**Code**

#include <iostream>

#include <limits.h>

using namespace std;

#define SIZE 15

class OBST {

int prob[SIZE] = {}; //Probabilities with which we search for an element

int keys[SIZE] = {}; //Elements from which OBST is to be built

int weight[SIZE][SIZE] = {}; //Weight weight[i][j]’ of keys tree having root ’root[i][j]’

int cost[SIZE][SIZE] = {}; //Cost ‘cost[i][j] of keys tree having root ‘root[i][j]

int root[SIZE][SIZE] = {}; //represents root

int n; // number of nodes

public:

void get\_data();

int Min\_Value(int, int);

void build\_OBST();

void build\_tree();

void print(int [][SIZE], int);

};

/\* This function accepts the input data \*/

void OBST::get\_data() {

int i;

cout << "\nOptimal Binary Search Tree \n\nEnter the number of nodes: ";

cin >> n;

cout << "\nEnter " << n << " nodes: ";

for (i = 1; i <= n; i++)

cin >> keys[i];

cout << "\nEnter " << n << " probabilities: ";

for (i = 1; i <= n; i++)

cin >> prob[i];

}

/\* This function returns keys value in the range ‘r[i][j-1]’ to ‘r[i+1][j]’so

that the cost ‘cost[i][k-1]+cost[k][j]’is minimum \*/

int OBST::Min\_Value(int i, int j) {

int l, k;

int minimum = INT\_MAX;

for (l = root[i][j - 1]; l <= root[i + 1][j]; l++) {

if ((cost[i][l - 1] + cost[l][j]) < minimum) {

minimum = cost[i][l - 1] + cost[l][j];

k = l;

}

}

return k;

}

/\* This function builds the table from all the given probabilities It

basically computes cost,root,weight values \*/

void OBST::build\_OBST() {

int i, j, k, l;

for (i = 0; i < n; i++) {

//initialize

weight[i][i] = root[i][i] = cost[i][i] = 0;

//Optimal trees with one node

weight[i][i + 1] = cost[i][i + 1] = prob[i + 1];

root[i][i + 1] = i + 1;

}

weight[n][n] = root[n][n] = cost[n][n] = 0;

//Find optimal trees with ‘m’ nodes

for (l = 2; l <= n; l++) {

for (i = 0; i <= n - l; i++) {

j = i + l;

weight[i][j] = weight[i][j - 1] + prob[j];

k = Min\_Value(i, j);

cost[i][j] = weight[i][j] + cost[i][k - 1] + cost[k][j];

root[i][j] = k;

}

}

cout << "\nCost are: \n";

print(cost, n);

cout << "\nRoot are: \n";

print(root, n);

}

/\* This function builds the tree from the tables made by the OBST function \*/

void OBST::build\_tree() {

int i, j, k;

int queue[20], front = -1, rear = -1;

cout << "\nThe Optimal Binary Search Tree For the Given Nodes Is…\n";

cout << "\nThe Root of this OBST is:: " << keys[root[0][n]];

cout << "\nThe Cost of this OBST is:: " << cost[0][n];

cout << "\n\n\tNODE\tLEFT CHILD\tRIGHT CHILD";

cout << "\n";

queue[++rear] = 0;

queue[++rear] = n;

while (front != rear) {

i = queue[++front];

j = queue[++front];

k = root[i][j];

cout << "\n\t" << keys[k];

if (root[i][k - 1] != 0) {

cout << "\t\t" << keys[root[i][k - 1]];

queue[++rear] = i;

queue[++rear] = k - 1;

}

else

cout << "\t\t";

if (root[k][j] != 0) {

cout << "\t" << keys[root[k][j]];

queue[++rear] = k;

queue[++rear] = j;

}

else

cout << "\t";

}

cout << "\n";

}

void OBST::print(int arr[][SIZE], int n) {

int i, j;

for(i = 0; i <= n; i++) {

for(j = 0; j <= n; j++)

cout << arr[i][j] << '\t';

cout << '\n';

}

}

int main() {

OBST obj;

obj.get\_data();

obj.build\_OBST();

obj.build\_tree();

return 0;

}