CS 61B Spring 2026

Shortest Paths and MSTs

Exam-Level Discussion 10: March 30, 2026

1 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.

2 Multiple MSTs

Recall a graph can have multiple MSTs if there are multiple spanning trees of minimum weight.

(a) For each subpart below, select the correct option and justify your answer. If you select "never" or "always," provide a short explanation. If you select "sometimes," provide two graphs that fulfill the given properties.

1. If **some** of the edge weights are **identical**, there will

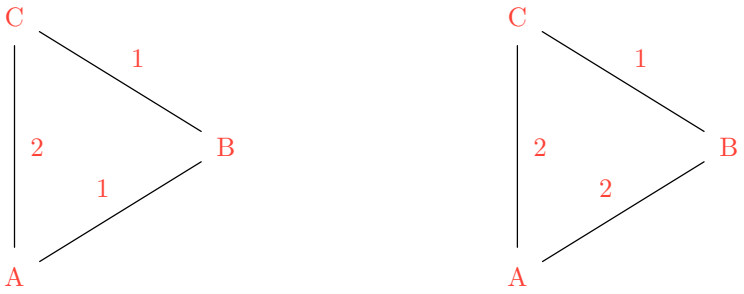
   ◯ never be multiple MSTs in G.

   ◯ sometimes be multiple MSTs in G.

   ◯ always be multiple MSTs in G.

   Justification:

   Solution: If **some** of the edge weights are **identical**, there will **sometimes** be multiple MSTs in G.



   Justification:

   In the graph on the left, the only MST is [AB, BC]. In the graph on the right, two MSTs exist — [AB, BC] and [AC, BC].

2. If **all** of the edge weights are **identical**, there will

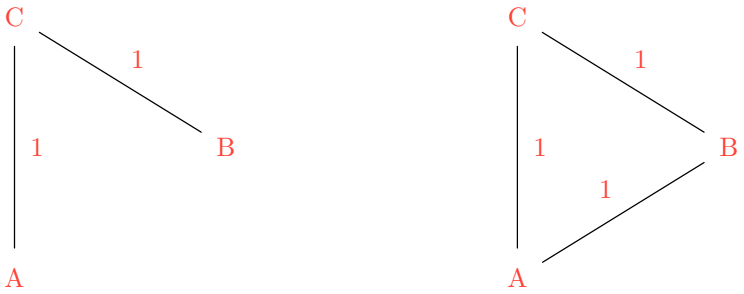
   ◯ never be multiple MSTs in G.

   ◯ sometimes be multiple MSTs in G.

   ◯ always be multiple MSTs in G.

   Justification:

   Solution: If **some** of the edge weights are **identical**, there will **sometimes** be multiple MSTs in G.



   Justification:

   In the graph on the left, the only MST is [AB, AC]. Note that for any tree, we only have one MST, since the tree itself is the MST! In the graph on the right, three MSTs exist — [AB, BC], [AC, BC], and [AB, AC].

(b) Suppose we have a connected, undirected graph (G) with (N) vertices and (N) edges, where all the **edge weights are identical**. Find the maximum and minimum number of MSTs in (G) and explain your reasoning.

Minimum:

Maximum:

Justification:

Solution:

Minimum: 3

Maximum: N

Justification: Notice that if all the edge weights are the same, an MST is just a spanning tree. Let's begin by creating a tree, i.e. a connected graph with $N-1$ edges. Now, notice that there is only one spanning tree, since the graph is itself a tree.

As such, the problem reduces to: how many spanning trees can the insertion of one edge create? If we add an edge to a tree, it will create a cycle that can be of length at minimum 3 and at maximum N. Then, notice that we can only remove **any** edge from a cycle to create a spanning tree, so we have at minimum 3 and at maximum N possible MSTs in G.

(c) It is possible that Prim's and Kruskal's find different MSTs on the same graph G (as an added exercise, construct a graph where this is the case!).

Given any graph G with integer edge weights, modify the edge weights of G to *ensure* that

(1) Prim's and Kruskal's will output the same results, and

(2) the output edges still form a MST correctly in the original graph.

You may not modify Prim's or Kruskal's, and you may not add or remove any nodes/edges.

Hint: Look at subpart 1 of part (a).

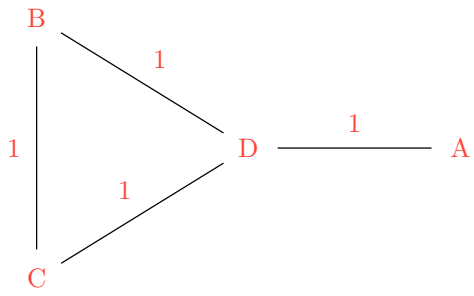
Solution:

To ensure that Prim's and Kruskal's will always produce the same MST, notice that if G has unique edges, only one MST can exist, and Prim's and Kruskal's will always find that MST! So, what if we modify G to ensure that all the edge weights are unique?

To achieve this, let's strategically add a small, unique offset between 0 and 1, exclusive, to each edge. It is important that we choose an offset between 0 and 1. This is to ensure that the edges picked in the modified graph is still a correct MST in the original graph, since all the edge weights are integers. It is also important that the offset is unique for each edge, because then we ensure each weight is distinct. Pseudocode for such a change is shown below:

```
E = number of edges in the graph
    offset = 0
    for edge in graph:
        edge.weight += offset
        offset += 1 / E
```

In regard to the added exercise, here is a simple graph G where Prim's and Kruskal's produce different MSTs. Prim's starting from A will select AD, BD, and CD, whereas Kruskals will select AD, BC, and BD.



3 Sticky Railroads

Two cities, Chicago and Berkeley, are located in the United States. The railroad system connecting them can be modeled as a **weighted directed graph**, with V vertices, E edges, and weights representing the length of the railway. Ethan wishes to take a railway from Chicago to Berkeley, and needs to determine the shortest railway distance between them.

Define the set C to be all cities in Chicago, and B to be all cities in Berkeley. There can be cities that belong to neither region along the way. The shortest distance between the two cities is the shortest distance between any city c_C in Chicago and c_B in Berkeley.

Describe an algorithm that computes the minimum railway distance from Chicago to Berkeley, in $O((V + E) \log V)$ time. You are able to utilize all graph algorithms you learned in class.

Hint: Consider modifying the graph so that running a graph algorithm yields an equivalent answer to solving the original problem.

Solution: Create a dummy vertex d. For each city in **Chicago**, create an edge from d to that city with edge weight 0. This creates a new graph G'. Run Dijkstra's algorithm, starting from the dummy vertex d, on the graph G'. This yields the distance from the dummy vertex d to all cities in the network. Now, because we are only interested in the minimum distance to any city in **Berkeley** (B), we can iterate through all cities in B and take the one with the minimum distance.