**Difference Between Maximum and Minimum Price Sum Using Dynamic**

**Programming**

**A PROJECT REPORT**

***Submitted by* S.SIDDHARTHA REDDY REGNO:192210598**

***Under the guidance of***

# MR.RAJAGOPAL

***in partial fulfillment for the completion of course CSA0516-Design and Analysis of algorithms for ROUTING***

***SIMATS ENGINEERING***

# THANDALAM



MARCH 2024

**Title: Difference Between Maximum and Minimum Price Sum Using Dynamic**

**Programming**

**Problem Statement:**

There exists an undirected and initially unrooted tree with n nodes indexed from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree. Each node has an associated price. You are given an integer array price, where price[i] is the price of the ith node. The price sum of a given path is the sum of the prices of all nodes lying on that path.

The tree can be rooted at any node root of your choice. The incurred cost after choosing root is the difference between the maximum and minimum price sum amongst all paths starting at root. Return the maximum possible cost amongst all possible root choices.

* An integer 'n' (number of nodes) and a 2D integer array 'edges' of length n - 1, where

edges[i] = [ai, bi] indicates an edge between nodes ai and bi.

* An integer array 'price' of length n, where price[i] is the price of the ith node.

The tree can be rooted at any node, and the goal is to find the root that maximizes the incurred cost.

# Proposed design work:

This proposed design focuses on creating modular and reusable components.

# Identified Key Components:

* **Node Structure:**Representing a node in the tree with its associated price and children.
* **Tree Representation**:Storing the tree structure using an array or linked list.
* **Path Sum Calculation**:Function to calculate the sum of prices along a given path.
* **Cost Calculation**:Function to calculate the difference between the maximum and minimum path sums.
* **Tree Building**:Constructing the tree based on the given edges.
* **Rooting:**Iterating through each node as a potential root to find the maximum incurred cost..

# Functionality:

* **CalculatePathSum(Node root, int price)**:Calculates the sum of prices along a path starting from the given root node.
* **CalculateCost(Node root, int price):**Computes the difference between the maximum and minimum path sums when rooted at the given node.
* **BuildTree(int n, int edges[][2], int price):**Constructs the tree based on the provided number of nodes, edges, and prices.
* **MaxIncurredCost(int n, int edges[][2], int price**):Iterates through each node as a potential root to find and return the maximum incurred cost.

# Architectural Design:

* **Node Structure:**

struct Node { int value;

struct Node\* children; int childCount;

};

# Path Sum Calculation Function:

int calculatePathSum(struct Node\* root, int\* price);

# Cost Calculation Function:

int calculateCost(struct Node\* root, int\* price);

# Tree Building Function:

struct Node\* buildTree(int n, int edges[][2], int\* price);

# Maximum Incurred Cost Function:

int maxIncurredCost(int n, int edges[][2], int\* price);

# Main Function (Example Usage):

int main() { int n = 4;

int edges[3][2] = {{0, 1}, {1, 2}, {2, 3}};

int price[4] = {1, 2, 3, 4};

struct Node\* tree = buildTree(n, edges, price); int result = maxIncurredCost(n, edges, price); printf("Maximum Incurred Cost: %d\n", result);

// Free allocated memory for the tree

// (Implementation of freeing resources is not shown in this example return 0;

}

# UI Design:

This UI design aims to provide a flexible, user-friendly interface with a well-thought-out layout, feasible elements, and clear functionality. The color selection and positioning of elements contribute to an accessible and visually pleasing experience.

# Layout Design:

* + **Flexible Layout:**
    - Utilize a responsive design to adapt to various screen sizes.
    - Allow for dynamic adjustments based on the number of nodes in the tree.

# User-Friendly:

* + - Use an intuitive layout that guides users through the process of providing input and viewing results.
    - Clearly label input fields and provide informative messages or tooltips.

# Color Selection:

* + - Choose a color scheme that is visually appealing and enhances readability.
    - Use distinct colors for different elements, such as input fields, buttons, and result displays.

# Feasible Elements Used:

* + **Elements Positioning:**
    - Position input fields, buttons, and result displays logically for a smooth user experience.
    - Group related elements together to enhance user understanding.

# Accessibility:

* + - Ensure that all UI elements are accessible, especially for users with disabilities.
    - Use appropriate contrast and text sizes for readability.

# Elements Function:

* + **Input Fields:**
    - Allow users to input the number of nodes, edge connections, and node prices.
    - Validate and provide feedback on user input to prevent errors.

# Buttons:

* + - Include buttons for actions such as calculating the maximum incurred cost.
    - Clearly label each button to convey its purpose.

# Result Display:

* + - Present the maximum incurred cost prominently.
    - Provide clear feedback on the result and its interpretation.

# Feedback/Informational Messages:

* + - Display informative messages for successful actions or potential errors.
    - Clarify the steps users need to take for successful interaction.

# Navigation:

* + - If applicable, include navigation elements to allow users to easily explore different functionalities.

# Login Template:

1. Login Process/Sign Up Process:

Plain text

| Login |

[Username or Email: ] [Password: ]

[ Forgot Password? ]

[ Sign In ]

Don't have an account? [Sign Up]

Sign-Up Template:

Plain text

| Sign Up |

[Full Name: ]

[Email Address: ] [Choose Password:]

[ Sign Up ]

Already have an account? [Login]

# HTML login template followed by a placeholder for the problem statement you provided:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

<style>

body {

font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 0;

display: flex;

align-items: center; justify-content: center; height: 100vh;

}

form {

background-color: #fff; padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); width: 300px;

}

label {

display: block; margin-bottom: 8px;

}

input {

width: 100%; padding: 8px;

margin-bottom: 16px; box-sizing: border-box;

}

button {

background-color: #4caf50; color: #fff;

padding: 10px; border: none; border-radius: 4px; cursor: pointer;

}

</style>

</head>

<body>

<form action="login.php" method="post">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<button type="submit">Login</button>

</form>

</body>

</html>

# Placeholder for the Problem Statement:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Difference Between Maximum and Minimum Price Sum</title>

<style>

body {

}

h1 {

}

font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 0;

padding: 20px;

color: #333;

p {

color: #555;

}

</style>

</head>

<body>

<h1>Difference Between Maximum and Minimum Price Sum</h1>

<p>

There exists an undirected and initially unrooted tree with n nodes indexed from 0 to n - 1.

You are given

the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an

edge between nodes ai and bi in the tree. Each node has an associated price. You are given an integer

array price, where price[i] is the price of the ith node. The price sum of a given path is the sum of the

prices of all nodes lying on that path.

</p>

<p>

The tree can be rooted at any node root of your choice. The incurred cost after choosing root is the

difference between the maximum and minimum price sum amongst all paths starting at root.

Return the

maximum possible cost amongst all possible root choices.

</p>

</body>

</html

# Template other than HTML, here is an example in plain text format:

* + Implement a function that takes inputs n, edges, and price, and returns the maximum possible incurred cost.

# Input:

* + An integer n (number of nodes).
  + A 2D integer array edges of length n - 1, representing edges between nodes.
  + An integer array price of length n, representing the price of each node.

# Output:

* + Return the maximum possible incurred cost.

# Example: Input:

n = 4

edges = [[0, 1], [1, 2], [2, 3]]

price = [1, 2, 3, 4]

# Output:

7

# Explanation:

Choosing node 2 as the root results in the maximum cost. The path sums are:

- Path 1: 2-1 = 3

- Path 2: 2-1-0 = 6

- Path 3: 2-1-0-3 = 10

The difference between the maximum and minimum path sums is 10 - 3 = 7.

# Conclusion:

In conclusion, the problem involves determining the maximum incurred cost in an undirected tree when rooted at various nodes. The goal is to find the root that maximizes the difference between the maximum and minimum price sum along all paths starting at that root. To achieve this, a C program was presented with key components such as node structure, path sum calculation, cost calculation, tree building, and a function to find the maximum incurred cost.

The proposed UI design emphasizes a flexible and user-friendly layout, with feasible elements strategically positioned. It incorporates accessible design principles and clear functionality, providing users with an intuitive interface for inputting tree information and obtaining maximum incurred cost results.

Overall, the combined programming solution and UI design aim to offer an efficient and user- centric experience in solving the problem of finding the maximum possible cost in a rooted tree.

The code provides a working solution for the problem, but there is room for refinement to improve efficiency, error handling, and code readability. Depending on specific requirements and constraints, further adjustments could be made. Overall, the code serves as a solid foundation for solving the stated problem.