ICS 311 Spring 2016, Problem Set 09, Topics 15-17

Due by midnight Tuesday 4/05

Adam Butac

#1. Peer Credit Assignment

1 Point Extra Credit for replying

Please list the names of the other members of your peer group for class this past week and the number of points you think they deserve for their participation in group work on the two days combined.

- You have a total of 6 points to allocate across all of your peers.
- You can distribute the points equally, give them all to one person, or do something in between.
- You need not allocate all the points available to you.
- *You cannot allocate any points to yourself!* Points allocated to yourself will not be recorded.

Marco de Lannoy Kobayashi 2

Nicole Hanabusa 2

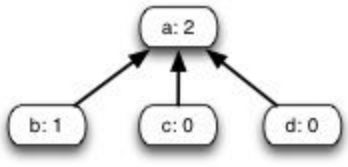
Jessie Flores 2

---

2. (2 pts) Constructing a Branchy Disjoint-Set Forest

Start with this disjoint-set forest:

Using only the operations Union and Find, write the sequence of calls that would create this disjoint set representation from the above nodes:



The only detailed definition for Union I could find was in podcast Topic 16A, 19m25s, which is also in the book, page 571. In that definition, Union calls Link and Find-Set.

```
Union(a,b)
Union(b,c)
Union(a,d)
Find(c)
```

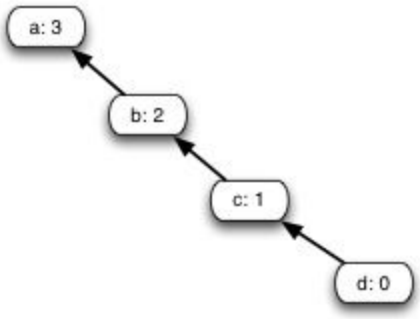
---

3. (4 pts) Linear Disjoint-Set Forest

With binary search trees there was always the possibility that a bad sequence of data would lead to a linear tree. Here we explore whether we need to worry about this with disjoint-set forests.

Start with this disjoint-set forest:

a. Using only the operations Union and Find, write the sequence of calls that would create this disjoint set representation from the above nodes:




```
Union(a,b)
Union(b,c)
Union(c,d)
```

b. In general, what is required to get a node of rank k?

```
It is not clear what is being asked.
If this is about time complexity, getting a node of rank k would
require climbing up a chain of nodes, which will take linear
time on the order of k, so θ(k) time.
```

c. Why will this never happen when union-find is used as a utility by Connected-Components and Kruskal's algorithm?

```
This would never happen because of the compression that occurs
```

when Union-Find is called.

---

4. (8 pts) MST on Restricted Range of Integer Weights

Suppose edge weights are restricted as follows. For each restriction,

- Describe a modification to Kruskal's algorithm that takes advantage of this restriction, and
- Analyze its runtime to prove that your modified version runs faster.

a. All edge weights are integers in the range from 1 to C for some constant C.

There is no longer a need to sort the edges which takes O(E*lg(E)) at the average case with the best known sorting algorithms.

Knowing that the edge weights range from 1 to C allows us to loop from 1 to C, which takes constant time, over each edge(u,v), which is O(E).

b. All edge weights are integers in the range from 1 to |V|.

Same as above, except now it is O(|V|), and since the Make-Set also costs O(|V|) total execution time is now O(|V|)

---

5. (4 pts) Suppose we change the representation of edges from adjacency lists to <u>matrices</u>. *(This is the Kruskal follow-up to what we did for Prim's in class.)*

a. What line(s) would have to change in Kruskal's algorithm, and how?

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

```
4 for u = 1 to G.E.length
5   for v = 1 to G.E[u].length
6     min = ∞
7     for i = v to G.E[u].length
8       if min <= G.E[u][i]
9         min = G.E[u][i]
10    if min < ∞ and Find-Set(u) ≠  Find-Set(v)
...
```

b. What would be the resulting asymptotic runtime of Kruskal's algorithm, and why?

$$O(E * \sum_{1}^{E} i) \approx O(E * E(E-1)/2) \approx O(E^3)$$

In my example three loops are required. The two outermost loops are needed to iterate over all possible edges(this could be optimized). The innermost loop is used to find the minimum edge within the current adjacency array.

It would *probably* be more efficient to sort the edges, and record their indices in nondecreasing order, but that is boring.

---

6. (6 pts) Building Low Cost Bridges

Suppose you are in Canada's Thousand Islands National Park, and in one particular lake there are *n* small islands that park officials want to connect with floating bridges so that people can experience going between islands without a canoe. The cost of constructing a bridge is proportional to its length. Assume the distance between every pair of islands is given to you as a two dimensional matrix (an example of such a table for *k*=8 islands is shown below).

Design an algorithm that, given a table such as shown below, determines which bridges they should build to connect the islands at minimal cost. Write down the pseudocode and analyze the runtime of your algorithm. (You are allowed to build on algorithms we have already studied.)

```
O-Canada(G)
1 min = ∞
2 for v in G.V
3   S = MST-Kruskal(G,v)
4   cost = 0
5   for each edge in S
6     cost += edge.value
7   if min > cost
8   min = cost
9 return min
```

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
A	–	240	210	340	280	200	345	120
B	–	–	265	175	215	180	185	155
C	–	–	–	260	115	350	435	195
D	–	–	–	–	160	330	295	230
E	–	–	–	–	–	360	400	170
F	–	–	–	–	–	–	175	205
G	–	–	–	–	–	–	–	305
H	–	–	–	–	–	–	–	–

---

7. (6 pts) Divide and Conquer MST

Consider the following divide-and-conquer algorithm for computing minimum spanning trees:

Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2) and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Prove that this algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

`The algorithm fails when there is more than one optimal edge across the cut.`