Tree Money Exchange: Editorial

~ *Omkar*

February 22, 2024

1 Detailed Problem Statement

The problem, *Tree Money Exchange*, involves a tree where nodes represent family members. Given two arrays, one defining parent-child relationships and another listing nodes with initial wealth, the task is to model a wealth redistribution process. Children pass wealth to their parents under specific conditions, aiming to determine which nodes possess money after the redistribution. This scenario explores wealth flow within hierarchical structures, requiring an efficient algorithm to track and update the wealth status of each node.

2 Comprehensive Approach

The solution strategy involves marking nodes with initial wealth, then iteratively redistributing money from children to their first parent without wealth, ensuring no parent receives money more than once. This approach mimics a depth-first search (DFS), propagating wealth up the tree and updating nodes' wealth status dynamically.

3 Detailed Pseudocode

```
Initialize a boolean array 'loaded' to track nodes with money.
Mark initial wealthy nodes in 'loaded' based on 'Arr2'.
For each wealthy node, attempt to pass money to its parent:
  If the parent already has money, stop the redistribution.
  Otherwise, transfer wealth up, marking the child as without money and
      the parent as with money.
Recursively apply this logic to ensure money is passed as far up the
    tree as possible.
After completion, list nodes that ended with money, representing the
    final wealth distribution.
```

Listing 1: Detailed Pseudocode

4 Implementations

4.1 Java Solution

```java
import java.util.ArrayList;
import java.util.Scanner;

public class TreeMoneyExchange {
    static boolean[] loaded;
    static int[] fam, earners;
    static ArrayList<Integer> list;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        loaded = new boolean[n + 1];
        fam = new int[n];

        for (int i = 0; i < n; i++) {
            fam[i] = sc.nextInt();
        }

        int m = sc.nextInt();
        earners = new int[m];
        for (int i = 0; i < m; i++) {
            earners[i] = sc.nextInt();
            loaded[earners[i]] = true;
        }

        list = new ArrayList<>();

        for (int earner : earners) {
            passMoney(earner);
        }

        for (int i = 0; i <= n; i++) {
            if (loaded[i]) {
                System.out.print(i + " ");
            }
        }
    }

    private static void passMoney(int i) {
        if (fam[i] == -1 || loaded[fam[i]]) {
            return;
        }

        loaded[i] = false;
        loaded[fam[i]] = true;
        passMoney(fam[i]);
    }
}
```

Listing 2: Java Implementation

4.2 C++ Solution

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

vector<bool> loaded;
vector<int> fam, earners, list;

void passMoney(int i) {
    if (fam[i] == -1 || loaded[fam[i]]) {
        list.push_back(i);
        return;
    }
    loaded[i] = false;
    loaded[fam[i]] = true;
    passMoney(fam[i]);
}

int main() {
    int n, m;
    cin >> n;
    loaded.resize(n + 1);
    fam.resize(n);
    for (int i = 0; i < n; i++) {
        cin >> fam[i];
    }
    cin >> m;
    earners.resize(m);
    for (int i = 0; i < m; i++) {
        cin >> earners[i];
        loaded[earners[i]] = true;
    }
    for (int i : earners) {
        passMoney(i);
    }
    for (int i = 0; i <= n; i++) {
        if (loaded[i]) {
            cout << i << " ";
        }
    }
}
```

Listing 3: C++ Implementation

4.3 Python Solution

```python
def pass_money(i):
    if fam[i] == -1 or loaded[fam[i]]:
        list.append(i)
        return
    loaded[i] = False
    loaded[fam[i]] = True
    pass_money(fam[i])

n = int(input())
loaded = [False] * (n + 1)
fam = [int(x) for x in input().split()]
m = int(input())
earners = [int(x) for x in input().split()]

for i in earners:
    loaded[i] = True

list = []

for i in earners:
    pass_money(i)

for i in range(n + 1):
    if loaded[i]:
        print(i, end=" ")
```

Listing 4: Python Implementation

4.4 JavaScript Solution

```javascript
let loaded, fam, earners, list;

function passMoney(i) {
    if (fam[i] === -1 || loaded[fam[i]]) {
        list.push(i);
        return;
    }
    loaded[i] = false;
    loaded[fam[i]] = true;
    passMoney(fam[i]);
}

function main(input) {
    let lines = input.split('\n');
    let n = parseInt(lines[0]);
    loaded = new Array(n + 1).fill(false);
    fam = lines[1].split(' ').map(x => parseInt(x));
    let m = parseInt(lines[2]);
    earners = lines[3].split(' ').map(x => parseInt(x));

    earners.forEach(i => { loaded[i] = true; });

    list = [];

    earners.forEach(i => {
        passMoney(i);
    });

    for (let i = 0; i <= n; i++) {
        if (loaded[i]) {
            process.stdout.write(`${i} `);
        }
    }
}

// Example usage with Node.js
// main(`5\n-1 0 0 1 1\n2\n3 4`);
```

Listing 5: JavaScript Code

5 Complexity Analysis Revisited

Considering the DFS-like approach for each initially wealthy node, the time complexity is **O(m * depth of tree)**, where m is the number of nodes starting with money, and $depth - of - tree$ is the maximum depth of the tree. This reflects the worst-case scenario of traversing from leaf to root for each wealthy node. The space complexity remains **O(n)** for tracking nodes' wealth status.