

Ramdeobaba University, Nagpur
Department of Computer Science and Engineering
Session: 2024-2025

Design and Analysis of Algorithms Lab III Semester PRACTICAL NO. 6

Name: Ananya Biyani

Roll number: 02

Section: A4-B1

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:

```
public class OptimalBST {

    public static double optimalBST(double[] p, double[] q, int n)

    { double[][] e = new double[n + 2][n + 2];

      double[][] w = new double[n + 2][n + 2];

      int[][] root = new int[n + 2][n + 2];

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

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

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

      }

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

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

              int j = i + l - 1;

              e[i][j] = Double.MAX_VALUE;

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

              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[l][j] = t;

        root[l][j] = r;

    }

}

}

}

return e[1][n];

}

public static void main(String[] args) {

    int n = 4;

    double[] p = {0.1, 0.2, 0.4, 0.3};

    double[] q = {0.05, 0.1, 0.05, 0.05, 0.1};

    double minCost = optimalBST(p, q, n);

    System.out.printf("Minimum expected search cost: %.4f\n",
minCost);

}

}

```

OUTPUT:

```
Minimum expected search cost: 2.9000
```

Task 2:

<https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/>

The screenshot shows a coding platform interface with two main panels. The left panel, titled 'Output Window', displays the following information:

- Compilation Results:** Custom Input, Y.O.S.U. (W. Set)
- Problem Solved Successfully:** (Green checkmark icon)
- Test Cases Passed:** 104 / 104
- Attempts:** Correct / Total: 1 / 2
- Accuracy:** 50%
- Points Scored:** 8 / 8
- Time Taken:** 0.25
- Your Total Score:** 8 (Green plus icon)
- Solve Next:** Fixing Two nodes of a BST, Strictly Increasing Array, Word Wrap
- Stay Ahead With:**

The right panel shows the C++ code for the solution, which is a dynamic programming approach for finding the optimal binary search tree. The code is as follows:

```
1 class Solution {
2     int optimalSearchTree(int keys[], int freq[], int n) {
3         int dp[n][n];
4         int sum = new int[n];
5         sum[0] = freq[0];
6         for (int i = 1; i <= n; i++) {
7             sum[i] = sum[i - 1] + freq[i];
8         }
9         for (int i = 0; i <= n; i++) {
10             dp[i][i] = freq[i];
11         }
12         for (int l = 2; l <= n; l++) {
13             for (int h = 0; h <= n - l; h++) {
14                 int i = h + l - 1;
15                 dp[i][i] = Integer.MAX_VALUE;
16                 int freqlen = sum[i] - (h > 0 ? sum[h - 1] : 0);
17                 for (int r = h; r <= i; r++) {
18                     int costleft = (r > 0) ? dp[h][r - 1] : 0;
19                     int cost = costleft + costright + freqlen;
20                     dp[i][i] = Math.min(dp[i][i], cost);
21                 }
22             }
23         }
24         return dp[n - 1][n - 1];
25     }
26 }
```