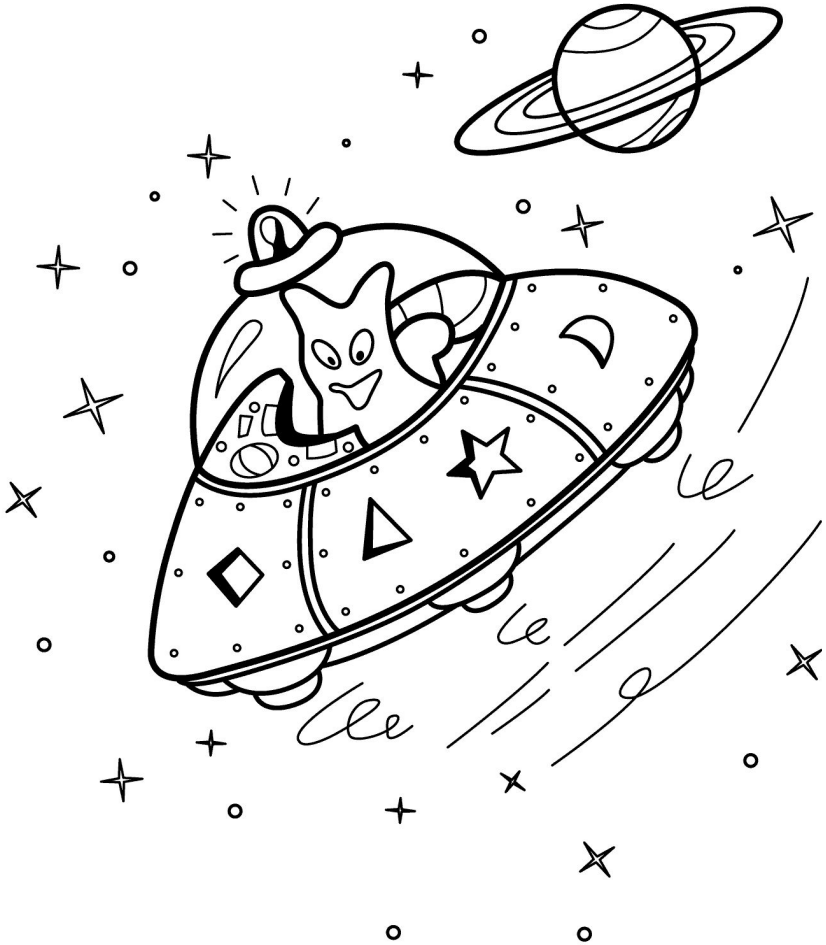


We'll start on Berkeley time, but till then feel free to ask questions, color this lil alien, or ponder this riddle:

What do a dollar and the moon have in common?

Shortest Paths and MSTs

Discussion 10

Announcements

Week 10

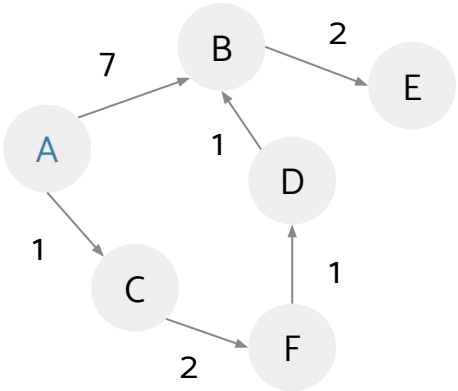
- **Project 3: BYOW partnership exemption form: due Monday (3/29)**
- Project 2: Gitlet presentation (Testing/Debugging): at 1:00 PM on Tuesday 3/30
- Project 2: Gitlet presentation (Merge): at 4:00 PM on Tuesday 3/30
- **Midterm 2 regrade requests:** open Monday 6PM, **due Wednesday 3/31**
- **Project 3: BYOW partnership form: due Wednesday 3/31**
- Project 2: Gitlet full grader logs available on Gradescope starting Thursday 4/1
- **Project 2: Gitlet due Friday 4/2**

Content Review

Dijkstra's Algorithm

We previously learned that BFS can help us find paths from the start to other nodes with the minimum number of edges. However, neither BFS nor DFS accounted for finding the shortest paths based off edge weight.

Dijkstra's algorithm is a method of finding the shortest path from one node to every other node in the graph. You use a priority queue that sorts vertices based off of their distance to the root node.



Steps:

1. Pop node from the top of the queue - this is the current node.
2. Add/update distances of all of the children of the current node in the PQ.
3. Re-sort the priority queue (technically the PQ does this itself).
4. Finalize the distance to the current node from the root.
5. Repeat while the PQ is not empty.

Dijkstra's Algorithm

Dijkstra's can be thought of as a traveler without a map, who arrives at one city and reads a street sign to see what roads there are to other cities, and how long they are (similar to how we pop off a node, and then update the distances).

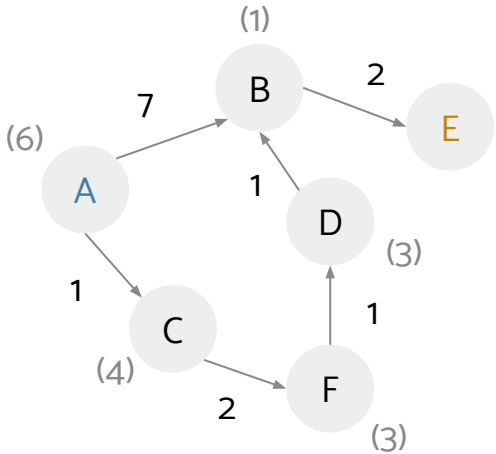
The traveler doesn't accept a distance as final however, until it is clear there are no potential shorter routes (similar to how we only finalize a node once we pop it off the queue which means it had the next shortest distance).

A*

A* is a method of finding the shortest path from one node to a specific other node in the graph. It operates similarly to Dijkstra's except for that we use a (given) heuristic to estimate a vertex's distance from the goal.

Steps:

1. Pop node from the top of the queue - this is the current node.
2. Add/update distances of all of the children of the current node. This distance will be the sum of the distance up to that child node and our guess of how far away the goal node is (our heuristic).
3. Re-sort the priority queue.
4. Check if we've hit the goal node (if so we stop).
5. Repeat while the PQ is not empty.

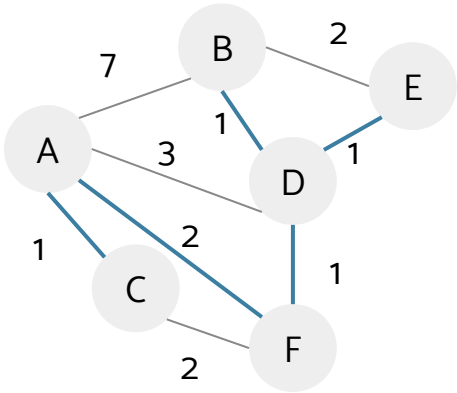


Minimum Spanning Trees

Minimum Spanning Trees are set of edges that connect all the nodes in a graph while being of the smallest possible weight.

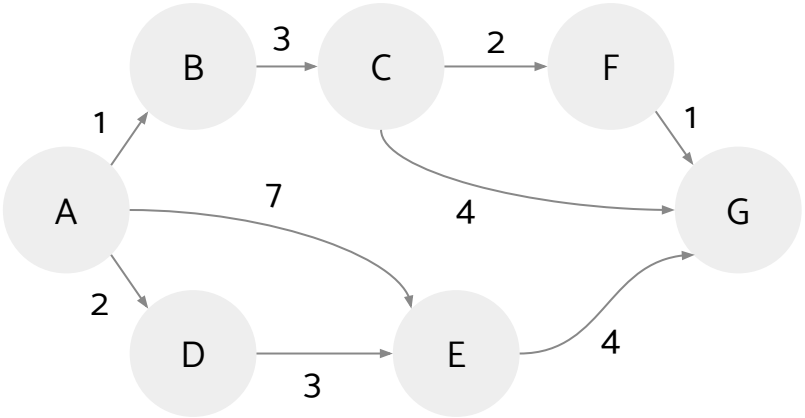
MSTs may not be unique if there are multiple edges of the same weight.

There are two main algorithms for finding MSTs in this class:
Prim's and Kruskal's.



Worksheet

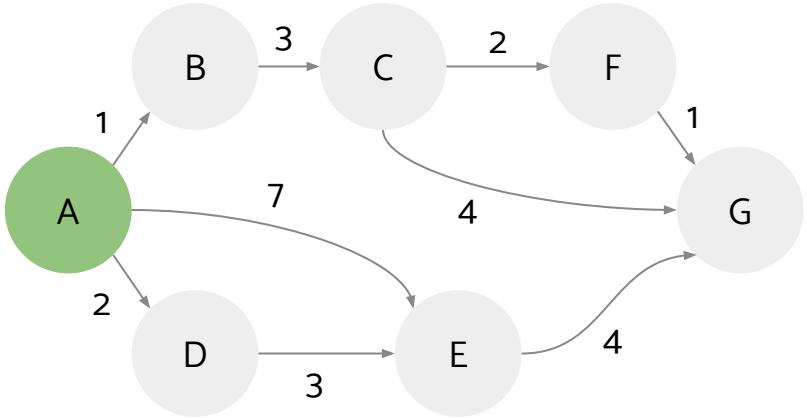
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	∞	∞	∞	∞	∞	∞	∞
EdgeTo	-	-	-	-	-	-	-

Priority Queue:
A : 0

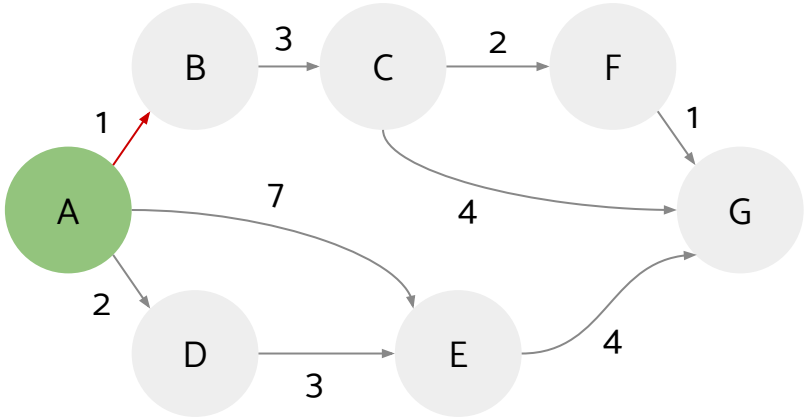
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	∞	∞	∞	∞	∞	∞
EdgeTo	-	-	-	-	-	-	-

Priority Queue:

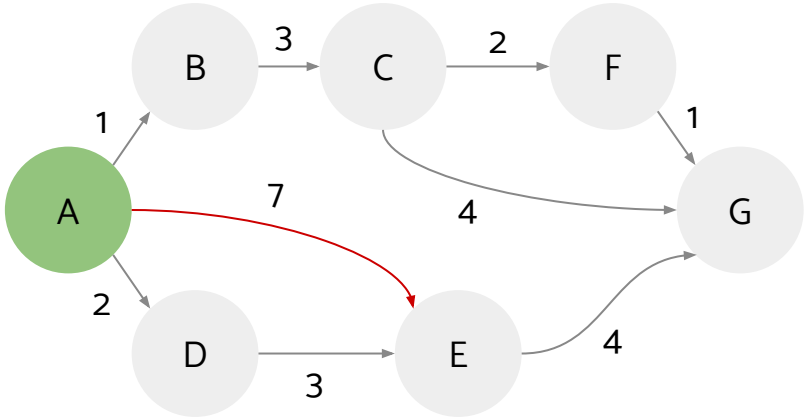
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	∞	∞	∞	∞	∞
EdgeTo	-	A	-	-	-	-	-

Priority Queue:
B : 1

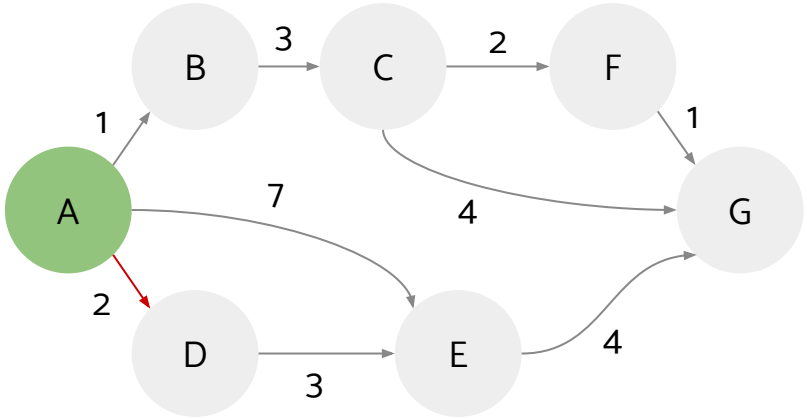
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	∞	∞	7	∞	∞
EdgeTo	-	A	-	-	A	-	-

Priority Queue:
B : 1
E : 7

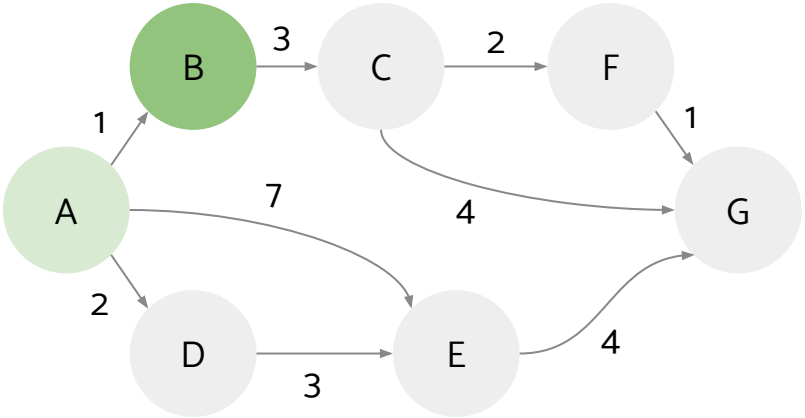
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	∞	2	7	∞	∞
EdgeTo	-	A	-	A	A	-	-

Priority Queue:
B : 1
D : 2
E : 7

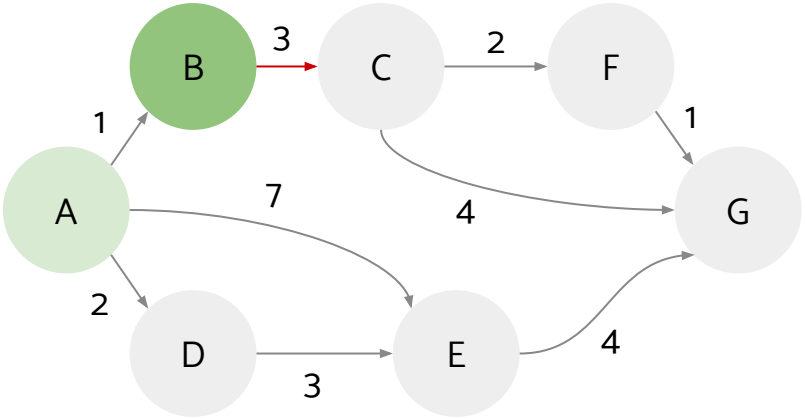
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	∞	2	7	∞	∞
EdgeTo	-	A	-	A	A	-	-

Priority Queue:
D : 2
E: 7

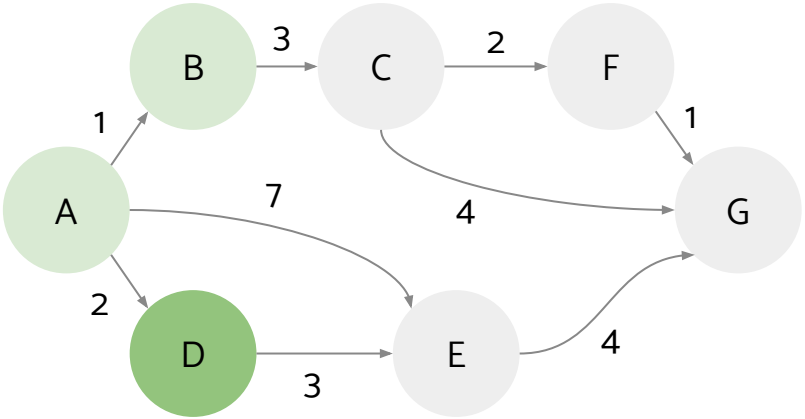
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	∞	∞
EdgeTo	-	A	B	A	A	-	-

Priority Queue:
D : 2
C : 4
E: 7

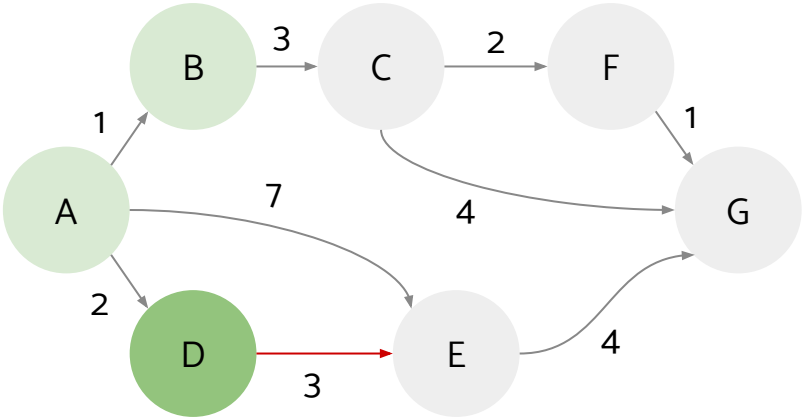
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	∞	∞
EdgeTo	-	A	B	A	A	-	-

Priority Queue:
C : 4
E: 7

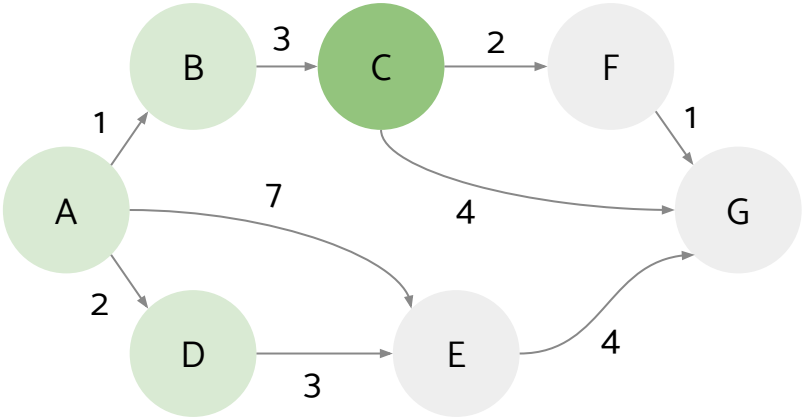
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	∞	∞
EdgeTo	-	A	B	A	D	-	-

Priority Queue:
C : 4
E : 5

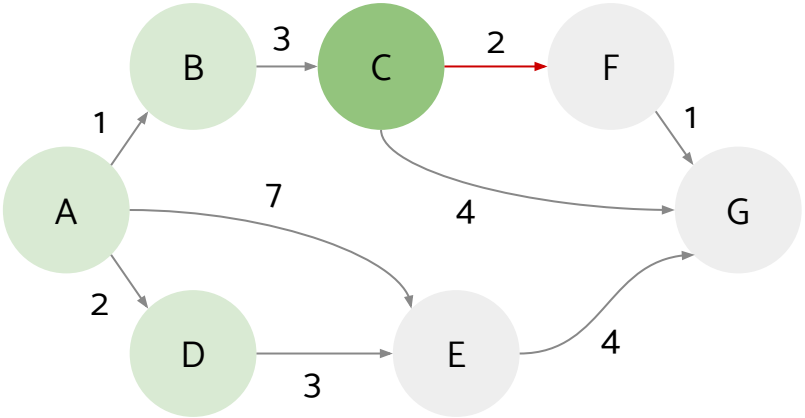
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	∞	∞
EdgeTo	-	A	B	A	D	-	-

Priority Queue:
E : 5

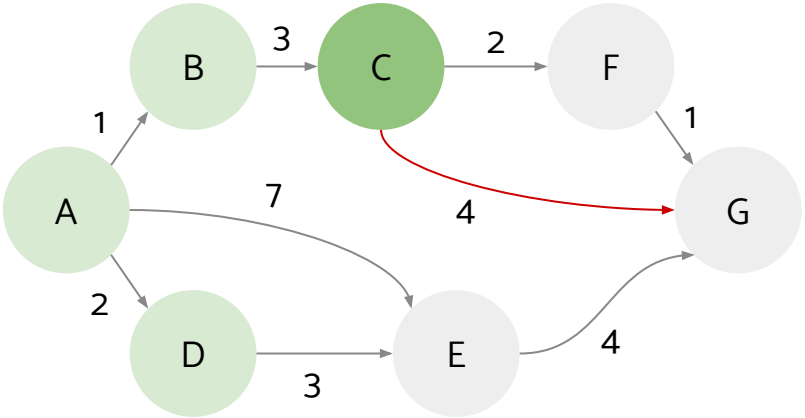
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	∞
EdgeTo	-	A	B	A	D	C	-

Priority Queue:
E : 5
F : 6

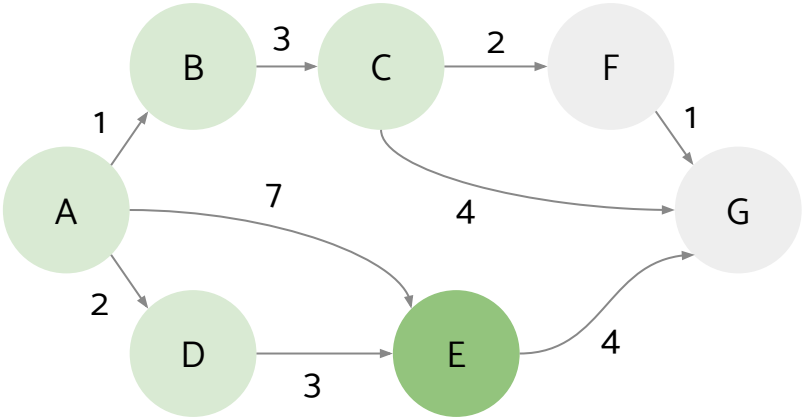
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	8
EdgeTo	-	A	B	A	D	C	C

Priority Queue:
E : 5
F : 6
G : 8

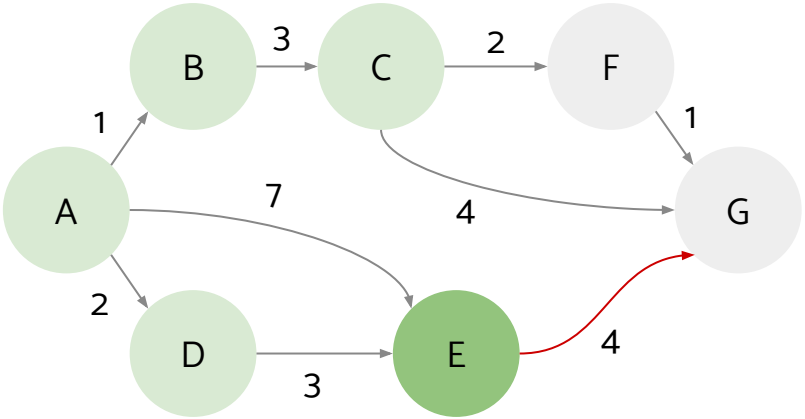
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	8
EdgeTo	-	A	B	A	D	C	C

Priority Queue:
F : 6
G : 8

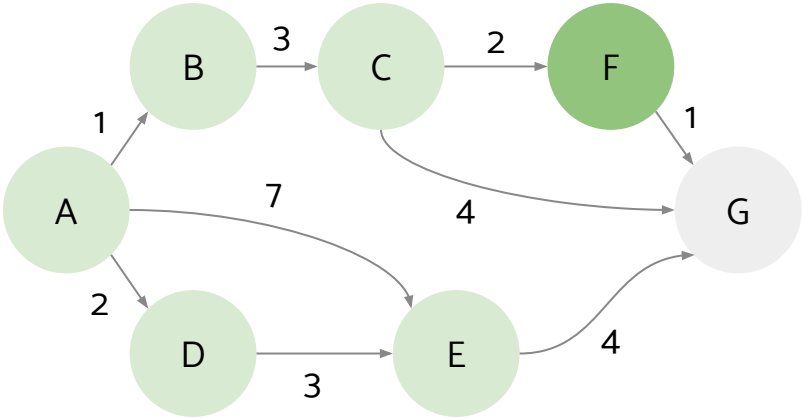
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	8
EdgeTo	-	A	B	A	D	C	C

Priority Queue:
F : 6
G : 8

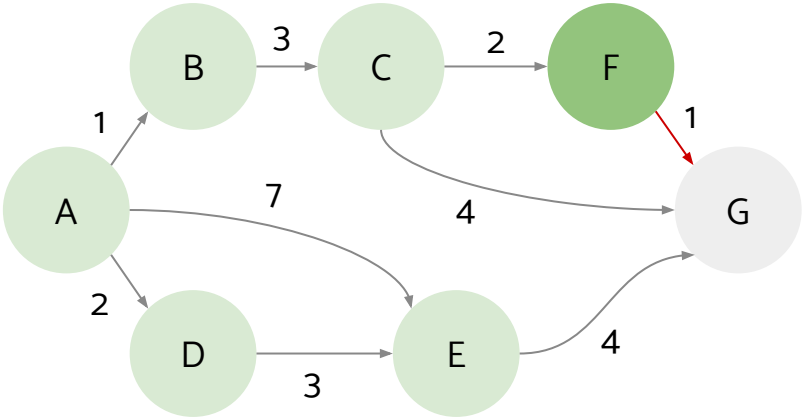
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	8
EdgeTo	-	A	B	A	D	C	C

Priority Queue:
G : 8

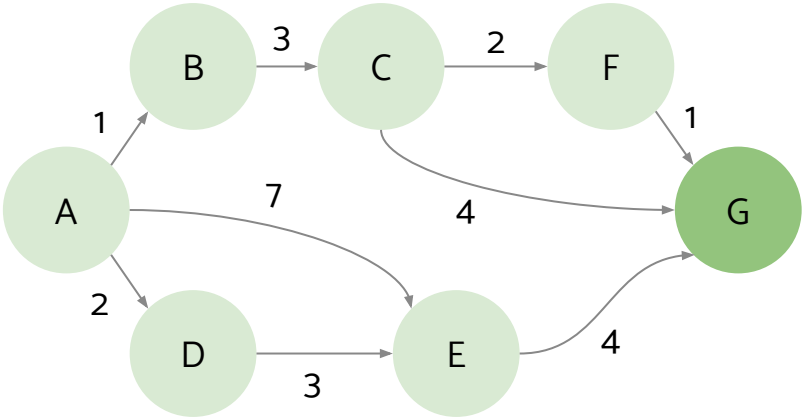
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	7
EdgeTo	-	A	B	A	D	C	F

Priority Queue:
G : 7

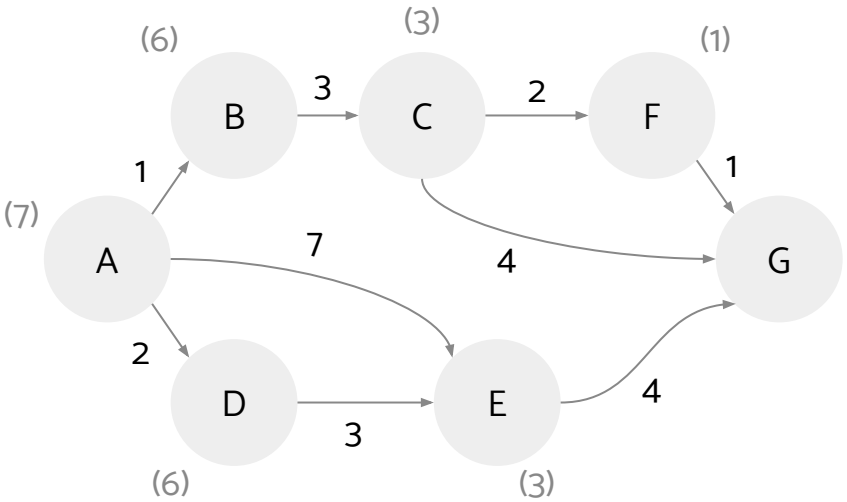
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	7
EdgeTo	-	A	B	A	D	C	F

Priority Queue:

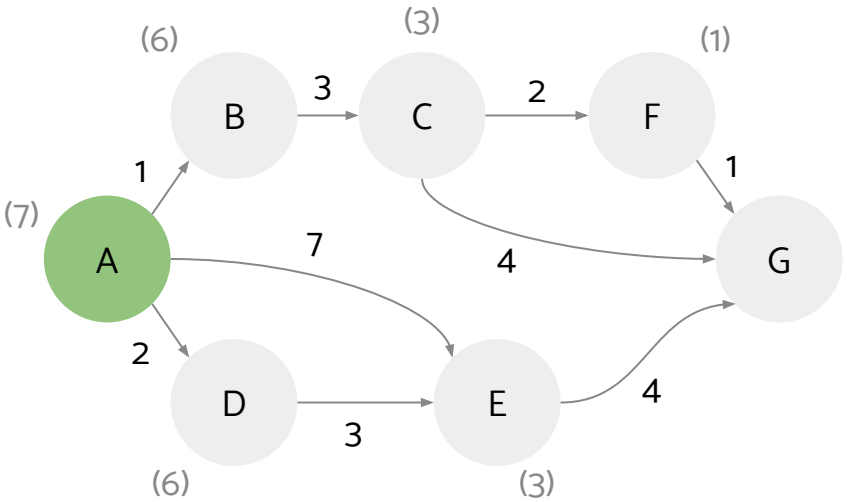
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	∞	∞	∞	∞	∞	∞	∞
EdgeTo	-	-	-	-	-	-	-

Priority Queue:
A : 7

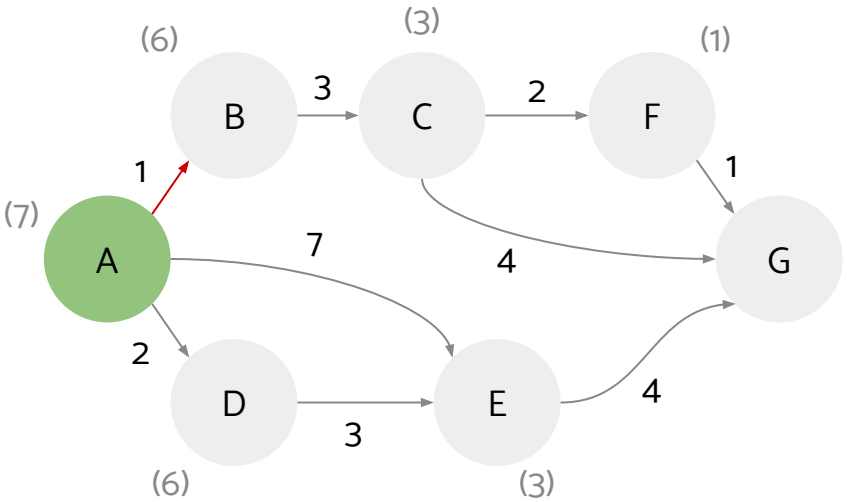
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	∞	∞	∞	∞	∞	∞
EdgeTo	-	-	-	-	-	-	-

Priority Queue:

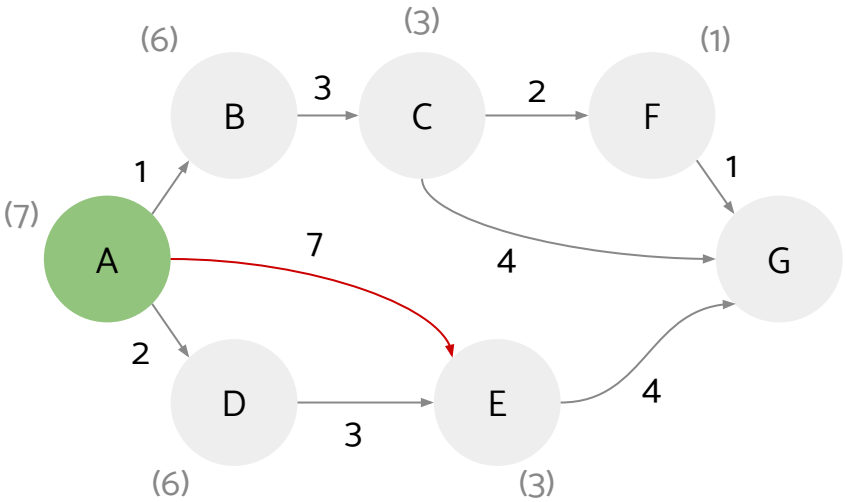
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	∞	∞	∞	∞	∞
EdgeTo	-	A	-	-	-	-	-

Priority Queue:
B : 7

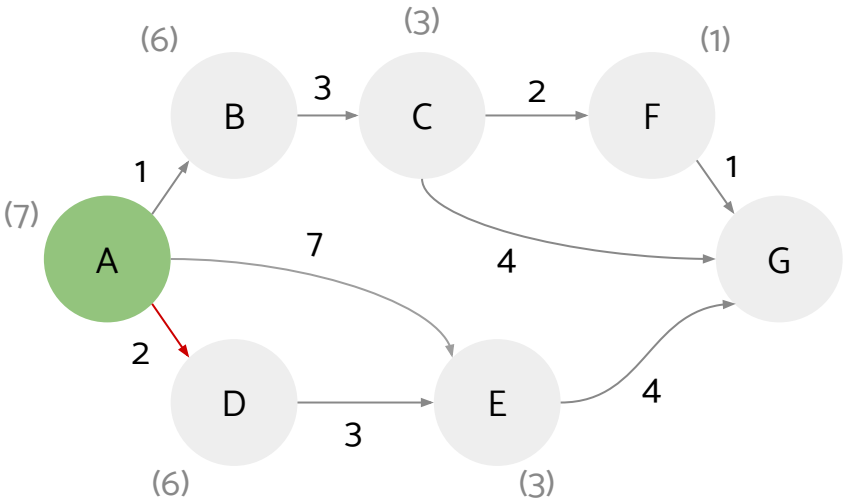
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	∞	∞	7	∞	∞
EdgeTo	-	A	-	-	A	-	-

Priority Queue:
B : 7
E : 10

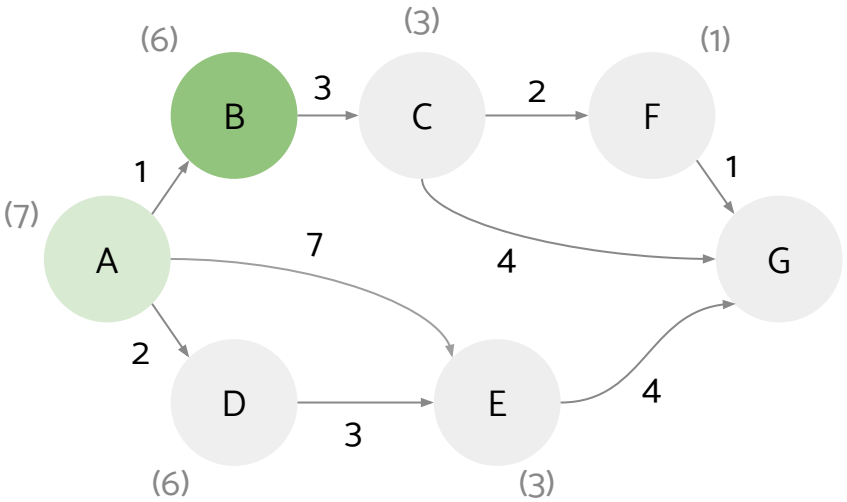
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	∞	2	7	∞	∞
EdgeTo	-	A	-	A	A	-	-

Priority Queue:
B : 7
D : 8
E : 10

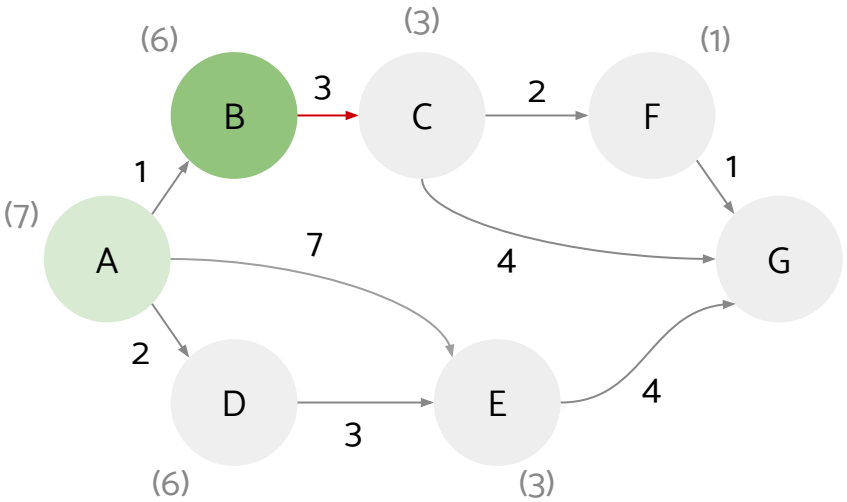
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	∞	2	7	∞	∞
EdgeTo	-	A	-	A	A	-	-

Priority Queue:
D : 8
E : 10

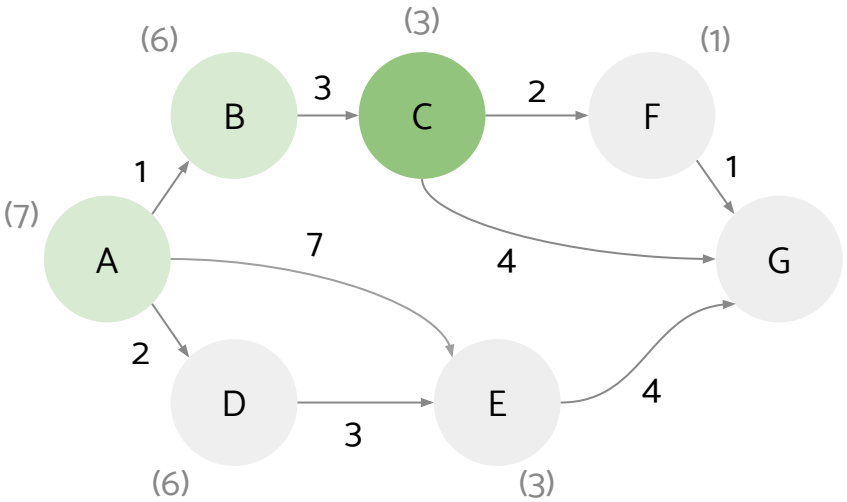
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	∞	∞
EdgeTo	-	A	B	A	A	-	-

Priority Queue:
C : 7
D : 8
E : 10

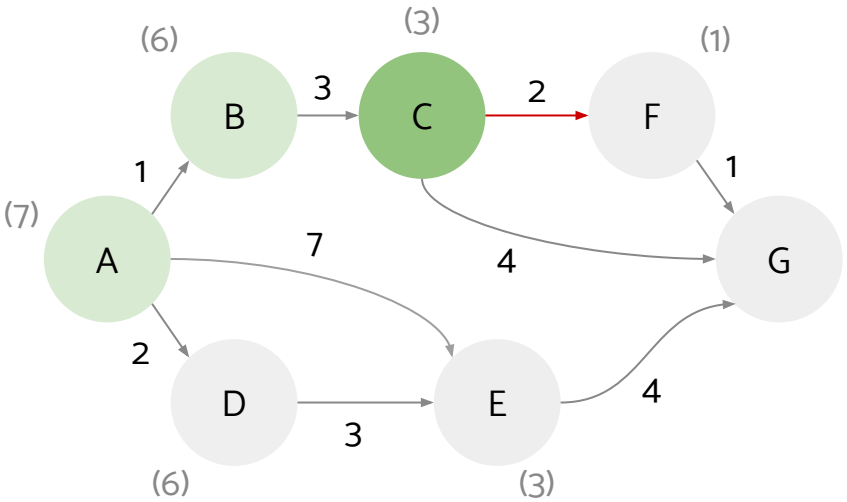
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	∞	∞
EdgeTo	-	A	B	A	A	-	-

Priority Queue:
D : 8
E : 10

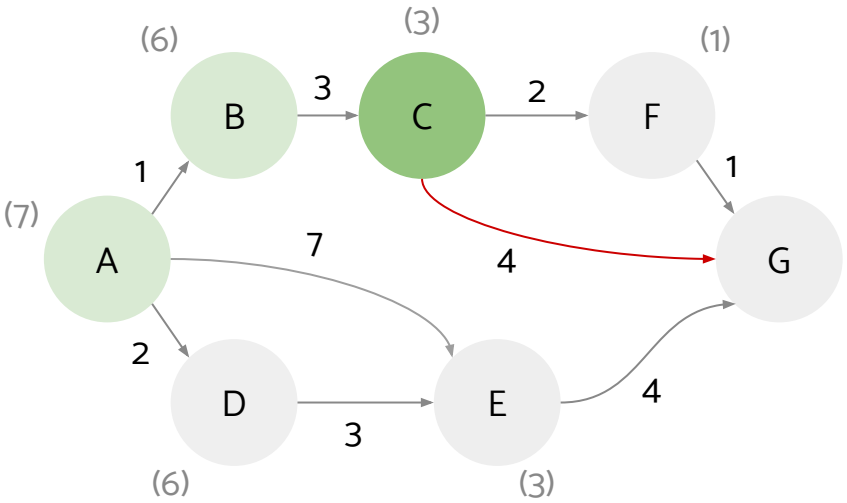
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	6	∞
EdgeTo	-	A	B	A	A	C	-

Priority Queue:
F : 7
D : 8
E : 10

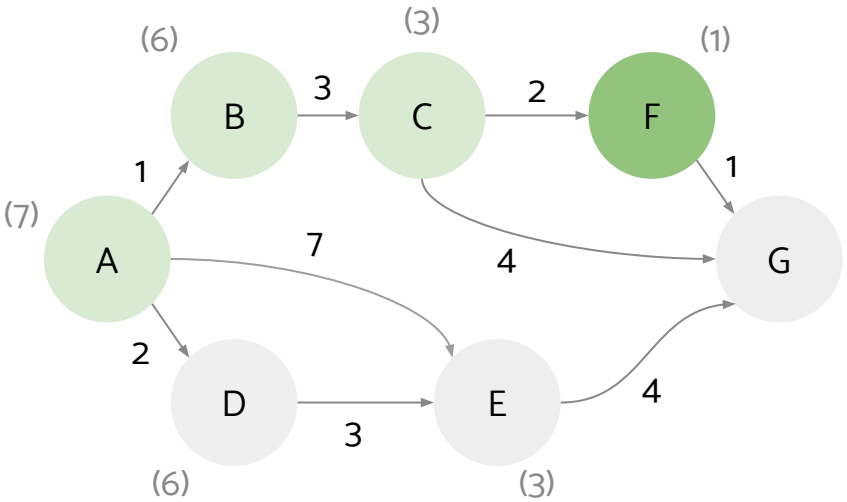
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	6	8
EdgeTo	-	A	B	A	A	C	C

Priority Queue:
F : 7
D : 8
G : 8
E : 10

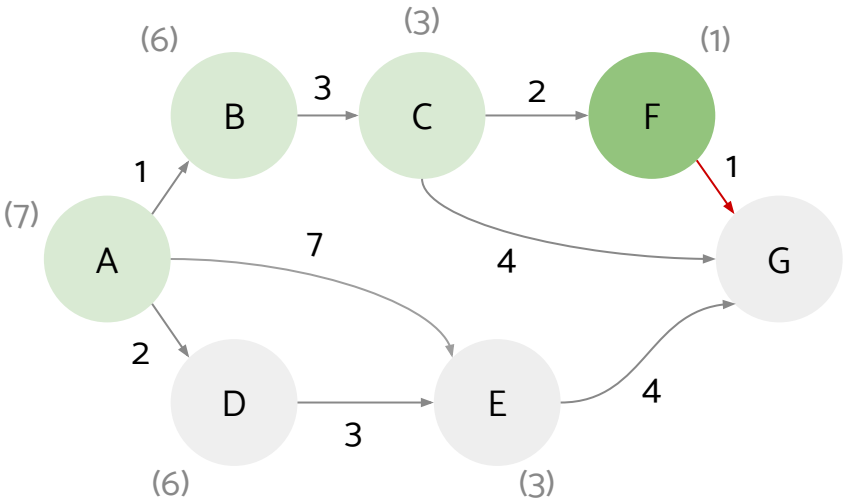
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	6	8
EdgeTo	-	A	B	A	A	C	C

Priority Queue:
D : 8
G : 8
E : 10

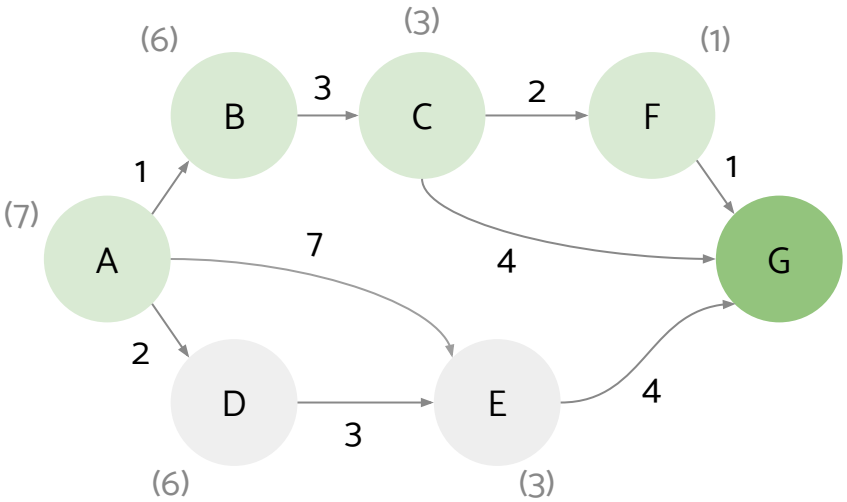
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	6	7
EdgeTo	-	A	B	A	A	C	F

Priority Queue:
G : 7
D : 8
E : 10

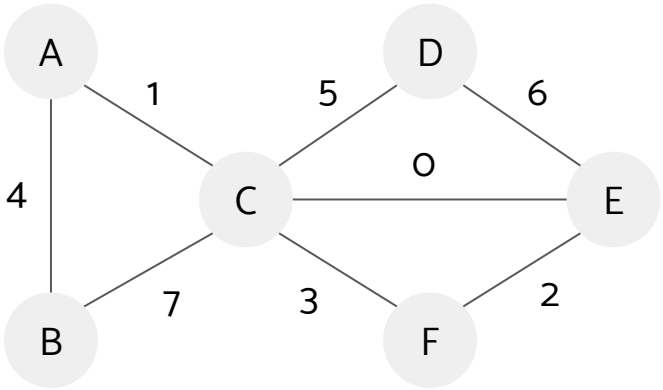
1B The Shortest Path To Your Heart *Extra* Run A* from A to G.



	A	B	C	D	E	F	G
DistTo	0	1	4	2	7	6	7
EdgeTo	-	A	B	A	A	C	F

Priority Queue:
D : 8
E : 10

2A Minimalist Moles

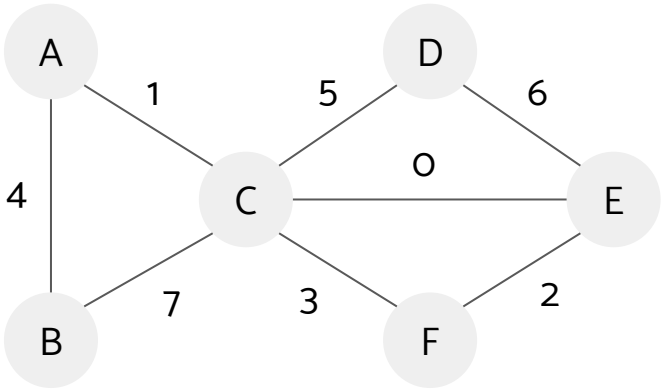


Mindy the mole wants to dig a network of tunnels connecting all of their secret hideouts. There are a set few paths between the secret hideouts that Mindy can choose to possibly include in their tunnel system, shown to the left. However, some portions of the ground are harder to dig than others, and Mindy wants to do as little work as possible. In the diagram shown, the numbers next to the paths correspond to how hard that path is to dig for Mindy.

How can Mindy figure out a tunnel system to connect their secret hideouts while doing minimal work?

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



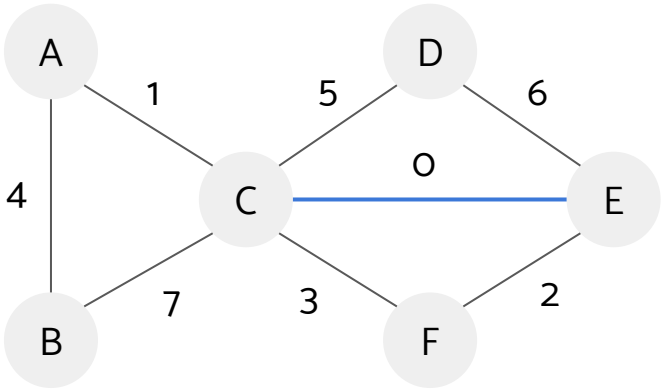
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



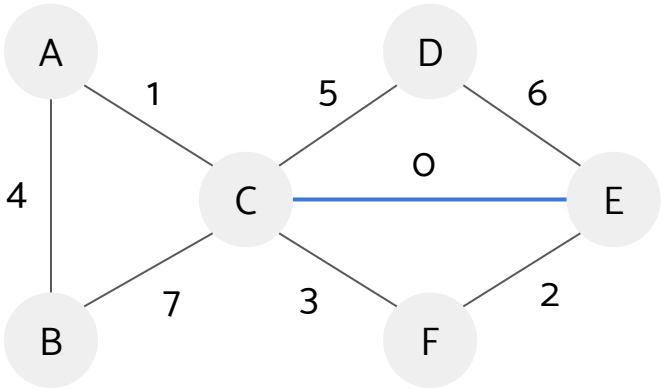
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



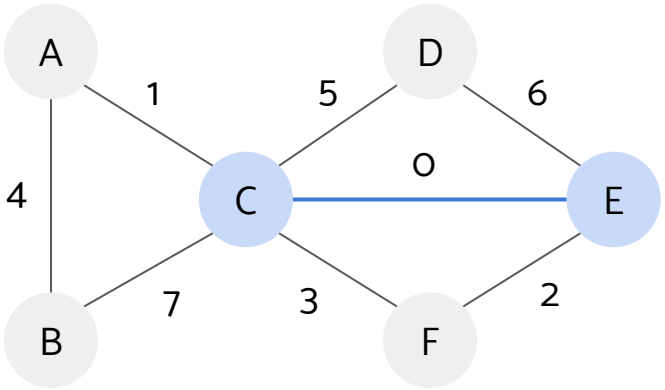
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



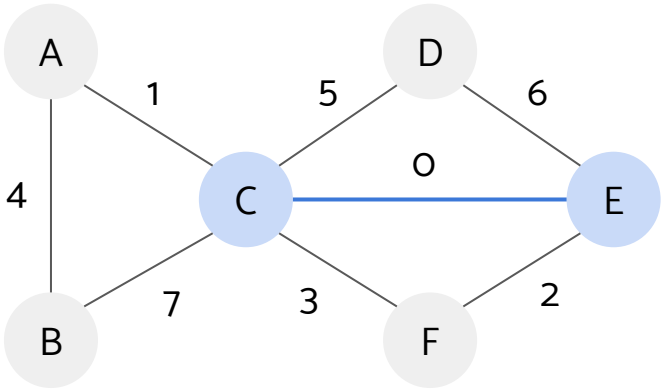
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



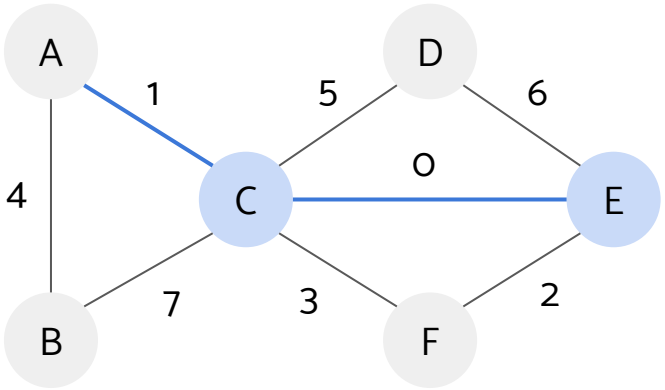
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



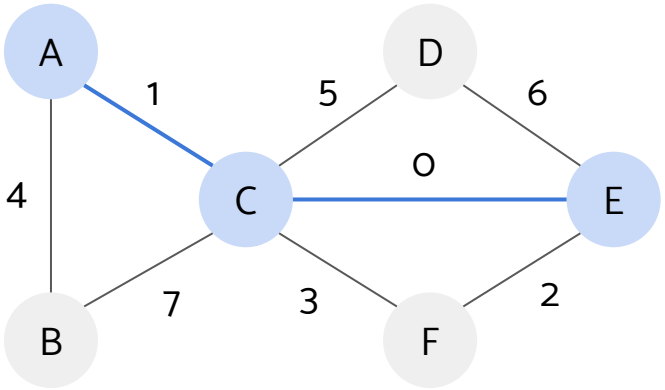
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



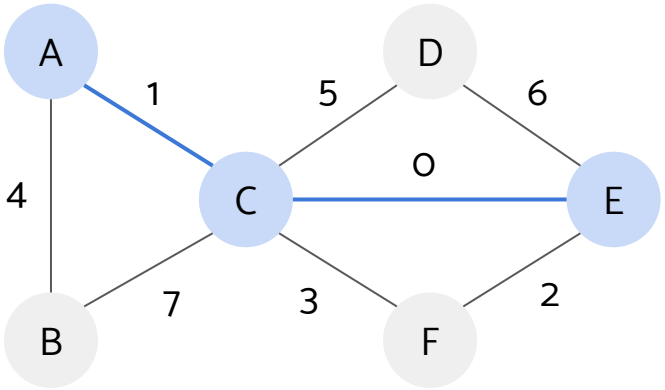
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



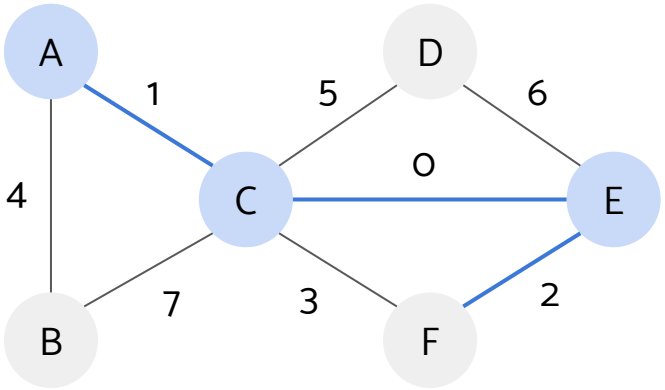
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



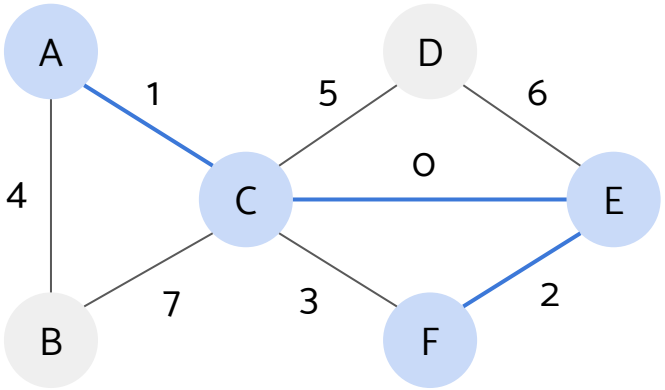
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



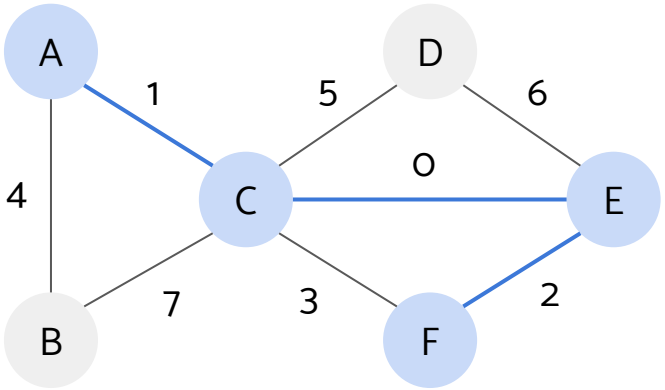
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



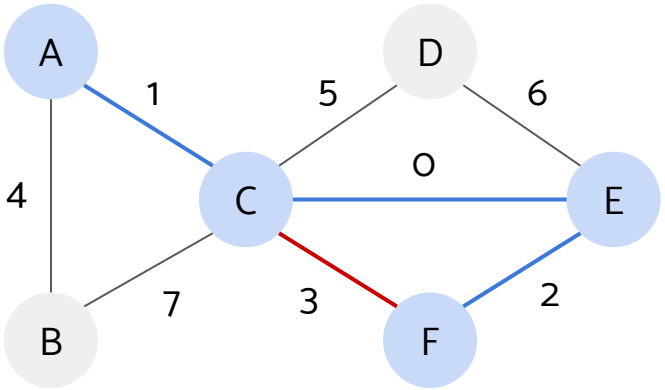
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



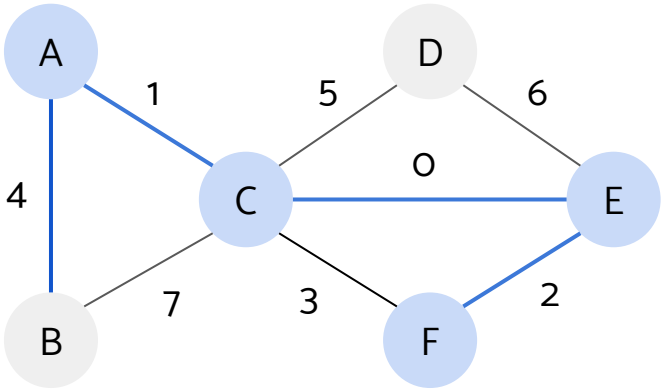
Kruskal's Algorithm

While there are still nodes not in the MST:

- *Add the lightest edge that doesn't create a loop.*
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



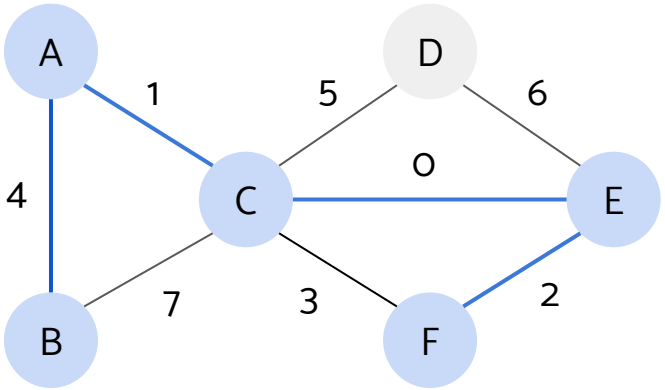
Kruskal's Algorithm

While there are still nodes not in the MST:

- *Add the lightest edge that doesn't create a loop.*
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



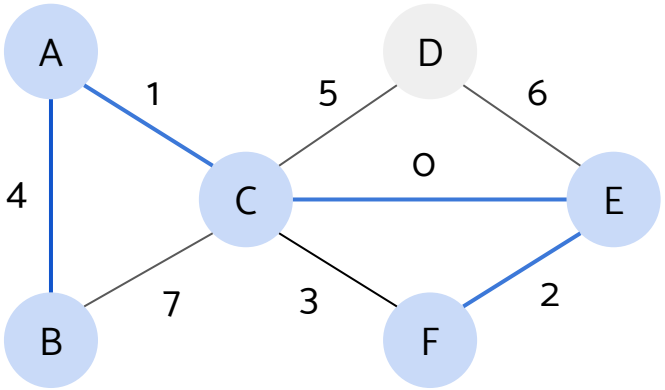
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



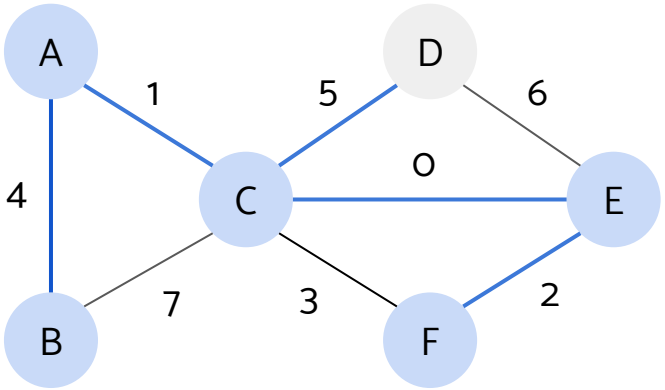
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



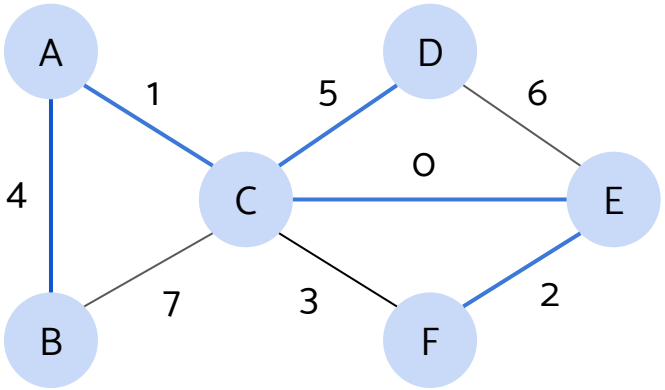
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



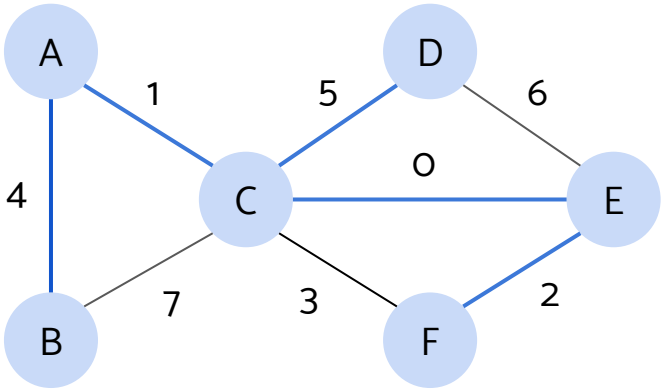
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



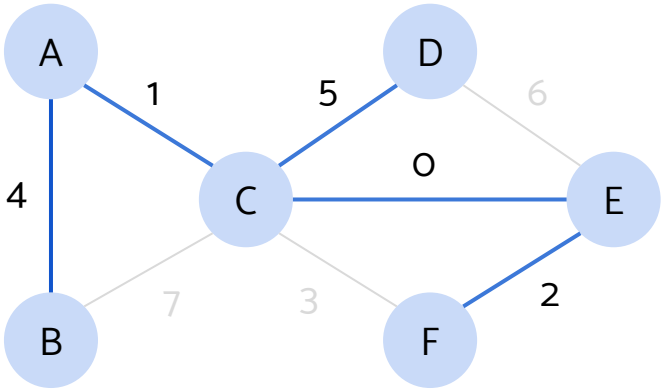
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



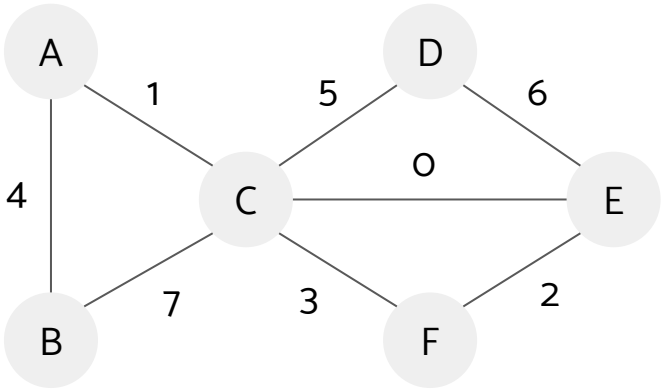
Kruskal's Algorithm

While there are still nodes not in the MST:

- Add the lightest edge that doesn't create a loop.
- Add the endpoints of that edge to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

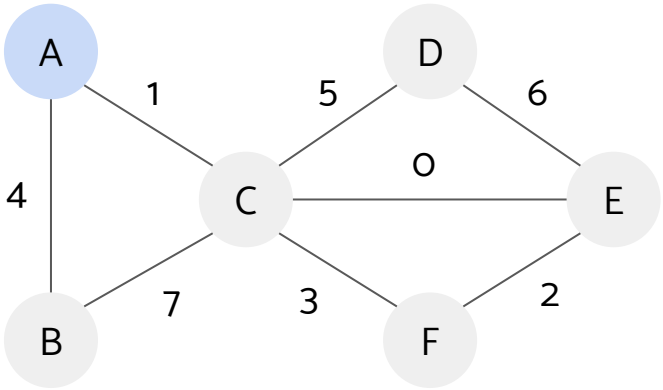


Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that leads to a node that is unvisited.
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

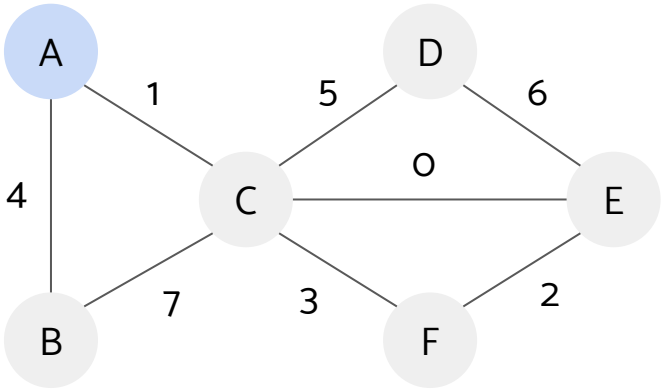


Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that leads to a node that is unvisited.
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



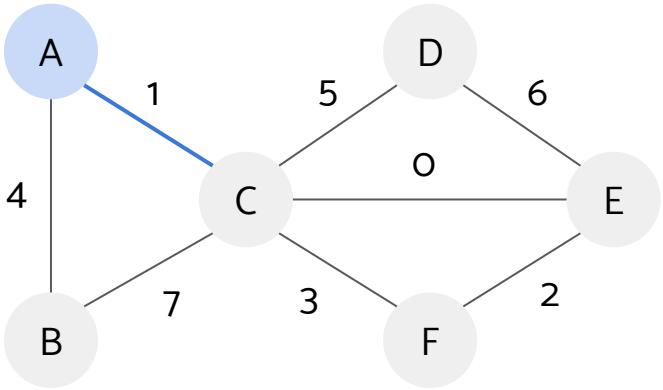
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



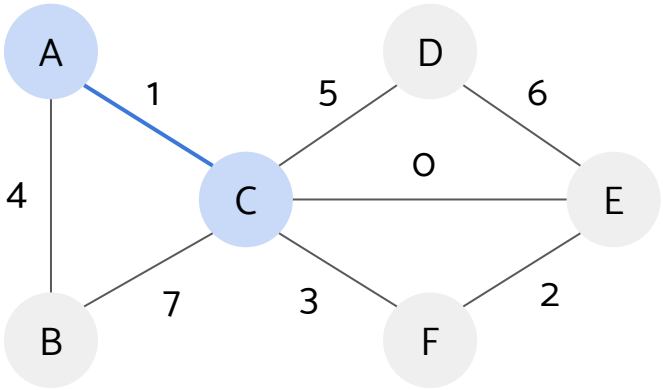
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

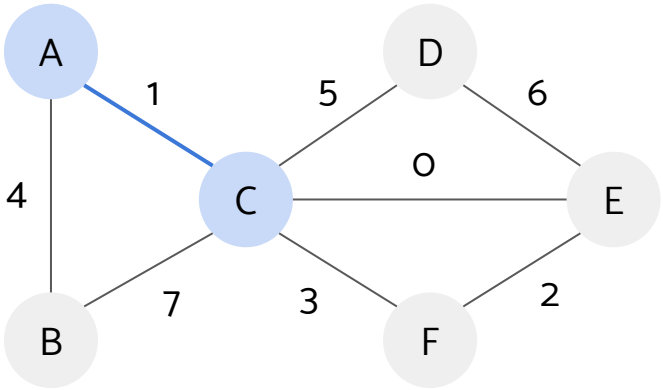


Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



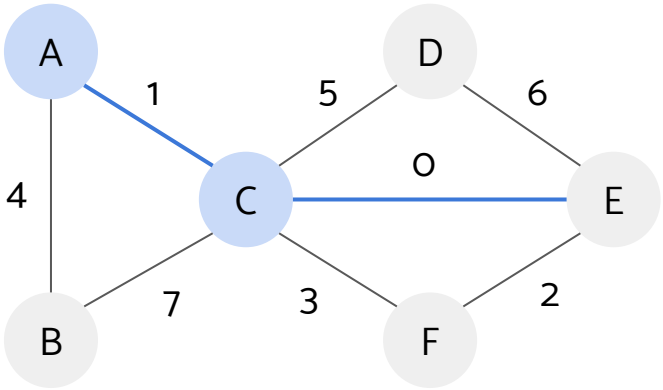
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



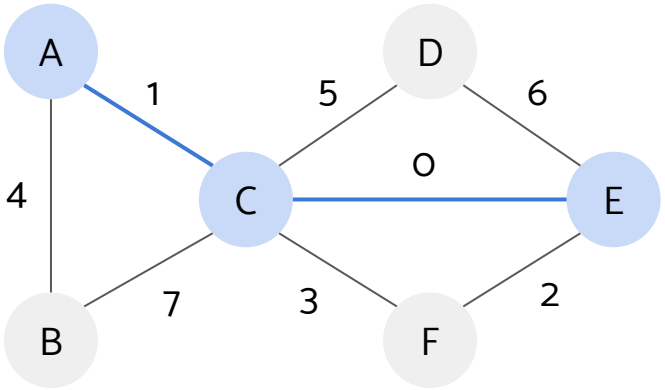
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

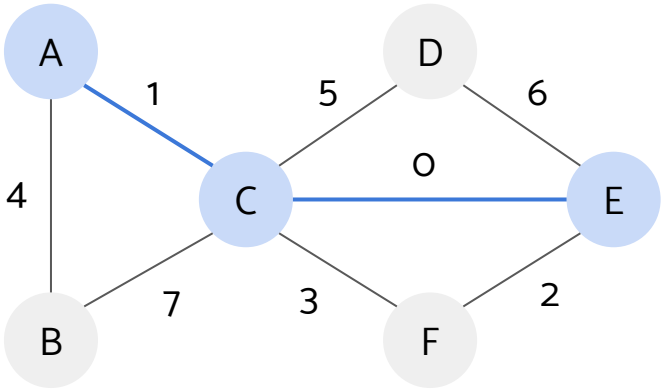


Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



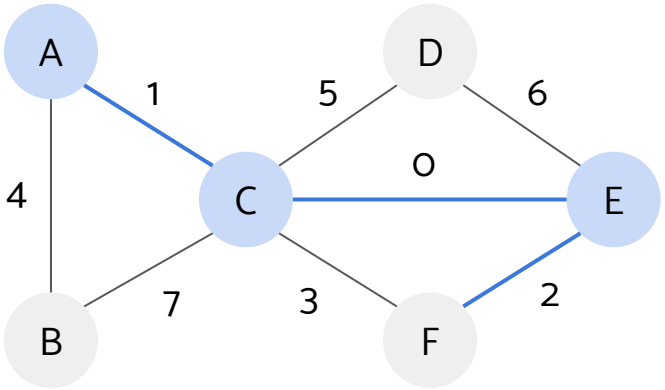
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

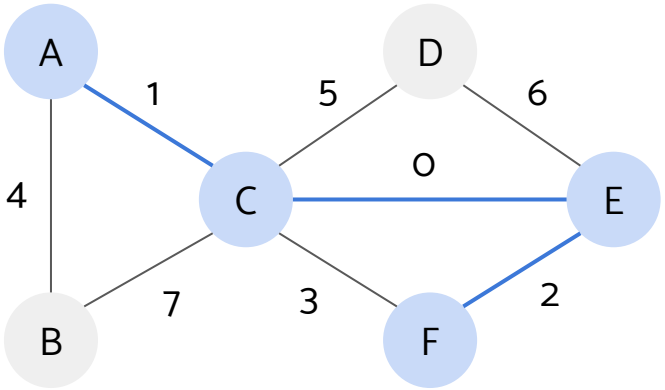



```
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in
the MST.
While there are still nodes not in
the MST:
  - Add the lightest edge that
    connects a visited node to an
    unvisited one (sprawl outward)
  - Add the new node to the set of
    nodes in the MST.
```

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

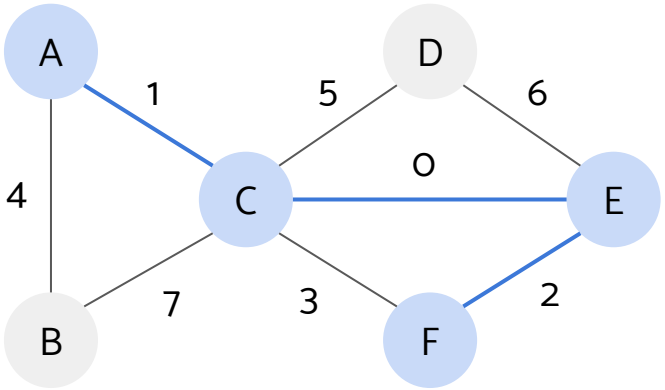


Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



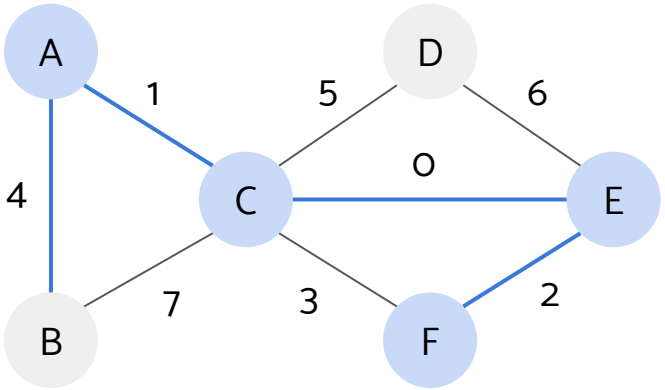
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

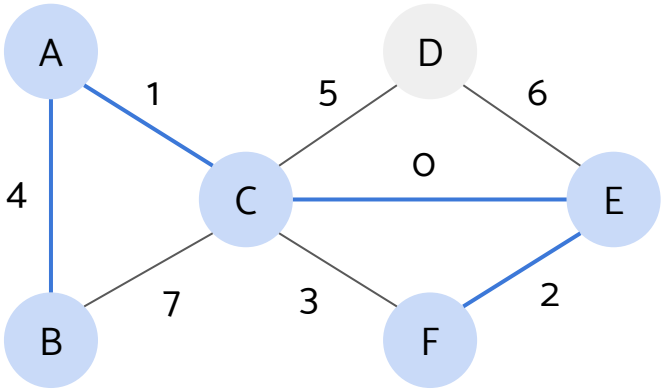



```
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in
the MST.
While there are still nodes not in
the MST:
  - Add the lightest edge that
    connects a visited node to an
    unvisited one (sprawl outward)
  - Add the new node to the set of
    nodes in the MST.
```

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

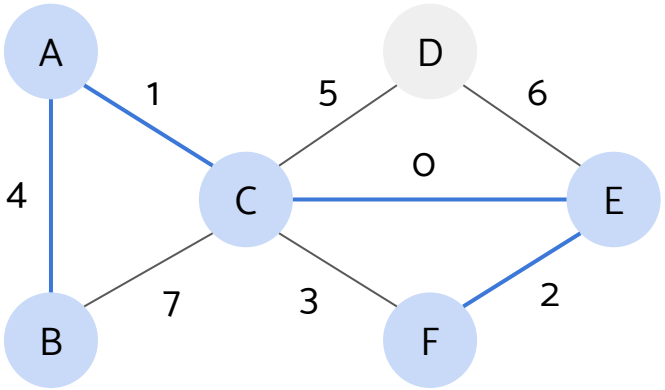


Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



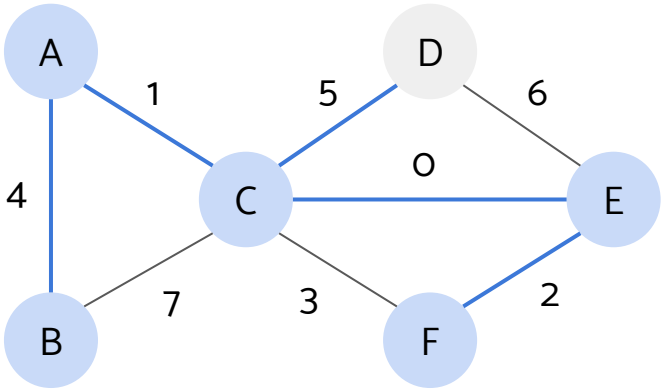
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

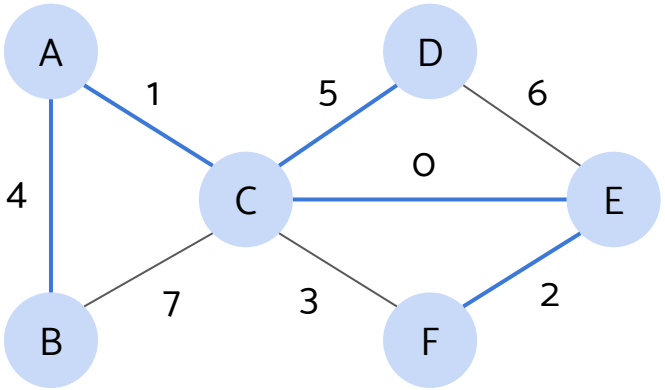



```
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in
the MST.
While there are still nodes not in
the MST:
  - Add the lightest edge that
    connects a visited node to an
    unvisited one (sprawl outward)
  - Add the new node to the set of
    nodes in the MST.
```

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.

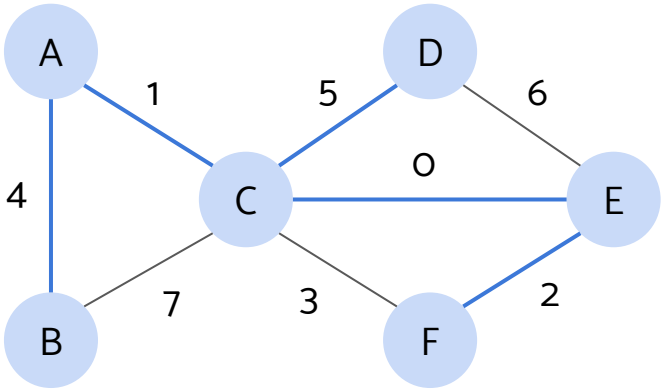


Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



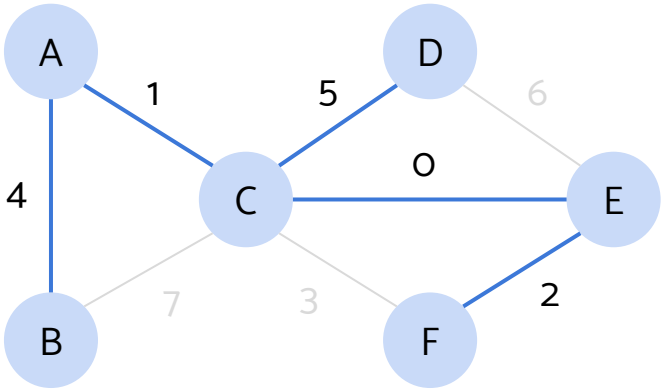
Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:

- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.



Prim's Algorithm

Start with any node.
Add that node to the set of nodes in the MST.
While there are still nodes not in the MST:
- Add the lightest edge that connects a visited node to an unvisited one (sprawl outward)
- Add the new node to the set of nodes in the MST.

Minimum Spanning Trees: *An Aside*

Running both algorithms side by side shows that Prim's and Kruskal's take two different and valid approaches to solve the same problem.

Prim's builds MSTs by dividing the graph into two sets: the vertices incorporated into the MST, and those yet to be included. It continually adds the lightest edge connecting those two sets, sprawling outward and claiming more and more nodes into the MST, until all nodes are included.

Kruskal's builds MSTs by sorting all the edges, and then adding them to our MST in order–but at every step, if adding an edge would cause a cycle then it is not included. Kruskal's may jump around the graph.

2C Minimalist Moles

Did Kruskal's and Prim's find different MSTs, or the same MST? Is there a different tie-breaking scheme that would change your answer?