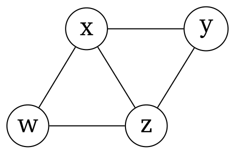
MSTs & Disjoint Sets

Due date: Friday May 21, 2021 at 11:59 pm PDT

Instructions: Submit your responses to the "EX4: Graphs, MSTs, and Disjoint Sets" assignment on Gradescope here: https://www.gradescope.com/courses/259163/assignments/1154092. Make sure to log in to your Gradescope account using your UW email to access our course.

1. Minimum Spanning Trees

Consider the graph below, which has four vertices (w, x, y, z) and five undirected edges.



1.1.

Label the edges in the graph with non-negative weights **1, 2, 3, 4, 5** such that the shortest path from w to y does not entirely consist of edges that also fall within a minimum spanning tree of the graph. You should use each weight on exactly one edge.

1.2.

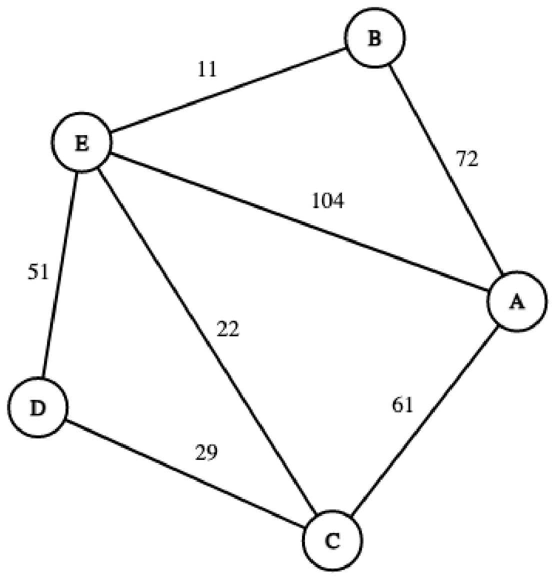
Label the edges in the graph with non-negative weights **1, 2, 3, 4, 5** such that all of the edges in the shortest path tree starting from w are also edges in a minimum spanning tree of the graph. You should use each weight on exactly one edge.

1.3.

Label the edges in the graph with weights **1, 2, 3, 4, -5** such that Dijkstra's algorithm starting from w would compute an incorrect shortest path from w to y. You should use each weight on exactly one edge.

2. Running MST Algorithms

Consider the graph below.



2.1.

What order are edges added to the MST when running **Kruskal's algorithm**? There is no particular formatting we are looking for, as long as it's clear the order in which the edges are added.

2.2.

What order are edges added to the MST when running **Prim's algorithm** starting from vertex B? There is no particular formatting we are looking for, as long as it's clear the order in which the edges are added.

2.3.

Is the following statement always, sometimes, or never true. Explain your reasoning. We make this question a file upload so that you can include a picture with your answer if you want, but that is not a requirement.

Running Prim's on a graph will always return the same MST, regardless of the start vertex. Assume there is a well-defined ordering of the nodes so that we always break ties based on the ordering of the destination vertex (e.g., alphabetical ordering).

3. Disjoint Sets

3.1. WeightedQuick Union (no Path Compression)

Consider we initialize a disjoint set structure over the elements A,B,C,D,E,F. When unioning sets together, use the approach described for WeightedQuickUnion (no path compression). If there is ever a tie between which tree should be placed below another, put the tree for the first argument to union under the tree for the second argument.

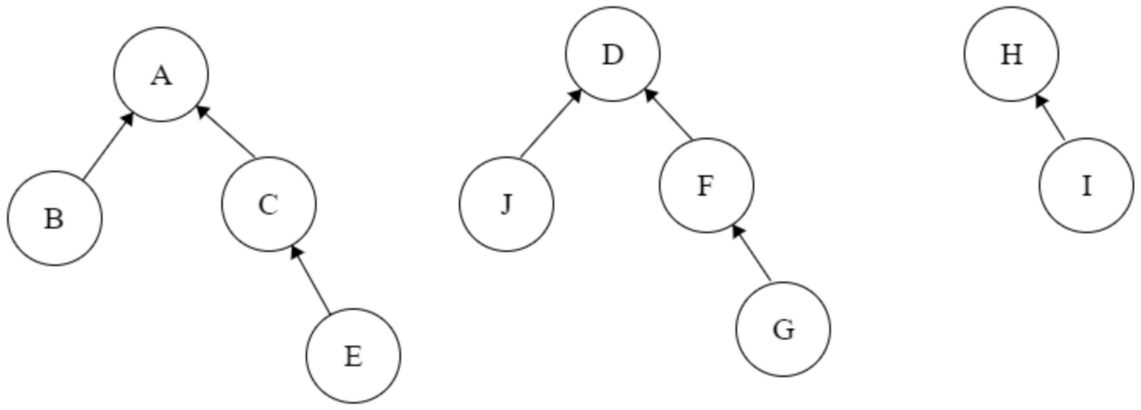
Draw the state of the up-trees in WeightedQuickUnion after the following calls (assuming the disjoint set has been initialized).

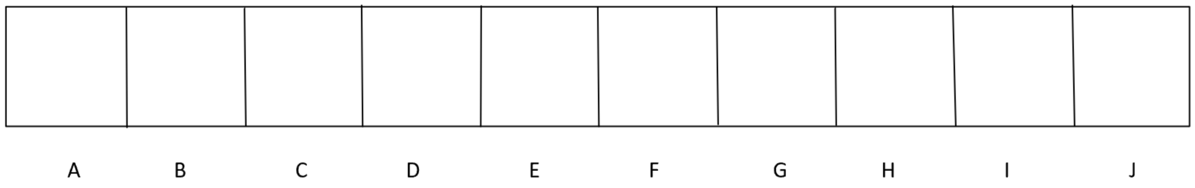
```
union(A, B)
union(A, C)
union(D, E)
union(E, C)
```

Upload a image for your answer. While not required, it may help to show your work.

3.2. Array WeightedQuickUnion with Path Compression

Consider the Array WeightedQuickUnion (with Path Compression) implementation disjoint sets. We show the abstract view of the state as up-trees in the image below.





(a) Give the array representation for the disjoint sets above. Note you should write the index as a number since arrays are indexed as such. The picture above shows the mapping from which value goes to which index.

(b) Draw the array representation for data structure after the call `union(G, H)`.