

PRACTICAL NO. – 06

NAME : RANVEER M. BHORTEKAR

SECTION/BATCH : A4/B1

ROLL NO. : 06

SUBJECT : DAA

GFG PROFILE ID : [Ranveer Mahesh Bhortekar - Ramdeobaba University | GeeksforGeeks Profile](#)

AIM : Construction of OBST

Problem Statement: Smart Library Search Optimization

TASK : 01

Scenerio : 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).

CODE : IN TEXT FORMAT

```
import math
```

```
def OBST(p, q, n):  
    e = [[0 for _ in range(n + 2)] for _ in range(n + 2)]  
    w = [[0 for _ in range(n + 2)] for _ in range(n + 2)]
```

```
# Base initialization
```

```
for i in range(1, n + 2):
```

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

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

OBST DP

```
for l in range(1, n + 1): # l = length of chain
```

```
    for i in range(1, n - l + 2):
```

```
        j = i + l - 1
```

```
        e[i][j] = math.inf
```

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

```
        for r in range(i, j + 1):
```

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

```
            if t < e[i][j]:
```

```
                e[i][j] = t
```

```
return e[1][n]
```

---- Test Input ----

```
n = 4
```

```
keys = [10, 20, 30, 40]
```

```
p = [0.1, 0.2, 0.4, 0.3]
```

```
q = [0.05, 0.1, 0.05, 0.05, 0.1]
```

```
min_cost = OBST(p, q, n)
```

```
print(f"Minimum cost of Optimal Binary Search Tree: {min_cost:.4f}")
```

CODE AND OUTPUT SCREENSHOT :

The screenshot shows a Google Colab notebook titled "DAA Pract 06.ipynb". The code implements the OBST algorithm using dynamic programming. It defines a function `OBST` that takes parameters `p`, `q`, and `n`. The function initializes arrays `e` and `w` with zeros. It then performs base initialization for the first node (`i=1`) by setting `e[i][i-1] = q[i-1]` and `w[i][i-1] = q[i-1]`. The main loop iterates over chain lengths `l` from 1 to `n+1`. For each length, it iterates over all possible splits `i` from 1 to `n-l+2`. For each split, it calculates the cost of merging nodes `j` and `j+1` as `w[i][j] + p[j-1] + q[j]`. It then iterates over all possible insertion points `r` between `i` and `j` (i.e., `i < r < j+1`). For each insertion point, it calculates the total cost `t = e[i][r-1] + e[r+1][j] + w[i][j]`. If `t` is less than the current value in `e[i][j]`, it updates `e[i][j] = t`. Finally, the function returns `e[1][n]`. Below the function, test input values are defined: `n = 4`, `keys = [10, 20, 30, 40]`, `p = [0.1, 0.2, 0.4, 0.3]`, and `q = [0.05, 0.1, 0.05, 0.05, 0.1]`. The minimum cost is printed as 2.9000.

```
import math

def OBST(p, q, n):
    e = [[0 for _ in range(n + 2)] for _ in range(n + 2)]
    w = [[0 for _ in range(n + 2)] for _ in range(n + 2)]

    # Base initialization
    for i in range(1, n + 1):
        e[i][i - 1] = q[i - 1]
        w[i][i - 1] = q[i - 1]

    # OBST DP
    for l in range(1, n + 1): # l = length of chain
        for i in range(1, n - l + 2):
            j = i + l - 1
            e[i][j] = math.inf
            w[i][j] = w[i][j - 1] + p[j - 1] + q[j]
            for r in range(i, j + 1):
                t = e[i][r - 1] + e[r + 1][j] + w[i][j]
                if t < e[i][j]:
                    e[i][j] = t

    return e[1][n]

# ---- Test Input ----
n = 4
keys = [10, 20, 30, 40]
p = [0.1, 0.2, 0.4, 0.3]
q = [0.05, 0.1, 0.05, 0.05, 0.1]

min_cost = OBST(p, q, n)
print(f"Minimum cost of Optimal Binary Search Tree: {min_cost:.4f}")


```

TASK : 02

GFG QUESTION ID : <https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1>

QUESTION SCREENSHOT

Optimal binary search tree | Practi

https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1

Search...

Courses Tutorials

☰ Problem Editorial Submissions Comments

Optimal binary search tree

Difficulty: Hard Accuracy: 50.02% Submissions: 11K+ Points: 8

Given a sorted array **keys[0.. n-1]** of search keys and an array **freq[0.. n-1]** of frequency counts, where freq[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

Let us first define the cost of a BST. The cost of a BST node is level of that node multiplied by its frequency. Level of root is 1.

Example 1:

Input:
n = 2
keys = {10, 12}
freq = {34, 50}

Output: 118

Explanation:
There can be following two possible BSTs

```
    10          12
     \         /
      12        10
```

The cost of tree I is $34*1 + 50*2 = 134$
The cost of tree II is $50*1 + 34*2 = 118$

Optimal binary search tree | Practi

https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1

Search...

Courses ▾ Tutorials ▾

☰ Problem Editorial Submissions Comments

Example 2:

Input:
 $N = 3$
 $\text{keys} = \{10, 12, 20\}$
 $\text{freq} = \{34, 8, 50\}$

Output: 142

Explanation: There can be many possible BSTs

```
    20
     /
    10
     \
     12
```

Among all possible BSTs,
cost of this BST is minimum.
Cost of this BST is $1*50 + 2*34 + 3*8 = 142$

Your Task:
You don't need to read input or print anything. Your task is to complete the function **optimalSearchTree()** which takes the array **keys[], freq[]** and their size **n** as input parameters and returns the total cost of all the searches is as small as possible.

Expected Time Complexity: $O(n^3)$
Expected Auxiliary Space: $O(n^2)$

Constraints:
 $1 \leq N \leq 100$

CODE : IN TEXT FORMAT

class Solution:

```
def optimalSearchTree(self, keys, freq, n):
    cost = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n):
```

```
cost[i][i] = freq[i]
```

```
for L in range(2, n + 1):
```

```
    for i in range(n - L + 1):
```

```
        j = i + L - 1
```

```
        cost[i][j] = float('inf')
```

```
total_freq = sum(freq[i:j + 1])
```

```
for r in range(i, j + 1):
```

```
    c = 0
```

```
    if r > i:
```

```
        c += cost[i][r - 1]
```

```
    if r < j:
```

```
        c += cost[r + 1][j]
```

```
    c += total_freq
```

```
    if c < cost[i][j]:
```

```
        cost[i][j] = c
```

```
return cost[0][n - 1]
```

SUBMISSION SCREENSHOT

Optimal binary search tree | Practice

https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1

Courses Tutorials Practice Jobs

Problem Submissions Comments

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 3 / 3 Accuracy: 100%

Time Taken 1.4

You get marks only for the first correct submission if you solve the problem without viewing the full solution.

Solve Next

Fixing Two nodes of a BST Strictly Increasing Array Word Wrap

Stay Ahead With:

Build 21 Projects in 21 Days

Build real-world ML, Deep Learning & Gen AI projects

Register Now +

Python3 Start Timer

```
1* class Solution:
2*     def optimalSearchTree(self, keys, freq, n):
3*         cost = [[0 for _ in range(n)] for _ in range(n)]
4*
5*         for i in range(n):
6*             cost[i][i] = freq[i]
7*
8*         for L in range(2, n + 1):
9*             for i in range(n - L + 1):
10*                 j = i + L - 1
11*                 cost[i][j] = float('inf')
12*
13*                 total_freq = sum(freq[i:j + 1])
14*
15*                 for r in range(i, j + 1):
16*                     c = 0
17*                     if r > i:
18*                         c += cost[i][r - 1]
19*                     if r < j:
20*                         c += cost[r + 1][j]
21*                     c += total_freq
22*
23*                     if c < cost[i][j]:
24*                         cost[i][j] = c
25*
26*         return cost[0][n - 1]
27*
```

Custom Input Compile & Run Submit