Stefanie Shidoosh

CS 180: HW #4

1. A student suggested in class that if the array *f* is *sorted* in increasing order, the optimal binary tree can be constructed in $O(n)$ time. Give an $O(n)$ algorithm assume *f* is sorted and prove your algorithm works.

If *f* is sorted, a heap can be built in linear time (with respect to the optimal tree demonstrated in class):
Traverse the two lowest levels of the tree
Recursively order (swap files as needed) all subtrees on those levels as binary heaps
Do so for all levels of the tree until root of the tree is reached

This algorithm requires at most *n* swaps, where *n* is the number of files in the tree. Therefore, the time complexity is *O(n)*.

Stefanie Shidoosh

2. Given a list of n items such that the order of any two items can be determined by one comparison. Design an $O(n)$ algorithm to construct a heap contains all n items.

If the order of any two items can be determined by one comparison, then the maximum number of swaps is a maximum of n swaps for n items.

Traverse the list
Insert the element to the terminal level of the tree
    At first iteration, this implies inserting at root
Compare the current element with the next element
    If not in the correct order, swap
    Otherwise, iterate to next item in the list

Because the swap will only occur at the moment the item is inserted, as opposed to the case in which swaps could be needed for each level of the tree at a given iteration, then the time complexity will be $O(n)$.

Stefanie Shidoosh

3. Given a directed weighted graph $G = (V, E)$ with every edge e has weight $w(e) > 1$. Assume G contains a node s that can reach all other nodes.

a. Prove that there exists a directed tree T in G with root s as a subgraph in G, with the property that the path from s to any other node v in T is the shortest path from s to v in G.

Proof by contradiction:

Assume that there does not exist a directed tree T in G with root s as a subgraph in G, with the property that the path from s to any other node v in T is the shortest path from s to v in G.
Such a subgraph, G_1, is composed of a subset of $V(G)$ and a subset of $E(G)$, E1, that correspond to v in $V(G)$.
Each e in $E_1(G_1)$ has the same weight as the corresponding e in E(G).
If there exists a shortest path from s to any v in G, then a subset of these edges, E_1, to each of the vertices, V_1, can compose a subgraph $G_1 = (V_1, E_1)$. Thus, G_1 does exist and can be represented as a tree T, with the property that the path from s to any other node v in T is the shortest path from s to v in G.

b. Argue that further squaring will not change the shortest path tree. Describe this characterization of the way of comparing the length of paths.

The initial squaring may change the shortest path as a result of the more "weighty" path dominates the lesser "weighty" path, as in the example provided with 7,2 and 5,5, s.t. sq(7,2) > sq(5,5). However, further squaring will not change the result of the shortest path, as the more "weighty" path as a result of squaring with always be more "weighty" forever more. That is,

If $a > b$ and $x < y$, with respect to $(a, x), (b, y)$: As $k + 2 \rightarrow \infty, (a^{2k} > (b^{2k})$.

c. An MST by definition retains the property of uniqueness, and our subgraph T is the minimum cost subgraph. Therefore, if T is the MST and the minimum cost subgraph, then it is the MST if directions are ignored (as it then would just be a usual weighted graph)

d. Use the way we compared directed paths before to extend it to compare between two spanning tree which one is "better." Argue that the MST defined as the tree that has the minimum sum of weights of its edges, is the same tree that is smaller than all other trees if we just determine which is smaller among 2 trees by comparison rather than addition.

To determine which spanning tree is "better" means determine which is the MST. This follows that for every weighted graph there is only one MST. For our purposes, comparing which tree is smaller maintains the property of that tree being the minimum cost subgraph.

Stefanie Shidoosh

4a. Given a list of $O(n)$ operations of Union and Query. Design an algorithm that "processes" the sequence of commands in $O(n \log n)$ time. To process the sequence means to answer all the queries in the sequence. (Hint: keep a set as a rooted tree with depth that is $O(\log n)$. The Crux is how to union two trees and keep this property.)

Represent the sets as rooted trees, with depth as O(logn)
    Order the set, traversing and inserting elements into the tree, compatible with DFS
Sort the edges by weight
For each e in E(G)
    Check if ex and ey are in the same component, using the query operation
    If not, union the sets

This algorithm is O(logn) as the algorithm is bound by it processing each tree with respect to its depth, which is O(logn).

b. Given an undirected weighted graph $G = (V, E)$. Show an implementation of Kruskal's Minimum Spanning Tree(MST) algorithm by using the Union and Query commands. Your implementation should have $O(|E|\log|V|)$ time complexity. (Hint: In Kruskal's algorithm, it builds the MST by processing every edge in G and asks whether two endpoints of an edge belong to two disjoint trees. If so, the algorithm adds the edge making two disjoint trees into a single tree).

For each v in V(G)
    Make a set for v, processing every edge in G
Sort the edges weight
For each e in E(G)
    Query whether two endpoints of an edge belong to two disjoint trees (sets)
    If so, Union the sets. Otherwise, continue to the next e in E(G).
When finished, the resulting union is an MST.

Sorting takes (ElogE), the query operation is bound by O(V) for each time a vertex is union-ed to the intermediate spanning tree. Thus in total, the time complexity for this algorithm is $O(|E|\log|V|)$.