CS 61BL Summer 2026

Disjoint Sets, Shortest Paths, and MSTs

Exam Prep 04: July 31, 2026

1 Union Find

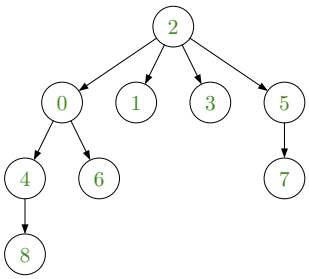
(a) Assume we have nine items, represented by integers 0 through 8. All items are initially unconnected to each other. Draw the union find tree, draw its array representation after the series of `union()` and `find()` operations, and write down the result of `find()` operations using WeightedQuickUnion without path compression. Break ties by choosing the smaller integer to be the root.

Note: find(x) returns the root of the tree for item x.

```
union(2, 3);
union(1, 2);
union(5, 7);
union(8, 4);
union(7, 2);
find(3);
union(0, 6);
union(6, 4);
union(6, 3);
find(8);
find(6);
```

`find()` returns `2, 2, 2` respectively.

The array is `[2, 2, -9, 2, 0, 2, 0, 5, 4]`.



(b) For each of the arrays below, write whether this could be the array representation of a weighted quick union with path compression and explain your reasoning. Assume the negative entries mean the node is root and its absolute value is the number of elements contained in the subtree.

```
     i:    0 1 2 3 4 5 6  7 8 9
   ---------------------------
A. a[i]:   1 2 3 0 1 1 1  4 4 5
B. a[i]:   9 0 0 0 0 0 9  9 9 -10
C. a[i]:   1 2 3 4 5 6 7  8 9 -10
D. a[i]: -10 0 0 0 0 1 1  1 6 2
E. a[i]: -10 0 0 0 0 1 1  1 6 8
F. a[i]:  -7 0 0 1 1 3 3 -3 7 7
```

A. Impossible: has a cycle 0-1, 1-2, 2-3, and 3-0 in the parent-link representation.

B. Impossible: the nodes 1, 2, 3, 4, and 5 must link to 0 when 0 is a root; hence, 0 would not link to 9 because 0 is the root of the larger tree.

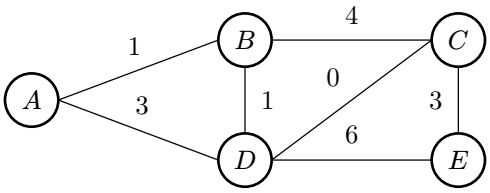
C. Impossible: tree rooted at 9 has height 9 $>\lg 10$.

D. Possible: 8-6, 7-1, 6-1, 5-1, 9-2, 3-0, 4-0, 2-0, 1-0.

E. Impossible: tree rooted at 0 has height 4 $>\lg 10$.

F. Impossible: tree rooted at 0 has height 3 $>\lg 7$.

2 Dijkstra's Algorithm



(a) Run Dijkstra's Algorithm on the graph above starting from vertex A. Break ties alphabetically.

i. Fill in how the priority values change below. When you remove a node from the priority queue, mark it with a check, and leave it blank for the subsequent rows.

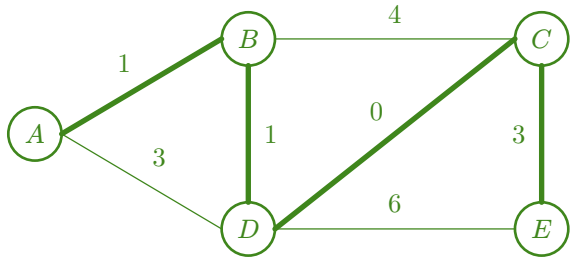
ii. Sketch the resulting shortest paths tree in the end.

Node	A	B	C	D	E
Start	0	∞	∞	∞	∞
Iter 1	✓				
Iter 2					
Iter 3					
Iter 4					
Iter 5					

Solution:

Node	A	B	C	D	E
Start	0	∞	∞	∞	∞
Iter 1	✓	1	∞	3	∞
Iter 2		✓	5	2	∞
Iter 3			2	✓	8
Iter 4			✓		5
Iter 5					✓

The SPT of the whole graph is drawn below for your reference. You should be able to "read-off" the `edgeTo` pointers by locating the last time the priority value of a node changes, and search for where the "checkmark" is for that row - that row would be the edge you came from in the SPT!



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.

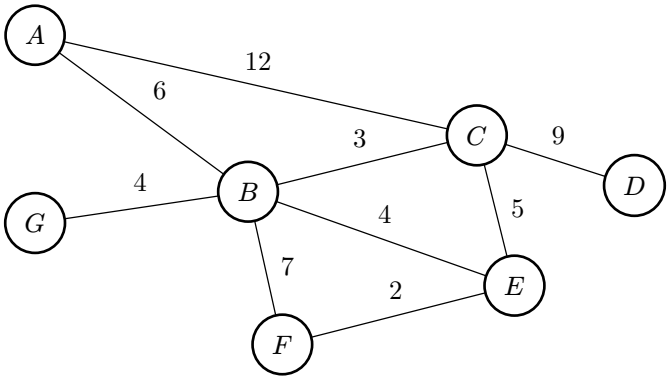
4 MST Algorithms

- **Prim's algorithm:** Start from any vertex. Repeatedly add the shortest edge that connects a vertex in the tree to a vertex not yet in the tree.
- **Kruskal's algorithm:** Sort all edges by weight. Add edges in order from smallest to largest, skipping any edge that would create a cycle.

(a) Fill in the blank with an expression using V (the number of vertices) and/or E (the number of edges):

Both algorithms stop when all V vertices are connected to the tree, which occurs when the tree contains _____ edges.

$V - 1$ edges. A tree with V vertices always has exactly $V - 1$ edges.



(b) For the graph above, list the edges in the order they're added to the MST by Kruskal's and Prim's algorithm. Assume Prim's algorithm starts at vertex A. Assume ties are broken in alphabetical order. Denote each edge as a pair of vertices (e.g. AB is the edge from A to B). The first edge is shown for you.

Prim's algorithm order: AB,

Kruskal's algorithm order: EF,

Solution: Prim's algorithm order: **AB, BC, BE, EF, BG, CD**
Kruskal's algorithm order: **EF, BC, BE, BG, AB, CD**

(c) True/False: Adding 1 to the smallest edge of a graph G with unique edge weights must change the total weight of its MST.

Solution: True, either this smallest edge (now with weight +1) is included, or this smallest edge is not included and some larger edge takes its place since there was no other edge of equal weight. Either way, the total weight increases.

(d) True/False: If all edge weights in a graph are unique, there is only one possible MST.

Solution: True, the cut property states that the minimum weight edge in a cut must be in the MST. Since all weights are unique, the minimum weight edge is always unique, so there is only one possible MST.

(e) True/False: The shortest path from vertex u to vertex v in a graph G is the same as the shortest path from u to v using only edges in T, where T is the MST of G.

Solution: False, consider vertices C and E in the graph above. The shortest path between C and E uses the edge CE, but it is not part of the MST of the graph.

5 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.