Shortest Paths and MSTs

Discussion 9

Example Agenda

- 1:10 - 1:15 ~ announcements
- 1:15 - 1:30 ~ content review
- 1:30 - 1:40 ~ question 1
- 1:40 - 1:55 ~ question 2
- Question 3 if time

Announcements

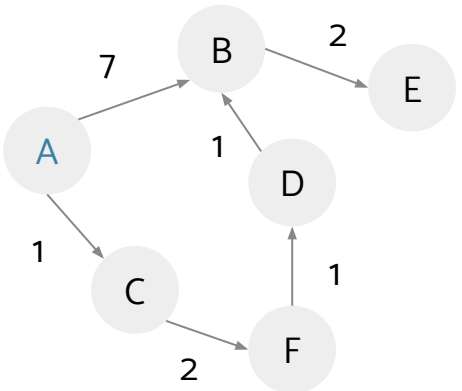
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	3/18 Homework 3 Due			3/21 **Midterm 2**		

Content Review

Dijkstra's Algorithm

We've learned that BFS can help us find paths from the start to other nodes with a minimum number of edges. However, neither BFS or DFS account for finding 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 front 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.

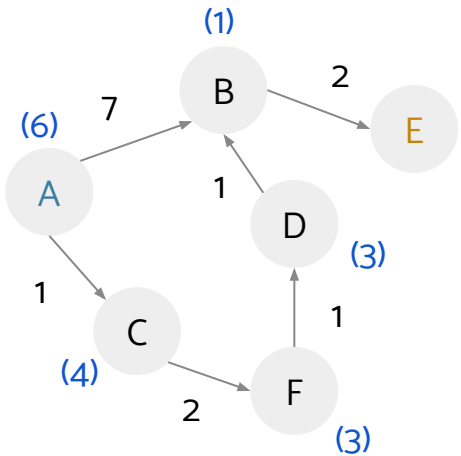
Dijkstra's Algorithm

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.



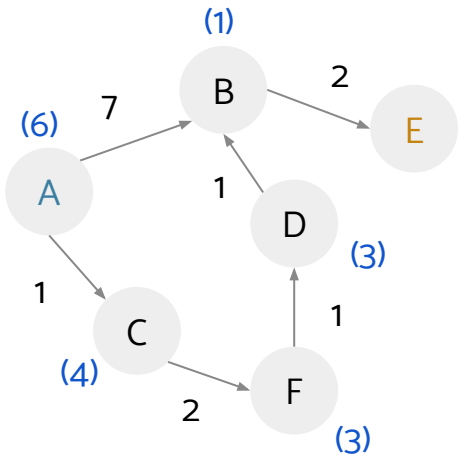
We're only guaranteed to get the shortest path if our heuristic is admissible (never overestimates the true distance to the goal) and consistent (estimate always <= the estimated distance from any neighboring vertex to the goal + the cost of reaching that neighbor).

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.



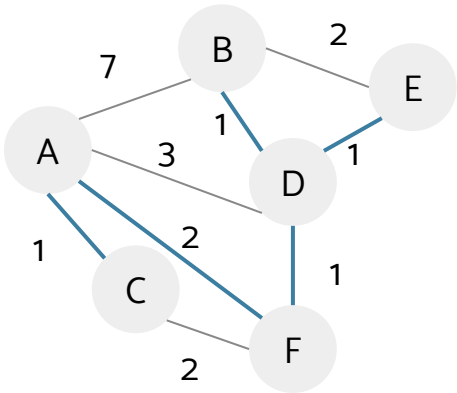
Note: the heuristic may not always be very good and might lead us down a path that isn't the shortest!

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. Both are based on the **cut property**: if we "cut" across any edges and separate the graph into two groups, the minimum weight edge that falls along that cut will be in some MST.



Minimum Spanning Trees: *An Aside*

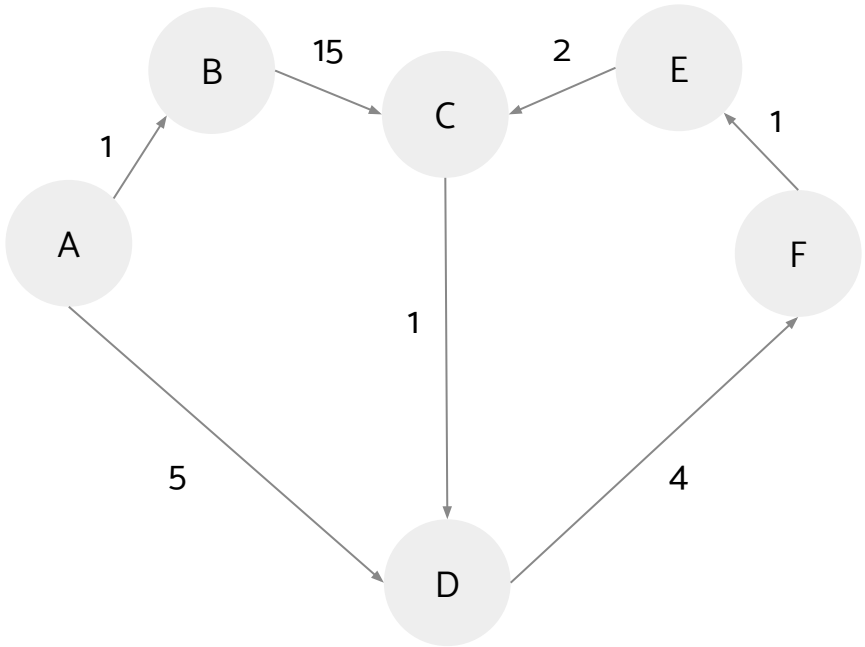
Running both algorithms side by side shows that Prim's and Kruskal's take two different and valid approaches to solve the same problem.

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.

Worksheet

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



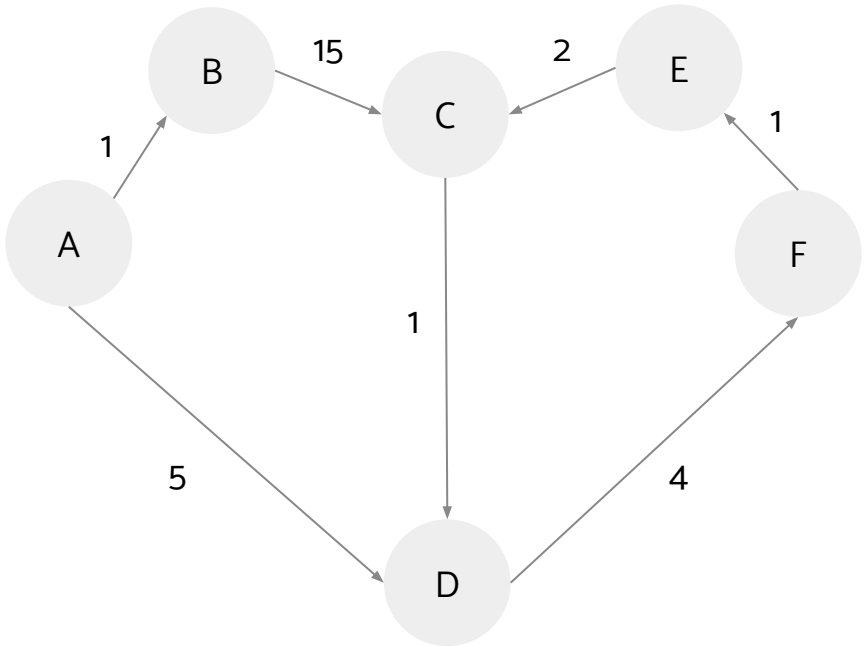

```
Dijkstra's algorithm:

PQ = new PriorityQueue()
PQ.add(A, 0)
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
edgeTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()
  for child in poppedNode.children:
    potentialDist = distTo[poppedNode] +
        edgeWeight(poppedNode, child)
    if potentialDist < distTo[child]:
      distTo.put(child, potentialDist)
      PQ.changePriority(child, potentialDist)
      edgeTo[child] = poppedNode
```

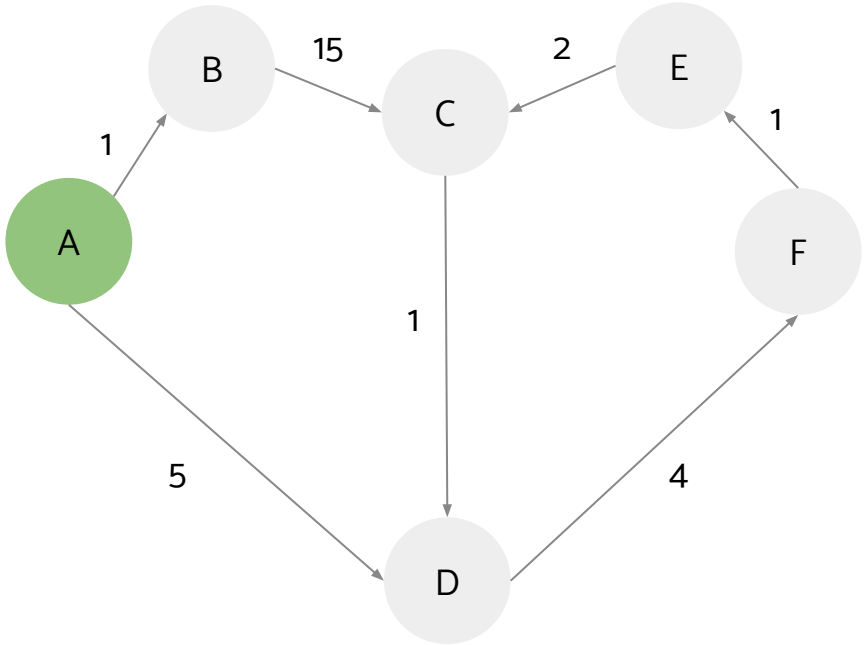

1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



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

Priority Queue:
A : 0 B: ∞ C: ∞ D: ∞ E: ∞ F: ∞

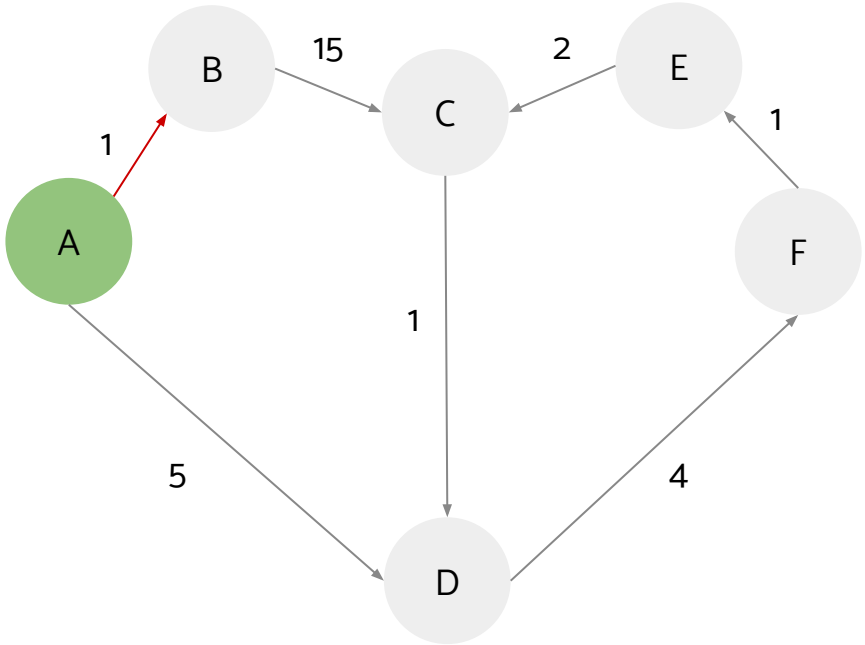
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



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

Priority Queue:
B: ∞ C: ∞ D: ∞ E: ∞ F: ∞

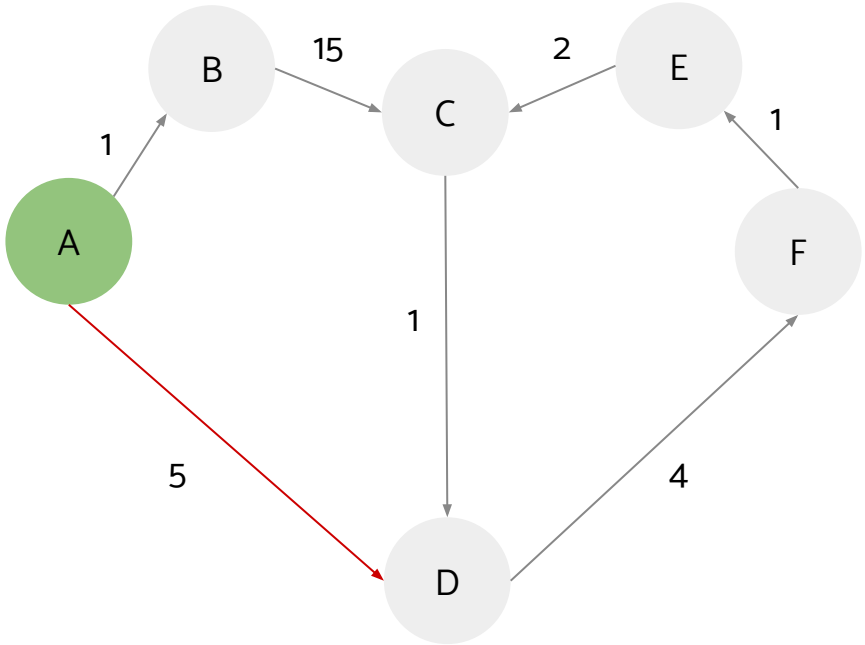
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



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

Priority Queue:
B: 1 C: ∞ D: ∞ E: ∞ F: ∞

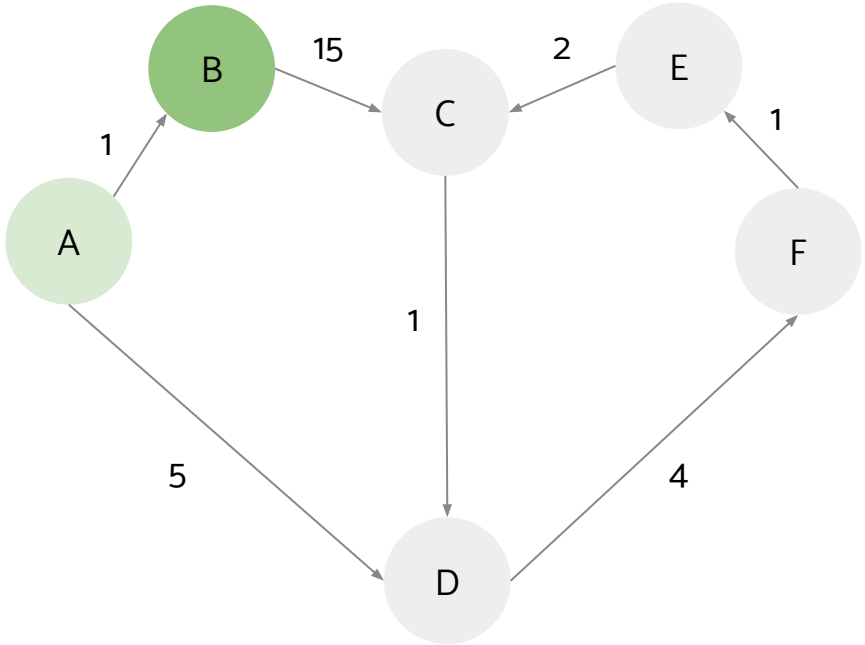
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



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

Priority Queue:
B: 1 D: 5 C: ∞ E: ∞ F: ∞

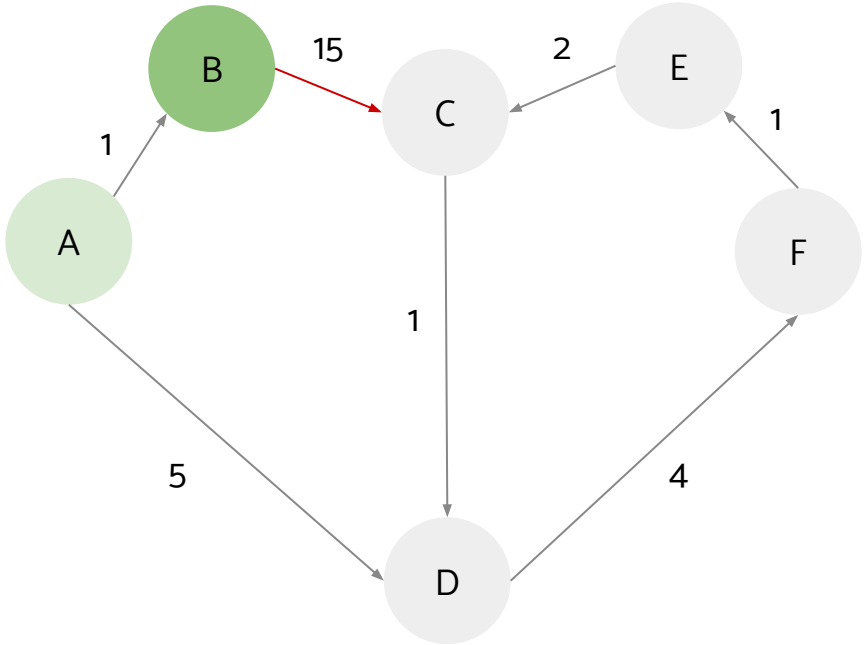
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



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

Priority Queue:
D: 5 C: ∞ E: ∞ F: ∞

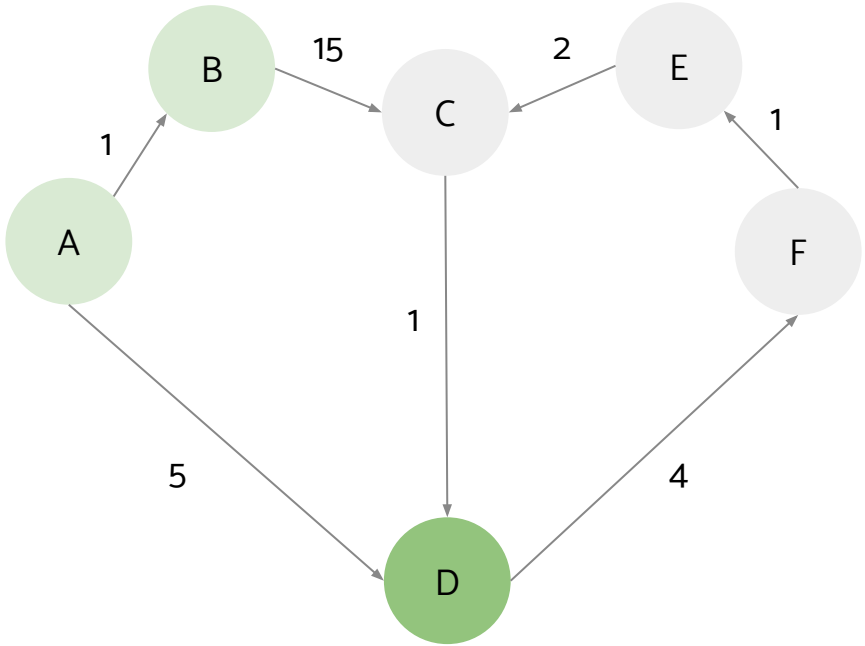
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F
DistTo	0	1	16	5	∞	∞
EdgeTo	-	A	B	A	-	-

Priority Queue:
D: 5 C: 16 E: ∞ F: ∞

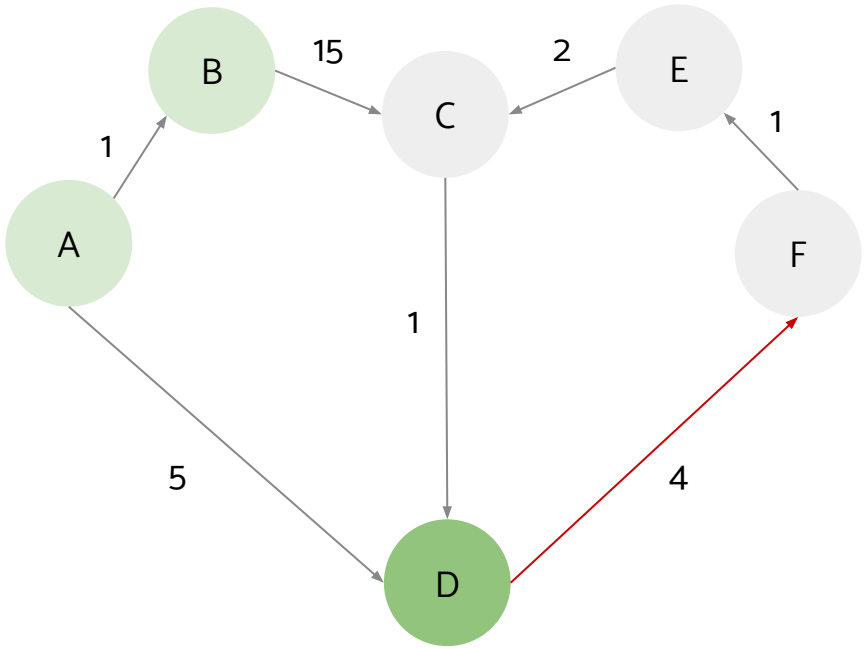
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F
DistTo	0	1	16	5	∞	∞
EdgeTo	-	A	B	A	-	-

Priority Queue:
D: 5 C: 16 E: ∞ F: ∞

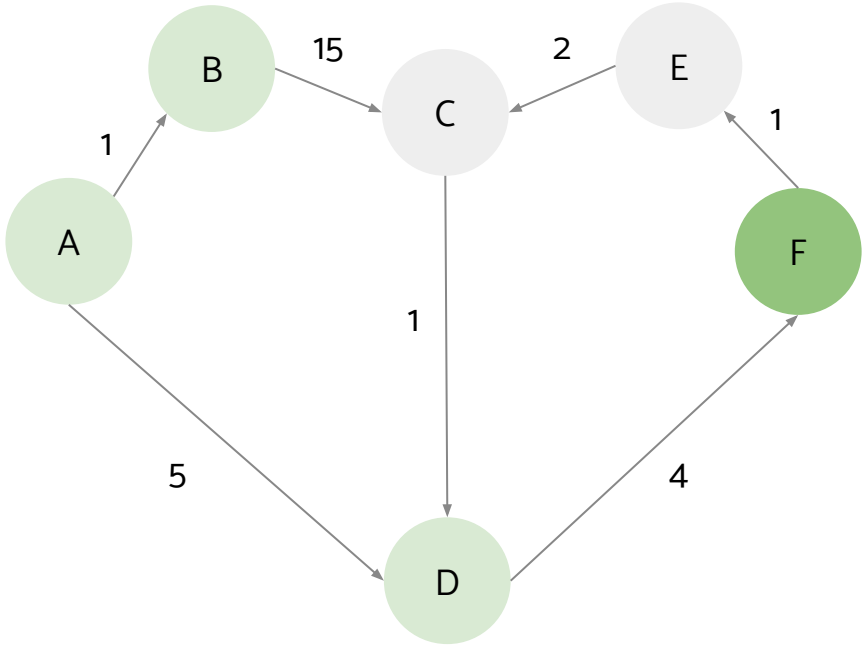
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F
DistTo	0	1	16	5	∞	9
EdgeTo	-	A	B	A	-	D

Priority Queue:
F: 9 C: 16 E: ∞

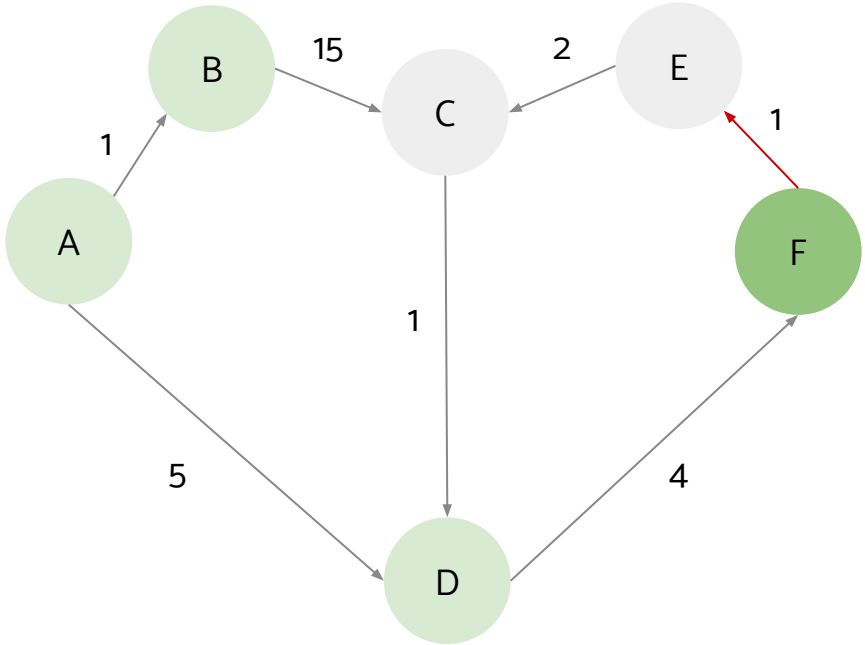
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F
DistTo	0	1	16	5	∞	9
EdgeTo	-	A	B	A	-	D

Priority Queue:
C: 16 E: ∞

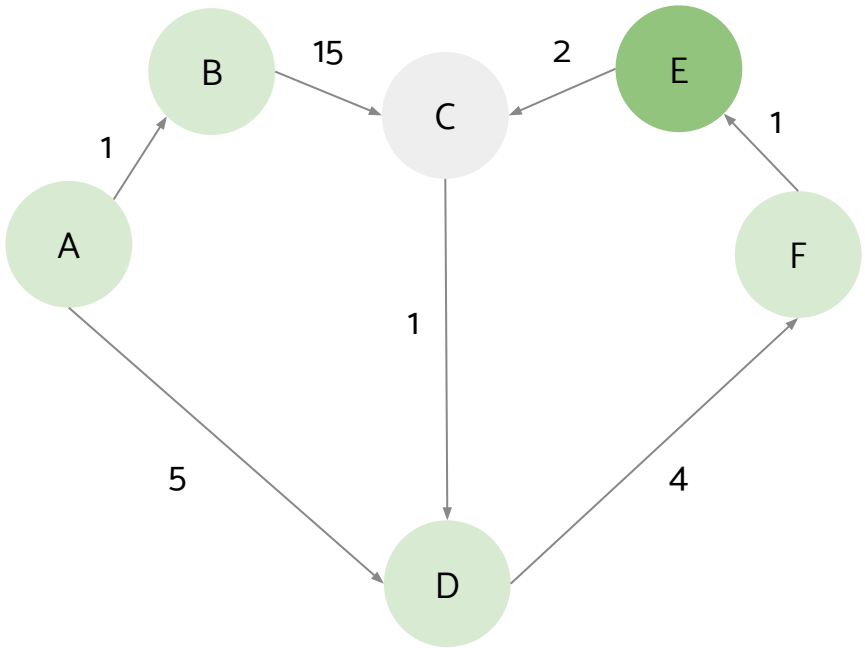
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F
DistTo	0	1	16	5	10	9
EdgeTo	-	A	B	A	F	D

Priority Queue:
E: 10 C: 16

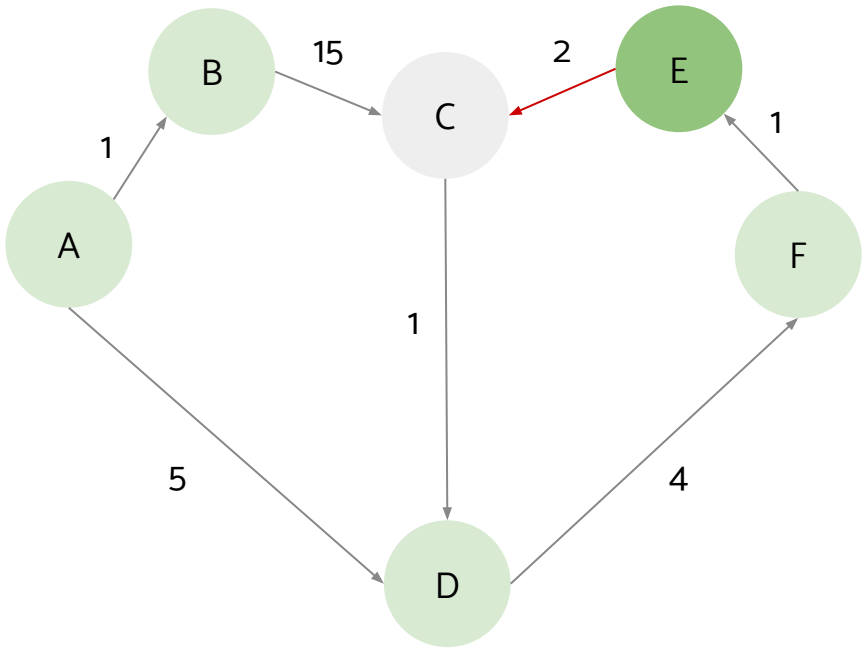
1A The Shortest Path To Your Heart **Run Dijkstra's, starting from A.**



	A	B	C	D	E	F
DistTo	0	1	16	5	10	9
EdgeTo	-	A	B	A	F	D

Priority Queue:
C: 16

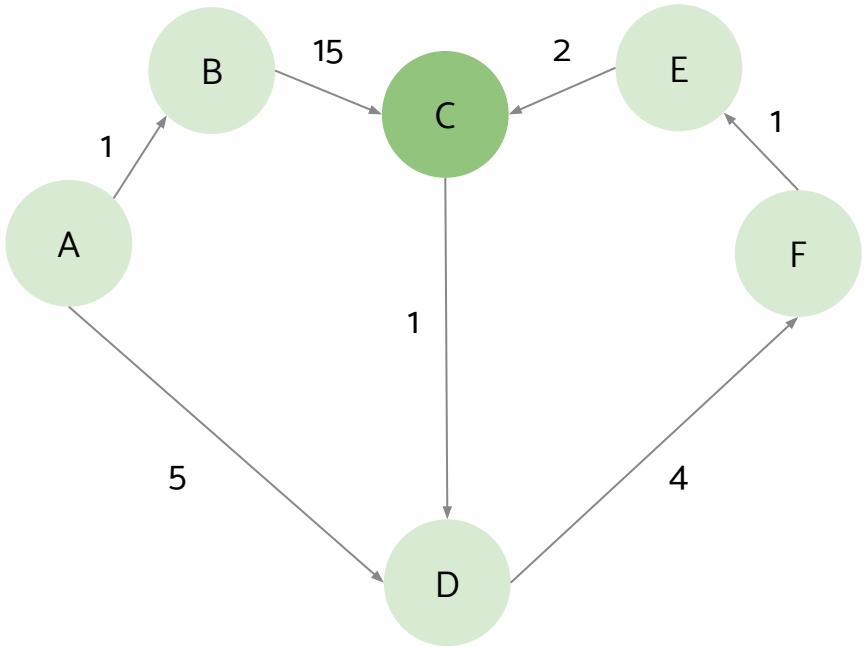
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F
DistTo	0	1	12	5	10	9
EdgeTo	-	A	E	A	F	D

Priority Queue:
C: 12

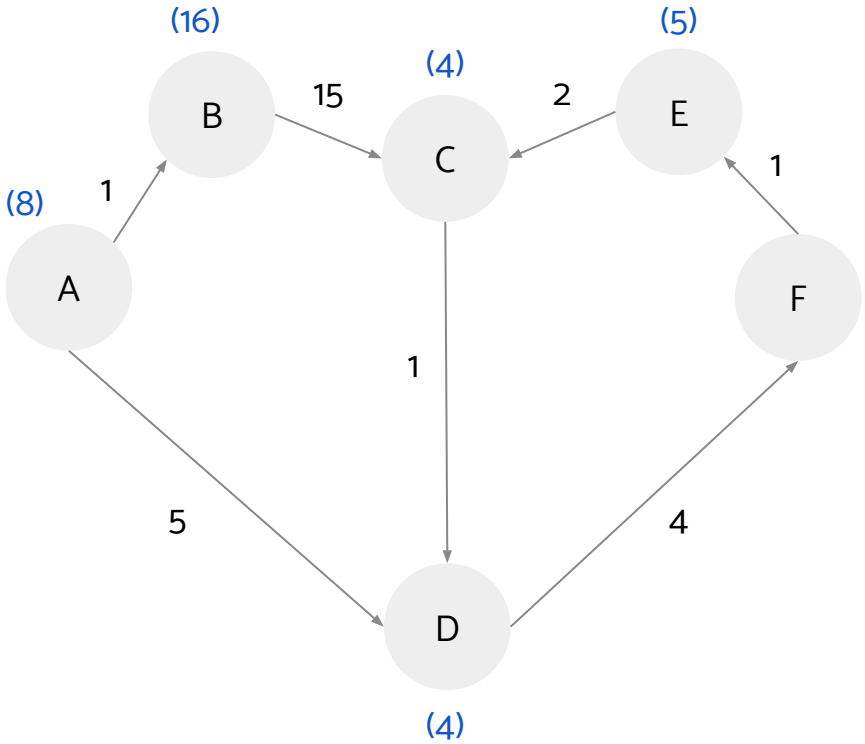
1A The Shortest Path To Your Heart Run Dijkstra's, starting from A.



	A	B	C	D	E	F
DistTo	0	1	12	5	10	9
EdgeTo	-	A	E	A	F	D

Priority Queue:

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



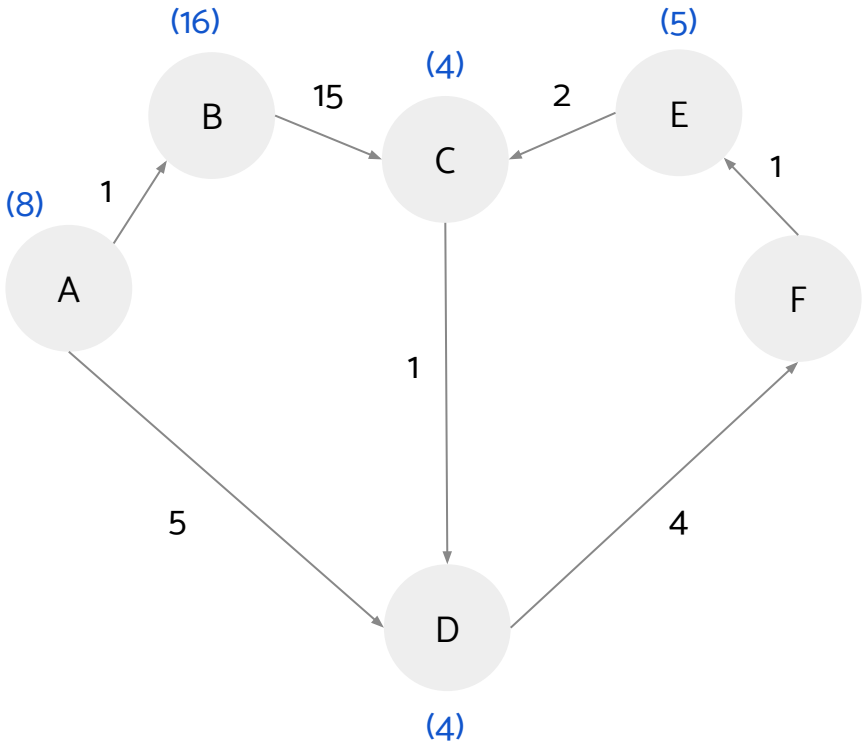

```
A*:

PQ = new PriorityQueue()
PQ.add(A, h(A, goal))
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()
  if (poppedNode == goal): terminate
    for child in poppedNode.children:
      potentialDist = distTo[poppedNode] +
          edgeWeight(poppedNode, child)
      if potentialDist < distTo[child]:
        distTo.put(child, potentialDist)
        PQ.changePriority(child,
potentialDist + h(child, goal))
        edgeTo[child] = poppedNode
```

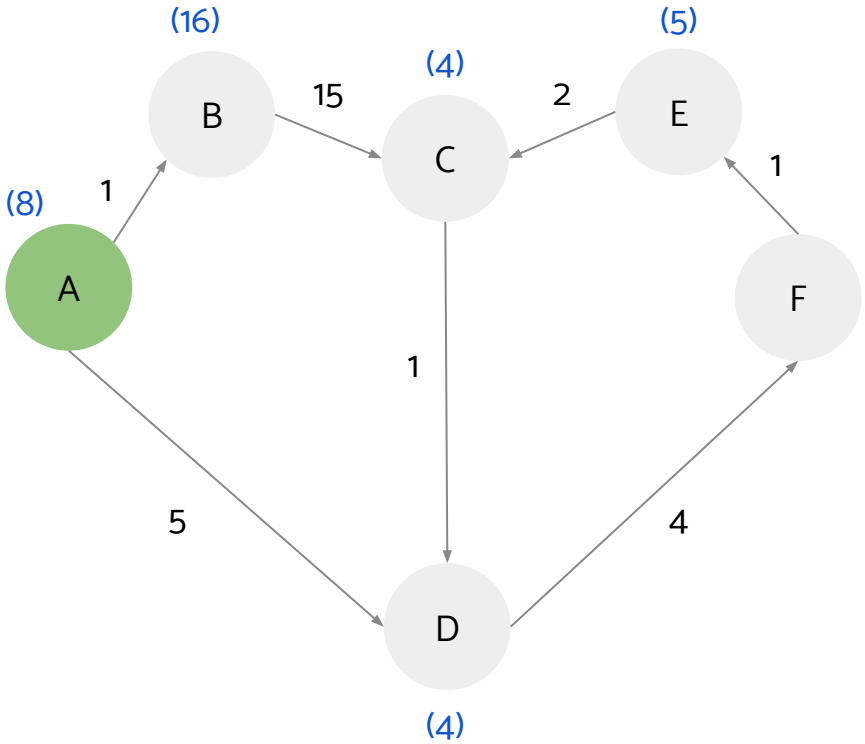

1B The Shortest Path To Your Heart Run A* from A to F.



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

Priority Queue:
A : 8 B: ∞ C: ∞ D: ∞ E: ∞ F: ∞

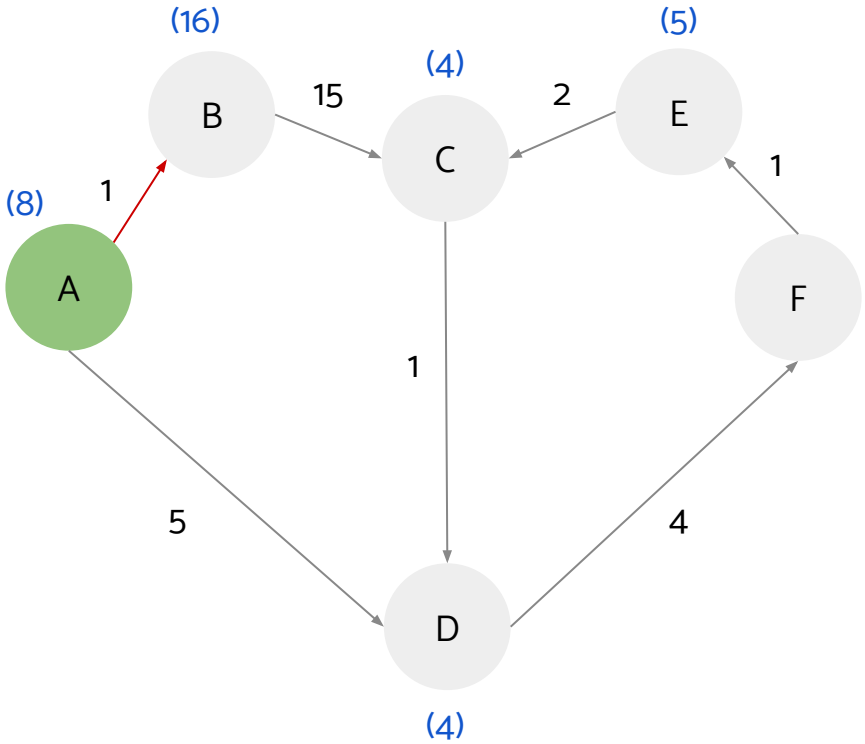
1B The Shortest Path To Your Heart Run A* from A to F.



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

Priority Queue:
B: ∞ C: ∞ D: ∞ E: ∞ F: ∞

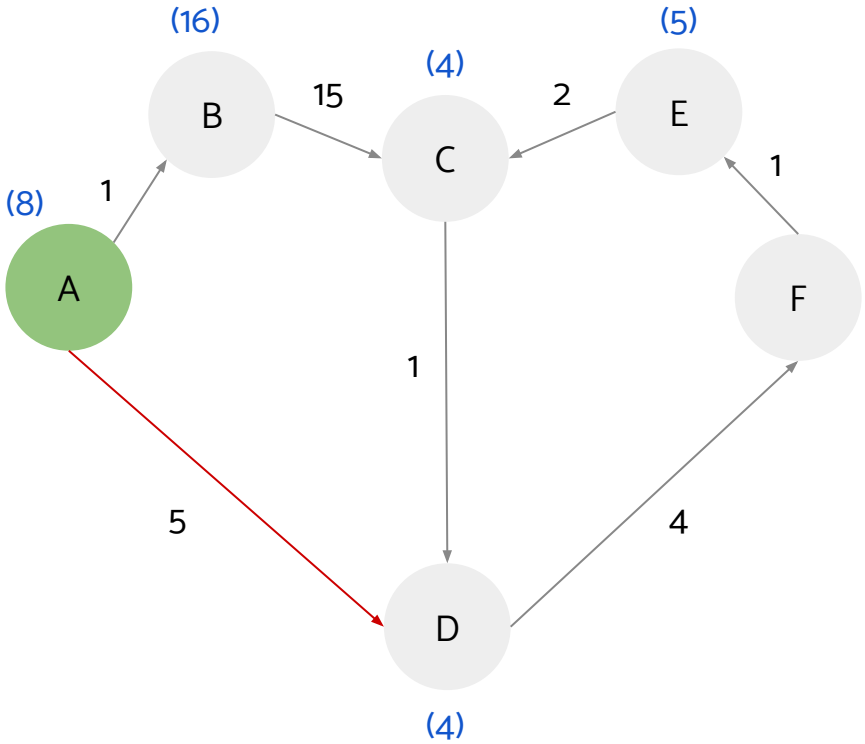
1B The Shortest Path To Your Heart Run A* from A to F.



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

Priority Queue:
B: 17 C: ∞ D: ∞ E: ∞ F: ∞

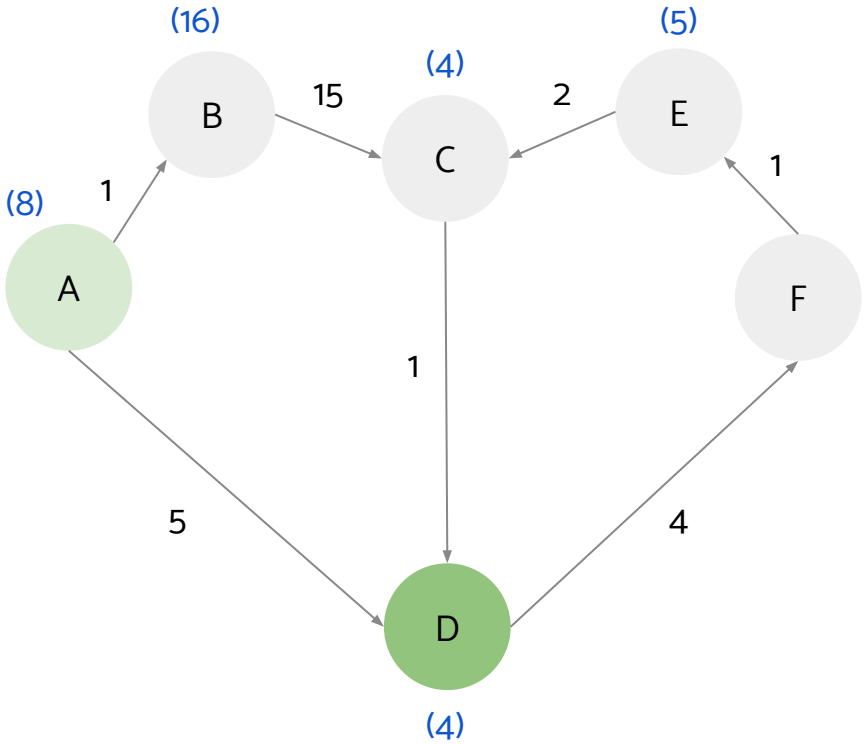
1B The Shortest Path To Your Heart Run A* from A to F.



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

Priority Queue:
D: 9 B: 17 C: ∞ E: ∞ F: ∞

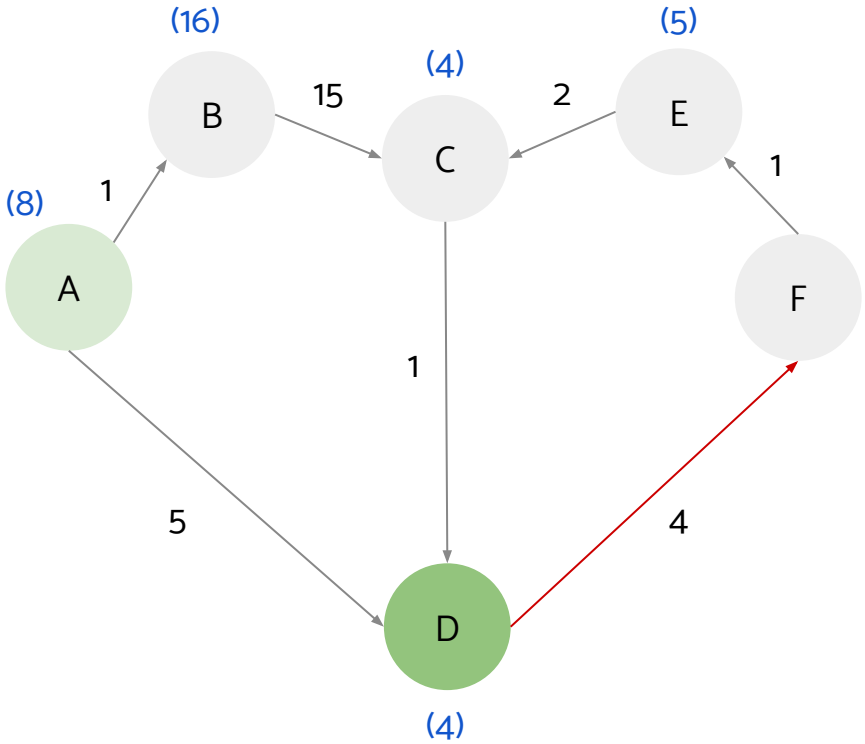
1B The Shortest Path To Your Heart Run A* from A to F.



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

Priority Queue:
B: 17 C: ∞ E: ∞ F: ∞

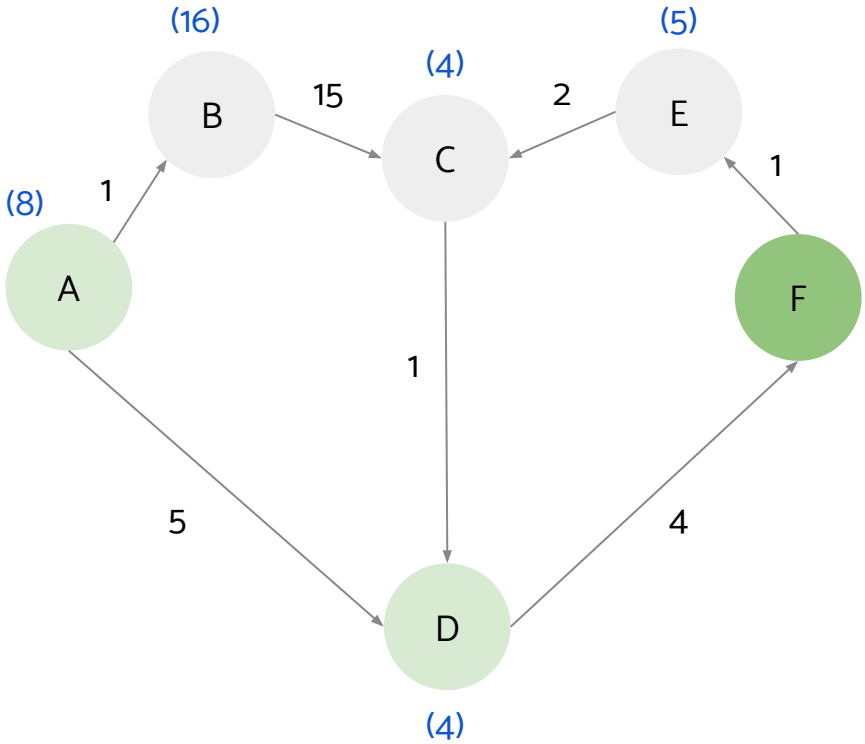
1B The Shortest Path To Your Heart Run A* from A to F.



	A	B	C	D	E	F
DistTo	0	1	∞	5	∞	9
EdgeTo	-	A	-	A	-	D

Priority Queue:
F: 9 B: 17 C: ∞ E: ∞

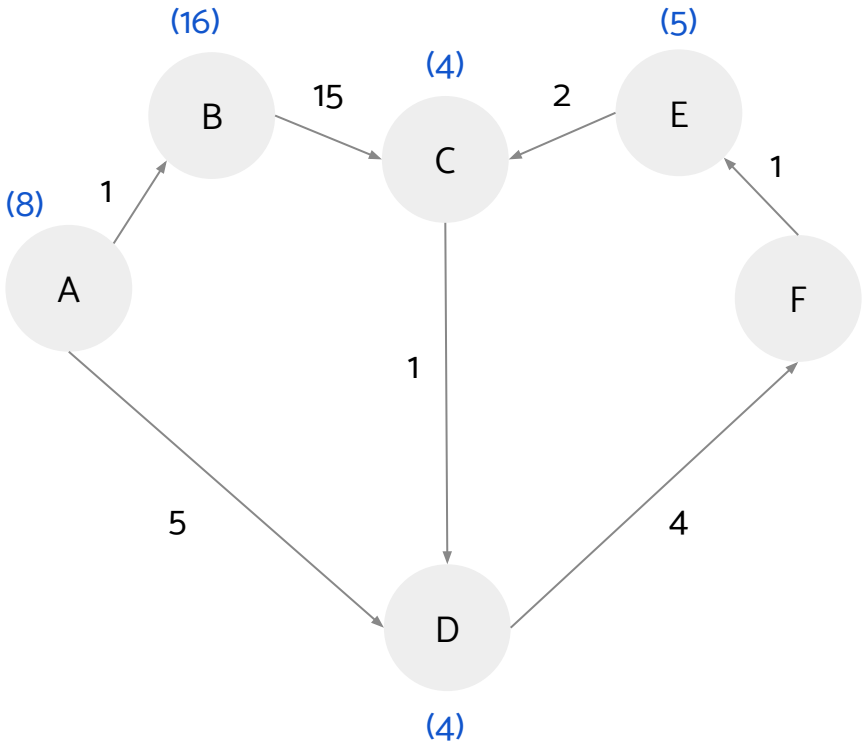
1B The Shortest Path To Your Heart Run A* from A to F.



	A	B	C	D	E	F
DistTo	0	1	∞	5	∞	9
EdgeTo	-	A	-	A	-	D

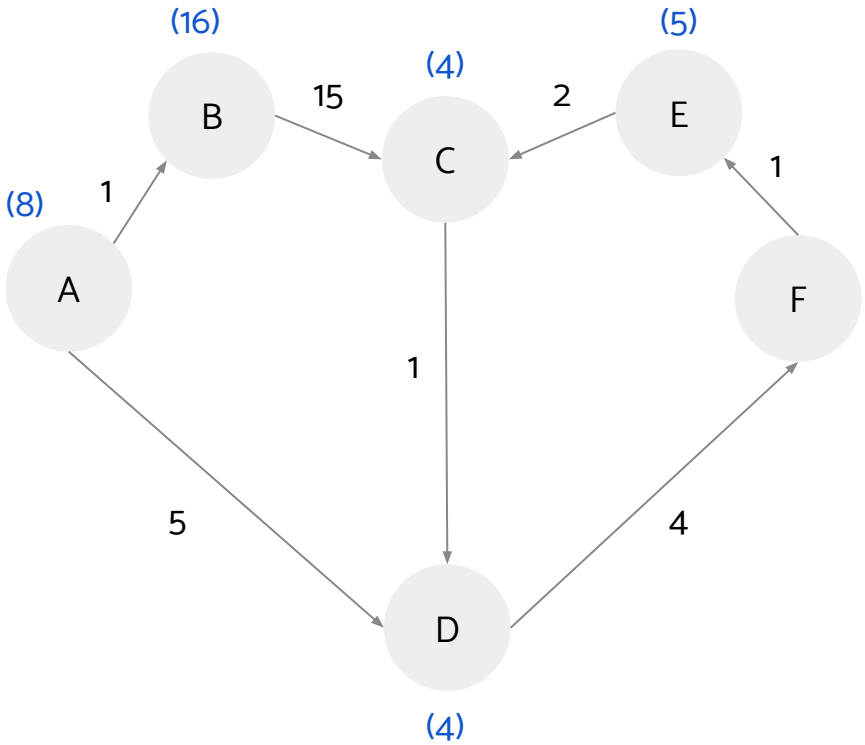
Priority Queue:
B: 17 C: ∞ E: ∞

1C The Shortest Path To Your Heart



Is the heuristic for this graph good? In other words, is it guaranteed that we will always find the shortest path from A to F?

1C The Shortest Path To Your Heart

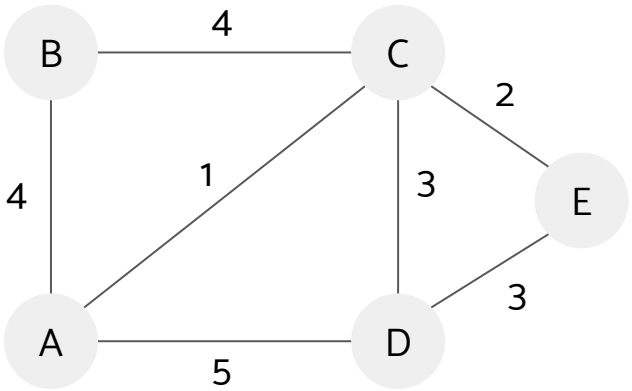


Is the heuristic for this graph good? In other words, is it guaranteed that we will always find the shortest path from A to F?

Yes, the given heuristic is both admissible and consistent.

2A 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.



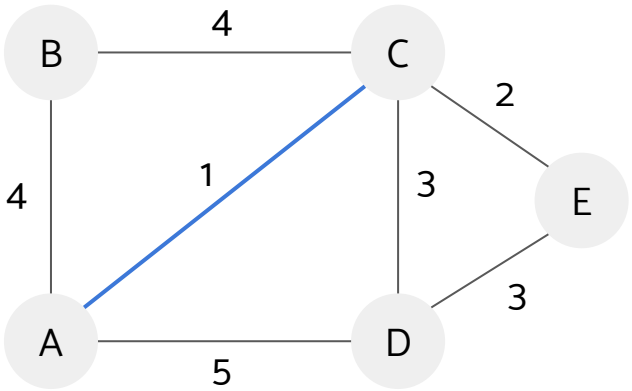
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



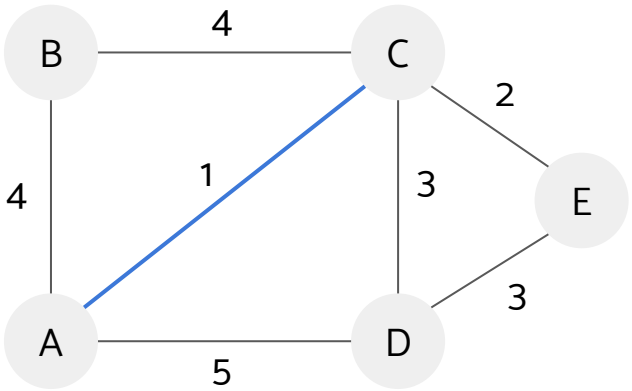
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



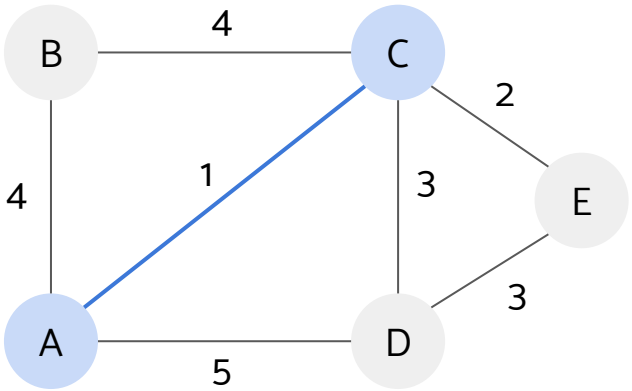
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



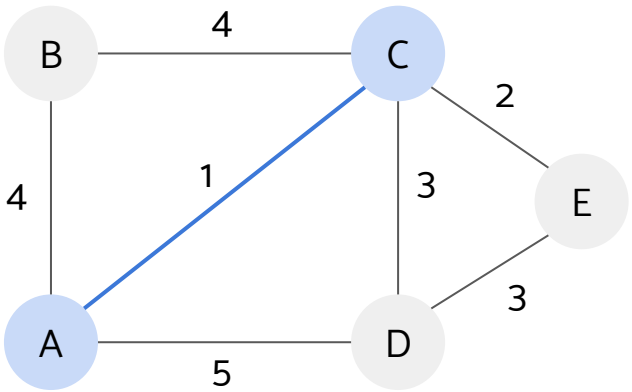
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.

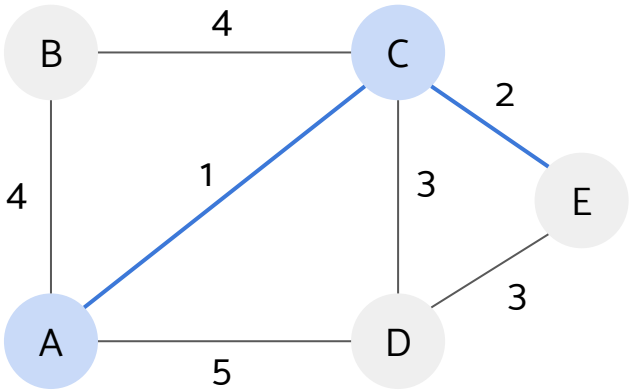


Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.**



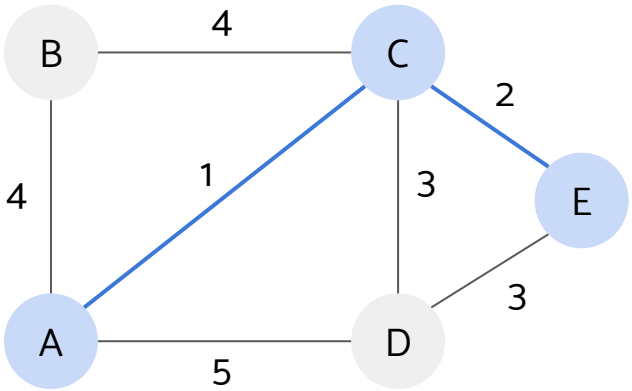
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



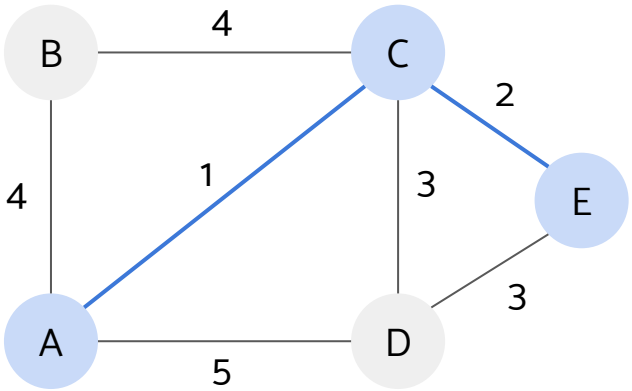
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



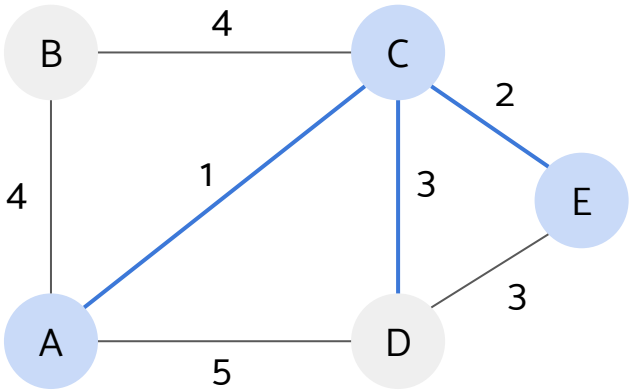
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.

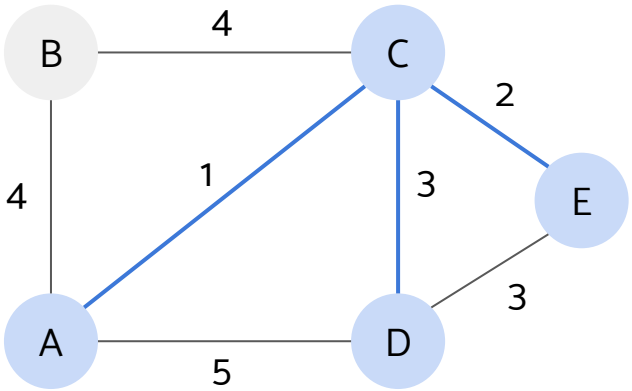


Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.**



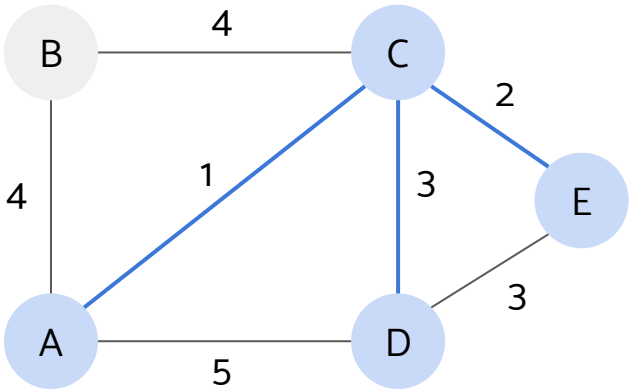
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



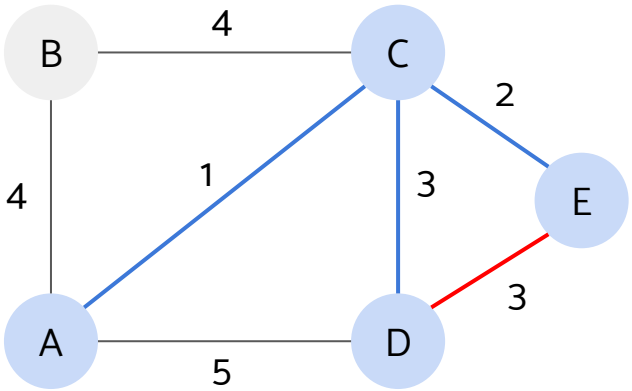
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



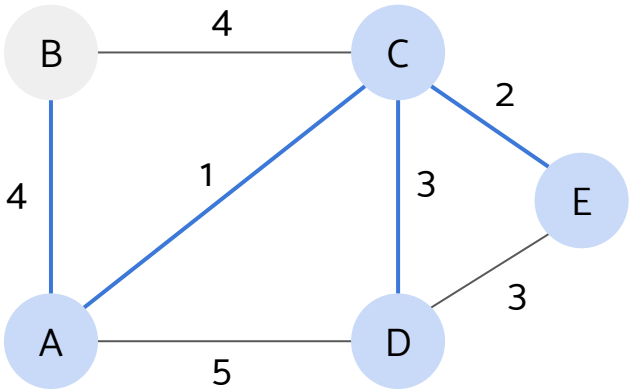
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.

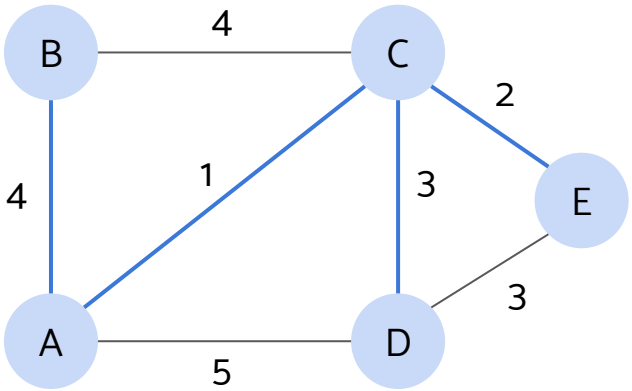


Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.**



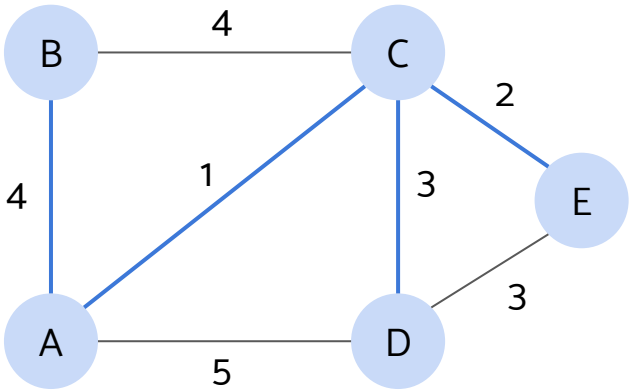
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.



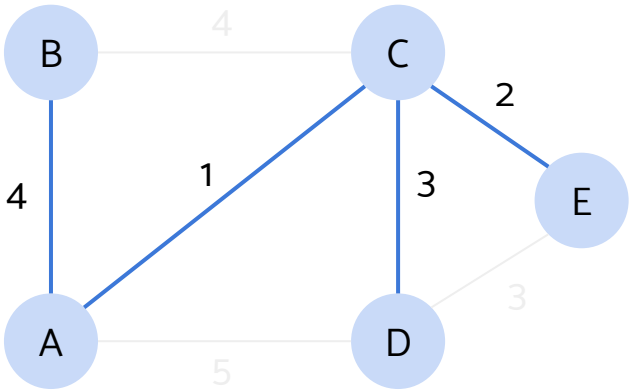
Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

2A 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.

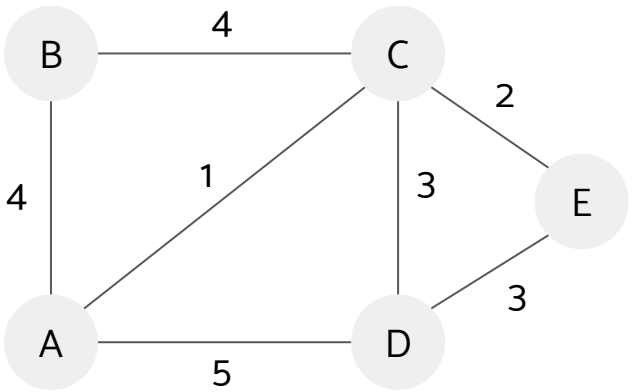


Kruskal's Algorithm

While there are less than V-1 edges in the MST:

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

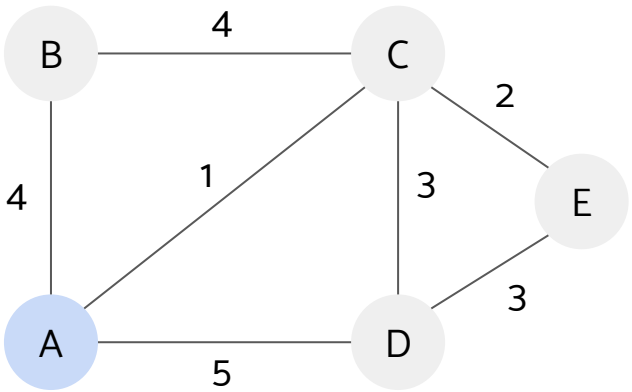
2A Minimalist Moles **Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.**



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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.**

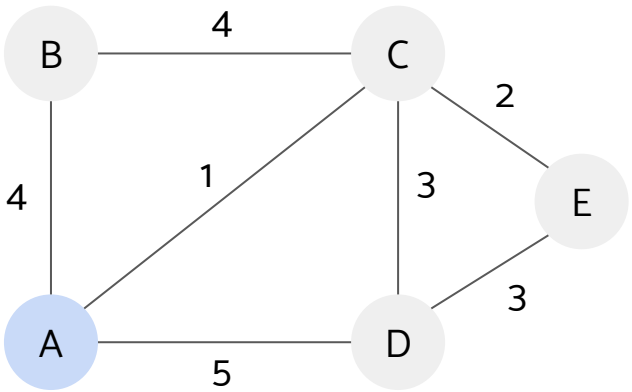


Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.

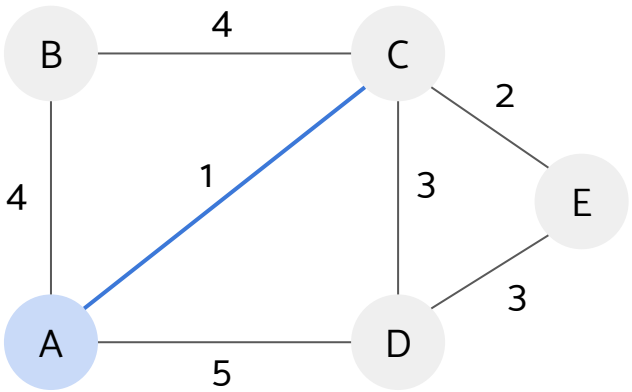


Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

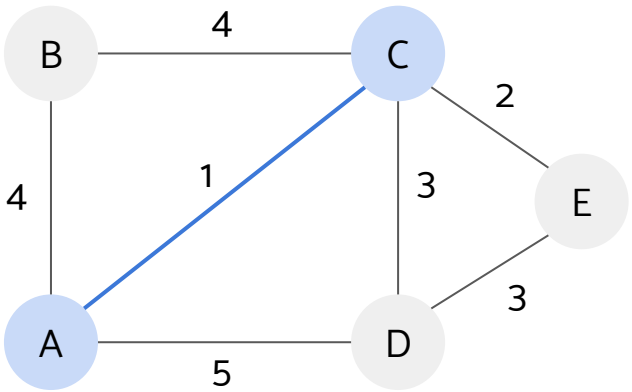
2A Minimalist Moles **Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.**



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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

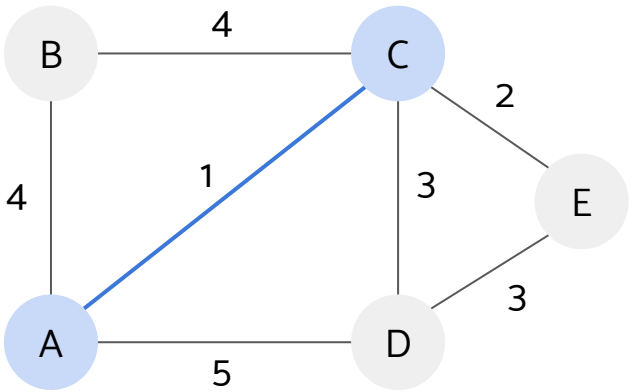
2A Minimalist Moles **Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.**



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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.**

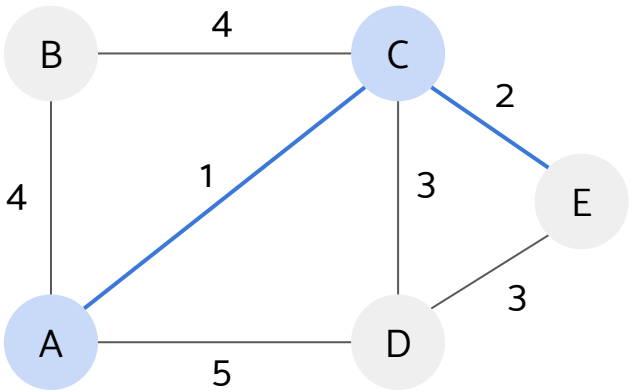


Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

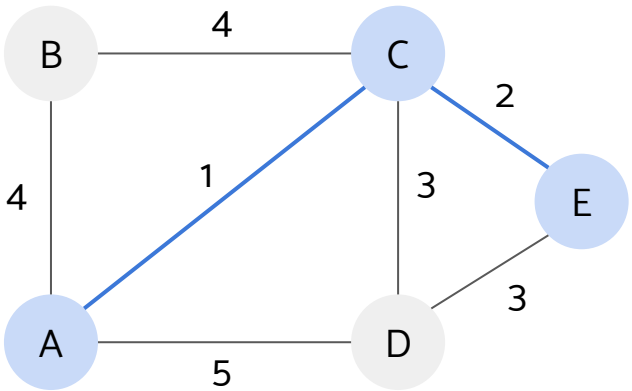
2A Minimalist Moles **Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.**



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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

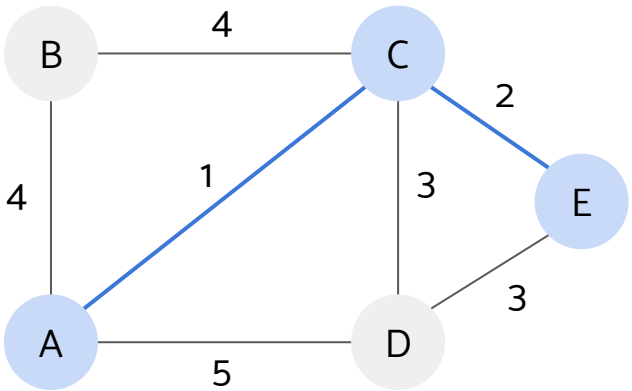
2A Minimalist Moles **Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.**



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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.

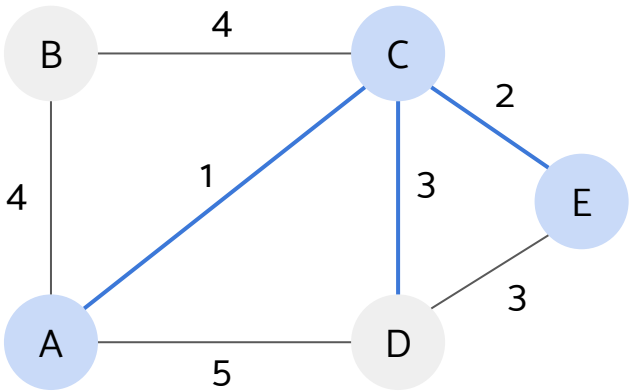


Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

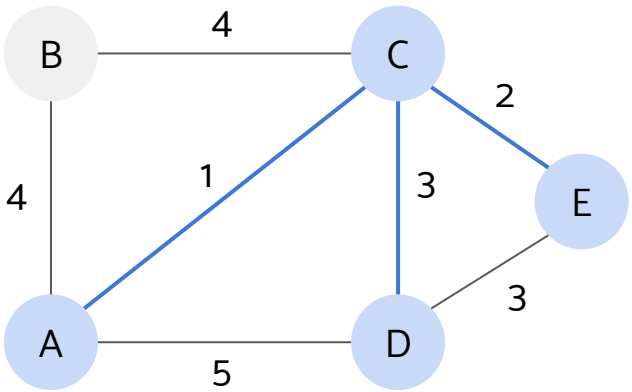
2A Minimalist Moles **Find a valid MST, using Kruskal's and then Prim's. For Prim's, take vertex A as your starting point. Break ties by choosing the edge that connects vertices of lower alphabetical order.**



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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.**

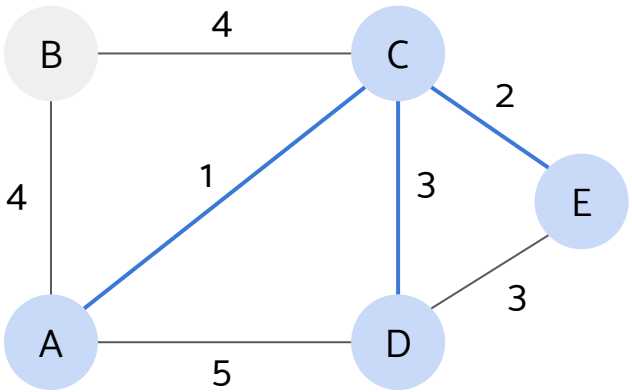


Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.



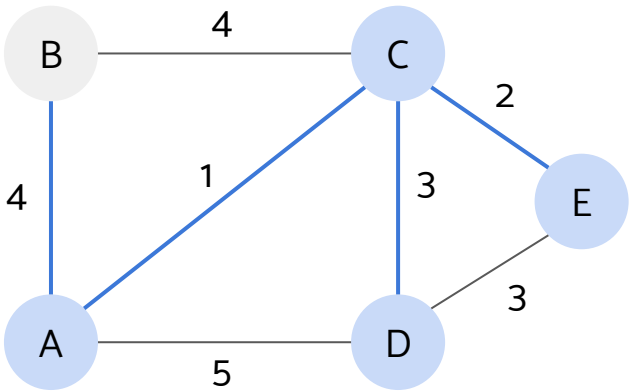
Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.

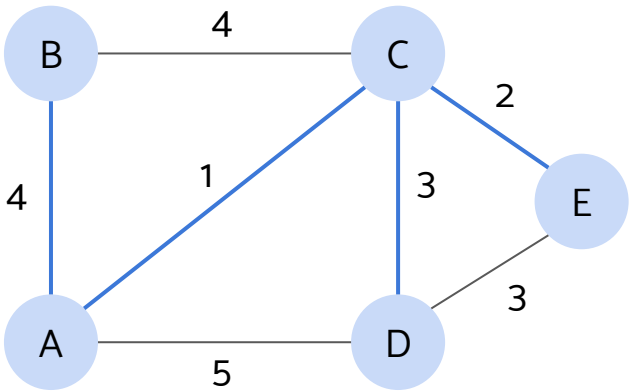


Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.**

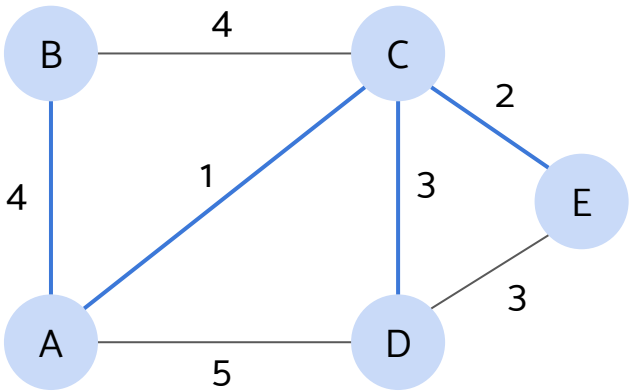


Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.



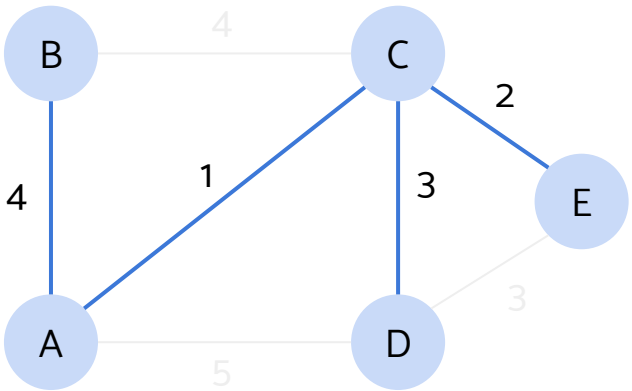
Prim's Algorithm

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 from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2A 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.



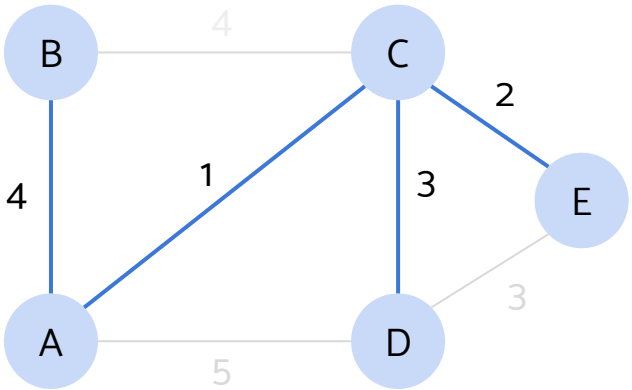
Prim's Algorithm

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

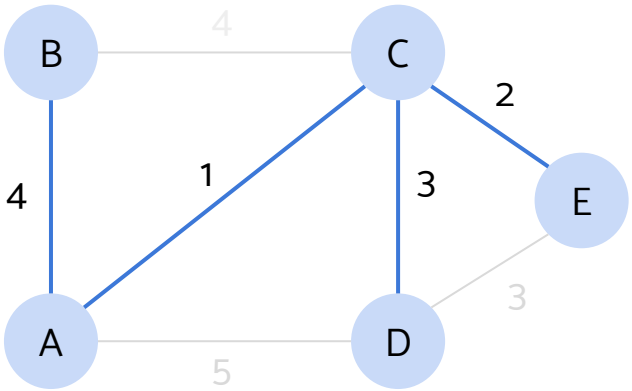
- Add the lightest edge from a node in the MST that leads to a new node that is unvisited.
- Add the new node to the set of nodes in the MST.

2B Minimalist Moles

Did Kruskal's and Prim's find different MSTs, or the same MST? More generally, is the MST for this graph unique? Describe a different tie-breaking scheme or make edits to the edge weights that would change your answer.



2B Minimalist Moles **Did Kruskal's and Prim's find different MSTs, or the same MST? More generally, is the MST for this graph unique? Describe a different tie-breaking scheme or make edits to the edge weights that would change your answer.**

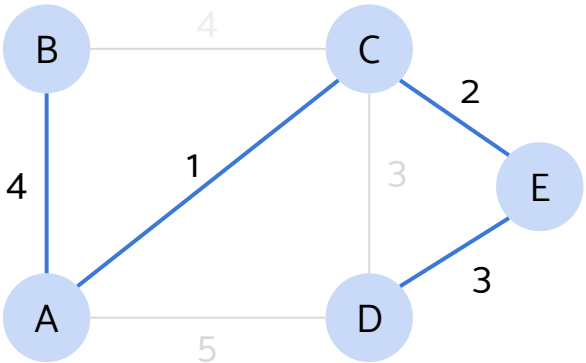


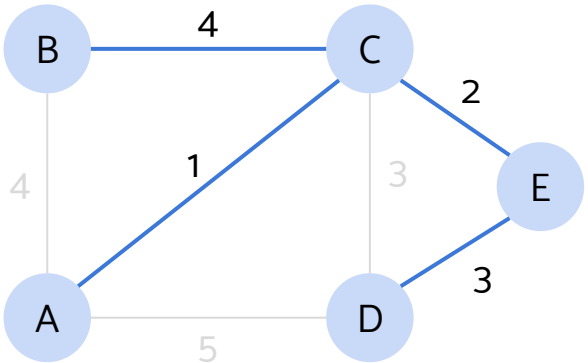
Both algorithms found the same MST.

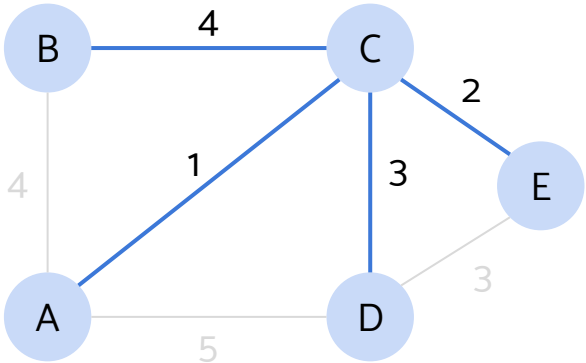
But there are duplicate edge weights, so the algorithms could find different MSTs if we used a different tiebreaking scheme!

2B Minimalist Moles **Did Kruskal's and Prim's find different MSTs, or the same MST? More generally, is the MST for this graph unique? Describe a different tie-breaking scheme or make edits to the edge weights that would change your answer.**

Examples:







3 Sticky Flights

Your airline company has been contracted to fly a large shipment of honey from Honeysville to the 61Bees in Apēs City. However, the airplane doesn't have enough fuel capacity to fly directly to Apēs City so it will stop at at least one of n airports along the way to refuel. Refueling takes an hour, and if the airport is one of k < n airports, your airplane will be grounded for six hours due to curfews (refueling is included in the six hours). The 61Bees want their honey as soon as possible so please design an algorithm to find the route that will allow your airplane to reach Apēs City in the least amount of hours.

Hint: Think of the n airports as a graph, where the paths between them are edges of weight equivalent to the number of hours it takes to fly from airport A to airport B. You may assume that the amount of time it takes to fly from A to B is equal to the amount of time it takes to fly from B to A.

3 Sticky Flights

We first create a graph with one node for each of the n airports and connect relevant airports with weighted edges corresponding to flight time.

From here, there are multiple ways we can adjust the graph to account for additional time costs! We can:

- Update the edge weights to account for refuel/grounding times
- Attach weights to the nodes themselves (ends up being similar to A*).
- Split airport nodes into two nodes connected by an edge with weight equal to refueling/grounding time

Then run Dijkstra's and backtrack to find the shortest path.