Renat Norderhaug

CS 477

HW #7

12/1/19

1. algorithm:

Step1: declare a Node class which will store vertices of the binary tree. Which has, edge, name, left and right properties. edge will store distance between parent and current node, name will store node name or label. left and right will store left and right child respectively.

Step2: make a max height method which will returrn 0 if a node is null, otherwise it will return the value of the edge + the edge of the left node and the right node.

Step3: make a increase\_edge method which will take in a node and “maxh” value, make the value of the node edge equal to the edge + maxh - max\_height of the node. If the the node to left is not null then recursively call the increase\_edge function on the left node, and the maxh value will subtracted by the left node. Repeat the same for the right node.

Step4: call the increase\_edge method with the root and the value of the max height of the root.

This will make sure the edges have the same value so there is no skew. Below is some pseudocode.

* max\_height(node):
  + if node is NULL return 0
  + else return node.edge + max(max\_height(node.left), max\_height(node.right))
* increase\_edge(node, maxh):
  + node.edge = node.edge + maxh – max\_height(node)
  + if node.left is not NULL:
    - increase\_edge(node.left, maxh – node.left.edge)
  + if node.right is not NULL:
    - increase\_edge(node.right, maxh – node.right.edge)
* main():
  + increase\_edge(root, max\_height(root))

2. Jobs are sorted in decreasing order of their PC time (*Pi , Fi )* and then the jobs are scheduled. The time complexity for this algorithm is O (n log n).

Proof: Suppose there are two scheduling strategies. One is named *S1* and the other is named *S2*. *S1* is the schedule created by our algorithm and *S2*is the schedule produced by the other algorithm. *T(S1)* and *T(S2)* are the time taken by both of these algorithms, respectively.

Must prove that *T(S1) ≤ T(S2)*. Assume that at time T, S1 schedules job *i* and S2 job *j*. Such that *fi* ≤ *fj .*

The time when both jobs will be completed is: Max{ *T + Pi + fi* , *T + Pi + Pj + fj*} = T + *Pi + Pj + fj*

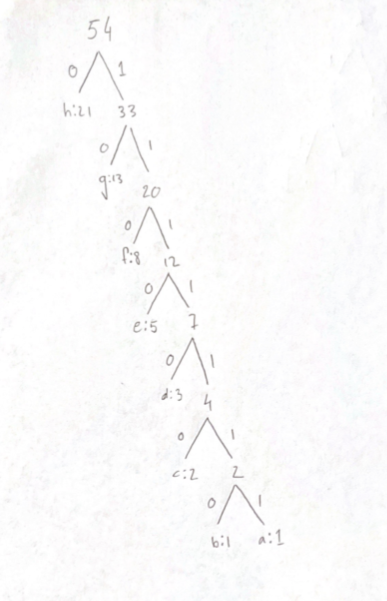
If we were to swap job *i* and job *j* by *S2* schedule then the job will be finished at: Max{*T + Pj + fj* , *T + Pj + Pj + fj*}

It can be concluded that: *T + Pj + fj* ≥ *T + Pi + fj*  AND: *T + Pj + Pj + fj* ≥ *T + Pi + Pj + fj*

Therefore, *T(S1) ≤ T(S2)*. And if we apply the bubble sort in *S2*, then we get *S1*. So the time taken by the strategy is *T(S1) ≤ T(S2)*. The best algorithm for this scheduler is *S1*.

3. [Extra Credit]: There are 8 letters in the set of frequencies and 7 merge steps are required to build the Huffman tree.

You first mergethe characters a,b and c, and after completing all the merge steps the resulting Huffman tree is:



The codeword for each letter is the sequence of edges (labeled 0 or 1) from the root of the tree to the character. These are the following Huffman codes for the 8 characters provided:

h: 0

g: 10

f: 110

e: 1110

d: 11110

c: 111110

b: 1111110 a: 1111111