Báo cáo giải bài tập: Cắt giảm Nhân sự bằng Thuật toán Tham lam

> **Nhóm 17 - Bùi Huỳnh Tây - Phạm Ngọc Thọ**

1. Đề bài

Bạn đang làm ở văn phòng quản lý nhân sự ở trong một công ty gồm n người.

Mỗi nhân viên sẽ có chính xác một người là quản lý của mình trừ nhân viên có chỉ số 1.

Người thứ i sẽ có một khối lượng công việc w_i.

Định nghĩa chỉ số phức tạp công việc của người thứ i là $C_i = w_i + \sum_j 1$ trong đó j là nhân viên mà i quản lý.

Bạn được giao nhiệm vụ cắt giảm nhân sự. Bạn có thể chọn một vài người để sa thải khỏi công ty.

- Tất cả nhân viên được quản lý bởi một người bị sa thải sẽ được tiếp nhận bởi quản lý của người đó.
- Đồng thời, khối lượng công việc của người bị sa thải cũng sẽ được chuyển giao cho quản lý của người đó.
- Công ty sẽ không muốn nhân viên của mình phải chịu quá nhiều trách nhiệm nên trong mọi thời điểm, độ phức tạp công việc của mỗi nhân viên không được vượt quá mức m.
- Hiển nhiên, nhân viên có chỉ số 1 là CEO của công ty nên bạn không thể sa thải.

Hãy tìm số lượng người tối đa mà bạn có thể sa thải !!

Input :

- Dòng đầu gồm 2 số nguyên n, m.

- Dòng tiếp theo gồm n số nguyên w_i.
- $n-1$ dòng tiếp theo, dòng thứ i gồm 1 số nguyên p_i là quản lý của nhân viên có chỉ số $i+1$.

Output:

- Gồm một số nguyên là số lượng người tối đa mà bạn có thể sa thải.

Ví dụ:

Input

```
4 6
0 6 2 2
1
1
3
```

Output

```
2
```

---

2. Phân tích và Giải quyết theo Tham lam

1. Mô hình hóa bài toán

- **Đối tượng**: Cấu trúc công ty được mô hình hóa thành một **cây có gốc**, với nhân viên `1` (CEO) là nút gốc.
- **Ràng buộc**: Chỉ số phức tạp công việc C_i của bất kỳ nhân viên `i` nào còn lại không được vượt quá giới hạn m.
- **Hàm mục tiêu**: **Tối đa hóa** tổng số nhân viên bị sa thải.

- Phân tích hành động: Khi sa thải nhân viên i, người quản lý p của họ sẽ gánh toàn bộ "trách nhiệm". Phân tích toán học cho thấy chỉ số phức tạp mới của p sẽ là:

$$C'_p = C_p + C_i - 1$$

Đây là công thức nền tảng cho thuật toán.

2. Xác định Tiêu chí Tham lam

- **Bài toán cục bộ**: Tại một nhân viên quản lý u, làm thế nào để sa thải được nhiều cấp dưới nhất có thể?
- **Phân tích lựa chọn**: Mỗi khi sa thải một cấp dưới v, "ngân sách" phức tạp của u sẽ bị tiêu tốn một lượng là $C_v - 1$. Để thực hiện được nhiều lần sa thải nhất, ta phải tiêu tốn ngân sách một cách tiết kiệm nhất.
- **Quy tắc tham lam (Greedy Criterion)**: Tại một nút quản lý u, ta sẽ luôn **ưu tiên xem xét sa thải những nhân viên cấp dưới v có chỉ số phức tạp C_v nhỏ nhất trước**. Bằng cách này, chúng ta thực hiện các lựa chọn "rẻ" nhất, giữ lại nhiều không gian phức tạp nhất có thể cho các lựa chọn sau.

3. Kiểm chứng tính đúng đắn

- Ta có:
  - Gọi G là lời giải bởi thuật toán tham lam.
  - Gọi O là một lời giải tối ưu bất kỳ.
  - Ta cần chứng minh số lượng sa thải của G và O là bằng nhau.
  - Xét nút u đầu tiên (theo thứ tự duyệt sâu) mà tại đó tập con bị sa thải của G và O khác nhau. Gọi các tập này là S_G và S_O.
- Lập luận Hoán đổi:
  1. Lấy v_i là đứa con có chỉ số nhỏ nhất (tức C thấp nhất) mà có trong S_G nhưng không có trong S_O.
  2. Vì O là tối ưu, để bù lại việc không sa thải v_i, O phải sa thải một đứa con v_j nào đó không có trong S_G.

3. Do cách chọn của G, ta suy ra $j > i$ và $C_{v_j} \geq C_{v_i}$.

4. Ta xây dựng một lời giải mới O' từ O bằng cách hoán đổi: sa thải v_i và không sa thải v_j. Tập sa thải mới là $S_{O'} = (S_O \setminus \{v_j\}) \cup \{v_i\}$.

- Phân tích lời giải mới O':
  - Số lượng sa thải: $|S_{O'}| = |S_O|$. O' vẫn tối ưu về mặt số lượng.
  - Tính hợp lệ: Lượng phức tạp cộng vào C_u trong O' là $\Delta_{O'} = \Delta_O + C_{v_i} - C_{v_j}$.
  - Vì $C_{v_j} \geq C_{v_i}$, ta có $\Delta_{O'} \leq \Delta_O$. Điều này có nghĩa là độ phức tạp cuối cùng của u trong O' nhỏ hơn hoặc bằng trong O. Do O hợp lệ, O' cũng hợp lệ.
- Kết luận chứng minh:

  Ta đã biến đổi một lời giải tối ưu O thành một lời giải tối ưu khác O' mà "giống" với lời giải tham lam G hơn. Bằng cách lặp lại quá trình này, ta có thể biến đổi O thành G mà không giảm số lượng sa thải. Do đó, G phải là một lời giải tối ưu.

4. Xây dựng quá trình tham lam

- **Luồng xử lý**: Xử lý từ lá lên gốc bằng cách duyệt cây theo thứ tự sau (post-order traversal).
- Cấu trúc dữ liệu:
  - vector< int> adj[]: Danh sách kề biểu diễn cây.
  - long long C[]: Mảng lưu chỉ số phức tạp.
  - Thuật toán: Tìm kiếm theo chiều sâu (DFS) để duyệt cây.

Mã nguồn C++ minh họa

```cpp
#include <bits/stdc++.h>

using namespace std;

int n;
long long m;
```

```cpp
vector<int> adj[700000];
long long w[700000];
long long C[700000];
int fired_count = 0;

void dfs(int u) {
    for (int v : adj[u]) {
        dfs(v);
    }

    vector<long long> child_complexities;
    for (int v : adj[u]) {
        child_complexities.push_back(C[v]);
    }

    sort(child_complexities.begin(), child_complexities.end());

    for (long long c_v : child_complexities) {
        if (C[u] + c_v - 1 <= m) {
            C[u] += c_v - 1;
            fired_count++;
        } else {
            break;
        }
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> n >> m;

    for (int i = 1; i <= n; ++i) {
        cin >> w[i];
    }

    for (int i = 2; i <= n; ++i) {
        int p;
        cin >> p;
        adj[p].push_back(i);
    }

    for (int i = 1; i <= n; ++i) {
        C[i] = w[i] + adj[i].size();
    }
```

```
    dfs(1);

    cout << fired_count << endl;

    return 0;
}
```

5. Tính toán độ phức tạp

Đây là phân tích độ phức tạp của thuật toán dựa trên các tham số đầu vào.

- **Tham số đầu vào chính**: N, tổng số nhân viên.

1. Độ phức tạp thời gian (Time Complexity)

Chúng ta sẽ phân tích chi phí của từng phần trong thuật toán:

1. **Khởi tạo và Đọc Dữ liệu**:
    - Đọc N, M: $O(1)$.
    - Đọc N giá trị w_i: $O(N)$.
    - Đọc $N-1$ quan hệ quản lý và xây dựng `adj` : Mỗi thao tác `push_back` tốn $O(1)$ (trung bình). Tổng cộng là $O(N)$.
    - Khởi tạo mảng C ban đầu: Lặp qua N nhân viên, mỗi bước tốn $O(1)$ (để truy cập `adj[i].size()`). Tổng cộng là $O(N)$.
    - *Chi phí khởi tạo:* $O(N)$
2. **Hàm `dfs(u)` (Duyệt theo chiều sâu)**:
    - Hàm `dfs` được gọi chính xác **một lần** cho mỗi nút u (từ 1 đến N).
    - Bên trong `dfs(u)` , gọi k_u là số lượng con trực tiếp của u (tức là `adj[u].size()`).
    - Tạo vector `child_complexities` : Tốn $O(k_u)$.

- **Sắp xếp `child_complexities`** : Đây là thao tác cốt lõi và tốn kém nhất. Tốn $O(k_u \log k_u)$.

- Vòng lặp tham lam (duyệt `child_complexities`): Tốn $O(k_u)$ trong trường hợp xấu nhất.

3. **Tổng độ phức tạp thời gian**:

   - Tổng thời gian của toàn bộ thuật toán là tổng chi phí khởi tạo cộng với tổng chi phí của hàm dfs cho tất cả các nút:

$$T(N) = O(N) \text{ (khởi tạo)} + \sum_{u=1}^{N} \text{Cost}(dfs(u))$$

$$T(N) = O(N) + \sum_{u=1}^{N} (O(k_u) + O(k_u \log k_u))$$

$$T(N) = O(N) + \sum_{u=1}^{N} O(k_u \log k_u)$$

   - Chúng ta có một ràng buộc quan trọng là $\sum_{u=1}^{N} k_u = N - 1$ (vì mỗi nút, trừ nút gốc, là con của đúng một nút khác).

   - Bài toán trở thành tìm chặn trên của $\sum k_u \log k_u$ với điều kiện $\sum k_u = N - 1$.

   - **Trường hợp xấu nhất (Worst Case)**: Xảy ra khi cây có cấu trúc hình sao (star graph), tức là nút gốc 1 có $N - 1$ con (tất cả nhân viên khác báo cáo trực tiếp cho CEO).

   - Trong trường hợp này, tổng $\sum$ trở thành $O((N-1)\log(N-1))$ (chi phí xử lý tại nút gốc) cộng với $\sum_{i=2}^{N} O(0)$ (chi phí xử lý tại các nút lá, vì $k_i = 0$ nên $k_i \log k_i = 0$).

   - Do đó, $\sum_{u=1}^{N} O(k_u \log k_u)$ được chặn trên bởi $O(N \log N)$.

   - **Kết luận**: $T(N) = O(N) + O(N \log N) = O(N \log N)$.

2. Độ phức tạp không gian (Space Complexity)

1. **Lưu trữ cây và dữ liệu**:

- Danh sách kề `adj` : Cần lưu trữ $N-1$ cạnh. Tốn $O(N)$.
- Các mảng `w` và `C` : Mỗi mảng có N phần tử. Tốn $O(N)$.

2. **Không gian phụ trợ (Auxiliary Space)**:
   - Vector `child_complexities` : Trong trường hợp xấu nhất (nút gốc có $N-1$ con), vector này sẽ lưu trữ $N-1$ phần tử. Tốn $O(N)$.
3. **Ngăn xếp đệ quy (Recursion Stack)**:
   - Trong trường hợp xấu nhất (cây suy biến thành một đường thẳng/danh sách liên kết, ví dụ: A quản lý B, B quản lý C,...), độ sâu của đệ quy `dfs` là N. Tốn $O(N)$.

- **Kết luận**: Tổng không gian là $S(N) = O(N) + O(N) + O(N) = O(N)$.

Phương án cài đặt này là rất hiệu quả và đáp ứng được tất cả các ràng buộc của bài toán.