DAA Lab

Practical 7

Amber Shukla

A1-B3-42

Aim: Construction of OBST

**Task 1:**

Code:

#include <bits/stdc++.h>

using namespace std;

typedef double db;

vector<db> keys, hit\_prob, miss\_prob;

int n;

db computeOBST()

{

    vector<vector<db>> e(n + 1, vector<db>(n + 1, 0.0));

    vector<vector<db>> w(n + 1, vector<db>(n + 1, 0.0));

    for (int i = 0; i <= n; i++) // Fill Diagonals

    {

        e[i][i] = miss\_prob[i];

        w[i][i] = miss\_prob[i];

    }

    for (int len = 1; len <= n; len++)

    {

        int j;

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

        {

            j = i + len;

            e[i][j] = INT\_MAX;

            w[i][j] = w[i][j - 1] + hit\_prob[j - 1] + miss\_prob[j];

            for (int r = i; r < j; r++)

            {

                db cost = e[i][r]       // Cost of left subtree [i, r) -> i to r-1

                          + e[r + 1][j] // Cost of right subtree [r+1, j) -> r+1 to j-1

                          + w[i][j];    // Total weight of current subtree

                e[i][j] = min(e[i][j], cost);

            }

        }

    }

    return e[0][n];

}

int main()

{

    cout << "Enter N: ";

    cin >> n;

    keys.resize(n);

    hit\_prob.resize(n);

    miss\_prob.resize(n + 1);

    cout << "Enter Keys: ";

    for (int i = 0; i < n; i++)

        cin >> keys[i];

    cout << "Enter P[] Array: ";

    for (int i = 0; i < n; i++)

        cin >> hit\_prob[i];

    cout << "Enter Q[] Array: ";

    for (int i = 0; i <= n; i++)

        cin >> miss\_prob[i];

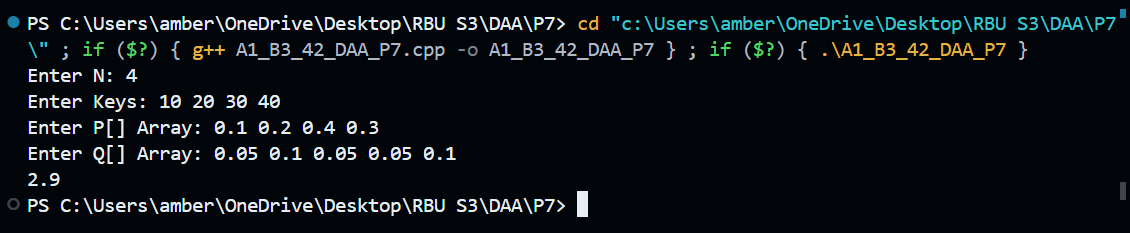
    db minCost = computeOBST();

    cout << setprecision(4) << minCost << '\n';

    return 0;

}

**Output:**



**Task 2 (Online Submission):**

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Solution

{

public:

    int optimalSearchTree(int keys[], int freq[], int n)

    {

        vector<vector<int>> dp(n, vector<int>(n, 0));

        vector<int> prefix(n + 1, 0);

        for (int i = 0; i < n; i++)

            prefix[i + 1] = prefix[i] + freq[i];

        for (int i = 0; i < n; i++)

            dp[i][i] = freq[i];

        for (int len = 2; len <= n; len++)

        {

            for (int i = 0; i + len - 1 < n; i++)

            {

                int j = i + len - 1;

                int totalFreq = prefix[j + 1] - prefix[i];

                int best = INT\_MAX;

                for (int r = i; r <= j; r++)

                    best = min(best, dp[i][r - 1] + dp[r + 1][j] + totalFreq);

                dp[i][j] = best;

            }

        }

        return dp[0][n - 1];

    }

};

**Output:**

