Basic Algorithms, Assignment 10

Due by 8 a.m. Wednesdy, April 14.
Send to Jingshuai: jj2903@nyu.edu.

I cannot live without people. – Pope Francis

1. Suppose we are given the Minimal Spanning Tree T of a graph G. Now we take an edge $\{x, y\}$ of G which is not in T and reduce its weight $w(x, y)$ to a new value w. Suppose the path from x to y in the Minimal Spanning Tree contains an edge whose weight is bigger than w. Prove that the old Minimal Spanning Tree is no longer the Minimal Spanning Tree.

2. **Do NOT submit this problem!** Suppose we ran Kruskal's algorithm on a graph G with n vertices and m edges, no two costs equal. *Assume* that the $n-1$ edges of minimal cost form a tree T.

   (a) Argue that T will be the minimal cost tree.

   (b) How much time will Kruskal's Algorithm take. *Assume* that the edges are *given* to you an array in increasing order of weight. Further, *assume* the Algorithm stops when it finds the MST. Note that the total number m of edges is irrelevant as the algorithm will only look at the first $n-1$ of them.

   (c) We define Dumb Kruskal. It is Kruskal without the SIZE function. For $UNION[u, v]$ we follow u, v down to their roots x, y as with regular Kruskal but now, if $x \neq y$, we simply reset $\pi[y] = x$. We have the same assumptions on G as above. How long could dumb Kruskal take. Describe an example where it takes that long. (You can imagine that when the edge u, v is given an adversary puts them in the worst possible order to slow down your algorithm.)

3. Consider Kruskal's Algorithm for MST on a graph with vertex set $\{1, \ldots, n\}$. Assume that the order of the weights of the edges begins $\{1, 2\}, \{2, 3\}, \{3, 4\}, \ldots, \{n-1, n\}$. (Note: When $SIZE[x] = SIZE[y]$ make the first value the parent of the second. In particular, set $\pi[2] = 1$, not the other way around.)

   (a) Show the pattern as the edges are processed. In particular, let $n = 100$ and stop the program when the edge $\{72, 73\}$ has been processed. Give the values of $SIZE[x]$ and $\pi[x]$ for all vertices x.

(b) Now let n be large and stop the program after $\{n-1, n\}$ has been processed. Assume the ordering of the weights of the edges was *given* to you, so it took zero time. How long, as an asymptotic function of n, would this program take. (Reasons, please!)

4. Here is an alternative approach [1] to Union-Find for Kruskal's Algorithm. You are asked to analyze the time. We have n vertices $1 \cdots n$. Initially each vertex is in its own component. *Assume* the edges have already been ordered by weight. Let E be the number of edges. *Assume* $E = 2V$ (other cases are similar.) Each node is given three attributes:

   (a) $LIST[i]$ will be a linked (important!) list of vertices, including i

   (b) $NAME[i]$ will be the designated vertex of the component containing i.

   (c) $SIZE[i]$ will be, when $NAME[i] = i$, the size of the component containing i. Further, when $NAME[i] = i$, $LIST[i]$ will consist of all the vertices of the component containing i.

   The initialization is easy:

   (a) $LIST[i] = i$ (the linked list with one item)

   (b) $NAME[i] = i$

   (c) $SIZE[i] = 1$

   Now we consider an edge x, y. If $NAME[x] = NAME[y]$ we reject the edge (x, y lie in the same component) and do nothing. The interesting part is when $NAME[x] \neq NAME[y]$ (which occurs $V - 1 = \Theta(V)$ times) so we need to merge. Reset $x \leftarrow NAME[x]$, $y \leftarrow NAME[y]$. (So the new x – same for y – is the designated vertex of the component which the old x lies in. This is analogous to the "sliding down the bannister" step but in only takes one step.) Reorder if necessary so that $SIZE[x] \leq SIZE[y]$. Now

   (a) (Rename step) For each z in $LIST[x]$ reset $NAME[z] = y$ (This will take the time!)

   (b) (size step) Set $SIZE[y] \leftarrow SIZE[y] + SIZE[x]$ (Assume addition takes time $\Theta(1)$.)

[1] From Aho/Hopcroft/Ullman, Data Structures and Algorithms – though original discovery uncertain.

(c) (append step) **Append** $LIST[x]$ to $LIST[y]$

Now for the time analysis questions (answers in Θ-land, please) :

(a) What is the total time for cases when x, y lie in the same component?

(b) What is the total time for the size steps?

(c) What is the total time for the append steps?

(d) Give an upper bound (similar to that in Union-Find) for the number of times a vertex z will be renamed. (Critically, it is the smaller component being renamed!!)

(e) What is the total time (using O) for the rename steps?

(f) Putting this all together – what is the total time as $O(g(V))$ for a nice function g.

If you look for things that are like the things that you have looked for before, then, obviously, they'll connect up. But they'll only connect up in an obvious sort of way, which actually isn't in terms of writing something new, very productive. So you have to take heterogeneous materials in order to get your mind to do something that it hasn't done before.
W. G. Sebald