Name: Harshit Jain
Access ID: hmj5262
Recitation: 8

Problem 0 **Points:**

Acknowledgements

(a) I worked with Yug Jarodiya.

(b) I did not consult with anyone in my group members.

(c) I did not consult any non-class materials.

Problem 1

Points:

Given: A Set $C = \{0, 1, ..., n-1\}$ of characters

To show: Represent any optimal prefix-free code on C using only $(2n-1+n\lceil \log n \rceil)$ bits

Proof:

$\Rightarrow$ An optimal prefix-free code on C has an associated full binary tree with n leaves and $(n-1)$ internal vertices. This tree can be coded by a sequence of $n+(n-1) = (2n-1)$ bits.

$\Rightarrow$ Height of full binary tree = $\lceil \log n \rceil$

$\Rightarrow$ To associate the n members of $C\{0, 1, ..., n-1\}$ with the n leaves of the tree, which can be done by listing them in the order in which pre-order traversal encounters them, $\lceil \log n \rceil$ is enough bits to represent each of the n members of C. This is because, each member in C (leaf of the tree) has the maximum possible depth of $\lceil \log n \rceil$ in a full binary tree. For n leaves, it require $n\lceil \log n \rceil$ bits to represent.

$\Rightarrow$ Total bits needed to represent optimal prefix-free code on $C =$

$$n \textit{ (for leaves)} + (n-1) \textit{ (for internal nodes)} + n\lceil \log n \rceil \Rightarrow \underline{2n-1+n\lceil \log n \rceil \text{ bits}}$$

Problem 2

Points:

The Idea: Put more frequent symbols at smaller depth

Greedy approach: Continually merge least frequent symbols/nodes until you have a full ternary tree encoding all symbols. In this case, take the three lowest frequency symbols, and merge them into one root R. Then from the set of the remaining nodes and R, take the three lowest nodes and merge them continuously until a full ternary tree is made.

Optimal Proof: Suppose we have a tree T with three lowest frequent symbols not as deep as possible. Then at least one has a smaller depth. Switch it with one of the deepest nodes that is more frequent.This improves the encoding length. Thus T is not optimal. Since T is not optimal, then we know that the three lowest frequencies must be at the largest depth.

Algorithm 1: Ternary Huffman Algorithm

Input : $f = f[1], \cdots, f[n]$; Γ has n symbols
Output: T

$T =$ empty tree;
$H =$ priority queue ordered by f;
for $i = 1$ *to* n **do**
  insert(H, i);
end for
for $k = (n+1)$ *to* $(2n - \lceil \frac{n}{2} \rceil)$ **do**
  $i =$ extract min(H);
  $j =$ extract min(H);
  $k =$ extract min(H);
  Create a node z in T with children $i, j,$ and k ;
  $f[z]$= $f[i]$ + $f[j]$ + $f[k]$;
  insert(H, z);
end for
return T

Problem 3

Points:

Proof by contradiction: Suppose greedy is not optimal solution.

Consider the solution given by the greedy algorithm as a sequence of packages, here represented by indexes: $1,2,3,...n$. Each package i has a weight, w_i, and an assigned truck t_i which is a non-decreasing sequence (as the k^{th} truck is sent out before anything is placed on the $(k+1)^{th}$ truck).

Note that: If $t_n = m$, that means our solution takes m trucks to send out the n packages.

If the greedy solution is non-optimal, then there exists another solution t'_i, with the same constraints, such that $t'_n = m' < t_n = m$. Consider the optimal solution that matches the greedy solution as long as possible, so $\forall i < k, t_i = t'_i$, and $t_k \neq t'_k$ (that means, look for the largest i such that: $t_i = t'_i$, and the number of trucks optimal solution takes to send out k packages is not equal to what greedy algorithm is going to take).

It leads to 2 cases:

Case 1: If $t'_k < t_k$, the greedy solution used more trucks than the optimal solution. As previously mentioned, $\forall\ i < k, t_i = t'_i$, we know that until $i = (k-1)$, the optimal solution and greedy solution were carrying the same packages, however, the greedy solution used another truck for the k^{th} package. Therefore, we have to add another truck in optimal solution too which leads to the contradiction as we get on the optimal solution with a larger t'_k.

Case 2: If $t'_k > t_k$, the optimal solution used more trucks than the greedy solution. We get a contradiction here because greedy solution carried k packages in less number of trucks than the optimal solution which is a contradiction of optimality that greedy solution used less number of trucks than the optimal solution.

(Note that, if $t'_k = t_k$, then greedy solution would be already considered as an optimal solution.)

Therefore, the greedy algorithm is the optimal solution that actually minimizes the number of trucks that are needed.