CS-206: Assignments 3

Soumen Pradhan – 1912176

10 . 05 . 2021

Assignment 3

16.1-3 Counter-examples for

Least Duration From activities

$$\{(0,6),(4,8),(7,14)\}$$

$(4,8)$ will be selected, which will eliminate the others resulting in an non-optimal solution.

Fewest Overlap From activities

$$\{(-1,1),(2,5),(0,3),(4,6),(6,9),(8,11),(10,12)\}$$

$(4,6)$ will be selected first due to 0 conflicts. But, the optimal solution

$$\{(-1,1),(2,5),(6,9),(10,12)\}$$

does not contain the activity, hence the optimal solution cannot be derived from the first choice.

Earliest Start From activities

$$\{(2,14),(3,5),(5,6),(8,11)\}$$

the earliest choice $(2,14)$ will overlap everythiong else, while the optimal solution will contain the other 2 activities.

16.3-2 An optimal prefix code corresponds to least cost of the Huffman Tree. Note, a full tree maens all nodes have either 2 children or none.
Let T be a binary tree with an optimal prefix code, and suppose T is not full. Let node p of T have a single child q. Let T' be the tree obtained by replacing p with q (a full tree). Let, m be a leaf node which is a descendent of q. Thus,

$$cost(T') \leq \sum_{c \in C-\{m\}} c.freq \cdot d_T(c) + m.freq \cdot (d_T(m) - 1) < \sum_{c \in C} c.freq \cdot d_T(c) = cost(T)$$

where, C is the set of characters in the encoded Huffman Tree.
This contradicts the fact that, T is optimal (hence, the least cost). $\therefore$ only full binary tree can generate optimal prefix code.

16.3-5 Suppose, we have 2 codes, w_1 and w_2 such that $w_1.freq > w_2.freq$, and $|w_1| > |w_2|$. That means, w_1 was involved in more merge operations during Huffman Tree construction (due to longer word length). However, only words with lower frequencies are merged together, contradicting the fact that w_1 has higher frequency than w_2.

15.5-2 The tables 3.1 are generated from the algorithm. The root table $root(i, j)$, gives the root of the subtree containing keys $k_i \dots k_j$. The OBST root is given by $root(1, n)$ as seen in fig 3.1. All the leaves are the dummy nodes $d_0 \dots d_n$. The cost of the BST is $e(1, 7) = 3.12$.

Expected Search Cost: $e(i, j)$

	0	1	2	3	4	5	6	7
1	0.06	0.28	0.62	1.02	1.34	1.83	2.44	3.12
2		0.06	0.3	0.68	0.93	1.41	1.96	2.61
3			0.06	0.32	0.57	1.04	1.48	2.13
4				0.06	0.24	0.57	1.01	1.55
5					0.05	0.3	0.72	1.2
6						0.05	0.32	0.78
7							0.05	0.34
8								0.05

Root Table: $root(i, j)$

	1	2	3	4	5	6	7
1	1	2	2	2	3	3	5
2		2	3	3	3	5	5
3			3	3	4	5	5
4				4	5	5	6
5					5	6	6
6						6	7
7							7

Table 3.1: Generated Tables from the algorithm

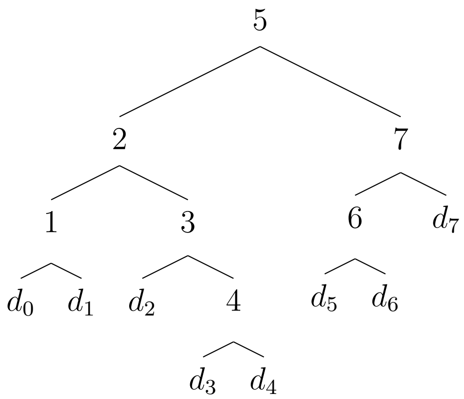

Figure 3.1: Optimal BST from root table

22.3-5 All the considerations for edge (u, v):

a. Since, we have $u.d < v.d$, u must be explored before v. Hence, (u, v) cannot be a back-edge. Also, $v.f < u.f$ implies we return from v before returning from u, which means v and u must be on the same DFS tree. This rules out it being a cross edge.

Alternatively, suppose (u, v) is indeed a tree edge. So, if u occurs before v, then $u.d < v.d$ and while traversing up, $v.f < u.f$.

b. Similar to part a., we have v as the ancestor of u in DFS tree, since v is discovered first and returned to last. Hence, it must be a back edge.
Alternatively, suppose (u, v) is indeed a back edge. That implies v is an ancestor of u or $v.d < u.d$ and $u.f < v.f$.

c. Given, $v.f < u.d$ implies either v is a descendent of u or v is on a branch explored before u. Also, $v.d < u.d$ implies u is a descendent of v or v is on a branch explored before u. Hence, combining the conditions make it a cross-edge.
Alternatively, suppose (u, v) is a cross edge. This means, v is explored completely and returned first, implying that $v.d < v.f < u.d$.