CSA0695 DESIGN AND ANALYSIS OF ALGORITHMS FOR OPEN ADDRESSING TECHNIQUES

-CAPSTONE PROJECT

DIFFERENCE BETWEEN MAXIMUM AND MINIMUM PRICE SUM

SUTHESAN VJ

192210542

DIFFERENCE BETWEEN MAXIMUM AND MINIMUM PRICE SUM

-PROBLEM OVERVIEW & OBJECTIVE

Problem Statement:

•You are given a tree with n nodes and n-1 edges.
•Each node has a price.
•The goal is to root the tree at any node, and compute the **difference between the maximum and minimum price sums** for paths starting from the root.
•Find the **maximum cost difference** across all root choices.

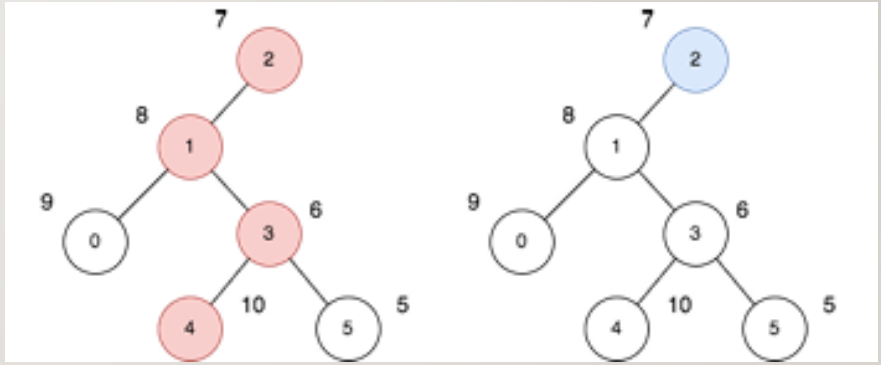
Example:

•Input: n = 6, edges = [[0,1], [1,2], [1,3], [3,4], [3,5]], price = [9,8,7,6,10,5]
•Output: **24**

SOLUTION APPROACH:

Algorithm Overview:

•Use **DFS (Depth-First Search)** to traverse the tree.
•**Adjacency List Representation** for tree structure.
•For each node as the root:
- Calculate **maximum** and **minimum** price sum paths using DFS.
- Compute the cost as the **difference** between these sums.



Objective:

•Maximize the cost difference across all root choices.

CODE IMPLEMENTATION (KEY PARTS):

DFS Traversal:

•The dfs function calculates maximum and minimum sums by visiting child nodes recursively.

```c
void dfs(int node) {
  visited[node] = 1;
  maxSum[node] = price[node];
  minSum[node] = price[node];
  for (int i = 0; i < adjSize[node]; ++i) {
    int child = adj[node][i];
    if (!visited[child]) {
      dfs(child);
      maxSum[node] = max(maxSum[node], maxSum[child] + price[node]);
      minSum[node] = min(minSum[node], minSum[child] + price[node]);
    }
  }
}
```

Cost Calculation:

•Calculate maximum cost for all possible roots using DFS.

COMPLEXITY & CONCLUSION:

Time Complexity:

•Each node and edge is visited once during the DFS traversal.
•**O(n)**, where n is the number of nodes.

Space Complexity:

•**O(n)** due to the adjacency list and recursion stack.

Key Takeaways:

•Efficient DFS approach for computing maximum cost difference.
•Applicable to real-world scenarios like **network optimization** or **resource allocation**.