

## Fundamental Algorithms, Assignment 10 Solutions

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.

**Solution:** We can replace the edge whose weight is bigger than  $w$  with the edge  $\{x, y\}$  resulting in a lower weight spanning tree.

2. Suppose we ran Kruskal's algorithm on a graph  $G$  with  $n$  vertices and  $m$  edges, no two costs equal. Suppose the the  $n - 1$  edges of minimal cost form a tree  $T$ .

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

**Solution:** From Kruskal's Algorithm we will accept all the edges of  $T$ . Then we have a spanning tree so no more edges are accepted.

- (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.

**Solution:** We do  $n$  operations  $UNION[x, y]$ , each takes time  $O(\ln n)$  so the total time is  $O(n \ln n)$ .

- (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.)

**Solution:** As  $UNION[x, y]$  must take time  $O(n)$  (as there are only  $n$  vertices) the whole algorithm will take time  $O(n^2)$ . This can happen. Suppose the edges were, in order,  $\{2, 1\}, \{3, 1\}, \{4, 1\}, \dots, \{n, 1\}$ . For the first edge we make  $\pi[1] = 2$ . The second edge we follow 1 down to root 2 and set  $\pi[2] = 3$ . Now for the third

edge we follow 1 to 2 to root 3 and set  $\pi[3] = 4$ . On the  $i$ -th step we are taking time  $\sim i$  so it is a  $\Theta(n^2)$  running time.  $\square$

- (d) Consider Kruskal's Algorithm for MST on a graph with vertex set  $\{1, \dots, n\}$ . Assume that the order of the weights of the edges begins  $\{1, 2\}, \{2, 3\}, \{3, 4\}, \dots, \{n-1, n\}$ . Assume that when in Kruskal's Algorithm we have a tie  $SIZE[x] = SIZE[y]$  we set the smaller of  $x, y$  to be the parent of the largest.

- i. Show the pattern as the edges are processed. In particular, let  $n = 100$  and stop the program when the edge  $\{1, 73\}$  has been processed. Give the values of  $SIZE[x]$  and  $\pi[x]$  for all vertices  $x$ .

**Solution:** First we set  $\pi[2] = 1$  and  $SIZE[1] = 2$ . Now for  $i = 3, 4, \dots$  when we process  $1, i$  we have  $\pi[i] = i$  and  $\pi[i-1] = 1$ . (In a formal mathematical sense this would be by induction, but its OK just to see the pattern.) So the WHILE loop sends  $i-1$  to 1 with  $SIZE[1] = i-1$  and  $i$  to itself with  $SIZE[i] = 1$  so we set  $\pi[i] = 1$  and reset  $SIZE[1] = i-1+1 = i$ . (That is, the  $SIZE[1]$  goes up by one for each iteration.) With  $n = 100$  after  $\{1, 73\}$  is processed we have  $\pi[i] = 1$  for all  $1 \leq i \leq 73$  and  $SIZE[1] = 73$  and  $SIZE[i] = 1$  for  $2 \leq i \leq 73$ . For the yet untouched  $i$  from 74 to 100 we still have the initial values  $SIZE[i] = 1, \pi[i] = i$ .

- ii. Now let  $n$  be large and stop the program after  $\{1, n\}$  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!)

**Solution:** It would be linear  $\Theta(n)$  time. At each iteration the WHILE loop is applied zero times for 1 and one time for  $i$  so it takes constant time – and we have to run the program through the  $n-1$  edges. **Remark:** This is quite special – in most cases the WHILE loops get long.