|  |
| --- |
| **Subject: DAA Class: S.E.(Comp)**    **Practical No.: 6 Date:** |

**AIM: Use dynamic programming algorithm to solve optimal binary search tree problem.**

**Title:** Write program for dynamic programming algorithm to solve optimal binary search tree problem.

**Theory:**

A Binary Search Tree (BST) is a tree where the key values are stored in the internal nodes. The external nodes are null nodes. The keys are ordered lexicographically, i.e. for each internal node all the keys in the left sub-tree are less than the keys in the node, and all the keys in the right sub-tree are greater.

When we know the frequency of searching each one of the keys, it is quite easy to compute the expected cost of accessing each node in the tree. An optimal binary search tree is a BST, which has minimal expected cost of locating each node

Search time of an element in a BST is ***O(n)***, whereas in a Balanced-BST search time is ***O(log n)***. Again the search time can be improved in Optimal Cost Binary Search Tree, placing the most frequently used data in the root and closer to the root element, while placing the least frequently used data near leaves and in leaves.

Here, the Optimal Binary Search Tree Algorithm is presented. First, we build a BST from a set of provided **n** number of distinct keys ***< k1, k2, k3, ... kn >***. Here we assume, the probability of accessing a key ***Ki*** is ***pi***. Some dummy keys (***d0, d1, d2, ... dn***) are added as some searches may be performed for the values which are not present in the Key set ***K***. We assume, for each dummy key ***di*** probability of access is ***qi***.

Time Complexity:

The algorithm requires **O (n3)** time, since three nested **for** loops are used. Each of these loops takes on at most **n** values.

Algorithm:

Begin

   define cost matrix of size n x n

   for i in range 0 to n-1, do

      cost[i, i] := freq[i]

   done

   for length in range 2 to n, do

      for i in range 0 to (n-length+1), do

         j := i + length – 1

         cost[i, j] := ∞

         for r in range i to j, done

            if r > i, then

               c := cost[i, r-1]

            else

               c := 0

            if r < j, then

               c := c + cost[r+1, j]

            c := c + sum of frequency from i to j

            if c < cost[i, j], then

               cost[i, j] := c

         done

      done

   done

   return cost[0, n-1]

End

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* left;

struct node\* right;

};

struct node\* createNode(value){

struct node\* newNode = malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct node\* insert(struct node\* root, int data)

{

if (root == NULL) return createNode(data);

if (data < root->data)

root->left = insert(root->left, data);

else if (data > root->data)

root->right = insert(root->right, data);

return root;

}

void inorder(struct node\* root){

if(root == NULL) return;

inorder(root->left);

printf("%d ->", root->data);

inorder(root->right);

}

int main(){

struct node \*root = NULL;

clrscr();

root = insert(root, 8);

insert(root, 3);

insert(root, 1);

insert(root, 6);

insert(root, 7);

insert(root, 10);

insert(root, 14);

insert(root, 4);

inorder(root);

}

**Output:**

