

EXPERIMENT NO: 9 (D)**Problem Statement:**

Given sequence $k = k_1 < k_2 < \dots < k_n$ of n sorted keys, with a search probability p_i for each key k_i . Build the Binary search tree that has the least search cost given the access probability for each key?

Objectives:

1. To understand concept of OBST.
2. To understand concept & features like extended binary search tree.

Outcomes:

1. Define class for Extended binary search tree using Object Oriented features.
2. Analyze working of functions.

Software Requirements:

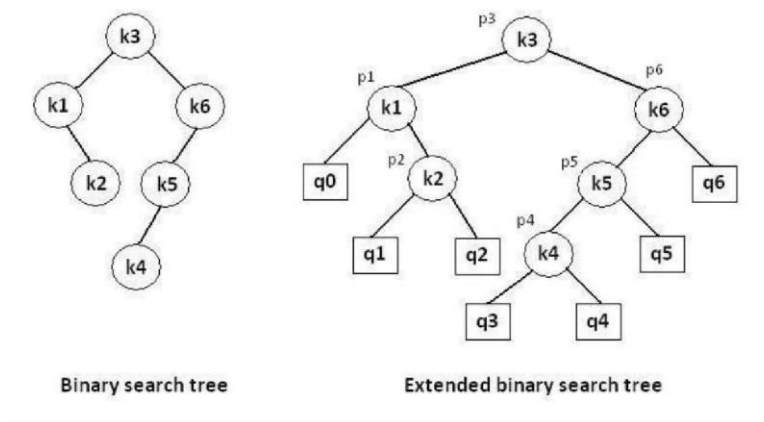
1. 64-bit Open source Linux or its derivative
2. Open Source C++ Programming tool like G++/GCC.
3. Turbo C++ compiler

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 k_1, k_2, \dots, 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 $k_1 < k_2 < \dots < k_n$.

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 ($p_1 \dots p_6$). 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 k_1 can be interpreted as representing all key values that are not stored and are less than k_1 . Similarly, the terminal node in the extended tree that is the right successor of k_n , represents all key values not stored in the tree that are greater than k_n . The terminal node that is successes between k_i and k_{i-1} in an inorder traversal represent all key values not stored that lie between k_i and k_{i-1} .

ALGORITHMS

We have the following procedure for determining $R(i, j)$ and $C(i, j)$ with $0 \leq i \leq j \leq n$:

PROCEDURE COMPUTE_ROOT($n, p, q; R, C$)

begin

for $i = 0$ to n do $C(i, i) \leftarrow 0$

$W(i, i) \leftarrow q(i)$ for $m = 0$ to n do

for $i = 0$ to $(n - m)$ do $j \leftarrow i + m$

$W(i, j) \leftarrow W(i, j - 1) + p(j) + q(j)$

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

end

The following function builds an optimal binary search tree

FUNCTION CONSTRUCT(R, i, j)

begin

*build a new internal node N labeled (i, j) $k \leftarrow R(i, j)$

if $i = k$ then

*build a new leaf node N labeled (i, i) else

* $N \leftarrow \text{CONSTRUCT}(R, i, k)$

* N is the left child of node N if $k = (j - 1)$ then

*build a new leaf node N'' labeled (j, j) else

* $N'' \leftarrow \text{CONSTRUCT}(R, k + 1, j)$

* N'' is the right child of node N return N

end

COMPLEXITY ANALYSIS:

The algorithm requires $O(n^2)$ time and $O(n^2)$ storage. Therefore, as „n“ increases it will run out of storage even before it runs out of time. The storage needed can be reduced by almost half by implementing the two-dimensional arrays as one-dimensional arrays.

Conclusion: Thus we have studied and implemented OBST concept with extended binary search tree feature.