Fundamental Algorithms - Spring 2018
Homework 10

Daniel Rivera Ruiz
Department of Computer Science
New York University
`drr342@nyu.edu`

1. Let's say that the edge whose weight is bigger than w is $e_{ab} = \{a, b\}$. The current path to go from x to y is given by $P = S_{xa} \cup e_{ab} \cup S_{by}$, where S_{xa} is the set of edges connecting x and a and S_{by} is the set of edges connecting b and y. Now, if we remove e_{ab} from the MST and replace it with the new edge e_{xy} we can still go from a to b by following the new path $P' = S_{ax} \cup e_{xy} \cup S_{yb}$. The new tree T' generated by doing this has a smaller weight than the original tree and therefore is the new MST of the graph G:

$$w(T') = w(T) - w(e_{ab}) + w(e_{xy}) \quad \& \quad w(e_{ab}) > w(e_{xy}) \Rightarrow w(T') < w(T)$$

2. (a) Let's consider the execution of Kruskal's algorithm on the graph G, when we get to the point of comparing x_f to y_f after `sliding-down-the-banister` for the original vertices x_i and y_i. At this point, the only conditions under which $x_f = y_f = p$ are 1) if $x_i = y_i$, or 2) if there is an edge $e = \{x_i, y_i\}$ connecting the original vertices. The first condition will never occur because we are dealing with a graph with no loops. The second condition will not occur in the first $n-1$ iterations of the algorithm because it would mean that there is a cycle in G connecting p, x_i and y_i, but according to the assumptions of the problem, *the* $n-1$ *edges of minimal cost form a tree* (which by definition can have no cycles). Finally, after $n-1$ iterations of the algorithm we must have added all the edges we encountered, since every time we had $x \neq y$. At this point the algorithm can terminate because by definition the MST (and any tree for that matter) can only have $n-1$ edges when there are n vertices.

   (b) The original time for Kruskal's algorithm is $O(E \log_2 V)$, where E is the number of edges and V the number of vertices. In this case, however, we can run the algorithm only for the first $n-1$ edges regardless of the total amount m (see the previous answer). Therefore the time complexity will be:

$$T = O(E \log_2 V) = O((n-1) \log_2 n) = O(n \log_2 n)$$

   (c) If we consider the dumb version of Kruskal's algorithm where there is no size function, `sliding-down-the-banister` can take as long as $O(V) = O(n)$, and therefore the overall complexity of the algorithm can be as bad as $O(n^2)$. This can be explained as follows:

   - `sliding-down-the-banister` executes as long as $v \neq \pi(v)$.
   - In the original algorithm we have the property (thanks to the size function) that $\pi(x) = y \Rightarrow size(y) \geq 2size(x)$. This means that `sliding-down-the-banister` can take (at most) $\log_2(V)$ steps.
   - The dumb algorithm, without the size function, has no upper bound to the steps `sliding-down-the-banister` can take other than the trivial $n-1$, which is the number of vertices in the MST. This means that in the worst case scenario `sliding-down-the-banister` can take time $O(n)$.

   To exemplify the statement above, let us consider a graph Γ with vertices $\alpha_1, \alpha_2, \ldots, \alpha_n$ where the $n-1$ minimal weight edges are of the form $e_i = \{\alpha_1, \alpha_i\}$ for $2 \leq i \leq n$ and $w(e_i) < w(e_j)$ if $i < j$. Under this conditions, dumb Kruskal's algorithm will traverse the edges in ascending order $e_2, e_3 \ldots, e_n$. Additionally, to consider the worst case scenario we assume that the parent function at the i^{th} iteration is given by $\pi(\alpha_i) = \alpha_{i+1}$ for $1 \leq i \leq n-1$. With all of the above, when the algorithm reaches the i^{th} edge $e_{i+1} = \{\alpha_1, \alpha_{i+1}\}$

`sliding-down-the-banister` will take one step for α_{i+1} but i steps for α_1. Summing over all values of i, the complexity of the algorithm is given by $\sum_1^{n-1} i$, which is in the order of $O(n^2)$ as expected.

3. (a) As the edges are processed, at the i^{th} iteration we will set $\pi(i+1) = 1$ and $size(1) = i+1$. This follows from the fact that each edge in the graph is of the form $\{i, i+1\}$: since the vertex $i+1$ is appearing for the first time, it will get 1 as its parent, which is the final node after `sliding-down-the-banister` from i. As a result of this, the value of $size(1)$ will increase from i to $i+1$.
   In the particular case where $n = 100$ and we stop the execution after processing the edge $\{72, 73\}$, the values of π and $size$ are defined as follows:

$$\pi(i) = \begin{cases} 1 & : \quad 1 \leq i \leq 73 \\ i & : \quad 74 \leq i \leq 100 \end{cases}$$

$$size(i) = \begin{cases} 73 & : \quad i = 1 \\ 1 & : \quad i \neq 1 \end{cases}$$

   (b) For a large value of n and if the edges are already ordered by increasing weight, the execution of the program will take time $O(n)$. The original value for the algorithm's time complexity is $O(n \log_2 n)$, which considers the worst case scenario where `sliding-down-the-banister` takes time $O(\log_2 n)$. In this particular case, however, we know that `sliding-down-the-banister` will always take constant time, since all nodes have 1 as their parent. Executing `sliding-down-the-banister` $n-1$ times results in the overall complexity $O(n)$.

4. Since the graph is complete, all the vertices are in the adjacency list of the root $r = 1$ and therefore will be added to the priority queue Q during the initialization with $\pi(j) = 1$ and $k(j) = (j-1)^2$ for $2 \leq j \leq n$.
   At the first iteration of the algorithm, the minimal element in Q is 2 because $k(j)$ is a strictly increasing function. After removing 2 from Q and adding it to S, we have to update π and k for all the values in Q because they are all in $adj(2)$ and they are all closer to 2 than they are to 1. Therefore, we have to make $\pi(j) = 2$ and $k(j) = (j-2)^2$ for $3 \leq j \leq n$.
   Following this intuition, at the i^{th} iteration of the algorithm the minimal element in Q will be i and after extracting it Q must be updated as follows: $\pi(j) = i$ and $k(j) = (j-i)^2$ for $i+1 \leq j \leq n$.
   (a) Under these conditions for a graph with n vertices where $n = 100$, the first 73 elements inserted in the MST have to be $\{1, 2, 3, \ldots, 73\}$.
   (b) After inserting the 73^{rd} element and updating π and k for the remaining elements in Q we will have $\pi(84) = 73$ and $k(84) = (84-73)^2 = 11^2 = 121$.