Nhóm 2

1. Mô tả thuật toán

Mỗi nhân viên u có:
- w[u]: tải trọng công việc (weight)
- f[u]: số nhân viên trực tiếp (direct reports)
- Độ phức tạp: C[u] = w[u] + f[u]

Khi sa thải nhân viên u:
- Tải trọng dồn lên cha p_id[u]
- Cập nhật: w[boss] += w[u], f[boss] = f[boss] - 1 + f[u]
=> Làm tăng độ phức tạp của boss lên Δ(u) = C[u] - 1.

2. Ý tưởng & cách tham lam

Mục tiêu: tối đa số nhân viên sa thải mà vẫn thỏa C[u] ≤ LIMIT cho mọi nhân viên được giữ.
Quan sát:
- Mỗi nhân viên bị sa thải sẽ làm tăng độ phức tạp của cha lên Δ(u).
- Cần chọn tập nhân viên để sa thải sao cho tổng Δ(u) ≤ LIMIT - C[boss].
Do đó, tại mỗi nút được giữ, ta nên chọn sa thải những nhân viên có Δ(u) nhỏ nhất trước (để được nhiều người nhất).

Vì sa thải con làm tăng độ phức tạp của cha, nên cần xử lý từ dưới lên (bottom-up).
Sử dụng hàng đợi ưu tiên (priority_queue) để chọn nhân viên sâu nhất và có Δ(u) nhỏ nhất để xử lý trước.

3. Cách cài đặt

- Duyệt DFS để tính độ sâu (dep) và số con trực tiếp (f[u]).
- Đưa tất cả nhân viên (trừ gốc) vào priority_queue với khóa ưu tiên:
  + Ưu tiên nhân viên sâu hơn (dep lớn hơn).
  + Nếu cùng độ sâu, ưu tiên nhân viên có C[u] nhỏ hơn.
- Mỗi bước:
  + Lấy u trên đỉnh PQ (priority_queue tức ta lấy thằng có dep lớn nhất).
  + Nếu cha boss của u có thể gộp u (C[boss] + C[u] - 1 ≤ LIMIT) thì:
    * Gộp u vào boss (sa thải u).
    * Cập nhật w, f, p_id và adj.
    * Đưa boss vào PQ (vì boss có thể thay đổi C[boss]).
  + Nếu không thể gộp u, bỏ qua.
- Kết thúc, biến res chứa số nhân viên bị sa thải tối đa.

4. Chứng minh tính đúng đắn

_ Xoá u làm C[boss] tăng đúng Δ(u) = C[u] - 1.
_ Nếu xoá hậu duệ trước u thì C[u] chỉ tăng, không giảm → tốt nhất xoá u trước.
_ Với mỗi boss, chọn nhân viên có Δ(u) nhỏ nhất trước là tối ưu cục bộ.
_ Nếu C[u] thay đổi (stale) thì bỏ qua u là đúng, vì sau khi dồn tải, u chỉ đắt hơn.
_ Dựa vô các ý trên ta chứng minh được thuật toán tham lam tối ưu

5. Phân tích độ phức tạp

Thời gian:
- Mỗi nút được đưa vào PQ tối đa 2 lần (ban đầu + khi cập nhật cha).
- Mỗi cạnh được xét tối đa 1 lần.
=> Tổng độ phức tạp: O(n log n).

Không gian:
- Lưu trữ cây (adj), mảng w, f, dep, p_id: O(n).
- Hàng đợi ưu tiên chứa O(n) phần tử.
=> Độ phức tạp không gian: O(n).

6. Code demo

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

using namespace std;

#define ll long long
//#define int ll

const int MAXN = 696969+3;

int n,LIMIT;
int p_id[MAXN],w[MAXN],f[MAXN];
vector<int> adj[MAXN];
int dep[MAXN];

struct T {
```

```
     int load;        // C_u hiện tại = w_u + số nhân viên trực tiếp đang hoạt động
     int id;

     bool operator < (const T& other) const {
     if(dep[id]!=dep[other.id]) return dep[id]<dep[other.id]; // sâu hơn -> nhỏ hơn
        if(load!=other.load)  return load>other.load;  // ít phức tạp hơn -> nhỏ hơn
     return id > other.id;
     }
};

// Hàm calc tính chỉ số phức tạp của nhân viên u
int calc(int u){
     if(w[u]<0) return -1;
     return w[u]+f[u];
}

void dfs(int u){
     for (int &v:adj[u]){
     dep[v] = dep[u]+1;
     dfs(v);
     }
}

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

     for(int v = 2; v <= n; ++v){
     int p; cin >> p;
```

```
    p_id[v] = p;
    adj[p].push_back(v);
    }
    for(int i = 1; i<=n; i++) f[i] = adj[i].size();

    dfs(1);

    priority_queue<T> pq;
    for (int u = 2; u <= n; ++u) pq.push({calc(u), u});

    int res = 0;

    while (!pq.empty()) {
    auto [load, u] = pq.top(); pq.pop();

    int curLoad = calc(u);
    if(curLoad!=load || curLoad<0) continue;

    int boss = p_id[u];
    if(w[boss]<0) continue;

    int val = w[boss]+w[u]+f[boss]-1+f[u];

    if(val>LIMIT) continue;
    ++res;
    w[boss] += w[u];
    w[u] = -1;
    f[boss] = f[boss]-1+f[u];
    f[u] = 0;

    for(int &v:adj[u]){
```

```
            p_id[v] = boss;
            adj[boss].push_back(v);
      }
      adj[u].clear();

      if(boss!=1){
            pq.push({calc(boss),boss});
      }
      }

      cout << res;
}

signed main() {
      ios_base::sync_with_stdio(0);
      cin.tie(0); // cout.tie(0);
      solve();
      return 0;
}
```

7. Kết luận

Tóm lại, thuật toán đạt được số nhân viên sa thải tối đa trong O(n log n) với cách cài đặt tham lam hợp lý.