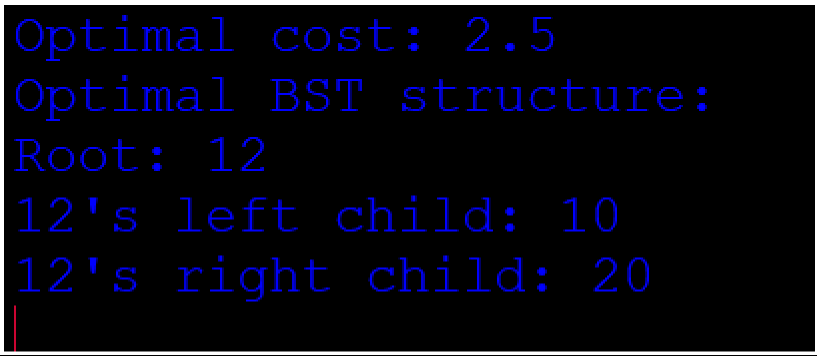
108. Optimal binary search tree

Code:

```
def optimal_bst(keys, p, q, n):
    cost = [[0 for _ in range(n + 1)] for _ in range(n + 1)]
    root = [[0 for _ in range(n + 1)] for _ in range(n + 1)]

    for i in range(n + 1):
        cost[i][i] = q[i]
    for length in range(1, n + 1):
        for i in range(1, n - length + 2):
            j = i + length - 1
            cost[i][j] = float('inf')
            w = sum(p[i-1:j]) + sum(q[i-1:j+1])
            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) + w
                if c < cost[i][j]:
                    cost[i][j] = c
                    root[i][j] = r

    return cost, root
```

```
def print_obst(root, i, j, keys, parent, is_left):
    if i > j:
        return

    r = root[i][j]
    if parent == -1:
        print(f"Root: {keys[r-1]}")
    else:
        if is_left:
            print(f"{keys[parent-1]}'s left child: {keys[r-1]}")
        else:
            print(f"{keys[parent-1]}'s right child: {keys[r-1]}")

    print_obst(root, i, r - 1, keys, r, True)
    print_obst(root, r + 1, j, keys, r, False)

keys = [10, 12, 20]
p = [0.2, 0.3, 0.5]
q = [0.1, 0.1, 0.1, 0.1]
n = len(keys)

cost, root = optimal_bst(keys, p, q, n)
print(f"Optimal cost: {cost[1][n]}")
print("Optimal BST structure:")
print_obst(root, 1, n, keys, -1, False)
```

Output:



```
Optimal cost: 2.5
Optimal BST structure:
Root: 12
12's left child: 10
12's right child: 20
```

Time Complexity:

- T(n)= O(n^3)