

Practical-6

TASK 1:

Code:

```
def OptimalBST(p, q, n):  
    E = [[0] * (n + 2) for _ in range(n + 2)]  
    W = [[0] * (n + 2) for _ in range(n + 2)]  
    R = [[0] * (n + 1) for _ in range(n + 1)]  
  
    for i in range(n + 1):  
        E[i + 1][i] = q[i]  
        W[i + 1][i] = q[i]  
  
    for l in range(1, n + 1):  
        for i in range(1, n - l + 2):  
            j = i + l - 1  
            W[i][j] = W[i][j - 1] + p[j - 1] + q[j]  
            E[i][j] = float('inf')  
            for k in range(i, j + 1):  
                cost = E[i][k - 1] + E[k + 1][j] + W[i][j]  
                if cost < E[i][j]:  
                    E[i][j] = cost  
                    R[i][j] = k  
  
    return E, W, R  
  
p_input = [0.15, 0.10, 0.05, 0.10, 0.20]  
q_input = [0.10, 0.05, 0.05, 0.10, 0.15, 0.10]  
n = len(p_input)  
  
E, W, R = OptimalBST(p_input, q_input, n)
```

Mandar Vivek Amte
A5-B2-27
DAA-Lab

```
def print_matrix(matrix, n, decimals=2):  
    for r in range(1, n + 2):  
        formatted_row = [round(matrix[r][c], decimals) for c in range(1, n + 2)]  
        print(formatted_row)  
  
print("Expected Cost Matrix (E):")  
print_matrix(E, n)  
  
print("\nWeight Matrix (W):")  
print_matrix(W, n)  
  
print("\nRoot Matrix (R):")  
for r in range(1, n + 1):  
    print(R[r][1:n + 1])
```

OUTPUT:

```
Expected Cost Matrix (E):  
[0.45, 0.85, 1.4, 2.25, 3.25, 0]  
[0.05, 0.3, 0.75, 1.5, 2.35, 0]  
[0, 0.05, 0.35, 0.95, 1.8, 0]  
[0, 0, 0.1, 0.6, 1.35, 0]  
[0, 0, 0, 0.15, 0.7, 0]  
[0, 0, 0, 0, 0.1, 0]  
  
Weight Matrix (W):  
[0.3, 0.45, 0.6, 0.85, 1.15, 0]  
[0.05, 0.2, 0.35, 0.6, 0.9, 0]  
[0, 0.05, 0.2, 0.45, 0.75, 0]  
[0, 0, 0.1, 0.35, 0.65, 0]  
[0, 0, 0, 0.15, 0.45, 0]  
[0, 0, 0, 0, 0.1, 0]  
  
Root Matrix (R):  
[1, 1, 2, 2, 4]  
[0, 2, 2, 3, 4]  
[0, 0, 3, 4, 4]  
[0, 0, 0, 4, 5]  
[0, 0, 0, 0, 5]
```

TASK 2:

The screenshot shows a coding platform interface with a dark theme. The top navigation bar includes links for Courses, Tutorials, Practice, and Jobs. The main area is divided into two panels. The left panel, titled 'Output Window', shows the 'Compilation Results' for a problem. It indicates that the problem was solved successfully with 104 out of 104 test cases passed, 8 out of 8 points scored, and a time taken of 1.63 seconds. The right panel displays the Python code for the solution, which is a class named 'Solution' with a method 'optimalSearchTree'. The code uses dynamic programming to calculate the minimum cost of a search tree for a given set of keys and frequencies.

Problem Solved Successfully ✓

Test Cases Passed: **104 / 104**

Attempts: Correct / Total: **1 / 1**

Accuracy: 100%

Points Scored: **8 / 8**

Your Total Score: 8 ↑

Time Taken: **1.63**

Solve Next

Fixing Two nodes of a BST | Strictly Increasing Array | Word Wrap

Stay Ahead With:

Build 21 Projects in 21 Days

```
1 class Solution:
2     def optimalSearchTree(self, keys, freq, n):
3         dp = [[0] * n for _ in range(n)]
4         sum_freq = [[0] * n for _ in range(n)]
5
6         for i in range(n):
7             sum_freq[i][i] = freq[i]
8             for j in range(i + 1, n):
9                 sum_freq[i][j] = sum_freq[i][j - 1] + freq[j]
10
11        for i in range(n):
12            dp[i][i] = freq[i]
13
14        for length in range(2, n + 1):
15            for i in range(n - length + 1):
16                j = i + length - 1
17                dp[i][j] = float('inf')
18                for k in range(i, j + 1):
19                    left_cost = dp[i][k - 1] if k > i else 0
20                    right_cost = dp[k + 1][j] if k < j else 0
21                    cost = left_cost + right_cost + sum_freq[i][j]
22                    dp[i][j] = min(dp[i][j], cost)
23
24        return dp[0][n - 1]
```