//code to display the obst

#include <iostream>

using namespace std;

#define MAX 50

#define INFINITY 999

#define Right 1

#define Left -1

int n;

int W[MAX][MAX]; // Weight matrix

int C[MAX][MAX]; // Cost matrix

int R[MAX][MAX]; // Root matrix

int P[MAX]; // Probabilities of successful search

int Q[MAX]; // Probabilities of unsuccessful search

void displayTree(int left, int right, int child, string indent){

int root = R[left][right];

cout << indent;

// Check the child direction and display the appropriate tree structure

if(child == Right){

cout << "├── ";

indent = indent + "│ ";

}

else if(child == Left){

cout << "└── ";

indent = indent + " ";

}

if(left == right){ // Check if it is a leaf node and display "NULL"

cout << "NULL" << endl;

return;

}

else{ // Display the root value of the subtree

cout << root << endl;;

}

if(right - left == 1){ // Check if there are only two nodes in the subtree (these two nodes will be NULL nodes)

return;

}

displayTree(root, right, Right, indent);

displayTree(left, root-1, Left, indent);

}

// Function to print a matrix

void Print\_Matrix(int M[MAX][MAX]){

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

for(int j=i; j<=n; j++){

cout << M[i][j] << " ";

}

cout << endl;

}

}

void Optimal\_Binary\_Search(){

int min, min\_preorderIndex;

// Initialization

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

W[i][i] = Q[i];

C[i][i] = 0;

R[i][i] = 0;

}

// I-Weighted Matrix

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

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

W[j][j+i] = P[j+i-1] + Q[j+i] + W[j][j+i-1]; // Calculate the weight of the sub-tree rooted at node j

}

}

// II-Cost Matrix and Root Matrix

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

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

min = INFINITY;

for(int k=j+1; k<=j+i; k++){

// Find the minimum cost of all possible sub-trees rooted at node j

if(C[j][k-1] + C[k][j+i] < min){

min = C[j][k-1] + C[k][j+i];

min\_preorderIndex = k;

}

}

C[j][j+i] = min + W[j][j+i]; // Calculate the cost of the sub-tree rooted at node j

R[j][j+i] = min\_preorderIndex; // Record the root of the sub-tree rooted at node j

}

}

cout << "Weight Matrix :-" << endl; Print\_Matrix(W);

cout << "Cost Matrix :-" << endl; Print\_Matrix(C);

cout << "Root Matrix :-" << endl; Print\_Matrix(R);

cout << "OBST :-" << endl; displayTree(0, n, 0,"");

}

int main(){

cout << "Enter the total number of nodes in the Binary Tree : ";

cin >> n;

cout << "Enter the Probabilities of Successful Search : ";

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

cin >> P[i];

}

cout << "Enter the Probabilities of Unsuccessful Search : ";

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

cin >> Q[i];

}

Optimal\_Binary\_Search();

return 0;

}

OUTPUT:

Enter the total number of nodes in the Binary Tree : 4

Enter the Probabilities of Successful Search : 10

20

30

40

Enter the Probabilities of Unsuccessful Search : 10

20

30

40

50

Weight Matrix :-

10 40 90 160 250

20 70 140 230

30 100 190

40 130

50

Cost Matrix :-

0 40 130 290 510

0 70 210 430

0 100 290

0 130

0

Root Matrix :-

0 1 2 3 3

0 2 3 3

0 3 4

0 4

0

OBST :-

3

├── 4

└── 2

├── NULL

└── 1