`

NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Cachar, Assam

B.Tech. IVth Sem

Subject Code: CS206

Subject Name: Algorithms

Submitted By:

Name : Subhojit Ghimire

Sch. Id. : 1912160

Branch : CSE – B

1. Determine the cost and structure of an optional binary search tree for a set of n= 7 keys with the following probabilities:

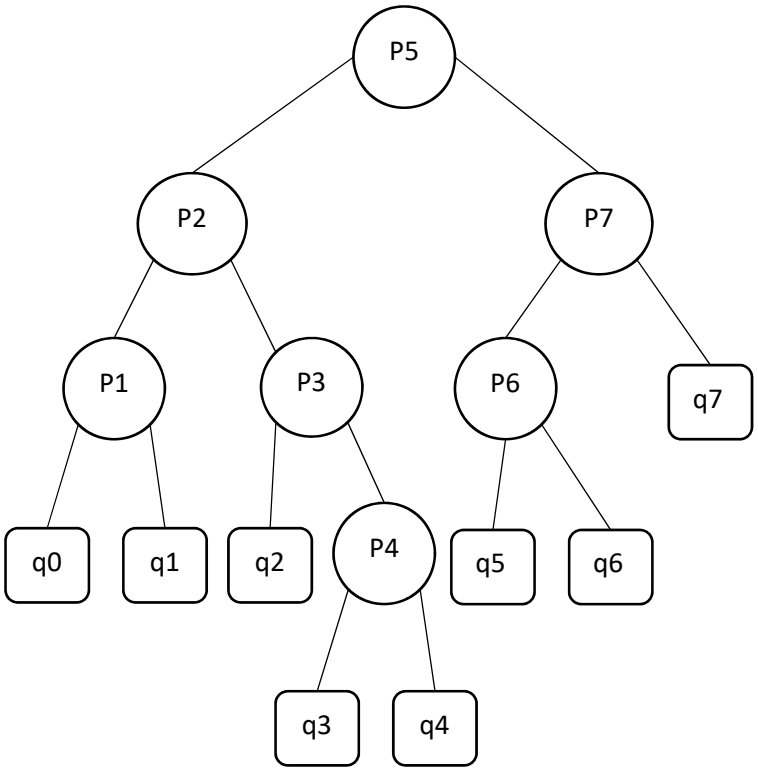
i	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***
p_i		**0.04**	**0.06**	**0.08**	**0.02**	**0.10**	**0.12**	**0.14**
q_i	**0.06**	**0.06**	**0.06**	**0.06**	**0.05**	**0.05**	**0.05**	**0.05**

➔ The matrix e is:

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	0.06	0.28	0.62	1.02	1.34	1.83	1.42	3.12
2		0.06	0.30	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	0.99	1.55
5					0.05	0.30	0.72	1.20
6						0.05	0.32	0.78
7							0.05	0.34
8								0.05

Therefore, the cost is 3.12

The Optimal Binary Search Tree structure is:



2. **Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.**

➔ Approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Example:
Suppose, our activity times are {(3, 5), (1, 4), (4, 7)}
Then, the greedy solution is {activity 1}.
The optimal solution is {activity 2, activity 3}.

Approach of always selecting the compatible activity that overlaps the fewest other remaining activities:
Suppose, our activity times are {(3, 5), (0, 2), (6, 8), (2, 4), (4, 6), (1, 3), (1, 3), (5, 7), (5, 7)}
Then, the greedy solution is {activity 1, activity 2, activity 3}
The optimal solution is {activity 2, activity 3, activity 4, activity 5}

Approach of always selecting the compatible remaining activity with the earliest start time:
Suppose, our activity times are {(1, 10), (2, 3), (4, 5)}
Then, the greedy solution is {activity 1}
The optimal solution is {activity 2, activity 3}

3. **Prove that a binary tree that is not full cannot correspond to an optimal prefix code.**

➔ Let, T be a binary tree corresponds to prefix code such that T is not full. Then, there must exist an internal node, say x, such that x has only one child y.

Construct another binary tree, say N, which has same leaves as T and has same depth as T, except for the leaves which are in the subtree rooted at y in T. These leaves will have depth in N, which implies T cannot correspond to an optimal prefix code.

4. **Prove that if we order the characters in an alphabet so that their frequencies are monotonically decreasing, then there exists an optimal code whose codeword lengths are monotonically increasing.**

➔ Let, two codewords be c_1 and c_2, where $c_1.freq > c_2.freq$
Let, another two codewords be x and y, such that x.length < y.length
Comparing the assignment costs for these two cases:

Shorter codeword to lower frequency. Cost is,

γ_1 = x.length * c_2.freq + y.length * c_1.freq

Shorter codeword to higher frequency

γ_2 = x.length * c_1.freq + y.length *c_2.freq

Taking difference,

$\gamma_1 - \gamma_2$ = (x.length*c_2.freq + y.length*c_1.freq) – (x.length*c_1.freq + y.length*c_2.freq)

= x.length*(c_2.freq – c_1.freq) + y.length*(c_1.freq – c_2.freq)

= (c_1.freq – c_2.freq) * (y.length – x.length)

> 0

Therefore, this corroborates our claim, i.e., if we order the characters in an alphabet so that their frequencies are monotonically decreasing, then there exists an optimal code whose codeword lengths are monotonically increasing.

5. **Show that edge (u, v) is**
   a. **A tree edge or forward edge if and only if u.d < v.d < v.f < u.f,**
   b. **A back edge if and only if v.d ≤ u.d < u.f ≤ v.f, and**
   c. **A cross edge if and only if v.d < v.f < u.d < u.f**

➔ Edge (u, v)
   a. v is a descendant of u.
   b. v is an ancestor of u.
   c. v has been finished when exploring (u, v).