```
#include <iostream>
#include "BST.cpp"
using namespace std;
#define INF  99999


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Function declarations~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~~~~~

void optsearchtree( );                          //Finds cheapest cost from Key_i to Key_j, and stores them ↙
    in costMatrix
double pSum(int i, int j);                      //To calculate Sum of probablities required by             ↙
    optsearchtree.
BSTNode* createBST(int i, int j);               //Creates an optimal Binary serach tree based on info from ↙
    rootMatrix[n][n]
BSTNode* search (BSTNode* tree, int key);       //searches a key in a tree. If it is found it returns a    ↙
    pointer to it, otherwise NULL

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Global Variables ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~~~~~

int keys[] = {10, 12, 20};                      //Keys have to be ordered in a  non-decreasing manner
double p[] = {.4, .1, .5};                      //Probablities corresponding to the keys
const int n = 3;                                //Size


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Output Matrices ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~~~~~

double costMatrix[n][n];                        // costMatrix[i][j] stores optimal cost for key_i to key_j
int    rootMatrix[n][n];                        // rootMatrix[i][j] stores optimal root for key_i to key_j


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Driver ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~~~~~
int main ()
{
    optsearchtree();

    //Print the cost of the tree. (most expensive node)
    cout<<"Cost of the optimal binary search tree: "<<costMatrix[0][n-1] << "\n"<< endl;


    //Print optimal cost b/w two nodes.
    cout<<"Printing costMatrix[][]" << endl;

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cout << costMatrix[i][j] << "    ";
        }
        cout << "\n\n";
    }


    //Print optimal root b/w two nodes.
    cout<<"Printing rootMatrix[][]" << endl;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cout << rootMatrix[i][j] << "    ";
        }
        cout << "\n \n";
    }

//Create BST and search a number
    cout<<"Creating BST" << endl;
    BSTNode* myTree = createBST(0, 2);
```

```
    if(myTree != NULL)
        cout<<"BST Created" << endl;

    cout<<"Searching a key" << endl;
    if(search (myTree, 23)  ==  NULL)
        cout<<"Key not found" << endl;
    else cout<<"Key not found" << endl;




}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ createBST( ) ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~~~
//Creates an optimal Binary serach tree based on info from rootMatrix[n][n]


BSTNode* createBST(int i, int j){
    int k;
    BSTNode* p;

    k = rootMatrix[i][j];
    if(k == 0)
        return NULL;
    else{
        p = new BSTNode(keys[k]);
        p-> left  = createBST(i, k-1);
        p-> right = createBST(k+1, j);

    return p;
    }
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ optsearchtree( ) ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~~~~
//Finds cheapest cost from Key_i to Key_j, and stores them in costMatrix
void optsearchtree( ) {

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            costMatrix[i][j] = 0;
            rootMatrix[i][j] = 0;
        }
    }

    // For cases when i = j
    // Also, to fill the diagonal first
    // costMatrix[][] requires the diagonal to filled first
    for (int i = 0; i < n; i++)
    {
        costMatrix[i][i] = p[i];
        rootMatrix[i][i] = i;
    }


    // For non-diagonal keys
    for (int L = 2; L <= n; L++) {
        // i (row) is starting index for key length L
        for (int i = 0; i < n-L+1; i++) {
            // j is the last index in i'th row
            int j = i+L-1;
            if (j < n)
                costMatrix[i][j] = INF;
            for (int r = i; r <= j; r++) {
                // c contains cost from i to j
```

```
                double c = ( ((r > i) ? costMatrix[i][r-1] : 0) +
                             ((r < j) ? costMatrix[r+1][j] : 0) +
                             pSum(i , j)
                    );
                if (costMatrix[i][j] > c){
                    costMatrix[i][j] = c;
                    rootMatrix[i][j] = r;
                }
            }
        }
    }
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ pSum( ) ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~~
//To  calculate Sum of probablities required by optsearchtree.
double pSum(int i, int j) {
    double sum = 0;
    for (int k = i; k <= j ; k++)
        sum = sum + p[k];
    return sum;
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ bool search(...) ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~↙
    ~~~~~~~

BSTNode* search (BSTNode* tree, int keyin){
    bool found = false;
    BSTNode* pointer = NULL;
    while(! found)
        if(pointer-> key == keyin)
        {
            found = true;
            return pointer;
        }
        else if (keyin < (pointer->key))
        {
            pointer = pointer->left;
        }
        else
        {
            pointer = pointer->right;
        }

    return NULL;
}
```