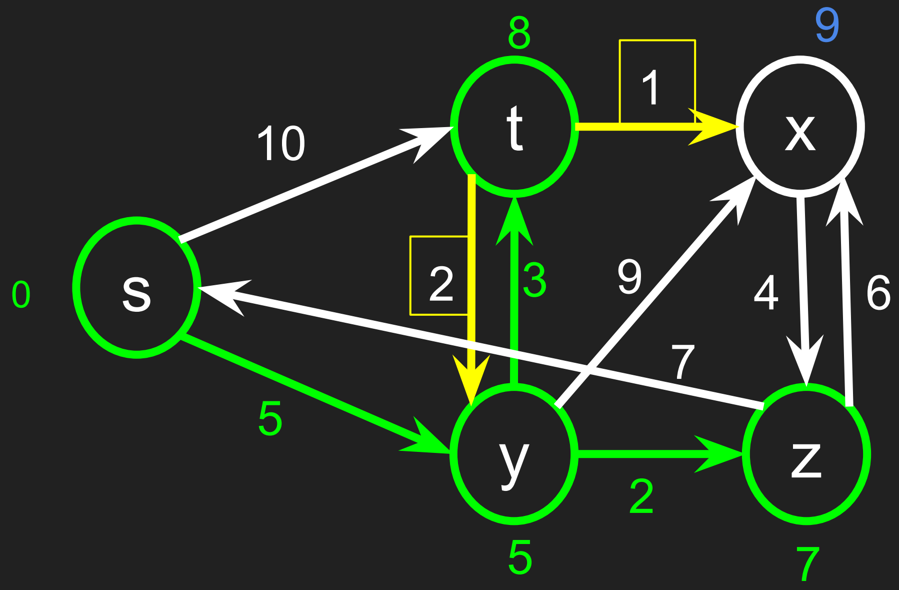
Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	F	8	y
y	T	5	s
x	F	13	z
z	T	7	y

4th iteration: t is visited



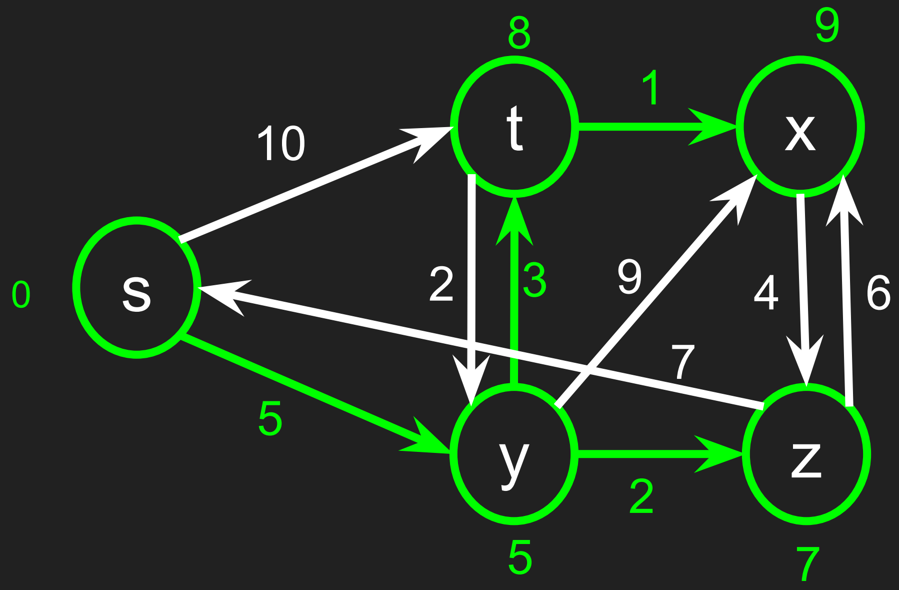
Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	T	8	y
y	T	5	s
x	F	13	z
z	T	7	y

4th iteration: the distance to x is relaxed



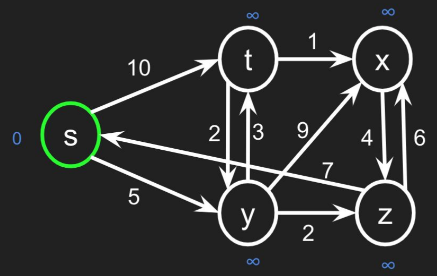
Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	T	8	y
y	T	5	s
x	F	9	t
z	T	7	y

5th and final iteration: x is visited



Key (k)	V[k]	dist[k]	p[k]
s	T	0	null
t	T	8	y
y	T	5	s
x	T	9	t
z	T	7	y

Dijkstra's algorithm



Initialize V with all vertices marked as unvisited,
dist with the source vertex distance value set to 0 and all others set to infinity, and
P with all values set to null.

While V has an unvisited vertex:

- Find an unvisited vertex u that has the minimum distance value in dist.
    - Note: This step is where an optimization can be done using a heap
- Mark that u has been visited in V.
- For each of u's edges to a vertex y:
    - If y is not visited in V,
        - <u>**Relax**</u> the distance in the table for y if necessary.
            - i.e. Select the min(current value for y in dist, new distance for y through u)
        - If the distance for y was relaxed, set y's predecessor value to u in P.

Poll Question #1:

Which of the following statements is not correct about Dijkstra’s algorithm?

A. Dijkstra’s algorithm uses a greedy strategy.
B. The input graph can have negative weight edges.
C. Dijkstra’s algorithm is a single source shortest path algorithm.
D. The input graph can be undirected or directed.

Poll Question #1:

Which of the following statements is not correct about Dijkstra's algorithm?

A. Dijkstra's algorithm uses a greedy strategy.
B. The input graph can have negative weight edges.
C. Dijkstra's algorithm is a single source shortest path algorithm.
D. The input graph can be undirected or directed.

Poll Question #2:

Given the following table of vertices and their predecessors from Dijkstra's algorithm, what is the shortest path between s and x?

A. s, y, t, x
B. s, y, z, x
C. s, t, x
D. s, y, x

Key (k)	d[k]	p[k]
s	0	null
t	8	y
y	5	s
x	9	t
z	7	y

Poll Question #3:

Given the following table of vertices and their predecessors from Dijkstra's algorithm, what is the shortest distance between s and x?

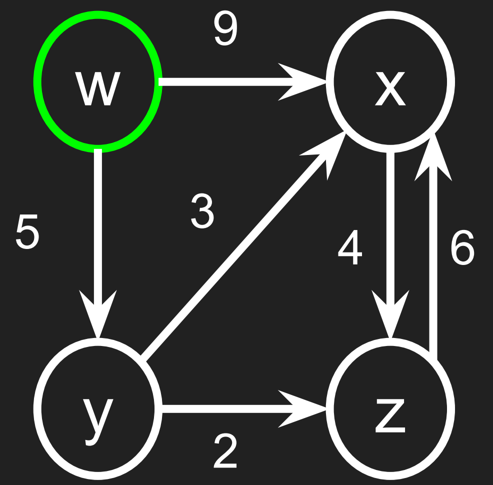
A. 9
B. 8
C. 7
D. 5

Key (k)	d[k]	p[k]
s	0	null
t	8	y
y	5	s
x	9	t
z	7	y

Poll Question #4:

Given the following graph with w as the source vertex, what will the distance values be in alphabetical order after the second iteration of Dijkstra's algorithm?

A. 0, 9, 5, ∞
B. 0, 3, 5, 2
C. 0, 8, 5, 7
D. 0, 9, 5, 2

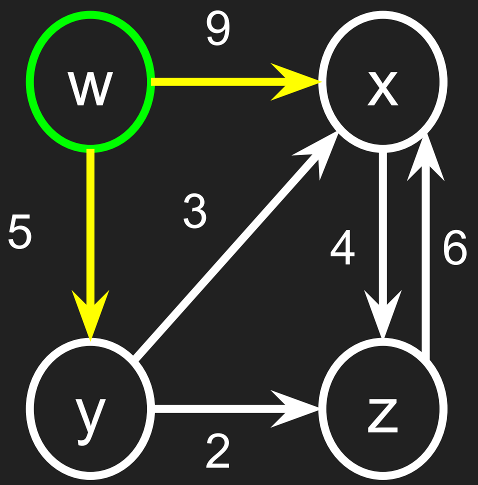


Key (k)	V[k]	dist[k]	p[k]
w	F	0	null
x	F	∞	null
y	F	∞	null
z	F	∞	null

Poll Question #4: After 1st iteration

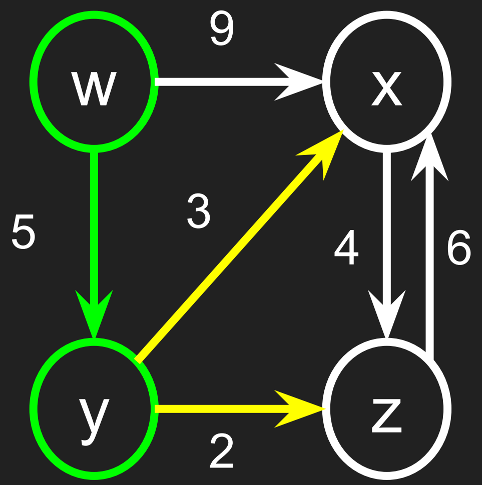
Given the following graph with w as the source vertex, what will the distance values be in alphabetical order after the second iteration of Dijkstra's algorithm?

A. 0, 9, 5, ∞
B. 0, 3, 5, 2
C. 0, 8, 5, 7
D. 0, 9, 5, 2



Key (k)	V[k]	dist[k]	p[k]
w	T	0	null
x	F	9	w
y	F	5	w
z	F	∞	null

Poll Question #4: After 2nd iteration



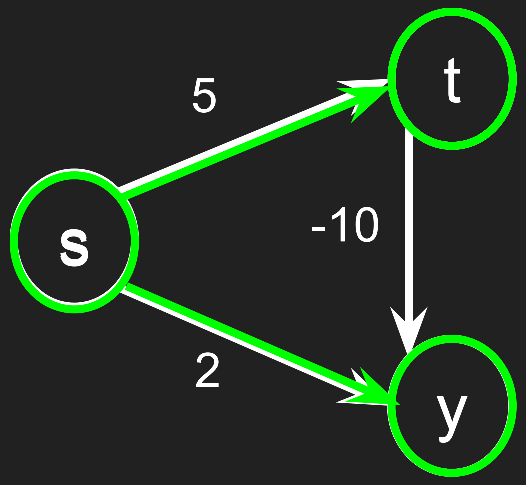
Given the following graph with w as the source vertex, what will the distance values be in alphabetical order after the second iteration of Dijkstra’s algorithm?

A. 0, 9, 5, ∞
B. 0, 3, 5, 2
C. 0, 8, 5, 7
D. 0, 9, 5, 2

Key (k)	V[k]	dist[k]	p[k]
w	T	0	null
x	F	8	y
y	T	5	w
z	F	7	y

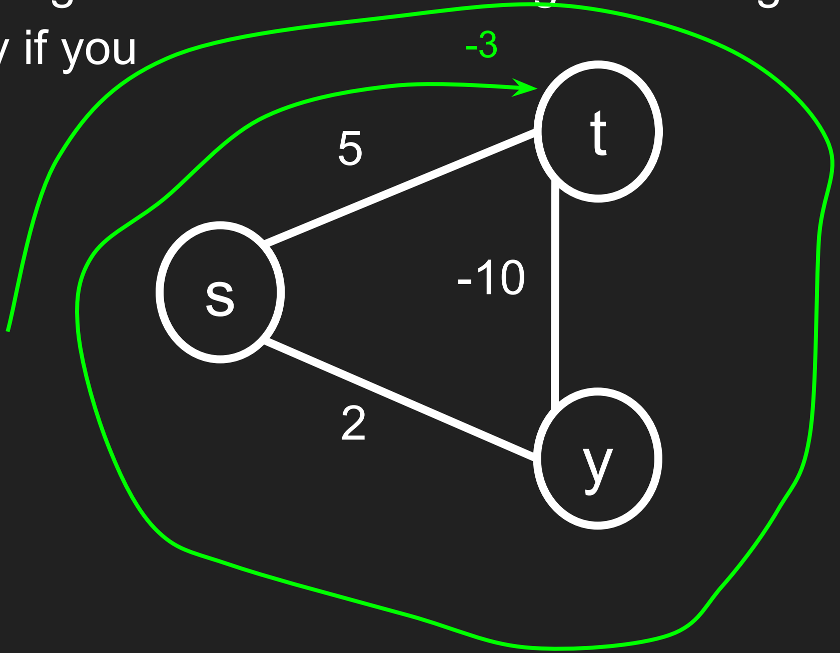
Negative Edge Weight

- Dijkstra's won't find the optimal solution in a graph with negative weights.
  - Due to its Greedy Approach



Negative Weight Cycle

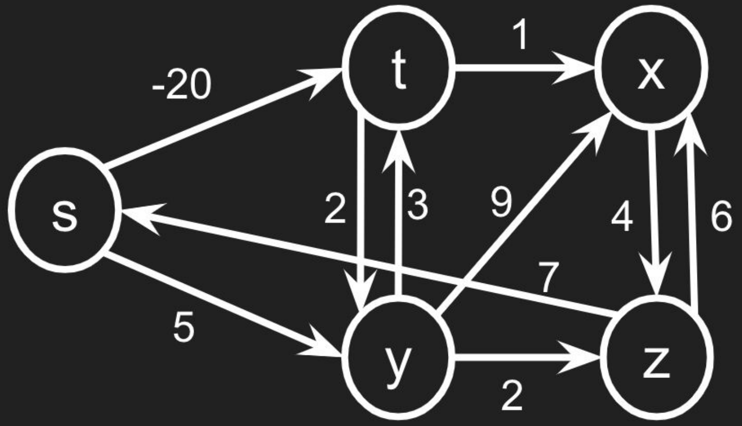
- A cycle with weights that sum to a negative number
- No Shortest-Path Algorithm can handle Negative Weight Cycles
- Makes sense why if you think of weights as "cost" instead of "distance"



Bellman-Ford Algorithm

- Slower ($O(|V|*|E|)$) and more complex, but more versatile than Dijkstra's
  - Can handle Negative Edge Weights
  - Can detect Negative Weight Cycles (NWC)
- Uses a *dist* map like Dijkstra's

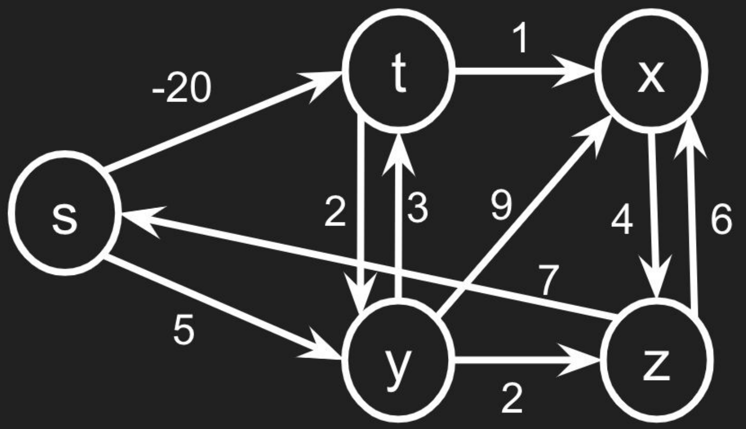
```
for i=0 to |V|
    for each edge (u,v) in E
        relax the path to v in dist
```

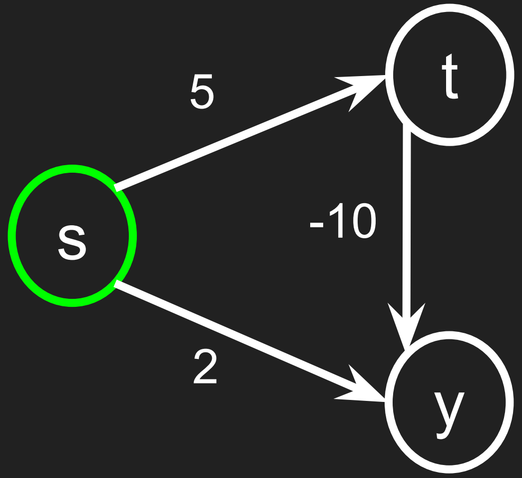


Bellman-Ford Algorithm

- **Relaxes** ALL edges at each iteration
  - Performs *exactly* |V| iterations
    - The *i*th iteration finds any Shortest Paths with i edges
  - Last iteration reveals presence of Negative Weight Cycles
  - The order in which you relax the edges *can* affect runtime performance

```
for i=0 to |V|
     for each edge (u,v) in E
          relax the path to v in dist
```

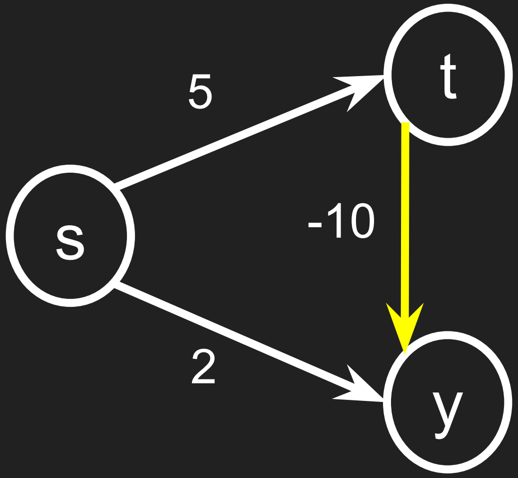




V	d[v]	p[v]
s	0	null
t	∞	null
y	∞	null

Edges: (t,y), (s,t), (s,y)

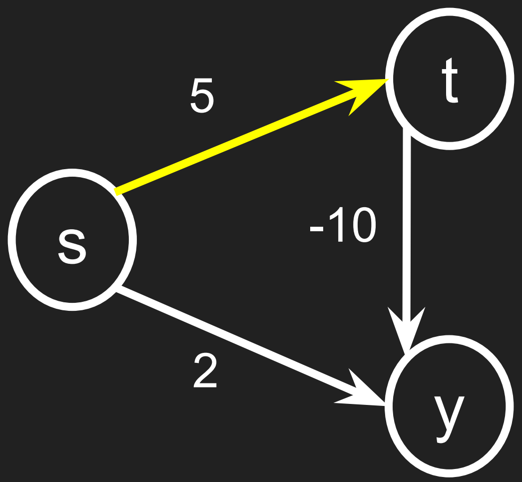
initialize the map.
We will do |V| = 3
iterations, the final one
only checks for NWC



V	d[v]	p[v]
s	0	null
t	∞	null
y	∞	null

Edges: (t,y), (s,t), (s,y)

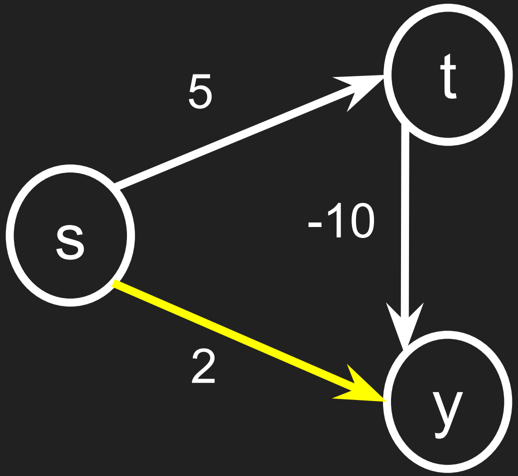
first iteration



V	d[v]	p[v]
s	0	null
t	5	s
y	∞	null

Edges: (t,y), (s,t), (s,y)

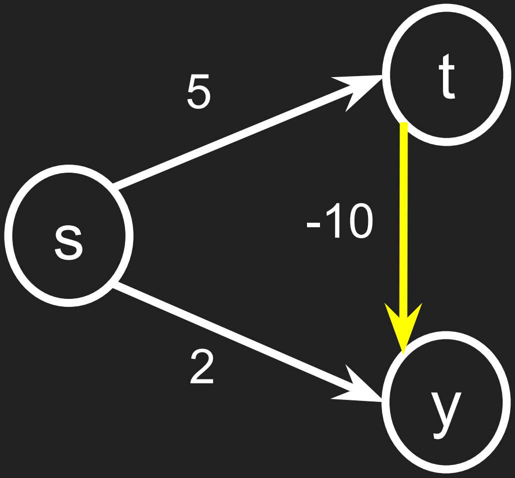
first iteration



V	d[v]	p[v]
s	0	null
t	5	s
y	2	s

Edges: (t,y), (s,t), (s,y)

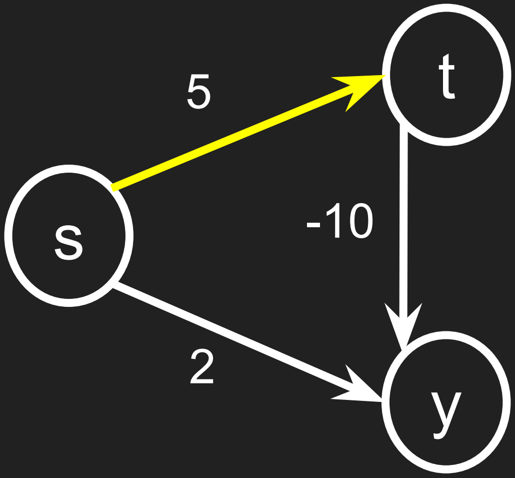
first iteration



V	d[v]	p[v]
s	0	null
t	5	s
y	-5	t

Edges: (t,y), (s,t), (s,y)

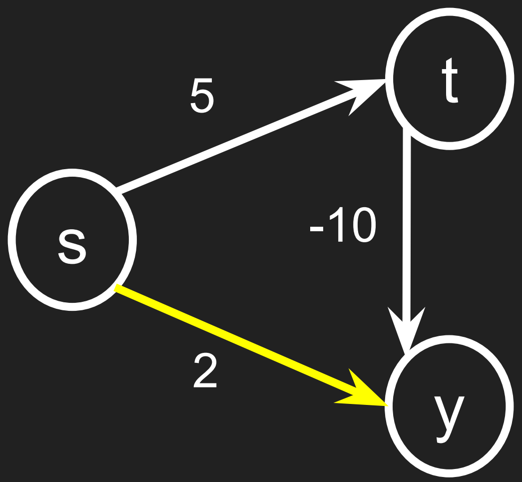
second iteration



V	d[v]	p[v]
s	0	null
t	5	s
y	-5	t

Edges: (t,y), (s,t), (s,y)

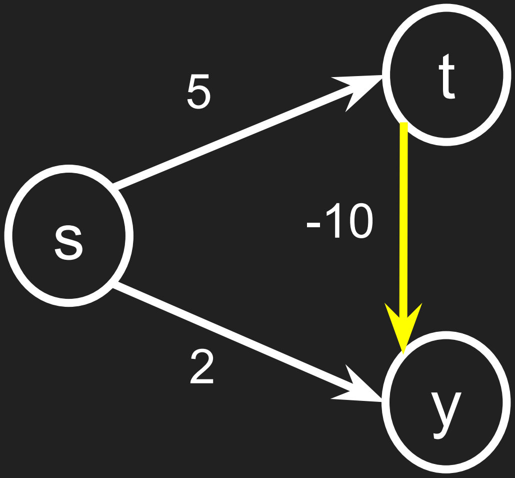
second iteration



V	d[v]	p[v]
s	0	null
t	5	s
y	-5	t

Edges: (t,y), (s,t), (s,y)

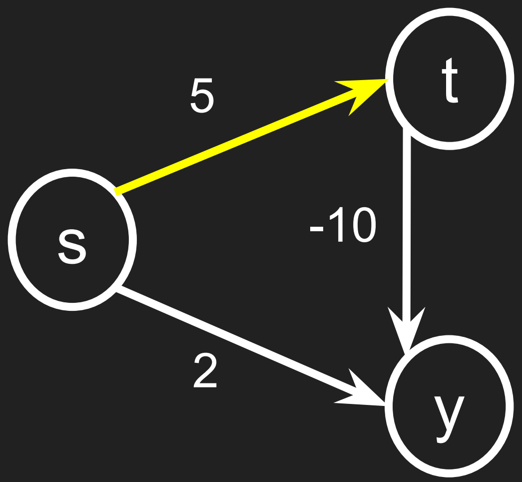
second iteration



V	d[v]	p[v]
s	0	null
t	5	s
y	-5	t

Edges: (t,y), (s,t), (s,y)

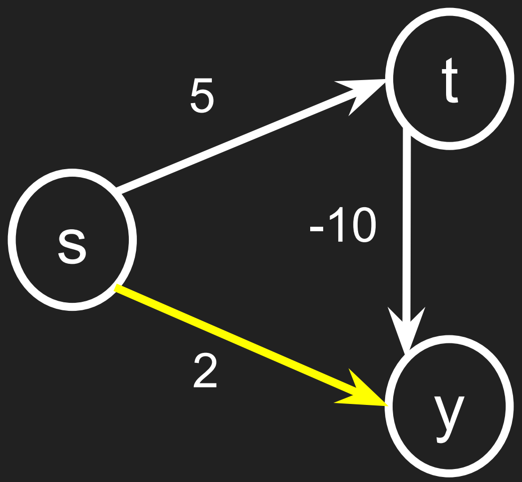
final iteration checks for NWC. Should be NO updates



V	d[v]	p[v]
s	0	null
t	5	s
y	-5	t

Edges: (t,y), (s,t), (s,y)

final iteration checks for NWC. Should be NO updates



V	d[v]	p[v]
s	0	null
t	5	s
y	-5	t

Edges: (t,y), (s,t), (s,y)

final iteration checks for NWC. Should be NO Relaxation

Bellman Ford Pseudocode

// Step 1: Initialize graph and map for distances and predecessors

```
// Step 2: Relax edges repeatedly
repeat |V| - 1 times:
    for each edge (u,v) with weight w in edges do
        if distance[u] + w < distance[v] then
            distance[v] = distance[u] + w
            predecessor[v] = u

// Step 3: Check for negative-weight cycles
for each edge (u,v) with weight w in edges do
    if distance[u] + w < distance[v] then
        error "Graph contains a negative-weight cycle"
```

Relax

Poll Question #5:

Bellman-Ford is able to find the Shortest Path in <u>any</u> graph with negative edge weights

A. True

B. False, not if there is a NWC

Poll Question #6:

Which real-world problem would be a better fit for the Bellman-Ford Algorithm (as opposed to Dijkstra's Algorithm)?

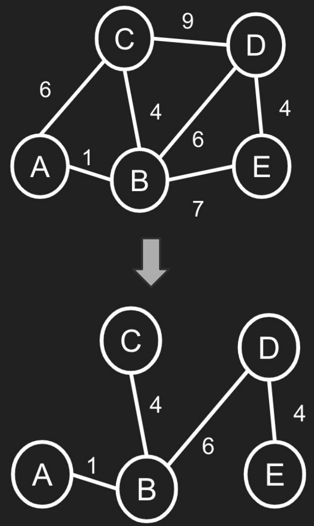
A. Shortest drive from Miami to Seattle
B. Cheapest drive from Miami to Seattle
C. Coldest drive from Miami to Seattle (temp can be negative)
D. Hottest drive from Miami to Seattle (longest Path, different set of Algorithms)

Questions about anything?

Minimum Spanning Tree Algorithms

Minimum Spanning Tree (MST)

- MST is a subset of the edges of a connected, weighted, and undirected graph that:
  - connects all the vertices together
  - without any cycles (acyclic)
  - with the minimum possible total edge weight
- Two famous MST algorithms are:
  - Prims' algorithm (tracks vertices)
  - Kruskal's algorithm (tracks edges)
- Both Prims' and Kruskal's algorithms are **greedy**
- MST of a graph is NOT necessarily unique
  - unique only if the edge weights are unique

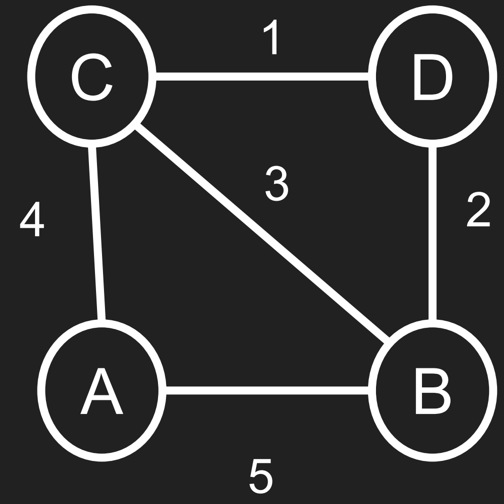


Poll Question #7

What is cost of the MST of the following Graph?

A. 15
B. 11
C. 7
D. 5

Graph:

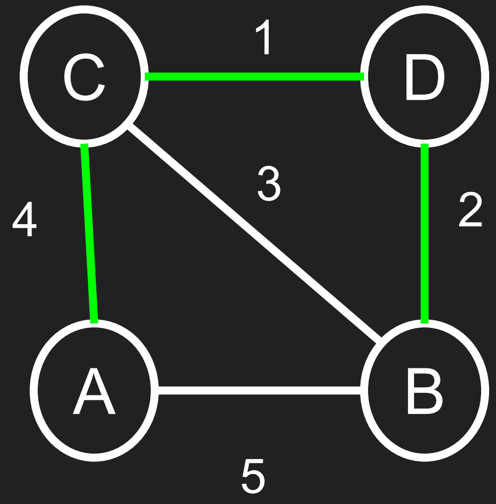


Poll Question #7

What is cost of the MST of the following Graph?

A. 15
B. 11
C. 7
D. 5

Graph:

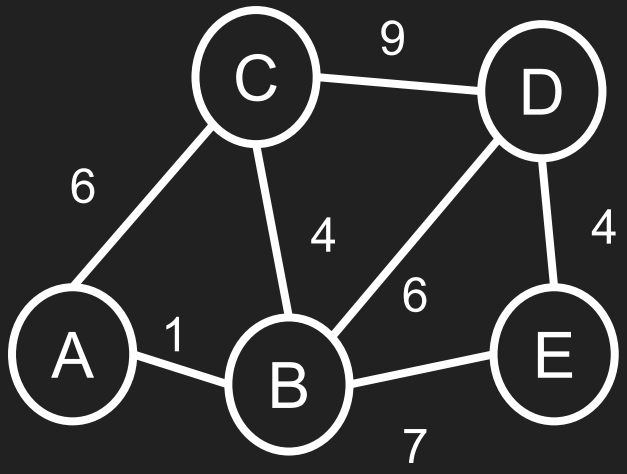


Prim's Algorithm

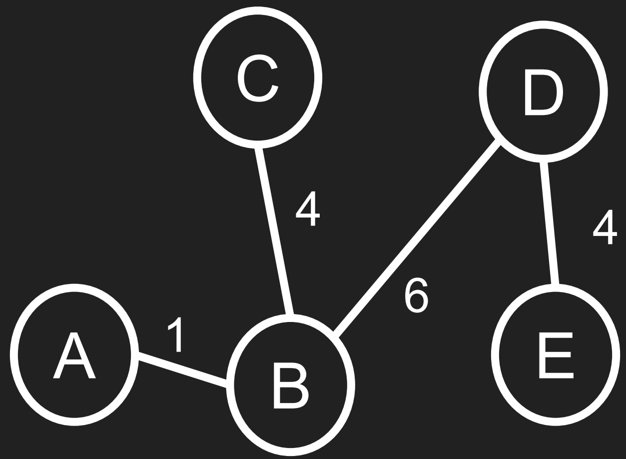
- Given a connected, weighted, and undirected graph Prim's results in a Minimum Spanning Tree (MST)!!

- Time complexity:
  - Adjacency Matrix:
    - O(|V|^2)
  - Adjacency List + heap:
    - O(|E| log(|V|))

Graph:



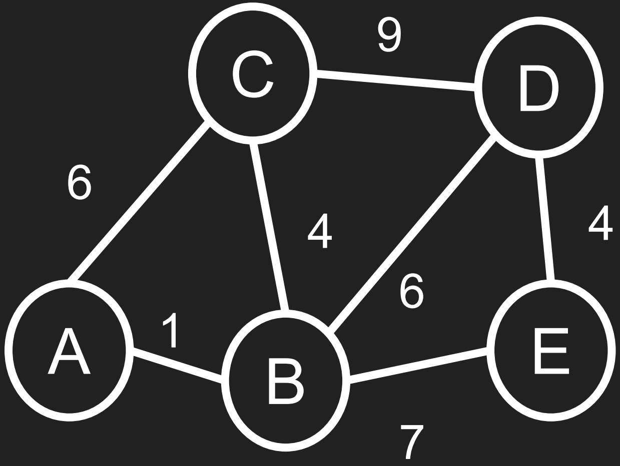
MST:



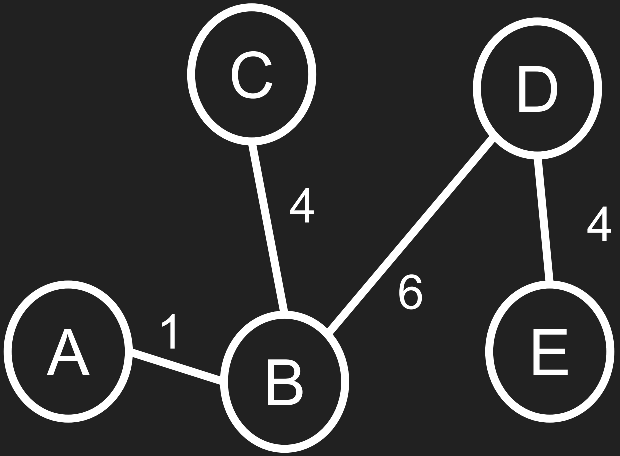
Prim’s Algorithm Pseudocode

1. Create an empty set
2. Put an arbitrary node in the set
3. While your set does not contain all vertices of the graph
   a. Add an adjacent vertex:
      i. with least edge weight
      ii. is not in the set already

Graph:



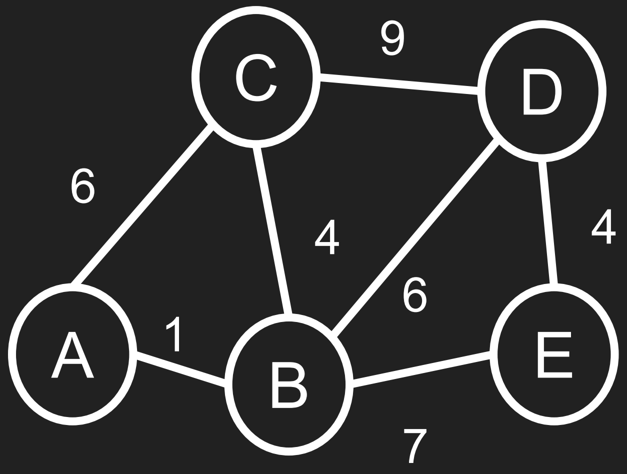
MST:



Prim's Algorithm Example

- Create a set (initially empty) of MST vertices

Graph:



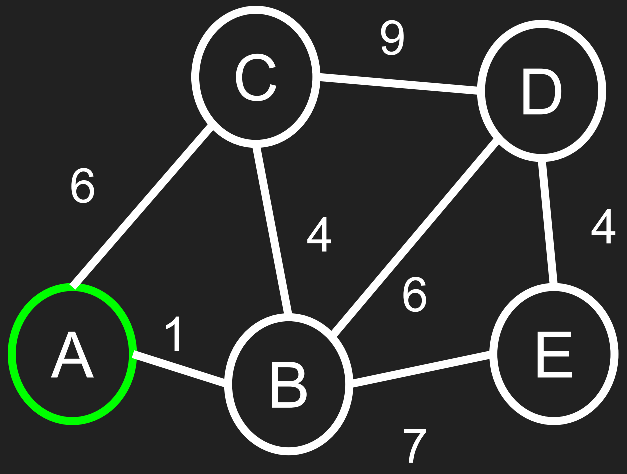
Set: {}

MST:

Prim’s Algorithm Example

- Pick any vertex (in this case A) and add it to your set

Graph:



Set: {A}

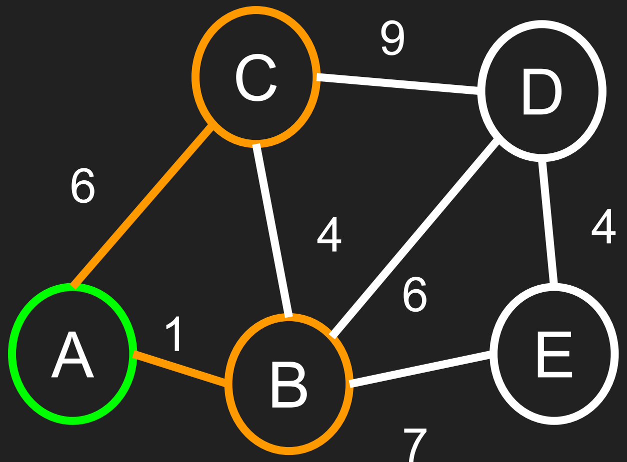
MST:

Prim's Algorithm Example

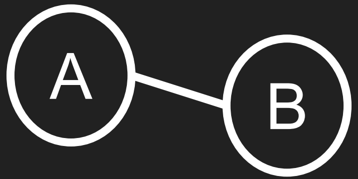
- While your set does not contain all vertices of the graph
  - Add an adjacent vertices:
    - with least edge weight
    - is not in the set already
- The adjacent vertices are: B, C. B's edge is the least weighted, so it is added to our set.

Set: {A, B}

Graph:



MST:

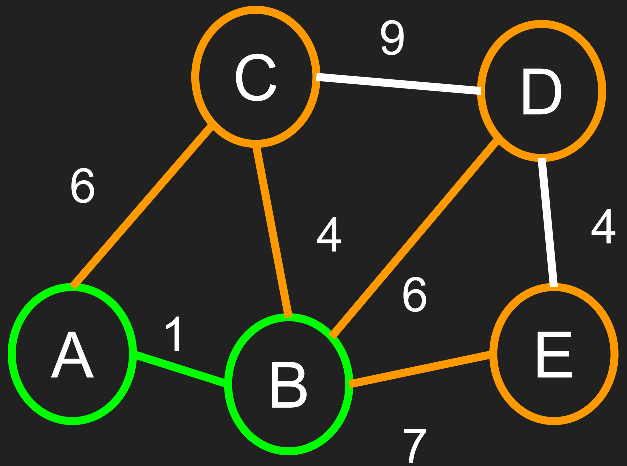


Prim’s Algorithm Example

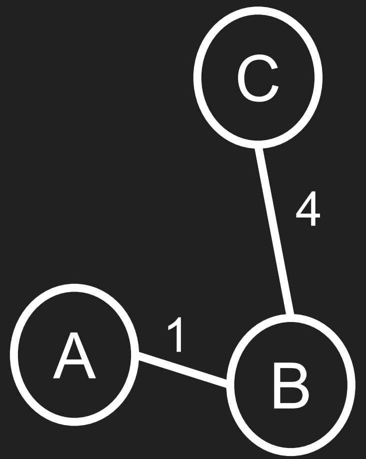
- While your set does not contain all vertices of the graph
  - Add an adjacent vertices:
    - with least edge weight
    - is not in the set already
- The adjacent vertices are: C, D, and E. C’s edge to B is the least weighted, so it is added to our set.

Set: {A, B, C}

Graph:



MST:

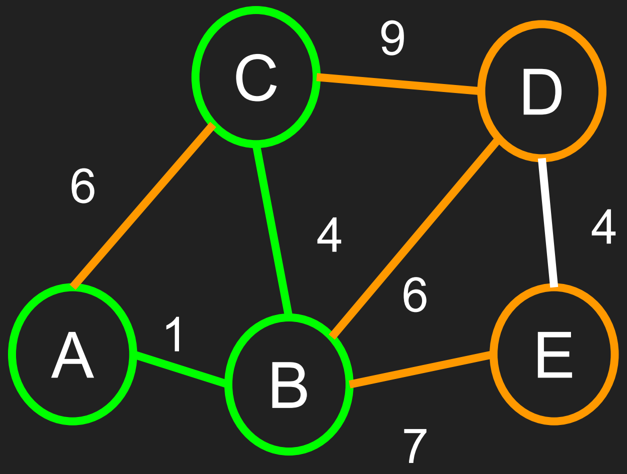


Prim's Algorithm Example

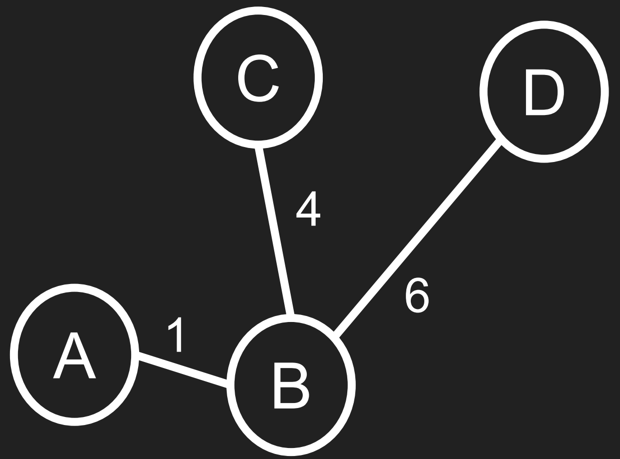
- While your set does not contain all vertices of the graph
  - Add an adjacent vertices:
    - with least edge weight
    - is not in the set already
- The adjacent vertices are: D, and E. D's edge to B is the least weighted, so it is added to our set.

Set: {A, B, C, D}

Graph:



MST:

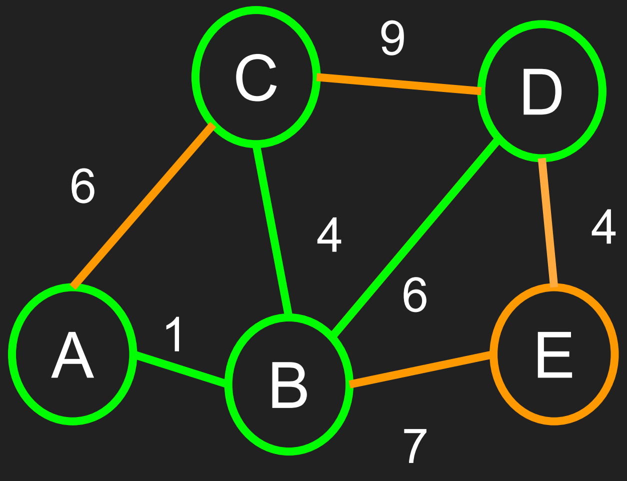


Prim's Algorithm Example

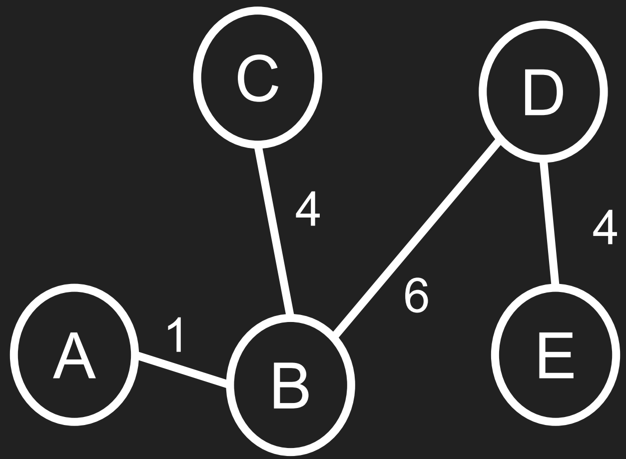
- While your set does not contain all vertices of the graph
  - Add an adjacent vertices:
    - with least edge weight
    - is not in the set already
- The only adjacent vertice is E E's edge to D is the least weighted, so it is added to our set.

Set: {A, B, C, D, E}

Graph:



MST:

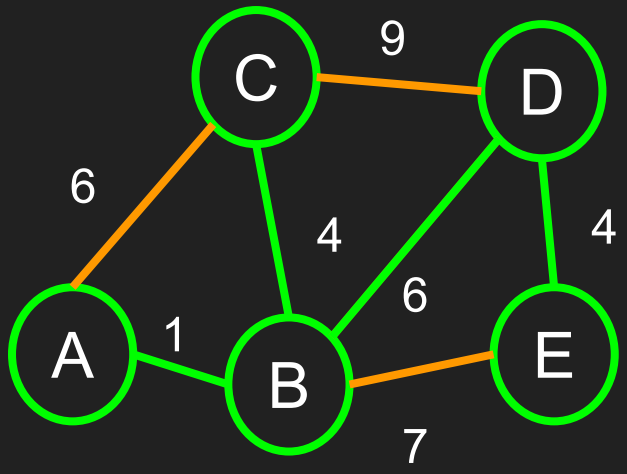


Prim's Algorithm Example

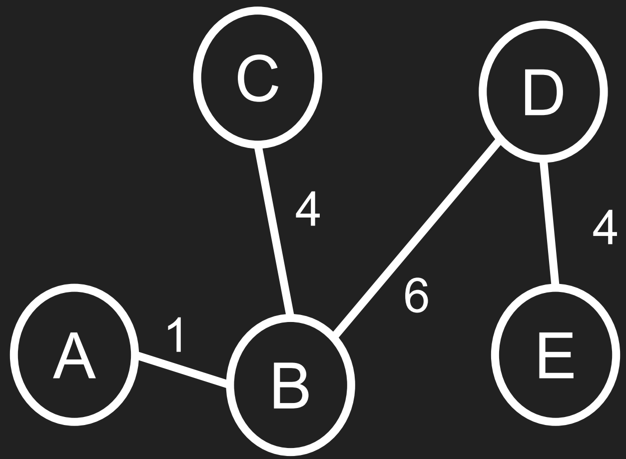
- Our set includes all vertices, so we exit the while loop and our MST is complete.

Set: {A, B, C, D, E}

Graph:



MST:



Poll Question #8:

You can use Prim’s Algorithm to find the MST of a graph with which of the following properties?

1. Cyclic
2. Weighted
3. Directed
4. Undirected
5. Connected

Poll Question #8:

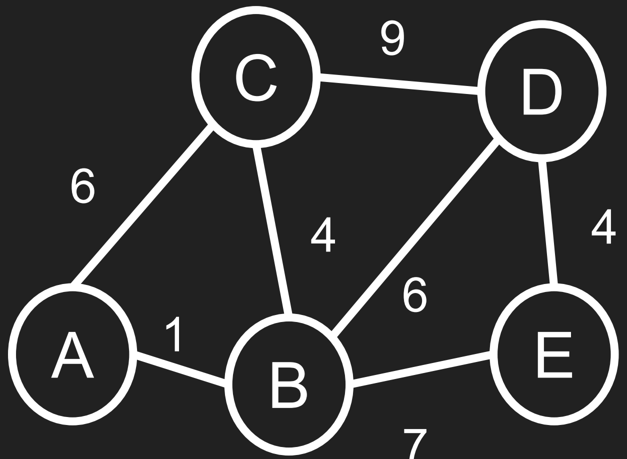
You can use Prim's Algorithm to find the MST of a graph with which of the following properties?

1. Cyclic
2. Weighted
3. Directed
4. Undirected
5. Connected

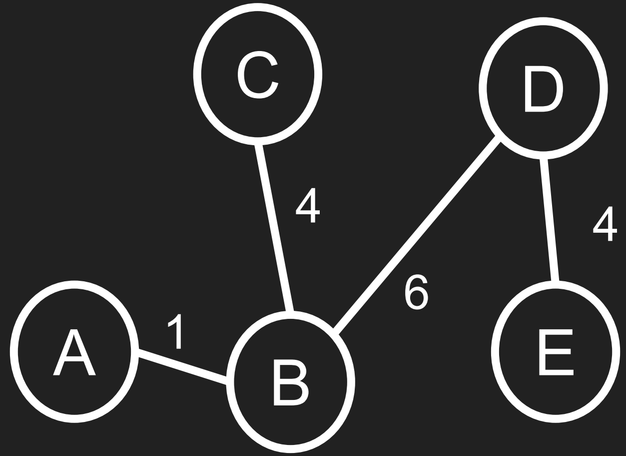
Kruskal's Algorithm

- Given a connected, weighted, and undirected graph Kruskal's results in a MST
- Kruskal's forms a MST by finding the least weighted edges that do NOT form cycles (as opposed to vertices with Prim's)
- Time complexity:
  - O(|E|log(|E|)), the limiting factor is sorting the edges

Graph:

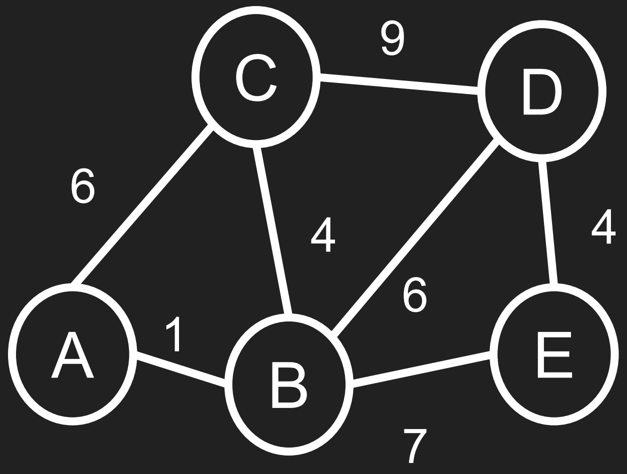


MST:



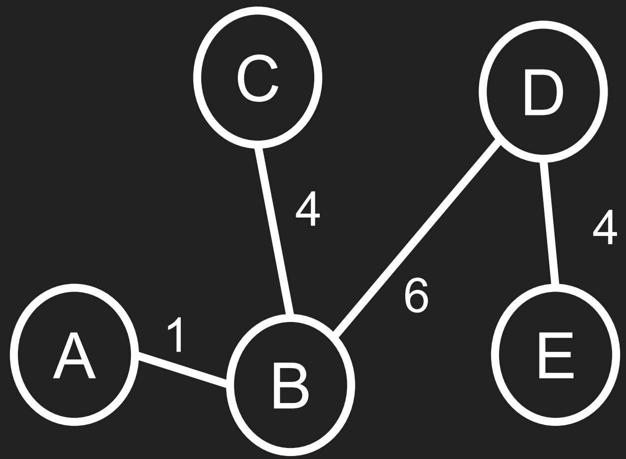
Kruskal’s Algorithm Pseudocode

Graph:



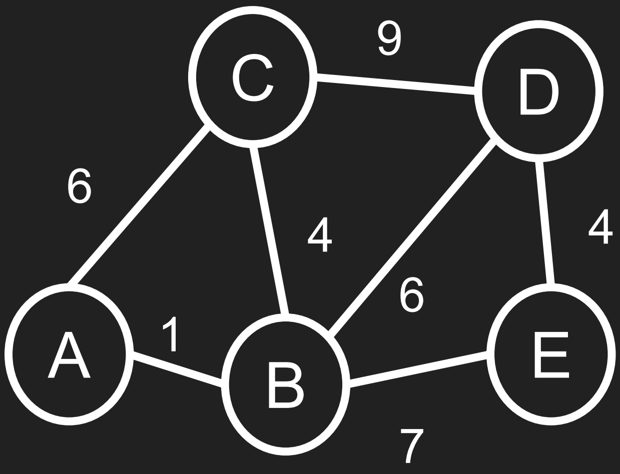
- Create an empty set of edges
- Create a list of the edges sorted by least weight
- While all the vertices are NOT connected by the edges in our set:
  - Add the least weighted edge that does not form a cycle with the other edges in the set (One way to do this is with union find)

MST:



Kruskal’s Algorithm Example

Graph:



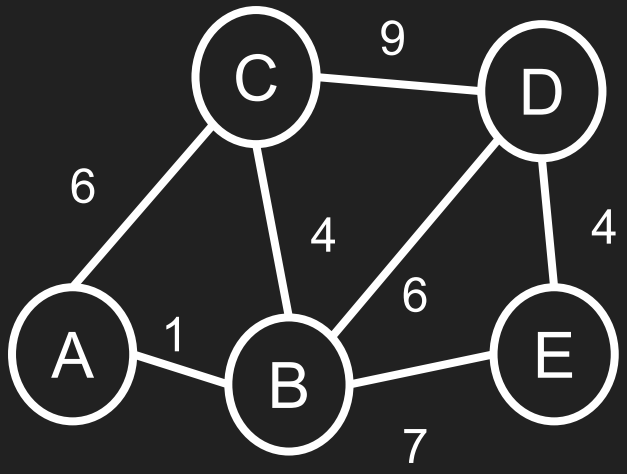
- Create an empty set of edges

Set: {}

MST:

Kruskal’s Algorithm Example

Graph:



- Create a list of the edges sorted by least weight

Edges:

A-B, C-B, D-E, A-C, B-D, B-E, C-D

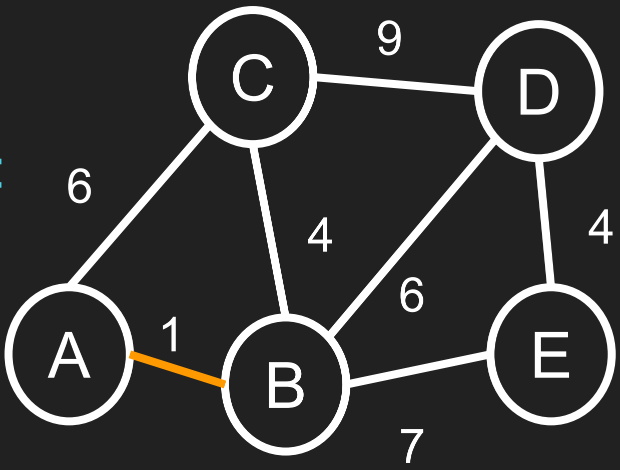
MST:

Set: {}

Kruskal’s Algorithm Example

- A-B is the least weighted edge, and it does not form a cycle
  - Add it to the set

Graph:

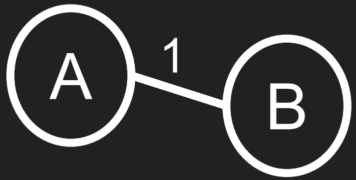


Edges:

A-B, C-B, D-E, A-C, B-D, B-E, C-D

Set: {A-B}

MST:



Kruskal’s Algorithm Example

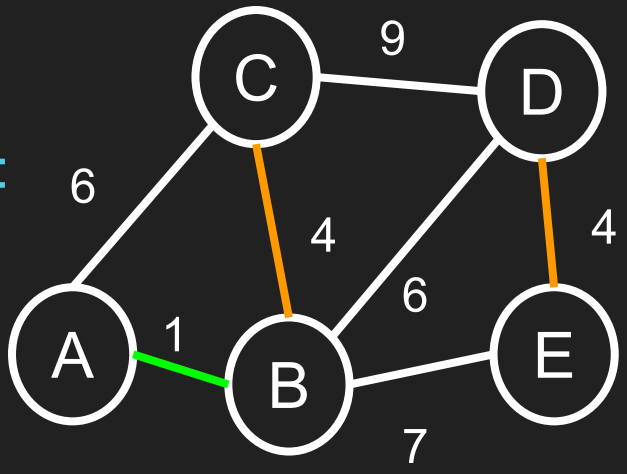
- C-B is tied with D-E for the least weighted edge that does NOT form a cycle, so it does not matter which you pick.
- We will add C-B

Edges:

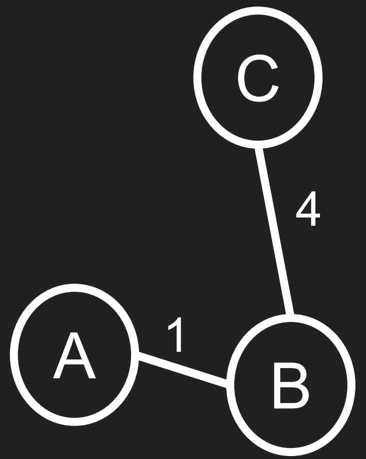
C-B, D-E, A-C, B-D, B-E, C-D

Set: {A-B, C-B}

Graph:



MST:



Kruskal’s Algorithm Example

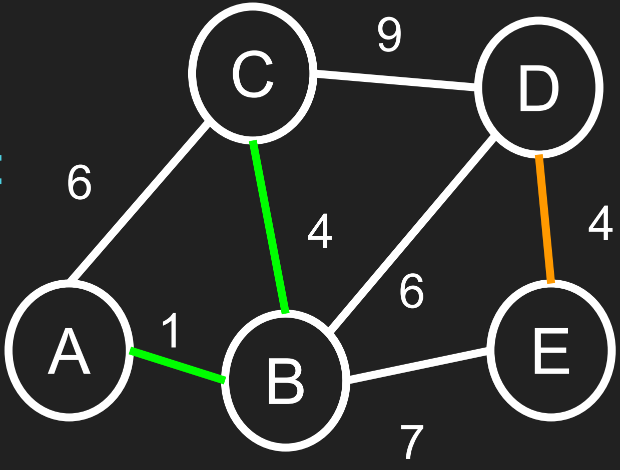
- D-E is the least weighted edge, and it does not form a cycle
  - Add it to the set

Edges:

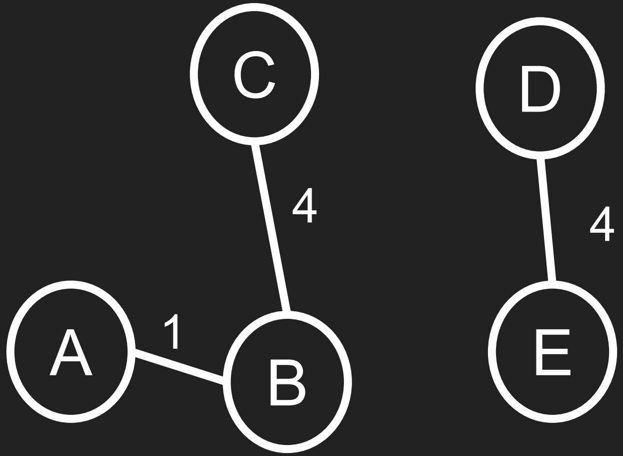
D-E, A-C, B-D, B-E, C-D

Set: {A-B, C-B, D-E}

Graph:



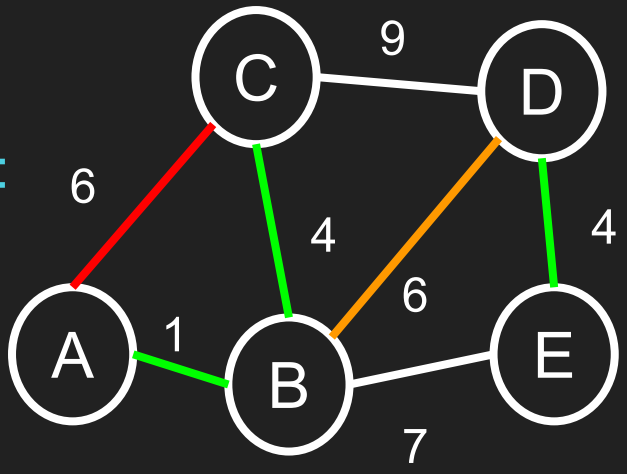
MST:



Kruskal’s Algorithm Example

- A-C is tied with B-D for the least weighted edge that does not form a cycle, BUT adding A-C would form a cycle, so we add B-D

Graph:

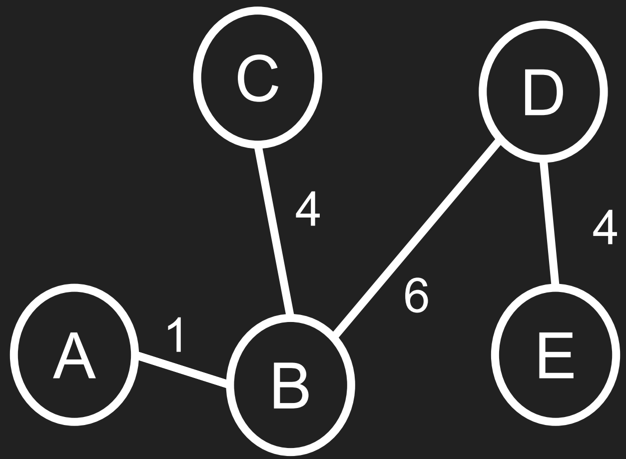


Edges:

A-C, B-D, B-E, C-D

Set: {A-B, C-B, D-E, B-D}

MST:



Kruskal’s Algorithm Example

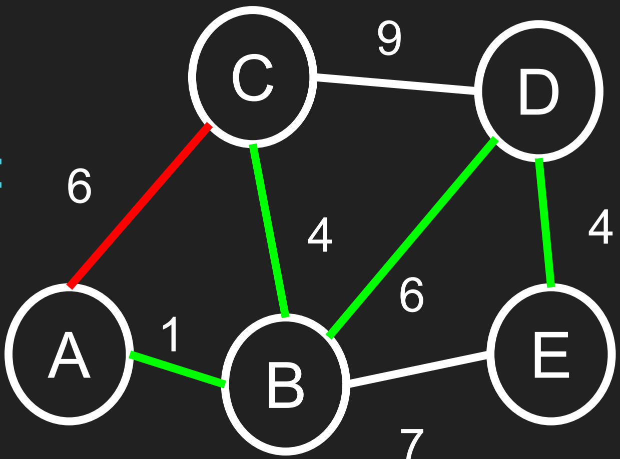
- All vertices have been connected, so we are done!

Edges:

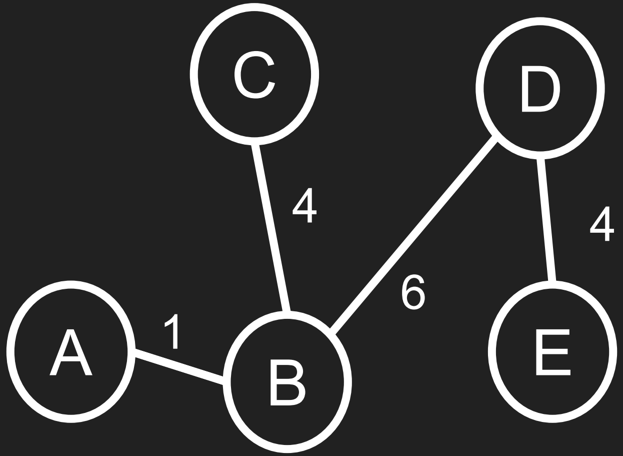
B-E, C-D

Set: {A-B, C-B, D-E, B-D}

Graph:



MST:

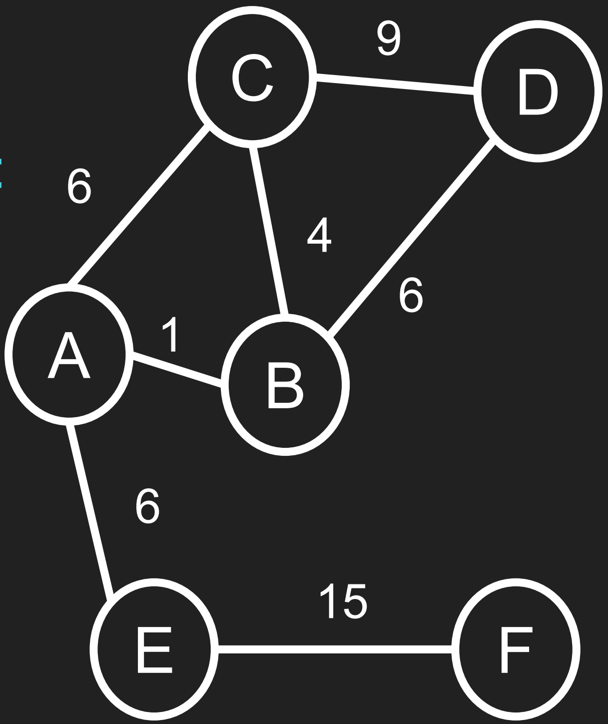


Poll Question #9

What is the MST of the following Graph:

1. (A, B), (C, B), (B, D), (A, E), (E, F)
2. (A, B), (A, C), (C, D), (A, B), (A, E)
3. (A, B), (A, C), (B, D), (A, E), (E, F)

Graph:

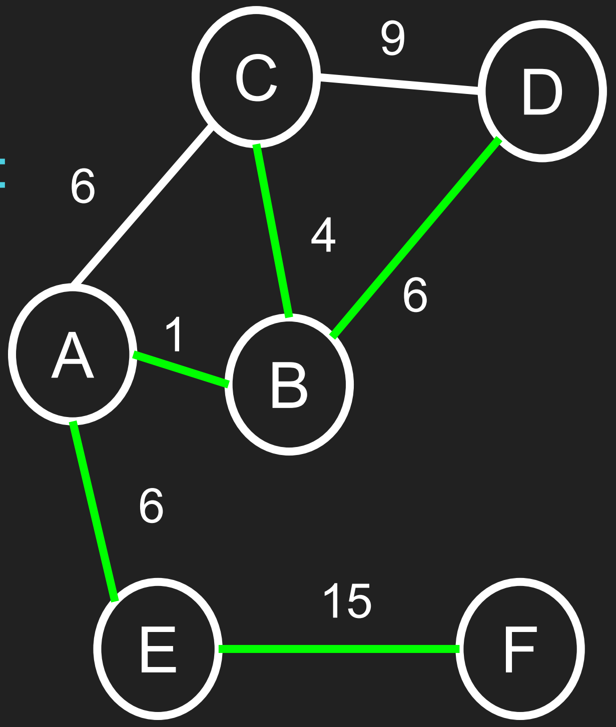


Poll Question #9

What is the MST of the following Graph:

1. (A, B), (C, B), (B, D), (A, E), (E, F)
2. (A, B), (A, C), (C, D), (A, B), (A, E)
3. (A, B), (A, C), (B, D), (A, E), (E, F)

Graph:

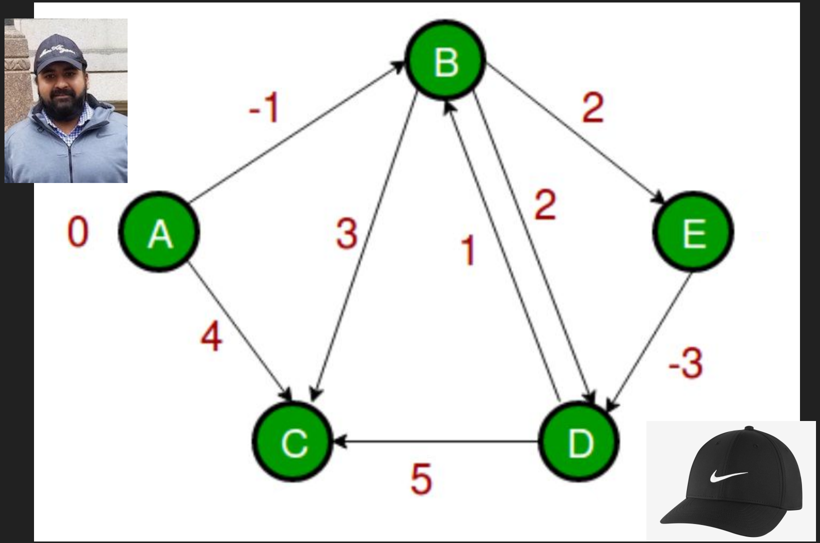


Participation Activity

Prof Aman has lost his favorite hat!! Use Bellman-Ford to help Prof Aman find the shortest path to it (A to D).

Note that this is a digraph.

A	B	C	D	E
0	∞	∞	∞	∞

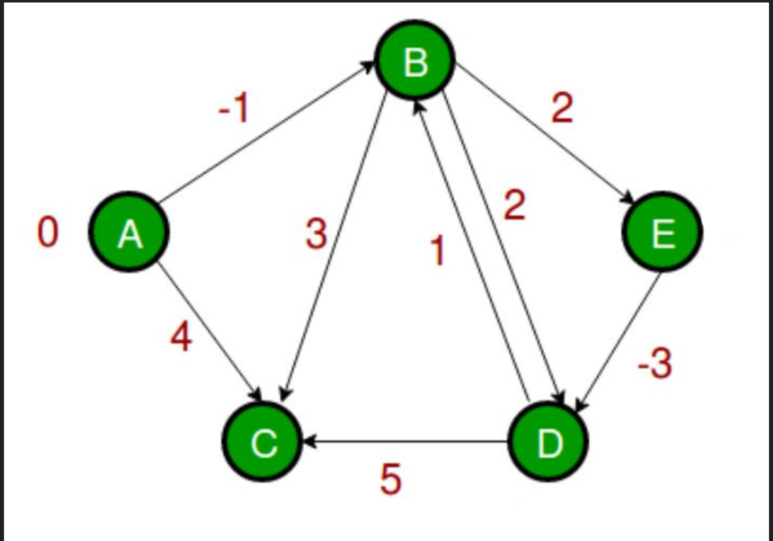


Use edge order: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

Participation Activity

Prof Aman has lost his favorite hat!! Use Bellman-Ford to help Prof Aman find the shortest path to it (A to D).

A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞

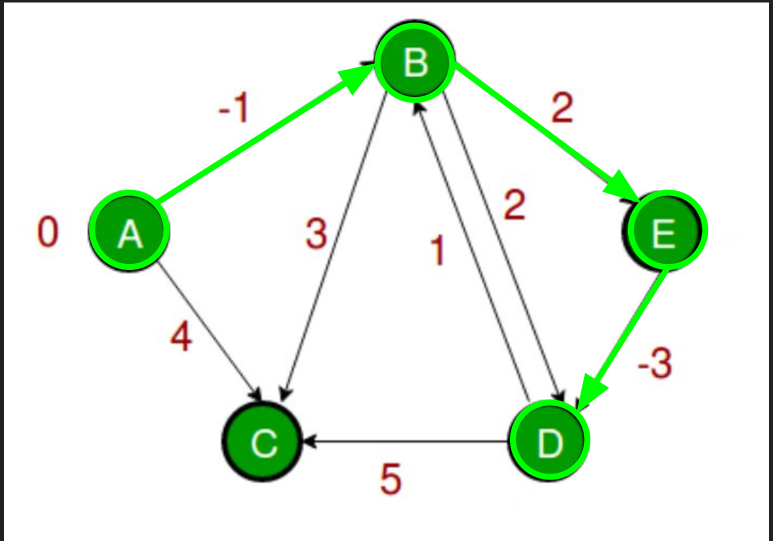


Iteration 1: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D)

Participation Activity

Prof Aman has lost his favorite hat!! Use Bellman-Ford to help Prof Aman find the shortest path to it (A to D).

A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1



Iteration 2: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D). **Iterations 3 and 4** won't update table.

Bellman Ford Walkthrough Video

https://www.youtube.com/watch?v=obWXjtg0L64

Extra Resources

[Min Heap Implementation of Dijkstra's Algorithm]

[Dijkstra's vs Bellman Ford]

GeeksforGeeks graph type overview:
https://www.geeksforgeeks.org/graph-types-and-applications/

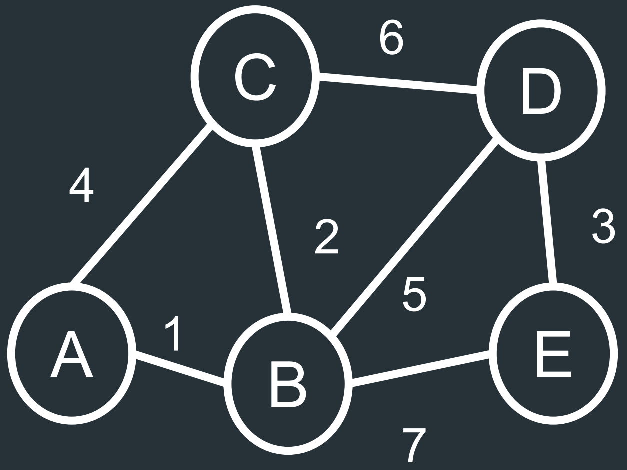
Shortest path algorithms: AFoolsPath

https://towardsdatascience.com/algorithm-shortest-paths-1d8fa3f50769

https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

Reminder:
You have an HonorLock quiz due this Friday by 11:59pm

Detecting Cycles with the Union-Find Data Structure



MAKE-SET(x)

UNION(x, y)

FIND-SET(x)

Solution to collaborative question

Using disjoint sets:

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v of V
  MAKE-SET(v)
sort the edges of E into nondecreasing order by weight w
for each edge (u, v)  E, taken in nondecreasing order by weight
  if FIND-SET(u) ≠ FIND-SET(v)
      then A ← A {(u, v)}
      UNION(u, v)
return A
```

Extra Material