# Giải Thích Thuật Toán

## Mô hình hoá bài toán

Bài toán yêu cầu chúng ta tìm số lượng phòng ban tối đa có thể sáp nhập vào phòng ban mẹ trực tiếp trong một công ty có cấu trúc dạng cây. Mỗi phòng ban $u$ có $a_u$ nhân viên.

Một phép sáp nhập phòng ban $u$ vào phòng ban mẹ $p = \text{par}(u)$ được coi là hợp lệ nếu "chi phí"của phòng ban $p$ sau khi sáp nhập không vượt quá giới hạn $m$.

**Định nghĩa chi phí:** Chi phí của một phòng ban $u$, ký hiệu là $Cost(u)$, được tính bằng tổng số nhân viên của nó và số lượng phòng ban con trực tiếp của nó.

$$Cost(u) = a_u + |\{v \mid \text{par}(v) = u \text{ và } v \text{ chưa bị sáp nhập}\}|$$

Khi sáp nhập $u$ vào $p$:

1. Số nhân viên của $p$ được cập nhật: $a_p \leftarrow a_p + a_u$.
2. Phòng ban $u$ bị loại bỏ.
3. Tất cả các phòng ban con của $u$ sẽ trở thành con trực tiếp của $p$.

Mục tiêu là thực hiện số phép sáp nhập hợp lệ nhiều nhất có thể.

## Xác định tiêu chí tham lam

Thay vì tiếp cận bằng quy hoạch động, ta có thể giải quyết bài toán bằng một chiến lược tham lam. Ý tưởng cốt lõi là tại mỗi bước, ta luôn thực hiện phép sáp nhập "tốt nhất"có thể. Một lựa chọn "tốt nhất"là lựa chọn có chi phí thấp và ít ảnh hưởng đến các khả năng sáp nhập trong tương lai.

Ta xác định thứ tự ưu tiên cho các phép sáp nhập như sau:

1. **Ưu tiên từ dưới lên (theo độ sâu):** Các phòng ban ở sâu hơn trong cây (gần lá) nên được xem xét trước. Khi ta xét một phòng ban $u$, mọi quyết định liên quan đến các cây con của nó đã được hoàn tất. Điều này đảm bảo rằng chi phí $Cost(u)$ ta đang tính là chi phí cuối cùng của nó trước khi được xem xét sáp nhập.
2. **Ưu tiên chi phí thấp:** Với các phòng ban có cùng độ sâu, ta nên ưu tiên sáp nhập phòng ban có $Cost(u)$ nhỏ hơn. Việc này giúp "tiết kiệm"ngân sách $m$ của phòng ban mẹ cho các cuộc sáp nhập tốn kém hơn sau này.

Để thực hiện chiến lược này, **hàng đợi ưu tiên (priority queue)** là một cấu trúc dữ liệu hoàn hảo. Ta sẽ lưu các bộ ba `(độ sâu, chi phí, id_phòng_ban)` và ưu tiên lấy ra các phần tử có độ sâu lớn nhất (giá trị số nhỏ nhất nếu quy ước gốc có độ sâu 0, con có độ sâu âm) và chi phí thấp nhất.

## Chứng Minh Tính Đúng Đắn

Ta sẽ chứng minh chiến lược tham lam trên là tối ưu bằng phương pháp hoán đổi (exchange argument).

Giả sử nghiệm của thuật toán tham lam là $S_G$ và một nghiệm tối ưu bất kỳ là $S_{OPT}$, với $|S_{OPT}| > |S_G|$.

Xét bước đầu tiên mà hai nghiệm này đưa ra quyết định khác nhau. Tại bước này:

- Thuật toán tham lam $G$ chọn sáp nhập phòng ban $u$ vào $p_u = \text{par}(u)$. Theo định nghĩa, $u$ là phòng ban có cặp (`độ sâu, chi phí`) nhỏ nhất lexicographically trong số tất cả các lựa chọn hợp lệ tại thời điểm đó.
- Nghiệm tối ưu $S_{OPT}$ lại chọn sáp nhập một phòng ban $v \neq u$ vào $p_v = \text{par}(v)$ (hoặc không sáp nhập gì cả).

Vì $G$ đã chọn $u$ thay vì $v$, ta chắc chắn có:

$$(\text{h}(u), Cost(u)) \leq_{lex} (\text{h}(v), Cost(v))$$

trong đó $h(u)$ là độ sâu của $u$.

Ta xét một nghiệm mới $S'_{OPT}$ được tạo ra bằng cách sửa đổi $S_{OPT}$: thay vì sáp nhập $v$, ta sáp nhập $u$ tại bước này, và giữ nguyên các quyết định sau đó nếu có thể.

- **Trường hợp 1: Phép sáp nhập $u$ vào $p_u$ hợp lệ trong trạng thái của $S_{OPT}$**
  Vì $Cost(u)$ và các tài nguyên nó tiêu thụ (nhân viên, số con) nhỏ hơn hoặc bằng của $v$, việc sáp nhập $u$ để lại một trạng thái "tốt hơn"hoặc tương đương cho phần còn lại của cây. Ngân sách còn lại của $p_u$ sẽ lớn hơn hoặc bằng so với ngân sách của $p_v$ sau khi sáp nhập $v$. Do đó, mọi phép sáp nhập tiếp theo trong $S_{OPT}$ vẫn có khả năng thực hiện được trong $S'_{OPT}$. Nghiệm $S'_{OPT}$ có số lần sáp nhập không ít hơn $S_{OPT}$.

- **Trường hợp 2: $S_{OPT}$ không sáp nhập gì cả, trong khi $G$ sáp nhập $u$**
  Điều này có nghĩa là mọi phép sáp nhập trong $S_{OPT}$ đều được thực hiện ở các bước sau. Ta có thể xây dựng một nghiệm mới $S'_{OPT}$ bằng cách thực hiện phép sáp nhập $u$ ngay bây giờ. Hành động này không ngăn cản các phép sáp nhập sau này trong $S_{OPT}$ vì nó chỉ ảnh hưởng cục bộ đến $p_u$. Ngược lại, việc giải quyết một cây con nhỏ có thể giải phóng tài nguyên và tạo điều kiện cho các sáp nhập khác.

Bằng cách lặp lại quá trình hoán đổi này, ta có thể dần dần biến đổi nghiệm $S_{OPT}$ thành nghiệm $S_G$ mà không làm giảm tổng số lần sáp nhập. Điều này dẫn đến $|S_G| \geq |S_{OPT}|$. Kết hợp với việc $S_{OPT}$ là tối ưu ($|S_{OPT}| \geq |S_G|$), ta suy ra $|S_G| = |S_{OPT}|$.

Vậy, thuật toán tham lam của chúng ta là đúng đắn.

## Cài Đặt và Độ Phức Tạp

1. **Cài đặt:**

   - Sử dụng DFS để tính độ sâu ban đầu $h(u)$ cho tất cả các nút.
   - Khởi tạo một hàng đợi ưu tiên `min-heap` để lưu các bộ $\{\{h(u), Cost(u)\}, u\}$.
   - Lặp trong khi hàng đợi không rỗng:
     - Lấy ra phòng ban $u$ có ưu tiên cao nhất.
     - **Lazy Update:** Kiểm tra xem chi phí lưu trong hàng đợi có còn khớp với chi phí thực tế `Cal(u)` hay không. Nếu không, bỏ qua vì đây là thông tin cũ.
     - Kiểm tra điều kiện sáp nhập $u$ vào $\text{par}(u)$.
     - Nếu hợp lệ, thực hiện sáp nhập: cập nhật $a_{\text{par}(u)}$, di chuyển con của $u$ sang $\text{par}(u)$, đánh dấu $u$ đã bị xóa (ví dụ $a[u] = -1$), và đẩy trạng thái mới của $\text{par}(u)$ vào hàng đợi ưu tiên.

2. **Độ phức tạp:**

   - DFS để tính độ sâu: $O(N)$.
   - Khởi tạo hàng đợi ưu tiên: $O(N \log N)$.
   - Mỗi phòng ban được sáp nhập thành công đúng một lần. Mỗi lần sáp nhập, phòng ban mẹ của nó có thể được cập nhật và đẩy lại vào hàng đợi. Một cạnh $(u, v)$ trong cây sẽ được "di chuyển"lên cấp trên khi $u$ bị sáp nhập. Tổng số lần di chuyển cạnh trên toàn bộ quá trình là $O(N)$. Do đó, tổng số lần đẩy vào hàng đợi ưu tiên bị giới hạn.
   - Mỗi thao tác trên hàng đợi tốn $O(\log N)$.
   - Độ phức tạp tổng thể của thuật toán là $\Theta(N \log N)$.