Báo cáo giải bài tập

Nhóm 13

1. Đề bài

Bạn là trưởng phòng nhân sự tại một công ty gồm n cán bộ nhân viên. Mỗi nhân viên có duy nhất một người quản lý trực tiếp, trừ nhân viên thứ nhất (Giám đốc Điều hành) không có cấp trên. Nhân viên thứ i được giao một khối lượng công việc cơ bản là w_i.

Chỉ số tải trọng công việc của nhân viên thứ i được xác định bằng: $C_i = w_i + |\{j : \text{j quản lý bởi i}\}|$, tức là khối lượng công việc cơ bản cộng với số lượng cấp dưới trực tiếp. Ví dụ, nếu nhân viên 5 có khối lượng công việc cơ bản $w_5 = 3$ và quản lý trực tiếp 2 nhân viên (7 và 10), thì $C_5 = 3 + 2 = 5$.

Bạn cần tối ưu hóa quy mô công ty bằng cách loại bỏ một số nhân viên. Khi loại bỏ nhân viên i, tất cả cấp dưới của i sẽ được chuyển giao cho cấp trên của i, và khối lượng công việc cơ bản của i cũng được chuyển sang cho cấp trên đó. Điều này có nghĩa rằng chỉ số tải trọng của cấp trên sẽ tăng lên một lượng là $(C_i - 1)$ (vì chúng ta mất đi một cấp dưới trực tiếp là i, nhưng lại nhận thêm số lượng cấp dưới của i và khối lượng công việc của i).

Ràng buộc: Chỉ số tải trọng C_i của bất kỳ nhân viên nào còn lại đều không được vượt quá mức m. Giám đốc (nhân viên thứ 1) không thể bị loại bỏ vì anh ta là người đứng đầu cả tổ chức.

Yêu cầu: Tìm số lượng nhân viên tối đa có thể bị loại bỏ khỏi công ty đồng thời duy trì tính hợp lệ của hệ thống.

Input:

- Dòng 1: Hai số nguyên n (số lượng nhân viên, $1 \leq n \leq 700000$) và m (ngưỡng tải trọng tối đa, $1 \leq m \leq 10^{18}$)
- Dòng 2: n số nguyên $w_1, w_2, \ldots, w_n$ ($0 \leq w_i \leq 10^9$) biểu diễn khối lượng công việc cơ bản của từng nhân viên
- Dòng 3 đến dòng $n + 1$: Mỗi dòng i (với $i = 2, 3, \ldots, n$) chứa một số nguyên p_i (cấp trên của nhân viên $i + 1$), với $1 \leq p_i \leq n$

Output:

Một số nguyên duy nhất là số lượng nhân viên tối đa có thể bị loại bỏ mà vẫn thỏa mãn toàn bộ ràng buộc của bài toán.

Ví dụ minh họa:

```
Input:
4 6
0 6 2 2
1
1
3

Output:
2

Giải thích: Cấu trúc quản lý là:
  Giám đốc (1: w=0)
   Nhân viên 2 (w=6)
   Nhân viên 3 (w=2)
   Nhân viên 4 (w=2, quản lý nhân viên 5)

Tải trọng ban đầu:
  C[1] = 0 + 3 = 3
  C[2] = 6 + 0 = 6
  C[3] = 2 + 0 = 2
  C[4] = 2 + 1 = 3

Có thể loại bỏ nhân viên 3 và 4 (chi phí 2-1=1 và 3-1=2).
```

2. Phân tích và Giải quyết theo Tham lam

1. Mô hình hóa bài toán

Cấu trúc dữ liệu: Cấu trúc quản lý của công ty được biểu diễn dưới dạng một cây có gốc (rooted tree), trong đó nhân viên 1 (Giám đốc) là nút gốc. Mỗi nhân viên thứ i có duy nhất một cấp trên trực tiếp là p_i (ngoại trừ giám đốc), và có thể quản lý một hoặc nhiều cấp dưới trực tiếp.

Định nghĩa chính thức:

- Đồ thị $G = (V, E)$ là cây có gốc, với $V = \{1, 2, \ldots, n\}$ là tập hợp các nhân viên
- Cạnh (p_i, i) tồn tại khi nhân viên i được quản lý bởi nhân viên p_i
- Nút gốc là nhân viên 1, gọi là $root = 1$

Các ràng buộc:

- **Ràng buộc tải trọng:** Mỗi nhân viên i còn lại phải thỏa mãn $C_i \leq m$
- **Ràng buộc không thể loại bỏ:** Nhân viên 1 (Giám đốc) phải luôn tồn tại trong công ty
- **Tính nhất quán cấu trúc:** Khi loại bỏ nhân viên i, tất cả cấp dưới của i sẽ được chuyển sang cấp trên p_i

Hàm mục tiêu: Tối đa hóa số lượng nhân viên bị loại bỏ, ký hiệu là $|S_{dismissed}|$, sao cho tất cả các nhân viên còn lại đều thỏa mãn ràng buộc tải trọng.

Phân tích hành động loại bỏ: Khi loại bỏ nhân viên i có cấp trên là p, chỉ số tải trọng của p sẽ thay đổi theo công thức:

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

Giải thích:

- C_i là toàn bộ khối lượng công việc và số cấp dưới của i (chuyển sang p)
- -1 vì chúng ta mất đi cấp dưới trực tiếp là i từ danh sách cấp dưới của p

2. Xác định Tiêu chí Tham lam

Bài toán cục bộ: Tại một nhân viên quản lý u (không phải là nhân viên bị loại bỏ), làm sao để loại bỏ được nhiều cấp dưới trực tiếp nhất mà vẫn tuân thủ ràng buộc tải trọng $C_u \leq m$?

Định nghĩa ngân sách: Lượng "không gian tải trọng"mà u có thể sử dụng để loại bỏ cấp dưới được gọi là ngân sách, định nghĩa bằng:

$$Budget_u = m - C_u$$

Khi loại bỏ một cấp dưới v của u, ngân sách sẽ giảm đi một lượng là $C_v - 1$. Nếu $C_v - 1 \leq Budget_u$, chúng ta có thể thực hiện loại bỏ v mà vẫn thỏa mãn ràng buộc.

Phân tích chi phí: Mỗi lần loại bỏ một cấp dưới v, "ngân sách"tải trọng của u bị tiêu tốn một lượng là $(C_v - 1)$. Để thực hiện được nhiều lần loại bỏ liên tiếp, ta phải sử dụng ngân sách một cách hiệu quả và tiết kiệm nhất.

Nguyên tắc Tham lam: Tại nút quản lý u, ta luôn ưu tiên xem xét loại bỏ những cấp dưới v có chỉ số tải trọng C_v **nhỏ nhất trước tiên**. Chiến lược này được gọi là "greedy choice lựa chọn tham lam. Bằng cách ưu tiên những "chi phí"thấp nhất, chúng ta giữ lại nhiều không gian tải trọng nhất có thể cho các lựa chọn loại bỏ tiếp theo, từ đó tối đa hóa tổng số nhân viên bị loại bỏ.

Chứng minh chiến lược tham lam tối ưu: Giả sử ta có một lựa chọn khác không theo chiến lược trên, ví dụ loại bỏ cấp dưới v_j có chi phí $(C_{v_j} - 1)$ cao hơn cấp dưới v_i có chi phí $(C_{v_i} - 1)$ thấp hơn. Nếu ta hoán đổi lựa chọn, ngân sách còn lại sẽ lớn hơn hoặc bằng, từ đó ta có thể loại bỏ được nhiều nhân viên hơn hoặc bằng. Điều này chứng tỏ chiến lược tham lam không kém hiệu quả so với bất kỳ chiến lược nào khác.

3. Kiểm chứng tính đúng đắn

Phát biểu bổ đề:

- Gọi G là lời giải được đưa ra bởi thuật toán tham lam.
- Gọi O là một lời giải tối ưu bất kỳ khác.
- Ta cần chứng minh số lượng nhân viên bị loại bỏ của G và O là bằng nhau hoặc G có nhiều hơn.

Xét nút đầu tiên có sự khác biệt: Sử dụng duyệt sâu (DFS) theo thứ tự sau (post-order), xét nút u đầu tiên mà tập hợp nhân viên bị loại bỏ của G và O khác nhau tại phạm vi quản lý của u. Ký hiệu các tập này là S_G và S_O.

Lập luận Hoán đổi (Exchange Argument):

1. **Bước 1 - Tìm phần tử khác biệt:** Lấy v_i là cấp dưới trực tiếp của u có chỉ số tải trọng nhỏ nhất (tức $C_{v_i} = \min\{C_v : v \in S_G \setminus S_O\}$), tức là nhân viên có chi phí loại bỏ thấp nhất mà có trong S_G nhưng không có trong S_O.

2. **Bước 2 - Phân tích lời giải tối ưu:** Vì O là tối ưu và không loại bỏ v_i, để tối đa hóa số lượng loại bỏ, O phải loại bỏ một cấp dưới v_j nào đó không có trong S_G. Từ cách chọn của G, ta suy ra $C_{v_j} \geq C_{v_i}$.

3. **Bước 3 - Xây dựng lời giải mới:** Ta xây dựng một lời giải mới O' từ O bằng cách hoán đổi: loại bỏ v_i thay vì v_j, và giữ lại v_j. Tập loại bỏ mới là:

$$S_{O'} = (S_O \setminus \{v_j\}) \cup \{v_i\}$$

4. **Bước 4 - Kiểm chứng lời giải mới:**

- **Số lượng loại bỏ:** $|S_{O'}| = |S_O| - 1 + 1 = |S_O|$, vẫn tối ưu về số lượng.
- **Tính hợp lệ của tải trọng:** Lượng tải trọng cộng vào u trong O' là:

$$\Delta_{O'} = \Delta_O - (C_{v_j} - 1) + (C_{v_i} - 1)$$

Vì $C_{v_i} \leq C_{v_j}$, ta có $\Delta_{O'} \leq \Delta_O \leq m$. Điều này có nghĩa tải trọng cuối cùng của u trong O' nhỏ hơn hoặc bằng trong O. Do O hợp lệ (tất cả nhân viên thỏa mãn tải trọng), O' cũng hợp lệ.

5. **Bước 5 - Kết luận chứng minh:** Bằng cách lặp lại quá trình hoán đổi từ nút u lên các nút cha của nó, ta có thể biến đổi O thành G mà không giảm số lượng loại bỏ. Do đó, G phải là một lời giải tối ưu, tức là không tồn tại lời giải nào tốt hơn G.

4. Xây dựng quy trình tham lam

Luồng xử lý toàn cục: Xử lý từ các nhân viên ở mức thấp nhất (lá cây) lên nhân viên ở mức cao hơn (gần gốc) bằng cách duyệt cây theo thứ tự sau (post-order traversal). Cách làm này đảm bảo rằng khi ta xử lý nhân viên u, tất cả cấp dưới trực tiếp của u đã được xử lý xong, nên chỉ số tải trọng cuối cùng của họ đã được xác định chính xác.

Cấu trúc dữ liệu chính:

- `vector<int> adj[700005]`: Danh sách kề biểu diễn cây quản lý. `adj[u]` chứa danh sách tất cả cấp dưới trực tiếp của nhân viên u.
- `long long w[700005]`: Mảng lưu trữ khối lượng công việc cơ bản của mỗi nhân viên.
- `long long C[700005]`: Mảng lưu trữ chỉ số tải trọng của mỗi nhân viên, được cập nhật động khi loại bỏ nhân viên.
- `int dismissed`: Biến đếm tổng số lượng nhân viên bị loại bỏ.

Thuật toán chi tiết: Sử dụng Tìm kiếm theo chiều sâu (DFS - Depth-First Search) để duyệt cây.

Mã nguồn C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int n;
long long m;
vector<int> adj[700005];
long long w[700005];
long long C[700005];
int dismissed = 0;

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

    C[u] = w[u] + adj[u].size();

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

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

    for (long long cost : child_costs) {
        if (C[u] + cost - 1 <= m) {
            C[u] += cost - 1;
            dismissed++;
        } 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 parent;
        cin >> parent;
        adj[parent].push_back(i);
    }

    dfs(1);

    cout << dismissed << endl;

    return 0;
}
```

5. Tính toán độ phức tạp

Phân tích chi tiết Thời gian (Time Complexity): O(N log N)

Độ phức tạp thời gian chủ yếu đến từ phép sắp xếp các cấp dưới tại mỗi nút. Cụ thể:

- **Vòng lặp DFS:** Mỗi nút được thăm duy nhất một lần, tổng cộng $O(n)$
- **Sắp xếp tại mỗi nút:** Tại nút u có d_u cấp dưới trực tiếp, chi phí sắp xếp là $O(d_u \log d_u)$. Tổng chi phí sắp xếp ở tất cả các nút là:

$$\sum_{u=1}^{n} O(d_u \log d_u) \leq \sum_{u=1}^{n} O(d_u \log n) = O(\log n) \sum_{u=1}^{n} d_u = O(n \log n)$$

  (vì $\sum_{u=1}^{n} d_u = n - 1$ là tổng số cạnh trong cây)
- **Vòng lặp tham lam:** Tại mỗi nút, duyệt qua tất cả cấp dưới của nó để kiểm tra xem có thể loại bỏ hay không. Tổng cộng mỗi cạnh chỉ được xét duy nhất một lần, nên độ phức tạp là $O(n)$.

Tính toán tổng:

$$T(n) = O(n) + O(n \log n) + O(n) = O(n \log n)$$

Thành phần $O(n \log n)$ từ phép sắp xếp là dominant, nên độ phức tạp thời gian tổng quát là $O(n \log n)$.

Phân tích chi tiết Không gian (Space Complexity): O(N)

Độ phức tạp không gian bao gồm:

- **Danh sách kề `adj[]`:** Lưu trữ tất cả các cạnh trong cây, tổng cộng $n - 1$ cạnh, nên độ phức tạp là $O(n)$
- **Các mảng dữ liệu:**
  - Mảng `w[]`: $O(n)$ để lưu khối lượng công việc cơ bản
  - Mảng `C[]`: $O(n)$ để lưu chỉ số tải trọng
  - Biến `dismissed`: $O(1)$
- **Ngăn xếp gọi đệ quy (Call Stack):** Độ sâu tối đa của cây là $O(n)$ trong trường hợp xấu nhất (khi cây suy biến thành một chuỗi dài)
- **Biến cục bộ trong hàm DFS:** Vector `child_costs` được tạo tạm thời tại mỗi nút, có kích thước bằng số cấp dưới của nút đó. Tính trung bình, mỗi vị trí bộ nhớ chỉ được sử dụng bởi một vector tại một thời điểm, tổng cộng $O(n)$ bộ nhớ.

Tính toán tổng:

$$S(n) = O(n) + O(n) + O(n) + O(1) + O(n) = O(n)$$

Độ phức tạp không gian là tuyến tính $O(n)$, rất hiệu quả.

Đánh giá hiệu quả thực tế: Với $n = 700000$:

- **Thời gian:**

$$T(700000) = O(700000 \times \log_2(700000)) \approx 700000 \times 20 \approx 1.4 \times 10^7 \text{ phép toán}$$

Ước tính thời gian chạy: khoảng $\mathbf{0.2 - 0.5}$ giây trên máy tính tiêu chuẩn (với tốc độ $\sim 10^8 - 10^9$ phép toán/giây).

- **Không gian:** Mỗi mảng long long chiếm 8 bytes. Tổng không gian:

$$S(700000) \approx 700000 \times 8 \text{ bytes} \times 3 = 16.8 \text{ MB}$$

Cộng thêm không gian cho danh sách kề và stack gọi hàm, tổng cộng khoảng $\mathbf{25 - 30}$ MB, hoàn toàn nằm trong giới hạn $256 - 512$ MB tiêu chuẩn của hệ thống thi đấu lập trình.

Kết luận về hiệu quả: Phương án cài đặt này vô cùng hiệu quả và hoàn toàn đáp ứng được tất cả các ràng buộc về thời gian chạy và bộ nhớ sử dụng của bài toán, cũng như các giới hạn tiêu chuẩn của hệ thống thi đấu lập trình hiện đại.