|  |  |
| --- | --- |
| Name : | Swati Kundu |
| Class : | A4-B3 |
| Roll No : | 48 |
| Subject : | Daa |

**Practical 6**

**Aim**: Construction of OBST

**Problem Statement**: Smart Library Search Optimization

**Task 1:**

**Scenario:**

A university digital library system stores frequently accessed books using a binary search

mechanism. The library admin wants to minimize the average search time for book lookups by

arranging the book IDs optimally in a binary search tree.

Each book ID has a probability of being searched successfully and an associated probability for

unsuccessful searches (when a book ID does not exist between two keys).

Your task is to determine the minimum expected cost of searching using an Optimal Binary

Search Tree (OBST).

**Input Format**

First line: integer n — number of book IDs.

Second line: n integers representing the sorted book IDs (keys).

Third line: n real numbers — probabilities of successful searches (p[i]).

Fourth line: n+1 real numbers — probabilities of unsuccessful searches (q[i]).

Keys: 10 20 30 40

P[i]: 0.1 0.2 0.4 0.3

Q[i]: 0.05 0.1 0.05 0.05 0.1

**Output Format**

Print the minimum expected cost of the Optimal Binary Search Tree, rounded to 4 decimal

places.

**CODE :**

#include <stdio.h>

#include <float.h>

// Function to find the minimum expected cost of OBST

void optimalBST(int n, int keys[], double p[], double q[]) {

double e[n + 2][n + 1]; // Expected cost

double w[n + 2][n + 1]; // Weight (sum of probabilities)

int root[n + 1][n + 1]; // Root table (for possible reconstruction if needed)

// Initialization for trees with zero keys

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

e[i][i - 1] = q[i - 1];

w[i][i - 1] = q[i - 1];

}

// Compute optimal cost for all chain lengths

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

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

int j = i + l - 1;

e[i][j] = DBL\_MAX;

w[i][j] = w[i][j - 1] + p[j - 1] + q[j];

// Try all possible roots for the subtree

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

double t = e[i][r - 1] + e[r + 1][j] + w[i][j];

if (t < e[i][j]) {

e[i][j] = t;

root[i][j] = r;

}

}

}

}

printf("\n---------------------------------------------\n");

printf(" Optimal Binary Search Tree Construction\n");

printf("---------------------------------------------\n");

printf("Minimum Expected Cost of Search = %.4f\n", e[1][n]);

printf("---------------------------------------------\n");

}

int main() {

int n;

printf("Enter number of book IDs: ");

scanf("%d", &n);

int keys[n];

double p[n], q[n + 1];

printf("Enter the sorted book IDs:\n");

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

scanf("%d", &keys[i]);

printf("Enter the probabilities of successful searches (P[i]):\n");

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

scanf("%lf", &p[i]);

printf("Enter the probabilities of unsuccessful searches (Q[i]):\n");

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

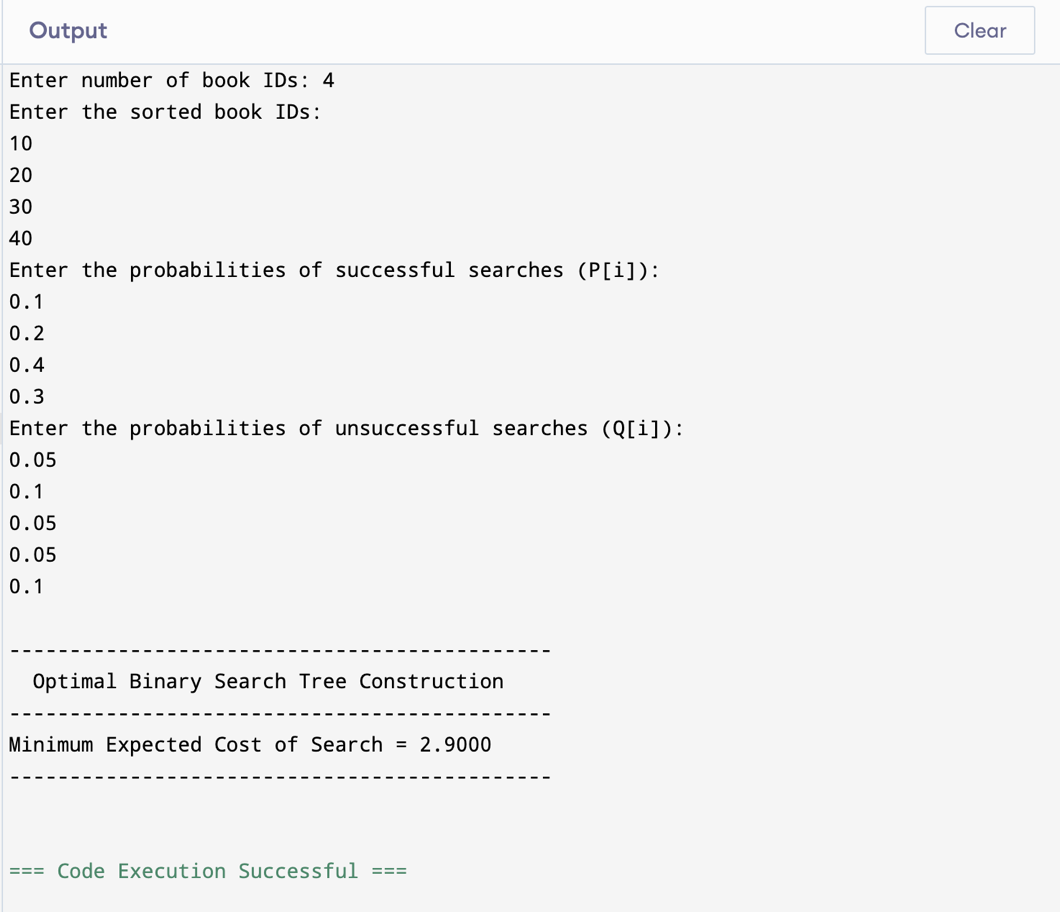
scanf("%lf", &q[i]);

optimalBST(n, keys, p, q);

return 0;

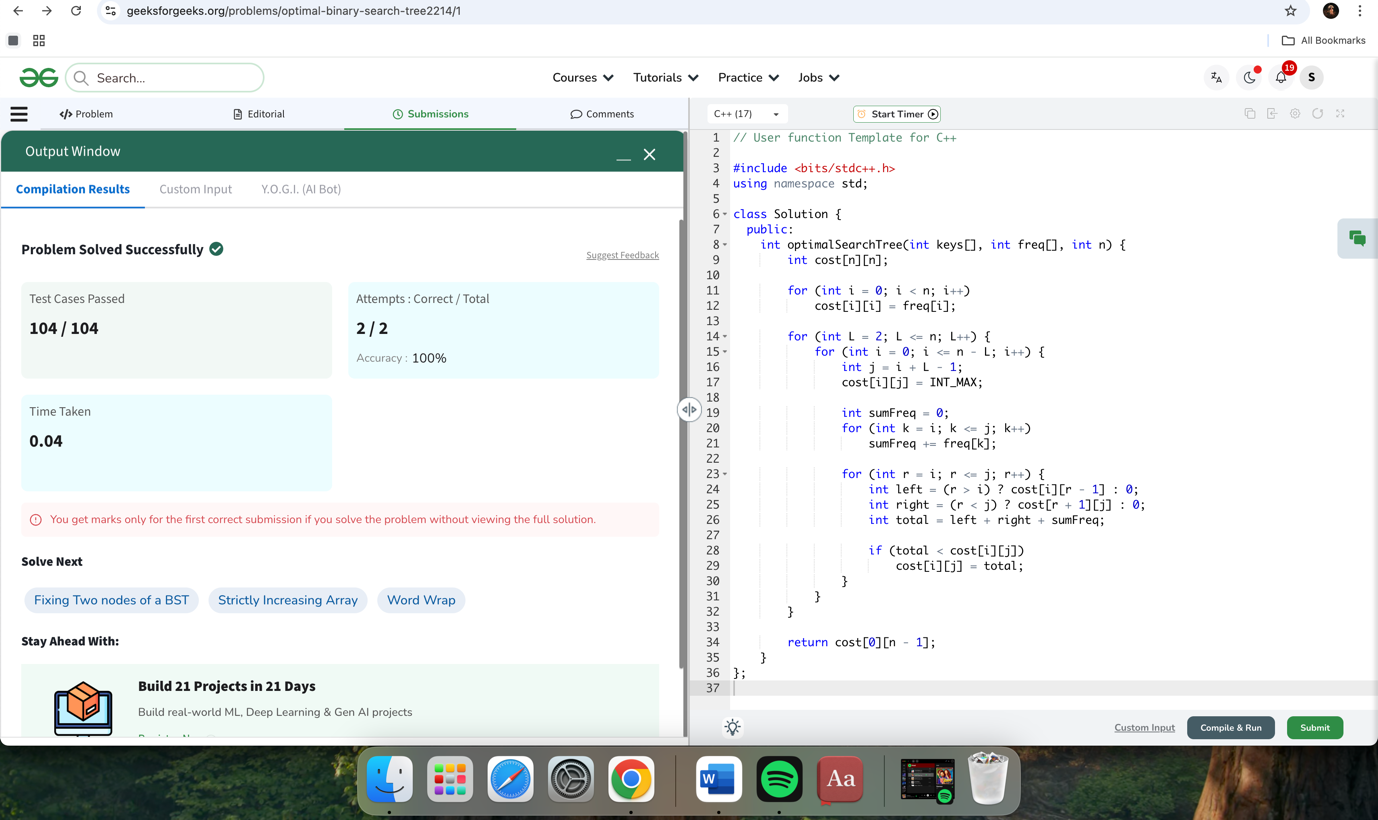
}

**OUTPUT:**



**Task 2:**

[**https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1**](https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1)

****

#include <bits/stdc++.h>

using namespace std;

class Solution {

public:

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

int cost[n][n];

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

cost[i][i] = freq[i];

for (int L = 2; L <= n; L++) {

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

int j = i + L - 1;

cost[i][j] = INT\_MAX;

int sumFreq = 0;

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

sumFreq += freq[k];

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

int left = (r > i) ? cost[i][r - 1] : 0;

int right = (r < j) ? cost[r + 1][j] : 0;

int total = left + right + sumFreq;

if (total < cost[i][j])

cost[i][j] = total;

}

}

}

return cost[0][n - 1];

}

};