# CS112 - Bài tập về cách tiếp cận tham lam

Nhóm 12
Hồ Hữu Tây - MSSV: 24520029
Dương Hoàng Việt - MSSV: 24520036

October 21, 2025

## 1 Đề bài:

Bạn đang làm ở văn phòng quản lý nhân sự 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 (CEO).

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 những 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 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 một mức pickle ball $m$.
- Hiển nhiên, nhân viên có chỉ số 1 là CEO của công ty nên 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 $w_1, w_2, \ldots, w_n$.
- $n-1$ dòng tiếp theo, dòng thứ $i$ (với $1 \leq i < n$) 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.

## Giới hạn

- **Subtask 1 (2 điểm):** $n \leq 18$.
- **Subtask 2 (2 điểm):** $n \leq 676767$, $m \leq 369$.
- **Subtask 3 (6 điểm):** Không có ràng buộc.
- Trong mọi bộ test: $n \leq 696969$; $m \leq 36363636$; $0 \leq w_i \leq m$; $p_i \leq i$.

# 2 Mô hình hóa bài toán

Để giải quyết bài toán một cách có hệ thống, chúng ta cần chuyển đổi các đối tượng và quy tắc từ đề bài thành một mô hình toán học cụ thể.

### 2.1 Cấu trúc công ty và các thuộc tính

Cấu trúc quản lý của công ty được mô hình hóa thành một **cây có gốc (rooted tree)**.

- Mỗi nhân viên tương ứng với một **nút (node)** của cây. Nhân viên số 1 là CEO, đóng vai trò là **nút gốc (root)**.
- Mối quan hệ quản lý được biểu diễn bằng một **cạnh có hướng**. Nếu nhân viên $p$ quản lý nhân viên $i$, sẽ có một cạnh đi từ nút $p$ đến nút $i$.
- Mỗi nút $i$ (nhân viên $i$) được gắn với các thuộc tính:
  - Khối lượng công việc ban đầu: $w_i$.
  - Số lượng nhân viên cấp dưới trực tiếp ban đầu: $k_i$.
  - **Chỉ số phức tạp (Complexity Index)** $C_i$: Theo định nghĩa của đề bài, $C_i = w_i + \sum_j 1$ (với $j$ là nhân viên $i$ quản lý). Do đó, ta có thể viết lại chỉ số phức tạp ban đầu của nhân viên $i$ là $C_i = w_i + k_i$.

### 2.2 Phép toán "Sa thải"

Khi một nhân viên $j$ (do $i$ quản lý) bị sa thải, chỉ số phức tạp của người quản lý $i$ sẽ được cập nhật. Giả sử trước khi sa thải, các chỉ số là $C_i = w_i + k_i$ và $C_j = w_j + k_j$. Sau khi sa thải $j$:

- Khối lượng công việc mới của $i$: $w'_i = w_i + w_j$.

- Số cấp dưới mới của $i$: $k'_i = (k_i - 1) + k_j$. (Trừ đi $j$, cộng thêm các cấp dưới của $j$).

Do đó, chỉ số phức tạp mới của $i$ là:

$$\begin{aligned} C'_i &= w'_i + k'_i \\ &= (w_i + w_j) + (k_i - 1 + k_j) \\ &= (w_i + k_i) + (w_j + k_j) - 1 \end{aligned}$$

Từ đó ta có công thức cập nhật cốt lõi:

$$C'_{\text{quản lý}} = C_{\text{quản lý}} + C_{\text{nhân viên bị sa thải}} - 1$$

### 2.3 Mục tiêu và ràng buộc

- **Mục tiêu:** Tối đa hóa tổng số nhân viên bị sa thải.
- **Ràng buộc:** Tại mọi thời điểm, chỉ số phức tạp của bất kỳ nhân viên nào còn lại trong công ty không được vượt quá $m$.
- **Ràng buộc đặc biệt:** CEO (nhân viên 1) không thể bị sa thải.

## 3 Xây dựng thuật toán tham lam (Greedy Approach)

Bài toán có thể được giải quyết hiệu quả bằng cách kết hợp quy hoạch động trên cây và thuật toán tham lam.

### 3.1 Hướng tiếp cận từ dưới lên (Bottom-up)

Chúng ta áp dụng phương pháp duyệt cây theo thứ tự xử lý từ các nút lá đi dần lên nút gốc. Khi xét một nút cha $u$, ta giả định rằng bài toán đã được giải quyết một cách tối ưu cho tất cả các cây con có gốc là con của $u$.

### 3.2 Tiêu chí tham lam

Tại một nút quản lý $u$, để có thể sa thải được nhiều cấp dưới nhất có thể, $u$ cần phải quản lý sự gia tăng chỉ số phức tạp của chính mình. Theo công thức cập nhật, khi sa thải một cấp dưới $v$, chỉ số phức tạp của $u$ tăng thêm một lượng là $C_v - 1$.

**Tiêu chí tham lam:** Tại mỗi nút $u$, để tối đa hóa số lượng người bị sa thải, ta luôn ưu tiên sa thải người cấp dưới $v$ có chỉ số phức tạp tổng hợp $C_v$ là **nhỏ nhất** trước, miễn là việc đó không vi phạm ràng buộc $C_u \leq m$.

### 3.3 Cài đặt thuật toán

1. **Duyệt cây:** Sử dụng thuật toán Tìm kiếm theo chiều sâu (DFS) để duyệt cây theo thứ tự sau.
2. **Lựa chọn tham lam:** Tại mỗi nút $u$, sử dụng một **hàng đợi ưu tiên (min-priority queue)** để lưu và truy xuất các nhân viên cấp dưới $v$ theo thứ tự tăng dần của chỉ số phức tạp $C_v$.
3. **Xử lý tại nút $u$:**
   - **Bước 1:** Gọi đệ quy DFS cho tất cả các con $v$ của $u$ để tính toán tối ưu chỉ số phức tạp $C_v$ của chúng.
   - **Bước 2:** Đưa tất cả các cặp $(C_v, v)$ vào hàng đợi ưu tiên.
   - **Bước 3:** Lần lượt lấy ra nhân viên $v$ có $C_v$ nhỏ nhất từ hàng đợi. Nếu việc sa thải $v$ (tức là $C_u + C_v - 1 \leq m$) là hợp lệ, ta tiến hành cập nhật $C_u$, tăng biến đếm kết quả, và tiếp tục. Nếu không, ta dừng lại vì mọi nhân viên còn lại trong hàng đợi đều có $C$ lớn hơn hoặc bằng.

## 4 Chứng minh tính đúng đắn

Chứng minh chiến lược tham lam là tối ưu.

- Gọi $G$ là lời giải được tạo ra bởi thuật toán tham lam và $O$ là một lời giải tối ưu bất kỳ.
- Xét một nút $u$ bất kỳ. Giả sử tại $u$, thuật toán $G$ quyết định sa thải tập cấp dưới $S_G$, trong khi lời giải $O$ sa thải tập $S_O$.
- Nếu $S_G \neq S_O$, phải tồn tại một nhân viên $v_i \in S_G$ nhưng $v_i \notin S_O$. Vì $|S_O|$ tối ưu, $O$ phải sa thải một nhân viên $v_j$ nào đó mà $G$ không sa thải ($v_j \in S_O$ và $v_j \notin S_G$).
- Theo cách chọn của $G$, ta có $C_{v_i} \leq C_{v_j}$.
- Bây giờ, ta xây dựng một lời giải mới $O'$ từ $O$ bằng: không sa thải $v_j$ mà thay vào đó sa thải $v_i$.
  - Số lượng người bị sa thải trong $O'$ bằng với $O$.
  - Tổng mức tăng chỉ số phức tạp của $u$ trong $O'$ nhỏ hơn hoặc bằng trong $O$, do $C_{v_i} \leq C_{v_j}$.
  - Vì $O$ là một lời giải hợp lệ, $O'$ cũng là một lời giải hợp lệ.
- Bằng cách lặp lại quá trình trao đổi này, chúng ta có thể biến đổi bất kỳ lời giải tối ưu $O$ nào thành lời giải tham lam $G$ mà không làm giảm số lượng người bị sa thải.
- Do đó, lời giải tham lam $G$ cũng là một lời giải tối ưu.

# 5 Kết luận

Lời giải đề xuất là một thuật toán hiệu quả và chính xác, tận dụng cấu trúc cây của bài toán.

- **Độ phức tạp thời gian:** Mỗi nút được duyệt một lần. Mỗi nút được thêm vào và lấy ra khỏi hàng đợi ưu tiên đúng một lần trong toàn bộ thuật toán. Do đó, độ phức tạp tổng thể là $O(N \log N)$.
- **Độ phức tạp không gian:** $O(N)$ để lưu trữ cây, các mảng phụ trợ và không gian cho hàng đợi ưu tiên.

## 5.1 Phân tích Độ phức tạp

### 5.1.1 Độ phức tạp thời gian: $O(N \log N)$

Độ phức tạp thời gian của thuật toán được quyết định bởi hai thành phần chính: quá trình duyệt cây và các thao tác trên hàng đợi ưu tiên (heap).

- **Duyệt cây (DFS):** Hàm `dfs(u)` được gọi chính xác một lần cho mỗi nút $u$ trong cây. Quá trình duyệt này có chi phí cơ bản là $O(N)$, với $N$ là tổng số nút.
- **Thao tác với Heap:** Đây là phần chiếm nhiều thời gian nhất.
  - Mỗi nút trong cây (trừ nút gốc) là con của **duy nhất một** nút cha. Do đó, mỗi nút sẽ được đẩy (`push`) vào heap của nút cha nó **đúng một lần** trong toàn bộ thuật toán.
  - Tương tự, mỗi nút sẽ được lấy ra (`pop`) khỏi heap **tối đa một lần**.
  - Tổng cộng, có $N - 1$ thao tác `push` và tối đa $N - 1$ thao tác `pop`. Chi phí cho mỗi thao tác trên heap có kích thước $k$ là $O(\log k)$. Vì kích thước của heap không bao giờ vượt quá $N$, chi phí cho mỗi thao tác là $O(\log N)$.

Tổng hợp lại, tổng chi phí cho tất cả các thao tác với heap là $(N - 1) \times O(\log N) = O(N \log N)$. Vì các phần khác chỉ tốn $O(N)$, độ phức tạp tổng thể của thuật toán bị chi phối bởi các thao tác heap và là $O(N \log N)$.

### 5.1.2 Độ phức tạp không gian: $O(N)$

Độ phức tạp không gian được tính dựa trên không gian lưu trữ cho cấu trúc dữ liệu và ngăn xếp đệ quy.

- **Lưu trữ cây và dữ liệu:**
  - Danh sách kề `root` để lưu trữ cấu trúc cây tốn $O(N)$ không gian.
  - Các mảng phụ trợ như `w`, `cnt_workload`, `cnt_reports` đều có kích thước $N + 1$, mỗi mảng tốn $O(N)$ không gian.

- **Ngăn xếp đệ quy (Recursion Stack):** Trong trường hợp xấu nhất, cây bị suy biến thành một chuỗi dài, chiều sâu của các lời gọi đệ quy có thể lên tới $N$. Do đó, không gian cho ngăn xếp lời gọi là $O(N)$.
- **Không gian cho Heap:** Tại một nút $u$, heap có thể chứa tất cả các con của $u$. Trong trường hợp cây hình sao (CEO quản lý trực tiếp $N-1$ người), heap có thể chứa tới $O(N)$ phần tử.

Tổng không gian lưu trữ là tổng của các thành phần trên, $O(N) + O(N) + O(N)$, do đó độ phức tạp không gian tổng thể là $O(N)$.

# 6 Code:

```
import sys
import heapq

sys.setrecursionlimit(int(1e6))

def solve():
    try:
        n, m = map(int, sys.stdin.readline().split())

        w = [0] + list(map(int, sys.stdin.readline().split()))

        root = [[] for _ in range(n + 1)]
        for i in range(2, n + 1):
            p = int(sys.stdin.readline())
            root[p].append(i)

    except (IOError, ValueError):
        return

    cnt_workload = list(w)

    cnt_reports = [len(r) for r in root]

    res_container = [0]

    def dfs(u):
        for v in root[u]:
            dfs(v)

        pq = []
        for v in root[u]:
            child_complexity = cnt_workload[v] + cnt_reports[v]
            heapq.heappush(pq, (child_complexity, v))
```

```
        while pq:
            child_complexity, v = heapq.heappop(pq)

            u_complexity = cnt_workload[u] + cnt_reports[u]

            if u_complexity + child_complexity - 1 <= m:
                cnt_workload[u] += cnt_workload[v]
                cnt_reports[u] += cnt_reports[v] - 1 # Cập nhật số cấp dưới

                res_container[0] += 1
            else:
                break

    dfs(1)

    print(res_container[0])

solve()
```