DAA PRACTICAL 6
SUCCESSFUL AND UNSUCCESSFUL SEARCH

NAME - SWARALI BAKRE
SECTION, BATCH, ROLL NO - A5-B4-60

Aim: Construction of OBST
Problem Statement: Smart Library Search Optimization

TASK 1
CODE -

```
import java.util.*;

public class Main {
    public static void optimalBST(double[] p, double[] q, int n) {
        double[][] E = new double[n + 1][n + 1];
        double[][] W = new double[n + 1][n + 1];
        int[][] R = new int[n + 1][n + 1];

        // Step 1: Initialization
        for (int i = 0; i <= n; i++) {
            E[i][i] = q[i];
            W[i][i] = q[i];
            R[i][i] = 0;
        }

        // Step 2: Compute optimal costs
        for (int d = 1; d <= n; d++) {
            for (int i = 0; i <= n - d; i++) {
                int j = i + d;
                E[i][j] = Double.MAX_VALUE;
                W[i][j] = W[i][j - 1] + p[j - 1] + q[j];

                for (int k = i + 1; k <= j; k++) {
                    double cost = E[i][k - 1] + E[k][j] + W[i][j];
                    if (cost < E[i][j]) {
                        E[i][j] = cost;
                        R[i][j] = k;
                    }
                }
```

```java
            }
        }

        // Step 3: Print results
        System.out.println("E Matrix:");
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= n; j++) {
                System.out.printf("%.2f ", E[i][j]);
            }
            System.out.println();
        }

        System.out.println("\nW Matrix:");
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= n; j++) {
                System.out.printf("%.2f ", W[i][j]);
            }
            System.out.println();
        }

        System.out.println("\nR Matrix:");
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= n; j++) {
                System.out.print(R[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int n = 3;
        double[] p = {0.15, 0.10, 0.05};
        double[] q = {0.05, 0.10, 0.05, 0.05};

        optimalBST(p, q, n);
    }
}
```

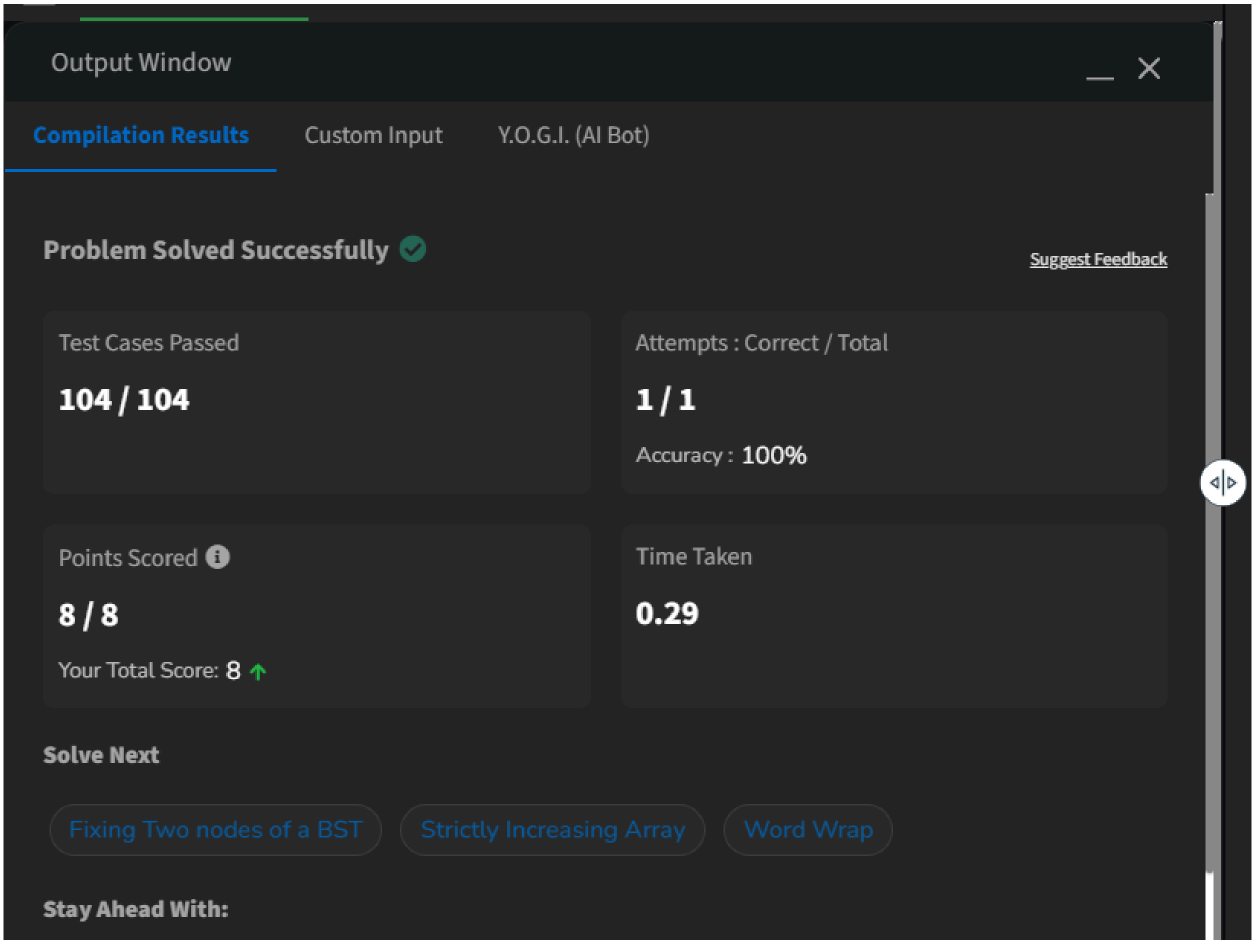
```
Output
E Matrix:
0.05 0.45 0.90 1.25
0.00 0.10 0.40 0.70
0.00 0.00 0.05 0.25
0.00 0.00 0.00 0.05

W Matrix:
0.05 0.30 0.45 0.55
0.00 0.10 0.25 0.35
0.00 0.00 0.05 0.15
0.00 0.00 0.00 0.05

R Matrix:
0 1 1 2
0 0 2 2
0 0 0 3
0 0 0 0
```

TASK 2 - GFG

CODE -

```
class Solution
{
    static int optimalSearchTree(int keys[], int freq[], int n)
    {
        // dp[i][j] will store the minimum cost of BST for the keys[i..j]
        int[][] dp = new int[n][n];

        // sum[i][j] will store the sum of frequencies from freq[i] to freq[j]
        int[][] sum = new int[n][n];

        // Step 1: Precompute the sum of frequencies for every subarray freq[i..j]
        for (int i = 0; i < n; i++)
        {
            sum[i][i] = freq[i];  // For a single key, sum is just its frequency
            for (int j = i + 1; j < n; j++)
```

```
            {
                sum[i][j] = sum[i][j - 1] + freq[j];  // Sum of frequencies from i to j
            }
        }

        // Step 2: Fill the dp array for the base case when there's only one key
        for (int i = 0; i < n; i++)
        {
            dp[i][i] = freq[i];  // Cost when there's only one key is just its frequency
        }

        // Step 3: Solve for larger subarrays
        for (int len = 2; len <= n; len++)
        {  // len is the subarray length
            for (int i = 0; i <= n - len; i++)
            {
                int j = i + len - 1;  // j is the end index of the subarray

                // Initialize dp[i][j] to a large value
                dp[i][j] = Integer.MAX_VALUE;

                // Try all possible roots for the current subarray keys[i..j]
                for (int r = i; r <= j; r++)
                {
                    // Cost if r is the root
                    int cost = (r > i ? dp[i][r - 1] : 0) + (r < j ? dp[r + 1][j] : 0) + sum[i][j];

                    // Update dp[i][j] with the minimum cost
                    dp[i][j] = Math.min(dp[i][j], cost);
                }
            }
        }

        // The answer is the minimum cost of the BST for the whole array
        return dp[0][n - 1];
    }
}
```


Output Window
Compilation Results
Custom Input
Y.O.G.I. (AI Bot)
Problem Solved Successfully
Suggest Feedback
Test Cases Passed
104 / 104
Attempts : Correct / Total
1 / 1
Accuracy : 100%
Points Scored
8 / 8
Your Total Score: 8
Time Taken
0.29
Solve Next
Fixing Two nodes of a BST
Strictly Increasing Array
Word Wrap
Stay Ahead With:

Compilation Completed

• Case 1

Input:

```
2
10 12
34 50
```

Your Output:

118

Expected Output:

118

---x---