**Assignment 7**

**Title of Assignment**: Given sequence k = k1 <k2 < … < kn of n sorted keys, with a search probability pi for each key ki . Build the Binary search tree that has the least search cost given the access probability for each key.

**Relevant Theory:**

An optimal binary search tree is a binary search tree for which the nodes are arranged on levels such that the tree cost is minimum. For the purpose of a better presentation of optimal binary search trees, we will consider “extended binary search trees”, which have the keys stored at their internal nodes. Suppose “n” keys k1, k2, … , k n are stored at the internal nodes of a binary search tree. It is assumed that the keys are given in sorted order, so that k1< k2 < … < kn. An extended binary search tree is obtained from the binary search tree by adding successor nodes to each of its terminal nodes as indicated in the following figure by squares



In the extended tree: the squares represent terminal nodes. These terminal nodes represent

unsuccessful searches of the tree for key values. The searches did not end successfully, that is, because they represent key values that are not actually stored in the tree the round nodes represent internal nodes; these are the actual keys stored in the tree; assuming that the relative frequency with which each key value is accessed is known, weights can be assigned to each node of the extended tree (p1 …p6). They represent the relative frequencies of searches terminating at each node, that is, they mark the successful searches.

If the user searches a particular key in the tree, 2 cases can occur:

1 – the key is found, so the corresponding weight ‘p’ is incremented;

2 – the key is not found, so the corresponding ‘q’ value is incremented.

GENERALIZATION: the terminal node in the extended tree that is the left successor of k1 can be interpreted as representing all key values that are not stored and are less than k1. Similarly, the terminal node in the extended tree that is the right successor of kn, represents all key values not stored in the tree that are greater than kn. The terminal node that is successed between ki and ki-1 in an inorder traversal represents all key values not stored that lie between ki and ki - 1.

Recursive algorithm. Consider the characteristics of any optimal tree. Of course it has a root and two subtrees. Both subtrees must themselves be optimal binary search trees with respect to their keys and weights. First, any subtree of any binary search tree must be a binary search tree. Second, the subtrees must also be optimal. Since there are “n” possible keys as candidates for the root of the optimal tree, the recursive solution must try them all. For each candidate key as root, all keys less than that key must appear in its left subtree while all keys greater than it must appear in its right subtree. Stating the recursive algorithm based on these observations requires some notations: OBST(i, j) denotes the optimal binary search tree containing the keys ki, ki+1, …, kj;

Wi, j denotes the weight matrix for OBST(i, j)

Wi, j can be defined using the following formula:



Ci, j, 0 ≤ i ≤ j ≤ n denotes the cost matrix for OBST(i, j)

Ci, j can be defined recursively, in the following manner:

Ci, i = Wi, j

Ci, j = Wi, j + mini<k≤j(Ci, k - 1 + Ck, j)

Ri, j, 0 ≤ i ≤ j ≤ n denotes the root matrix for OBST(i, j)

Assigning the notation Ri, j to the value of k for which we obtain a minimum in the above relations, the optimal binary search tree is OBST(0, n) and each subtree OBST(i, j) has the root kRij and as subtrees the trees denoted by OBST(i, k-1) and OBST(k, j).

\*OBST(i, j) will involve the weights qi-1, pi, qi, …, pj, qj.

ALGORITHMS IN PSEUDOCODE

We have the following procedure for determining R(i, j) and C(i, j) with

0 <= i <= j <= n:

PROCEDURE COMPUTE\_ROOT(n, p, q; R, C)

begin

for i = 0 to n do

C (i, i) \_ 0

W (i, i) \_ q(i)

for m = 0 to n do

for i = 0 to (n – m) do

j \_ i + m

W (i, j) \_ W (i, j – 1) + p (j) + q (j)

\*find C (i, j) and R (i, j) which minimize the tree cost

end

**Conclusion**: Understood and performed optimal binary search tree implementation