

```

1  #include <iostream>
2  #include <vector>
3  #include <limits> // For numeric_limits
4
5  using namespace std;
6
7  // Function to compute the optimal BST cost and structure
8  void optimalBST(const vector<int>& keys, const vector<double>& prob, int n, vector<
vector<int> >& root) {
9      vector< vector<double> > cost(n, vector<double>(n, 0));
10
11     // Initialize cost and root for single-key trees
12     for (int i = 0; i < n; i++) {
13         cost[i][i] = prob[i];
14         root[i][i] = i;
15     }
16
17     // Compute optimal costs for larger subtrees
18     for (int len = 2; len <= n; len++) { // Tree size
19         for (int i = 0; i <= n - len; i++) {
20             int j = i + len - 1;
21             cost[i][j] = numeric_limits<double>::max(); // Set to a large value
22
23             // Compute the total probability sum for the range
24             double totalWeight = 0;
25             for (int k = i; k <= j; k++) {
26                 totalWeight += prob[k];
27             }
28
29             // Try each key as a root and find the minimum cost
30             for (int k = i; k <= j; k++) {
31                 double leftCost = (k > i) ? cost[i][k - 1] : 0;
32                 double rightCost = (k < j) ? cost[k + 1][j] : 0;
33                 double totalCost = leftCost + rightCost + totalWeight;
34
35                 // Store the minimum cost and corresponding root
36                 if (totalCost < cost[i][j]) {
37                     cost[i][j] = totalCost;
38                     root[i][j] = k;
39                 }
40             }
41         }
42     }
43
44     cout << "Minimum search cost: " << cost[0][n - 1] << endl;
45 }
46
47 // Function to print the Optimal BST structure
48 void printBST(const vector<int>& keys, const vector< vector<int> >& root, int i, int
j, int parent, bool isLeft) {
49     if (i > j) return;
50
51     int r = root[i][j]; // Root of the subtree
52
53     if (parent == -1)

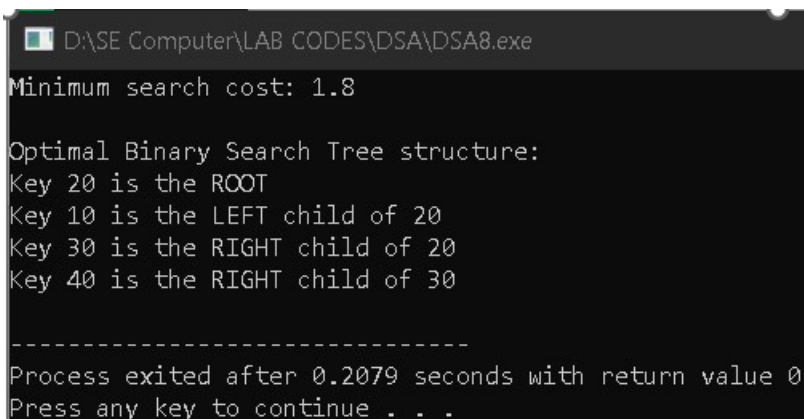
```

```

54         cout << "Key " << keys[r] << " is the ROOT" << endl;
55     else if (isLeft)
56         cout << "Key " << keys[r] << " is the LEFT child of " << keys[parent] <<
endl;
57     else
58         cout << "Key " << keys[r] << " is the RIGHT child of " << keys[parent] <<
endl;
59
60     // Recur for left and right subtrees
61     printBST(keys, root, i, r - 1, r, true);
62     printBST(keys, root, r + 1, j, r, false);
63 }
64
65 int main() {
66     // Sorted keys and corresponding search probabilities
67     int keyArr[] = {10, 20, 30, 40};
68     double probArr[] = {0.4, 0.3, 0.2, 0.1};
69     int n = sizeof(keyArr) / sizeof(keyArr[0]);
70
71     // Convert arrays to vectors (C++98-compatible)
72     vector<int> keys(keyArr, keyArr + n);
73     vector<double> prob(probArr, probArr + n);
74     vector< vector<int> > root(n, vector<int>(n, 0));
75
76     // Compute the optimal BST
77     optimalBST(keys, prob, n, root);
78
79     // Print the structure of the Optimal BST
80     cout << "\nOptimal Binary Search Tree structure:\n";
81     printBST(keys, root, 0, n - 1, -1, false);
82
83     return 0;
84 }
85

```

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 . . .

```