1st shift - learningwithram7349@gmail.com

TCS NQT-2025 - 31st march 2025(2nd shift questions analysis)

Q1)Write a program that takes two integer inputs, **input1** and **input2**, representing positions in the sequence of prime numbers. The program should find the **input1-th** and **input2-th** prime numbers, compute their product, subtract 1 from the result, and print the final output.

Example:

Input:

input1 = 2

input2 = 3

2,3,5,7,11

- **Process:**
    - The 2nd prime number is **3**.
    - The 3rd prime number is **5**.
    - Product of these primes: **3 × 5 = 15**.
    - Subtract 1: **15 - 1 = 14**.
- **Output:**

  14

Code:-

```
#include <iostream>
#include <vector>
using namespace std;
```

```
// Function to check if a number is prime
bool isPrime(int num) {
    if (num < 2) return false;
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) return false;
    }
    return true;
}
// Function to find the nth prime number
int getNthPrime(int n) {
    int count = 0, num = 1;
    while (count < n) {
        num++;
        if (isPrime(num)) count++;
    }
    return num;
}


int main() {
    int input1, input2;
    cin >> input1 >> input2;
    int prime1 = getNthPrime(input1);
    int prime2 = getNthPrime(input2);
    int result = (prime1 * prime2) - 1;
    cout << result << endl;
    return 0;
}
```

Q2)**Problem Statement:**

Rani is in a kingdom with **N** towers, where the towers are connected by **N-1** roads, forming a tree-like structure. Each tower has a certain **restoration cost** associated with it. Rani possesses a special **crystal** that allows her to restore a tower. If she restores a tower, its power spreads, and after breaking, it distributes its power to all the connected towers.

Your task is to determine the **maximum number of towers** Rani can restore, given that:

- Restoring a tower initially consumes its given **cost**.
- When a restored tower breaks, its restoration power is **evenly distributed** among its directly connected towers.

Input Format:

- An integer **N** (number of towers).
- A list of **N** integers representing the restoration cost of each tower.
- **N-1** lines, each containing two integers **u** and **v**, representing a road (connection) between tower **u** and tower **v**.

Output Format:

- A single integer representing the maximum number of towers that can be restored.

Example:

Input:

5

4 2 6 3 1

1 2

1 3

2 4

2 5

Explanation:

- There are **5 towers**.

- Their restoration costs are: [4, 2, 6, 3, 1].

The connectivity forms a tree-like structure:

```
  1
 / \
2   3
/ \
4   5
```

- If **Rani restores Tower 2**, its power will be distributed among Towers **1, 4, and 5**.
- The goal is to maximize the number of restored towers.

Output:

3

Code:-

```
/*Code Explanation:

Graph Representation:

The towers and roads are represented as an adjacency list.

The restoration costs of each tower are stored in an array.

Breadth-First Search (BFS):

The BFS function starts from a restored tower and propagates its restoration power to the connected towers.

It keeps track of restored towers using a boolean array.

Simulation for Maximum Towers Restored:

The program tries restoring each tower as the starting point.

The maximum number of restorations across all possible starting towers is determined.

*/
```

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const int MAX_N = 100005;
vector<int> adj[MAX_N]; // Adjacency list
vector<int> cost(MAX_N); // Restoration cost
vector<bool> restored(MAX_N, false); // Track restored towers

int bfs(int start) {
    queue<int> q;
    q.push(start);
    restored[start] = true;
    int savedTowers = 1;

    while (!q.empty()) {
        int tower = q.front();
        q.pop();
        for (int neighbor : adj[tower]) {
            if (!restored[neighbor]) {
                restored[neighbor] = true;
                savedTowers++;
                q.push(neighbor);
            }
        }
    }
    return savedTowers;
}
```

```
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> cost[i]; // Read tower restoration costs
    }

    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int maxSavedTowers = 0;
    for (int i = 1; i <= n; i++) {
        fill(restored.begin(), restored.end(), false); // Reset the restored status
        maxSavedTowers = max(maxSavedTowers, bfs(i));
    }

    cout << maxSavedTowers << endl;
    return 0;
}
```