```cpp
#include <iostream>
#include <vector>
#include <limits>  // For numeric_limits

using namespace std;

// Function to compute the optimal BST cost and structure
void optimalBST(const vector<int>& keys, const vector<double>& prob, int n, vector<
vector<int> >& root) {
    vector< vector<double> > cost(n, vector<double>(n, 0));

    // Initialize cost and root for single-key trees
    for (int i = 0; i < n; i++) {
        cost[i][i] = prob[i];
        root[i][i] = i;
    }

    // Compute optimal costs for larger subtrees
    for (int len = 2; len <= n; len++) {  // Tree size
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            cost[i][j] = numeric_limits<double>::max();  // Set to a large value

            // Compute the total probability sum for the range
            double totalWeight = 0;
            for (int k = i; k <= j; k++) {
                totalWeight += prob[k];
            }

            // Try each key as a root and find the minimum cost
            for (int k = i; k <= j; k++) {
                double leftCost = (k > i) ? cost[i][k - 1] : 0;
                double rightCost = (k < j) ? cost[k + 1][j] : 0;
                double totalCost = leftCost + rightCost + totalWeight;

                // Store the minimum cost and corresponding root
                if (totalCost < cost[i][j]) {
                    cost[i][j] = totalCost;
                    root[i][j] = k;
                }
            }
        }
    }

    cout << "Minimum search cost: " << cost[0][n - 1] << endl;
}

// Function to print the Optimal BST structure
void printBST(const vector<int>& keys, const vector< vector<int> >& root, int i, int
j, int parent, bool isLeft) {
    if (i > j) return;

    int r = root[i][j];  // Root of the subtree

    if (parent == -1)
```

```cpp
        cout << "Key " << keys[r] << " is the ROOT" << endl;
    else if (isLeft)
        cout << "Key " << keys[r] << " is the LEFT child of " << keys[parent] << endl;
    else
        cout << "Key " << keys[r] << " is the RIGHT child of " << keys[parent] << endl;

    // Recur for left and right subtrees
    printBST(keys, root, i, r - 1, r, true);
    printBST(keys, root, r + 1, j, r, false);
}

int main() {
    // Sorted keys and corresponding search probabilities
    int keyArr[] = {10, 20, 30, 40};
    double probArr[] = {0.4, 0.3, 0.2, 0.1};
    int n = sizeof(keyArr) / sizeof(keyArr[0]);

    // Convert arrays to vectors (C++98-compatible)
    vector<int> keys(keyArr, keyArr + n);
    vector<double> prob(probArr, probArr + n);
    vector< vector<int> > root(n, vector<int>(n, 0));

    // Compute the optimal BST
    optimalBST(keys, prob, n, root);

    // Print the structure of the Optimal BST
    cout << "\nOptimal Binary Search Tree structure:\n";
    printBST(keys, root, 0, n - 1, -1, false);

    return 0;
}
```

OutPut :

```
D:\SE Computer\LAB CODES\DSA\DSA8.exe
Minimum search cost: 1.8

Optimal Binary Search Tree structure:
Key 20 is the ROOT
Key 10 is the LEFT child of 20
Key 30 is the RIGHT child of 20
Key 40 is the RIGHT child of 30

--------------------------------
Process exited after 0.2079 seconds with return value 0
Press any key to continue . . .
```