Submission for Problem Set 5 (due Monday, November 5, 11:59 PM)

Name: Deepanshu Parihar

Problem 1. (5 points) Three-character Huffman codes

Consider the problem of Huffman codes where we use three characters from $\{0, 1, 2\}$ in our code, as opposed to the bits 0 and 1. Modify the Huffman encoding algorithm to determine a minimum-length compression of any sequence of characters from an alphabet A of size n, with the ith letter of the alphabet having frequency $f[i]$. Your algorithm should encode each character with a variable-length codeword over the values $\{0, 1, 2\}$ such that no codeword is a prefix of another codeword and so as to obtain the maximum possible compression. Prove that your algorithm is correct. Analyze the worst-case running time of your algorithm.

Answer: There are two possible cases , one when n is odd and one when n is even. In case of even we combine the last two frequencies as one to get the total to odd.
A is a set of n characters. So,

```
if(n is odd)
Initialize Min-priority Queue Q = A
n=size of (A)
for i=1 to n-1
   allocate a new node z
   z.left = p= EXTRACT-MIN(Q)
   z.right = q =EXTRACT-MIN(Q)
   z.center= r =EXTRACT-MIN(Q)
   z.freq = p.freq + q.freq +r.freq
   INSERT(Q,z)
return EXTRACT-MIN(Q)


else if(n is even)
Initialize Min-priority Queue Q = A
n=size of (A)
allocate a new node z
z.left = p= EXTRACT-MIN(Q)
z.right = q =EXTRACT-MIN(Q)
z.freq = p.freq + q.freq
INSERT(Q,z)          (now the size is odd, so we can do the same procedure again)
for i=1 to n-2       (as now the length of the queue has been reduced by one)
   Using Min-priority queue Q
   allocate a new node m
   m.left = p= EXTRACT-MIN(Q)
```

```
    m.right = q =EXTRACT-MIN(Q)
    m.center= r =EXTRACT-MIN(Q)
    m.freq = p.freq + q.freq +r.freq
    INSERT(Q,m)
return EXTRACT-MIN(Q)
```

Proof-
Extending the proof that was used for 2 bit encoding in class.

1. Suppose T' is an optimal tree.
2. x, y, z are least frequency letters.
3. If x, y, z are at maximum depth and siblings in T* , then algorithm is done.
4.Otherwise
5.T' has some 3 leaves p, q, r at maximum depth which are siblings.
6.Swap x with p.
7.Change in cost is dT'(p) f(x) + dT'(x) f(p) dT'(a) f(p) dT'(x) f(x) ≤ 0
8.This implies that the cost of T' can be reduced. 9.Therefore, T' is not an optimal tree.
10.This is a contradiction to our assumption.
Hence Proved

Running Time- All the extract and insert operations for a priority queue take O(log n) time individually. We do it here for n iterations. So the net complexity is O(nlog(n))

Problem 2. (5 points) Matching Widgets and Gadgets

You are given a set W of n widgets and a set G of n gadgets. Each widget w has a weight $W(v)$ and each gadget g has a weight $W(g)$. You would like to match each widget w in W to a unique gadget g in G so as to minimize the sum of the absolute values of the weight differences of the matched pairs. That is, you would like to find a perfect matching M between W and G that minimizes

$$\sum_{(w,g)\in M} |W(w) - W(g)|.$$

Design an efficient greedy algorithm to solve the given problem. Prove that your algorithm is correct. Analyze the worst-case running time of your algorithm.

Answer:
Algorithm-
Matching()
1. Sort set W of widgets in increasing order.
2. Sort set G of gadgets in increasing order.
3.n= number of widgets
4.Initialize a list L
5.for(i=0 to n)
6.   Add(Widget(i),gadget(i)) to L

7.Return L

Running Time-
Since sorting is the dominant function here the running time is O(nlogn).

Proof-
P is an optimal solution containing the pair(w1,g1) and (w2,g2).
1. Let's assume that the solution P' is an optimal solution containing pair(w1,g2) and (w2.g1)
2.If matching is same in both P and P' then we are done.
3.Otherwise
4.Swap the gadgets to get the pairs (w1,g1) and (w2,g2).
5.Now , calculate cost (—W(w1)W(g1)— - —W(w2)W(g2)—) - (—W(w1)W(g2)— - —W(w2)W(g1)—)
6. For all the possible cases of combination possible i.e.
(a) g1,g2 is greater than w1,w2.
(b) g1,g2 is less than w1,w2.
(c) g1,g2 lies between w1,w2.
(d) g1 is greater than w1, and g2 is less than w2.
(e) g1 is less than w1, and g2 is greater than w2.
7. For all the above cases the cost is less than or equal to 0.
8. This is a contradiction to our assumption that P' is optimal.
9. Hence proved by contradiction.

Problem 3. (3 + 2 = 5 points) Uniqueness of MSTs when all weights are distinct

(a) Suppose T_1 and T_2 are distinct minimum spanning trees for graph G. Let (u, v) be the lightest edge (smallest weight edge) among all edges that are in T_1 and but not in T_2. Let (x, y) be any edge that is in T_2 and not in T_1. Show that $w(x, y) \geq w(u, v)$.

Answer:
Considering all the uncommon edges in MST T1, T2.
1.Assume that there exists an edge e' in T2 such that the w(e') is less than the minimum edge in T1 i.e. w(u,v).
2.If this were true , then both T1 and T2 are MST of same graph G and this edge would be part of T1 as well. So adding it to T1 , we form a cycle.
3.So we remove another edge whose weight is greater than e' from T1 to reduce the weight and remove the cycle to get a spanning tree.
4.But now both T1 and T2 have a common edge e' which is not possible because if it were true then it would have been removed for being a part of common edges and a different MST of lower value would have existed.
5.Hence no such edge e' existed to begin with. So no such edge exists.
6.Proved by contradiction.

(b) Using part (a), prove that if the weights on the edges of a connected, undirected graph are distinct, then there is a unique minimum spanning tree.

Answer:
1.Suppose more than one MST exists for a connected, undirected graph whose edges are distinct.
2.So considering two distinct MST T1 and T2 of same weight.
3.Then based on (a) min weight w(u,v) in T1 $\leq$ w(x,y) in T2.
4.Also,min weight w(x,y) in T2 $\leq$ w(u,v) in T2.
5.As all the edges are distinct and from the above two steps the only possibility is that both the edges are same.
6.Now if we consider the next minimum weight by removing this as common.We again get the same results.
7. Now iteratively keep doing the above step.
8.In the end we will be left with nothing that is not common for T1 and T2.
9.This implies that both the MST T1 and T2 are not distinct but the same.
10.This is a contradiction to our assumption.

Hence Proved by contradiction.

Problem 4. (5 points) Leaf-Constrained Spanning Tree

Design an algorithm, which takes as input a connected undirected graph $G = (V, E)$, a weight function $w : E \to Z^+$, and a subset U of V, and returns minimum-weight spanning tree of G satisfying the property that every vertex in U is a leaf in T. If no such spanning tree exists, then your algorithm must indicate so. Analyze the worst-case running time of your algorithm.

(*Note:* The desired spanning tree may not be a minimum spanning tree of G. The spanning tree your algorithm returns must satisfy the desired property and have the minumum weight among all spanning trees satisfying the desired property.)

Answer:
So here I form a MST by using vertices that are a part of set V - U as subset U of vertices V are to be used as node and hence wont be connectors in MST.
If I dont get an MST then no spanning tree can be formed.
Else for vertices in U I find edges of minimum weight such that the edge links a vertex in U to a vertex of MST and if such edges cant be formed for all the vertices in U then also no spanning tree is formed. Else I return a spanning tree with connected leaves.

Algorithm-

1.Form a new graph G' formed by removing vertices that are present in set U and all its related edges.
2.Using Kruskal to find the MST.
3.If no MST exists , then no Spanning tree of desired property exists.
4.if(MST exists)
5.for(every vertex in U)

6. Find and connect minimum weighted edge e such that e connects a vertex in U to a vertex in MST.
7. if (no such edge exists)
8. no Spanning tree of desired property exists
9.Return MST.

Running Time-
Both Kruskal and find steps used in steps 2 and 6 respectively use O(mlog(n)) time i.e. O(vlog(e)). So worst case time is O(vlog(e)).