186. Implement the Optimal Binary Search Tree algorithm for the keys A,B,C,D with frequencies 0.1,0.2,0.4,0.3 Write the code using any programming language to construct the OBST for the given keys and frequencies. Execute your code and display the resulting OBST and its cost. Print the cost and root matrix. Input N =4, Keys = {A,B,C,D} Frequencies = {01.02.,0.3,0.4} Output : 1.7

PROGRAM:

```
def optimal_bst(keys, freq, n):
    # Create cost and root tables
    cost = [[0 for x in range(n)] for y in range(n)]
    root = [[0 for x in range(n)] for y in range(n)]

    # Initialize the cost table for single keys
    for i in range(n):
        cost[i][i] = freq[i]
        root[i][i] = i

    # Fill the cost table in a bottom-up manner
    for L in range(2, n+1):  # L is the chain length
        for i in range(n-L+1):
            j = i + L - 1
            cost[i][j] = float('inf')
            total_freq = sum(freq[i:j+1])

            # Try making all keys in interval keys[i..j] as root
            for r in range(i, j+1):
                c = (cost[i][r-1] if r > i else 0) + (cost[r+1][j] if r < j else 0) + total_freq
                if c < cost[i][j]:
                    cost[i][j] = c
                    root[i][j] = r

    # Printing cost table
    print("Cost Table:")
    for row in cost:
        print("\t".join(map(str, row)))

    # Printing root table
    print("\nRoot Table:")
    for row in root:
        print("\t".join(map(str, row)))

    return cost[0][n-1]

# Keys and frequencies
keys = ['A', 'B', 'C', 'D']
freq = [0.1, 0.2, 0.4, 0.3]
n = len(keys)
```

```
# Calculate the cost of the optimal BST
result = optimal_bst(keys, freq, n)
print("\nThe cost of the Optimal Binary Search Tree is:", result)

# Test cases
print("\nTest case 1:")
keys1 = [10, 12]
freq1 = [34, 50]
n1 = len(keys1)
result1 = optimal_bst(keys1, freq1, n1)
print("The cost of the Optimal Binary Search Tree is:", result1)
```

OUTPUT:

```
Cost Table:
0.1 0.4 1.1 1.7
0   0.2 0.8 1.4000000000000001
0   0   0.4 1.0
0   0   0   0.3

Root Table:
0   1   2   2
0   1   2   2
0   0   2   2
0   0   0   3

The cost of the Optimal Binary Search Tree is: 1.7

Test case 1:
Cost Table:
34  118
0   50

Root Table:
0   1
0   1
The cost of the Optimal Binary Search Tree is: 118

=== Code Execution Successful ===
```

TIME COMPLEXITY:O(N^3)