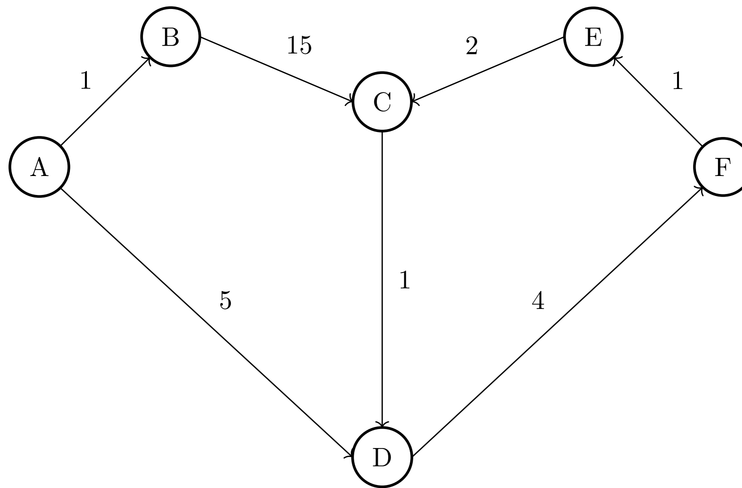
CS 61B
Spring 2025

Shortest Paths, MSTs
Discussion 09: March 31, 2025

1 The Shortest Path To Your Heart

For the graph below, let `g(u, v)` be the weight of the edge between any nodes `u` and `v`. Let `h(u, v)` be the value returned by the heuristic for any nodes u and v.



Below, the pseudocode for Dijkstra's and A∗ are both shown for your reference throughout the problem.

Dijkstra's Pseudocode

```
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
```

A∗ Pseudocode

```
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
```

(a) Run Dijkstra's algorithm to find the shortest paths from A to every other vertex. You may find it helpful to keep track of the priority queue. We have provided a table to keep track of best distances, and the adjacent vertex that has an edge going to the target vertex in the current shortest paths tree so far.

	A	B	C	D	E	F
distTo						
edgeTo						

Solution:

B = 1 ; D = 5 ; F = 9 ; E = 10 ; C = 12

Explanation: For the best explanation, it is recommended to check the slideshow linked on the website or watch the walkthrough video, as the text explanation is verbose.

We will maintain a priority queue and a table of distances found so far, as suggested in the problem and pseudocode. We will use {} to represent the PQ, and (()) to represent the distTo array.

{A:0, B:inf, C:inf, D:inf, E:inf, F:inf}. (()).

Pop A.

{B:inf, C:inf, D:inf, E:inf, F:inf}. ((A: 0)).

changePriority(B, 1). changePriority(D, 5).

{B:1, D:5, C:inf, E:inf, F:inf}. ((A: 0)).

Pop B.

{D:5, C:inf, E:inf, F:inf}. ((A: 0, B: 1)).

changePriority(C, 16).

{D:5, C:16, E:inf, F:inf}. ((A: 0, B: 1)).

Pop D.

{C:16, E:inf, F:inf}. ((A: 0, B: 1, D: 5)).

changePriority(F, 9).

{F: 9, C:16, E:inf, F:inf}. ((A: 0, B: 1, D: 5)).

Pop F.

{C:16, E:inf}. ((A: 0, B: 1, D: 5, F: 9)).

changePriority(E, 10).

{E:10, C:16}. ((A: 0, B: 1, D: 5, F: 9)).

Pop E.

{C:16}. ((A: 0, B: 1, D: 5, F: 9, E: 10)).

changePriority(C, 12).

{C:12}. ((A: 0, B: 1, D: 5, F: 9, E: 10)).

Pop C.

{}. ((A: 0, B: 1, D: 5, F: 9, E: 10, C: 12)).

At the end, our table looks like this:

	A	*B*	*C*	*D*	*E*	*F*
distTo	0	1	12	5	10	9
edgeTo	-	A	E	A	F	D

(b) Given the weights and heuristic values for the graph above, what path would A$*$ search return, starting from A and with F as a goal?

Edge Weights	**Heuristics**
$g(A,B)=1$	$h(A,F)=8$
$g(A,D)=5$	$h(B,F)=16$
$g(B,C)=15$	$h(C,F)=4$
$g(C,D)=1$	$h(D,F)=4$
$g(D,F)=4$	$h(E,F)=5$
$g(F,E)=1$	
$g(E,C)=2$	

	A	B	C	D	E	F
distTo						
edgeTo						

Solution: A∗ would return *A-D-F* . The cost here is 9.

	A	B	C	D	E	F
distTo	0	1	∞	5	∞	9
edgeTo	-	A	-	A	-	D

Explanation: A∗ runs in a very similar fashion to Dijkstra's. We got the same answer for the shortest path to F, though we actually explored less unnecessary nodes in the process (we never popped B, C, or E off the queue). The main difference is the priority in the priority queue. For A∗, whenever computing the priority (for the purposes of the priority queue) of a particular node n, always add *h(n)* to whatever you would use with Dijkstra's.

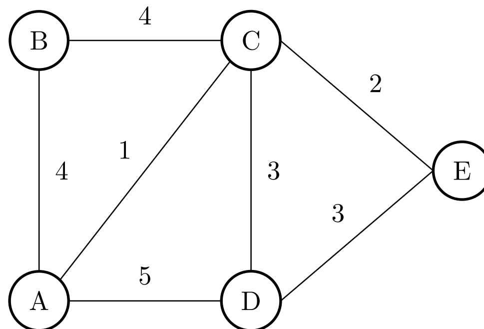
Additionally, note that A∗ will be run to find the shortest path to a particular goal node (as our heuristic is calculated as our estimate to our specific goal node), whereas Dijkstra's may be run with a specific goal, or it may be run to find the shortest paths to **ALL** nodes. In the solutions above, we found the shortest paths to all nodes, but if we only needed to know the shortest path to E, for example, we could have stopped after visiting E.

(c) Based on the heuristics for part b, is the A∗ heuristic for this graph good? In other words, will it always give us the actual shortest path from A to F? If it is good, give an example of a change you would make to the heuristic so that it is no longer good. If it is not, correct it.

Solution: The heuristic is admissible: for every node, the heuristic value is less than or equal to the shortest distance path from that node to the target node. It is also consistent: each estimate is less than or equal to the estimated distance from any neighboring vertex to the goal, plus the cost of reaching that neighbor. Because it is both admissible and consistent, we can say that the heuristic is good. If we changed the heuristic from D to F to be 6 (i.e. `h(D, F)` = 6), then the overall heuristic for the graph would no longer be admissible.

2 Minimalist Moles

Karen the mole wants to dig a network of tunnels connecting all of her secret hideouts. There are a few set paths between the secret hideouts that Karen can choose to possibly include in her tunnel system, shown below. However, some portions of the ground are harder to dig than others, and Karen wants to do as little work as possible. In the diagram below, the numbers next to the paths correspond to how hard that path is to dig. Lucky for us, she knows how to use MSTs to optimize the tunnel paths!



Below, the pseudocode for Kruskal's and Prim's are both shown for your reference throughout the problem.

Kruskal's Pseudocode

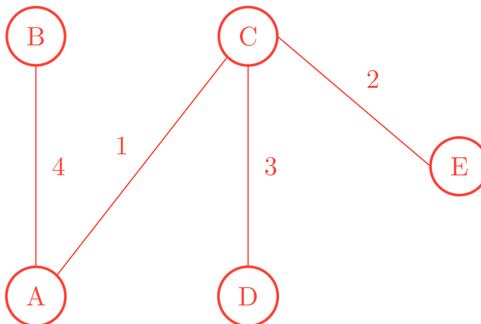
```
while there are still nodes not in the MST:
  add the lightest edge
      that does not create a cycle.
  add the new node to the
      set of nodes in the MST.
```

Prim's Pseudocode

```
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.
```

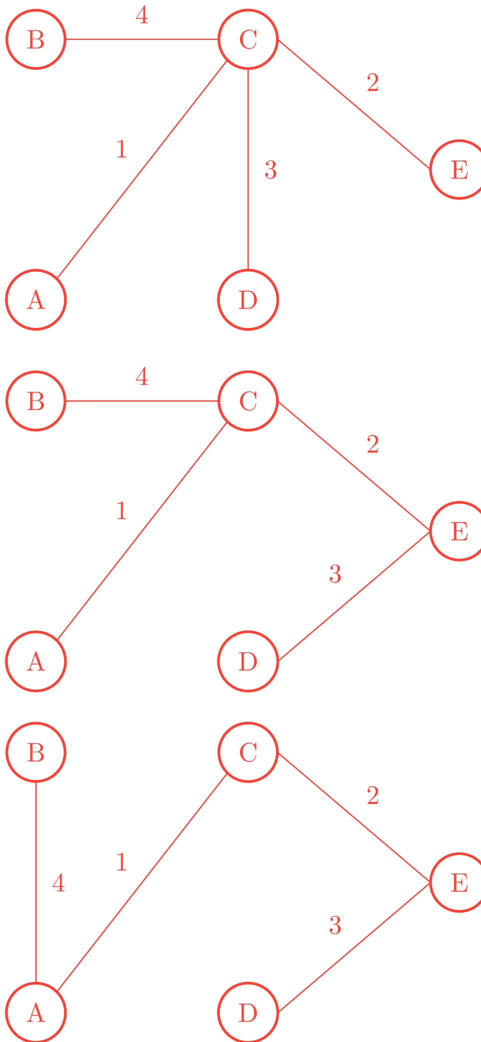
(a) Find a valid MST for the graph above using Kruskal's algorithm, then Prim's. For Prim's algorithm, take A as the start node. In both cases, if there is ever a tie, choose the edge that connects two nodes with lower alphabetical order.

Solution: Both Prim's and Kruskal's give the MST below.



(b) Are the above MSTs different or the same? If different, describe a tie-breaking scheme that would make them the same. If the same, describe a tie-breaking scheme that would make them different.

Solution: In this particular case, the trees for Prim's and Kruskal's are the same. However, because our graph has edges with duplicate weights, then **it would be possible for Prim's and Kruskal's to give different answers with a different tiebreaking scheme**. For example, if in the graph above, depending on which node we start Prim's from, as well as which tiebreaking scheme we use (ie. instead of lower alphabetical order, use higher alphabetical order, or randomness, or by most recently added node(s)), we could get other perfectly valid MSTs, like:



3 Sticky Flights

Your airline company has been contracted to fly a large shipment of honey from Honeysville to the 61Bees in Goldenhive City. However, the airplane doesn't have enough fuel capacity to fly directly to Goldenhive 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 Goldenhive 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.

Solution: Since we want to find a path of minimum time (weight), using a Shortest Paths Tree algorithm would make sense for this problem. The problem states that we can represent the airports as a graph, so we first create the graph. We have one node for each of the n airports and for all airports directly reachable from a particular airport, we create an undirected edge with the flight time (in hours) between the two airports as the edge weight. From here, there are multiple approaches we can take to adjust the graph.

1. We can increase the edge weights by the associated refueling or grounding time. Think of undirected edges as two directed edges pointing in opposite directions; we can increase the weight of the directed edge by the refueling/grounding time of the node it points to.

2. We can attach the additional weights to the nodes themselves and modify Dijkstra's algorithm to take into account both edge and node weight. This approach will end up looking very similar to A$*$.

3. For this option, we must create a directed graph rather than an undirected graph. We can split airports into two nodes. We attach all incoming edges to the original node to one node ("left" node) and all outgoing edges to the other ("right" node). Finally we connect the two nodes with a directed edge going from the left node to the right node of weight equal to the refueling/grounding time.

Once the graph is prepared, we run Dijkstra's algorithm starting at the node corresponding to Honeysville and terminate once the node corresponding to Goldenhive City is popped off the fringe. The `distTo` value of Goldenhive City is the minimum time and backtracking from Goldenhive City to Honeysville gives the shortest path.