HW5

Seyed Armin Vakil Ghahani
PSU ID: 914017982
CSE-565 Fall 2018
Collaboration with: Sara Mahdizadeh Shahri, Soheil Khadirsharbiyani, Muhammad Talha Imran

October 3, 2018

Problem 1. Huffman code

Solution

- (a) As it shown in Figure 1 the code for the alphabet will be as following: $a = 11111, b = 11110, c = 1110, d = 101, e = 100, f = 110, g = 01, h = 00$

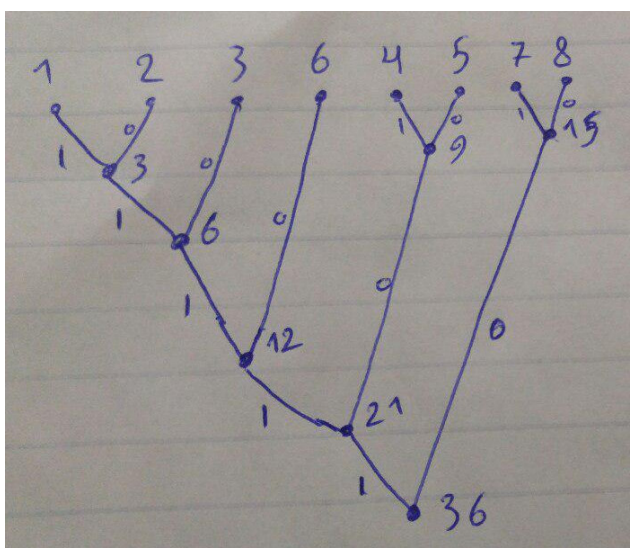

Figure 1: Huffman algorithm output

- (b) Suppose the alphabet $\{a_1, a_2, a_3, a_4, a_5\}$ with $\{5, 6, 7, 8, 15\}$ weights. The node generated from $/a_1, a_2/$ is going to merge with two possible nodes $\{a_3, a_4\}$ and a_5. This turns into two possible huffman code for this alphabet.

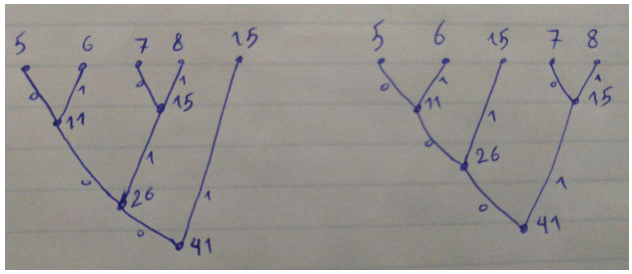

Figure 2: Two different Huffman codes

Problem 2. A greedy algorithm for El Goog

Solution This problem has a greedy solution that we can sort the jobs by their f_i, and run jobs based on their decreasing order of f_i.
Suppose that this solution S is not an optimal solution and the optimal solution is S^*. Suppose two consecutive jobs J_i and J_j in S^* that $f_i \leq f_j$. We are going to swap these two consecutive jobs to create a new solution S' and prove that this will not increase the completion time. The starting time of the jobs before and after these two jobs will not changed because they are going to start their execution at $p_i + p_j$ in both cases. The starting time of running the second part of second job in S^* and S' is $p_i + p_j$. Hence, if the job that is going to execute later has higher finishing time, the completion time will be higher. In S^* the second job that is going to execute has higher finishing time, and consequently, if we swap these two consecutive jobs this parameter will not increase because $f_i \leq f_j$.
By doing this procedure until there is not two consecutive jobs that the finishing time of the first job is lower than the second one, we have jobs in the decreasing order of their finishing time. Hence, the greedy solution is optimal. I want to add that in each swap procedure, the number of inversions in the array of finishing times will increase and swapping procedure will end.

The time complexity of this algorithm is $\theta(nlogn)$ because we just want to sort jobs based on their finishing times.

Problem 3. MST for near-tree

Solution By the cycle property, we know that in each cycle, the heaviest edge should not appear in any MST. As a result, if we found a cycle in the graph and remove the heaviest edge on it for nine times, the remained graph has $n-1$ edge, and it is connected yet, and consequently, it will be a tree. To find a cycle in this graph we can run a DFS algorithm, and whenever we see a back edge in the DFS algorithm, it will result to a cycle, and we can find the heaviest edge on that easily.

The time complexity of this algorithm is the number of DFS algorithms we run multiply to the time complexity of running the DFS algorithm and finding the heaviest edge. The time complexity of DFS algorithm is $O(|V| + |E|)$ and the time complexity of finding the heaviest edge is $O(|V|)$ because we should go through the edges in the DFS tree and find the heaviest edge. Moreover, the number of times we should run the DFS algorithm is nine times, and after that, the graph will not have any cycle.

Finally, the time complexity of this algorithm is $O(|V|+|E|)$ and because $|V|, |E| \in O(n)$, the time complexity is $O(n)$.