HOMEWORK 7

CS677 - Algorithm Analysis

Due Date : December 03, 2019

Dzung Bui

Exercise 1 - VLSI - timing circuit

Considering the following binary tree. Let l' and l" denote the maximum root-to-leaf distance over all leaves that are descendants of v', and v" respectively. An algorithm that increase the length of some edges such that the resulting tree without skew should work as following:

- If *l' > l"*, d'-d" will be added to the length of the v-to-v" edge and add nothing to the length of v-to-v' edge.
- If *l' < l"*, d"-d' will be added to the length of the v-to-v' edge and add nothing to the length of v-to-v" edge.
- if *l' = l"*, nothing will be added to the length of v-to-v' and v-to-v".

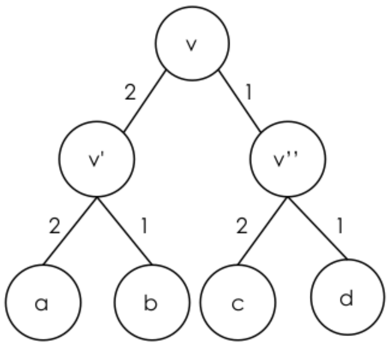


Figure 1: VLSI tree

The algorithm is applied to the subtrees rooted at each of v' and v".

Let T is a complete, balanced binary tree in the problem. There are two facts about the optimal solution:

1. Let *n* be an internal node in T, and let *e'* and *e"* be the two edges directly below *n*. If a solution adds non-zero length to both *e'* and e", then it is not optimal.
2. Let *n* be an internal node in T that is neither the root nor a leaf. If a solution increases the length of every path from *n* to a leaf below *n*, then the solution is not optimal.

Proof of 1:
Suppose that d' > 0 and d"> 0 are added distances to the lengths $l_e^{'}$ and $l_e^{''}$ of edges $e^{'}$ and $e^{''}$, respectively. Let d = min(d', d"), then solution which adds d'-d and d"-d to the lengths of these edges must also have zero skew and use less total length.
Proof of 2:
Suppose that $x_1, x_2, ..., x_k$ are leaves below w. Consider edges e in the subtree below w with the following property: the solution increases the length of e, and it does not increase the length of any edge on the path from w to e. Let F be the set of all such edges; we observe two thing about F. First, for each leaf x_i, the first edge on the $w - x_i$ path whose length has been increased must belong to F; thus there is exactly one edge from F on very $w - x_i$ path. Second, |F| ≥ 2, since a path in the left subtree below w shares no edges with a path form other side of subtree below w, and yet each contains an edge of F.

Let e_w be the edge entering w from its parent (w is not the root). Let d be the minimum amount of length added to any of the edges in F. If we subtract d from the length added to each edge in F, and add d to the edge above w, the length of all root-to-leaf paths remains the same, and so the tree remains zero-skew. But have have subtracted |F|d ≥ 2d from the total length of the tree, and added only d, so we get a zero-skew tree with less total length.

Exercise 2 - Schedule for SuperComputer
The suggested greedy schedule - G:

Run jobs in the order of decreasing finishing time f_i.

The schedule need to be proved as a optimal schedule. We will show that for any given schedule $S \neq G$, we can repeatedly swap adjacent jobs to convert S into G without increasing the completion time.

Consider a schedule S, which is not in order of G. It means it must contain at least a pair of tasks J_i and J_j; J_j runs directly after J_i on the supercomputer, and $f_i < f_j$. The running time R_T with schedule S as in figure 2, and the computation time for Task J_1 and J_j completed at t_4.

$$t_4 = p_j + p_i + f_j \tag{1}$$

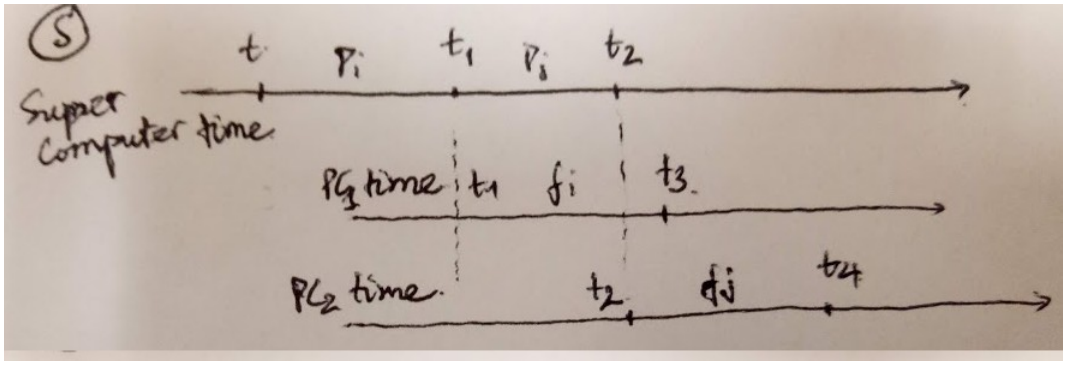


Figure 2: Complete timed of Task J_i and J_j on schedule S.

Let S' be the schedule S where we swap only the order of J_i and J_j. The computation time changes.

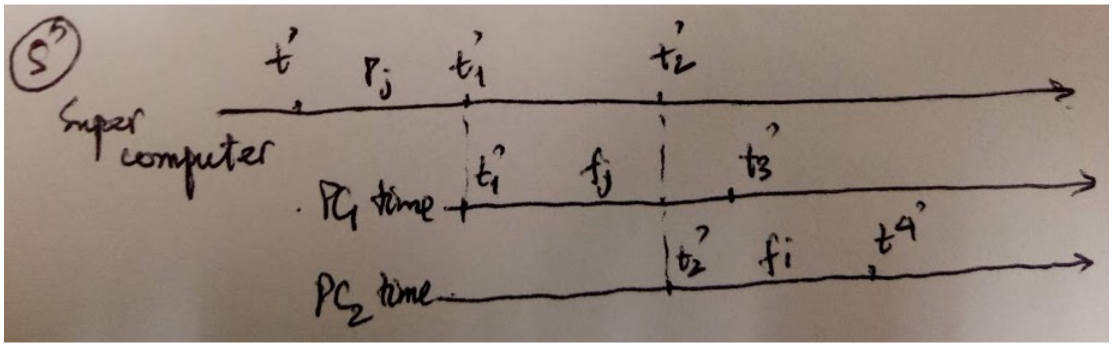


Figure 3: Completed time of Task J_i and J_j on schedule S'.

The running time with schedule S' as in figure 3, and the computation time for Task J_1 and J_j finishes at t'_4:

$$t'_4 = p_j + p_i + f_i \tag{2}$$

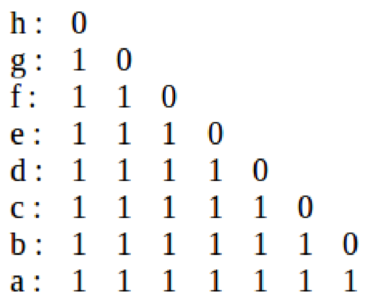
Assuming $t = t'$, and $f_j > f_i$, thus it is obvious that the computation time of tasks J_i and J_j on schedule S is smaller than on other schedule.

If there are multiple task pairs similar to the pair (J_i, J_j), we can apply the same procedure here - swapping the order of processing to make a new schedule whose completion time is not changed. Finally, we go to the schedule G, which is stated before hand. It means that the schedule G, which run the longer finishing time task first, is the optimal schedule.

Exercise 16.1-2 *Greedy Algorithm with selection of the last activity to start that is compatible with all previously selected activities*
Proof: Given a set $S = \{a_1, a_2, ..., a_n\}$ of activities, where $a_i = [s_i, f_i)$. Let create a set $S' = \{a'_1, a'_2, ..., a'_n\}$ where $s'_i = [f_i, s_i)$, that is, a'_i is a_i in reverse.

If a subset $\{a_{i_1}, a_{i_2}, ..., a_{i_k}\} \subseteq S$ is mutually compatible, then it is clearly that the corresponding subset $\{a'_{i_1}, a'_{i_2}, ..., a'_{i_k}\} \subseteq S'$ is also mutually compatible. Therefore, an optimal solution for S maps directly to an optimal solution for S' and vice versa. In another saying, if an greedy algorithm run on S, and provides an optimal solution, it is also able to run on S', and provide the same optimal solution. Thus, the solution on S' is also optimal.

Exercise 16.3-3 *Optimal Huffman Code*
Proof:
a) There are 8-letters int he alphabet, thus there initial queues size in n = 8, and 7 merge steps are required to build the tree. The final tree represents the optimal prefix code. The codeword for a letter is the sequence of the edge labels on the path from the root to the letter. Thus, the optimal Huffman code is:



h :	0
g :	1 0
f :	1 1 0
e :	1 1 1 0
d :	1 1 1 1 0
c :	1 1 1 1 1 0
b :	1 1 1 1 1 1 0
a :	1 1 1 1 1 1 1