CS 61B Spring 2022

WQU & Graphs

Discussion 14: April 25, 2022

1 Weight Times

A Quick Union data structure is used to handle set union and membership operations. The supported methods are:

1. `connect(a, b)` - connects the set of a to the set of b
2. `isConnected(a, b)` - returns true if a and b are in the same set

Example:

```
connect(a, b)
connect(b, c)
connect(a, d)
isConnected(c, d)
isConnected(d, b)
```

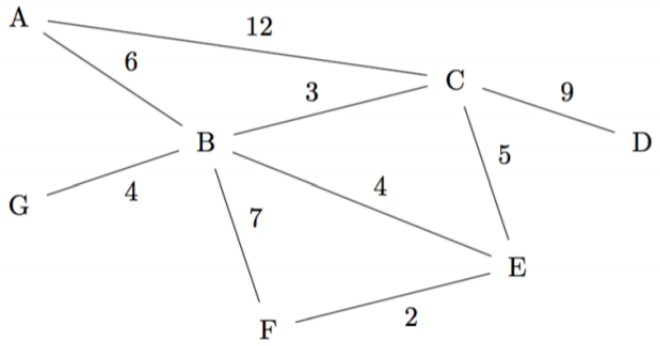
Internally, a Quick Union's sets are represented using trees. Sets can be connected by adding one set's tree to the root of another set's tree. Note that Weighted Quick Union data structures are similar to Quick Union data structures, except that a Weighted Quick Union will always add the shorter tree to the root of the taller tree during connect operations. You may break ties by setting the smaller number as the root and assume that the corresponding array holds either the parent of each node or the weight of the node the set is in if the node is the root.

(a) Draw the Weighted Quick Union object that results after the following four method calls:

```
connect(1, 3)
connect(0, 4)
connect(0, 1)
connect(0, 2)
```

(b) What is the resulting array of the Weighted Quick Union after the calls in (a) are executed?

2 Introduction to MSTs



(a) 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)

Prim's algorithm order:
Kruskal's algorithm order:

(b) Is there any vertex for which the shortest paths tree from that vertex is the same as your Prim MST? If there are multiple viable vertices, list all.

(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

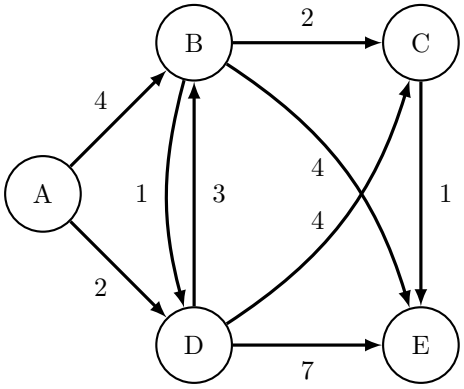
(d) True/False: The shortest path from vertex A to vertex B in a graph G is the same as the shortest path from A to B using only edges in T, where T is the MST of G.

(e) True/False: Given any cut, the maximum-weight crossing edge is in the maximum spanning tree.

3 Dijkstra's Algorithm

(a) Given the following graph, run Dijkstra's algorithm starting at node A. For each iteration, write down the entire state of the algorithm. This includes the value `dist(`v`)` for all vertices v as well as what node was popped off of the fringe for that iteration.

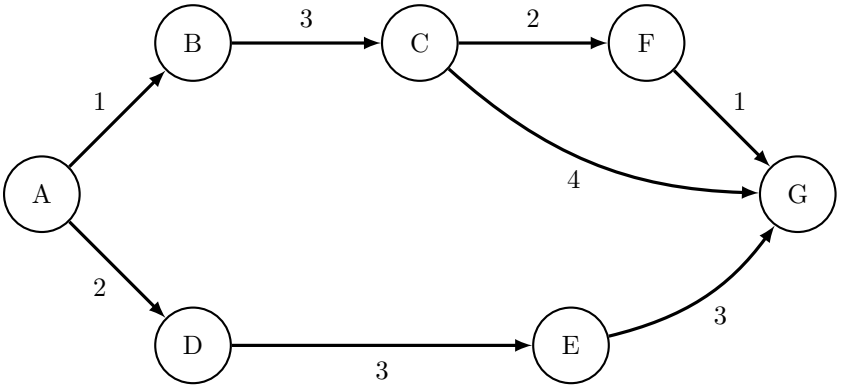
Note: If you want to keep track of the vertices traversed along the shortest paths from A to every other node in the graph, you will need to maintain an edgeTo array.



(b) What must be true about our graph in order to guarantee Dijkstra's will return the shortest path's tree to every vertex? Draw an example of a graph that demonstrates why Dijkstra's might fail if we do not satisfy this condition.

4 A* Search

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`. Remember the heuristic serves to estimate the distance between two nodes `u` and `v`.




```
Edge weights:   Heuristics:
g(A, B) = 1     h(A, G) = 8
g(B, C) = 3     h(B, G) = 6
g(C, F) = 2     h(C, G) = 5
g(C, G) = 4     h(F, G) = 1
g(F, G) = 1     h(D, G) = 6
g(A, D) = 2     h(E, G) = 3
g(D, E) = 3
g(E, G) = 3
```

(a) Given the weights and heuristic values for the graph above, what would A* search return as the shortest path from A to G?

(b) Is the heuristic admissible? Why or why not? A heuristic is admissible if it never overestimates the distance it is estimating.