LỜI GIẢI ĐỀ LUYỆN 01

(Dành cho đội tuyển quốc gia)

Bài 1. Dãy Spooky

Gọi đồ thị bạn bè là vô hướng trên N đỉnh với M cạnh. Vì quan hệ "bạn bè" là *bắc cầu*, mỗi **thành phần liên thông** tạo thành một nhóm bạn: hai người i, j là bạn khi và chỉ khi chúng thuộc cùng một thành phần.

Tách bài toán theo thành phần.

- Hai thành phần khác nhau **không ràng buộc** nhau về thứ tự.
- Bên trong *một* thành phần, thứ tự người phải **không giảm theo sức mạnh** A.
- Những người có **cùng** sức mạnh trong cùng thành phần có thể hoán đổi tự do.

Số cách trong một thành phần. Xét một thành phần, dựng bảng tần suất theo giá trị sức mạnh: với mỗi x xuất hiện $\mathrm{ct}[x]$ lần. Các phần tử có cùng x hoán đổi được, nên số cách sắp trong thành phần đó là

$$\prod_x (\mathrm{ct}[x])!\,.$$

Ghép các thành phần lại với nhau. Giả sử có k thành phần với kích thước $s_1, s_2, \ldots, s_k$ (đã cố định thứ tự nội bộ từng thành phần như trên). Khi trộn chúng vào một dãy độ dài N, chỉ còn việc chọn *vị trí* cho mỗi thành phần (các phần tử cùng "loại" là không phân biệt). Đây là số hoán vị đa thức:

$$\frac{N!}{s_1!\, s_2! \cdots s_k!}\,.$$

Một cách thấy: chọn $\binom{N}{s_1}$ vị trí cho thành phần 1, rồi $\binom{N-s_1}{s_2}$ cho thành phần 2, …, nhân và rút gọn cho ra công thức trên.

Khi làm việc mod 10^9+7, dùng giai thừa và nghịch đảo giai thừa (Fermat) để tính nhanh.

Thuật toán (DSU/BFS/DFS).

1. Tìm các thành phần liên thông (DSU hoặc BFS/DFS). Lưu kích thước s_t từng thành phần và gom các chỉ số người thuộc cùng thành phần.
2. Với mỗi thành phần C_t, đếm tần suất theo A_i để lấy $\prod_x (\mathrm{ct}_t[x])!$.
3. Tiền xử lý $\mathrm{fact}[0..N]$, $\mathrm{invfact}[0..N]$ mod 10^9+7.
4. Tính tích nội bộ các thành phần, rồi nhân với $\mathrm{fact}[N] \cdot \prod_t \mathrm{invfact}[s_t]$.

Độ phức tạp. Tìm thành phần: $\mathcal{O}(N + M)$. Đếm tần suất trong từng thành phần (tổng qua tất cả là N) và ghép: $\mathcal{O}(N)$. Tiền xử lý giai thừa: $\mathcal{O}(N)$. Tổng thể: $\mathcal{O}(N + M)$ thời gian, $\mathcal{O}(N)$ bộ nhớ.

Bài 2. Alice và cây LCS

Gọi $M = |S|$. Bài toán LCS kinh điển với hai xâu độ dài lần lượt N và M giải bằng quy hoạch động trong $\mathcal{O}(N \cdot M)$. Nếu cây là một *đường thẳng*, ta đúng là đang giải LCS cổ điển. Với cây tổng quát, mỗi đường đi $u \to v$ là ghép của một đoạn **đi lên** rồi một đoạn **đi xuống** qua LCA. Ý tưởng là tách việc so khớp S thành **tiền tố** (cho đoạn đi lên) và **hậu tố** (cho đoạn đi xuống).

Đặt gốc và phân rã đường đi. Đặt gốc cây tại đỉnh 1. Với cặp đỉnh bất kỳ (u, v), đặt $L = \mathrm{LCA}(u, v)$. Khi đó

$$\text{str}(u, v) = (\text{xâu trên đường đi } \textit{lên} \text{ từ } u \text{ tới } L) \ + \ (\text{xâu trên đường đi } \textit{xuống} \text{ từ } L \text{ tới } v).$$

Đoạn đi lên sẽ khớp với *một tiền tố* của S, và đoạn đi xuống khớp với *một hậu tố* của S.

Định nghĩa trạng thái. Ký hiệu $S[1..M]$.

- up$[u][i]$: độ dài LCS tốt nhất giữa *một đường đi bắt đầu trong cây con của u và đi lên tới u* với *tiền tố* $S[1..i]$.
- down$[u][i]$: độ dài LCS tốt nhất giữa *một đường đi bắt đầu ở u và đi xuống trong cây con của u* với *hậu tố* $S[M - i + 1..M]$ (tức "i ký tự cuối" của S).

Chuyển trạng thái cho up. Xét cạnh (u, v) với v là con của u, ký tự trên cạnh là c. Khi gộp đóng góp từ cây con v vào u cho mỗi i $(1 \le i \le M)$:

$$\text{up}[u][i] \leftarrow \max\Big(\text{up}[u][i], \ \underbrace{\text{up}[v][i]}_{\text{bỏ qua cạnh } (u,v)}, \ \underbrace{(S[i] = c) \ ? \ 1 + \text{up}[v][i-1] \ : \ \text{up}[u][i-1]}_{\text{dùng cạnh } (u,v) \text{ để khớp ký tự thứ } i}\Big),$$

trong đó quy ước up$[\cdot][0] = 0$. Diễn giải:

- Nếu không dùng cạnh (u, v) để khớp, ta giữ giá trị từ dưới v: up$[v][i]$.
- Nếu dùng cạnh (u, v) để khớp ký tự, có hai tình huống theo "LCS kinh điển":
  - Nếu $S[i] = c$: khớp c với $S[i]$ và cộng $1 +$ up$[v][i - 1]$.
  - Nếu $S[i] \neq c$: ký tự $S[i]$ không được dùng $\Rightarrow$ đẩy lùi về up$[u][i - 1]$.

Chuyển trạng thái cho down. Tương tự, khi đẩy từ u xuống các con v với ký tự cạnh c, ta so khớp với *i ký tự cuối* của S. Với mỗi i:

$$\text{down}[u][i] \leftarrow \max\Big(\text{down}[u][i], \ \text{down}[v][i], \ (S[M-i+1] = c) \ ? \ 1+\text{down}[v][i-1] \ : \ \text{down}[u][i-1]\Big),$$

với down$[\cdot][0] = 0$.

Kết hợp để cập nhật đáp án. Với một đỉnh chốt u làm LCA, ta ghép *đường đi lên* (khớp tiền tố độ dài i) với *đường đi xuống* (khớp hậu tố độ dài $M - i$):

$$\text{ứng viên đáp án tại } u: \quad \max_{0 \le i \le M} \ \text{up}[u][i] \ + \ \text{down}[u][M - i].$$

Lưu ý quan trọng: đường đi lên và đường đi xuống phải đi qua *hai nhánh con khác nhau* của u (nếu không, LCA sẽ nằm sâu hơn). Kỹ thuật đơn giản là *cập nhật đáp án trong lúc gộp con*: giả sử đang xử lý con mới v của u, tại thời điểm này các giá trị $\text{up}[u][\cdot]$, $\text{down}[u][\cdot]$ đang phản ánh *chỉ* các con đã xử lý trước đó (chưa gồm v). Ta dùng cặp (up từ v, down từ "cũ" của u) và ngược lại để cập nhật đáp án với điều kiện hai nhánh khác nhau; sau đó mới *merge* v vào DP của u.

Độ phức tạp. Mỗi đỉnh có $M+1$ trạng thái cho `up` và `down`; mỗi bước gộp con là $\mathcal{O}(M)$ với hằng số nhỏ theo công thức LCS chuẩn. Tổng thể $\boxed{\mathcal{O}(N \cdot M)}$ thời gian, và $\mathcal{O}(N \cdot M)$ bộ nhớ (có thể tối ưu bộ nhớ theo tầng nếu cần).

Gợi ý cài đặt.

- Duyệt `up`: DFS hậu tố từ lá lên gốc; gộp lần lượt từng con.
- Duyệt `down`: DFS tiền tố từ gốc xuống; gộp tương tự với chỉ số phía cuối S.
- Khi gộp con v vào u, trước khi cập nhật `up`/`down` của u, hãy dùng cặp giá trị "v" với "u-đã-xử-lý-trước" để thử mọi tách i và cập nhật đáp án toàn cục.

Bài 3. Reaper và các ngôi nhà Halloween

Trước hết xét bài toán **không có ràng buộc** ($M = 0$). Nếu ta thăm các nhà theo **thứ tự giảm dần** của A_i thì sẽ *phải bỏ* (không gặt) đúng một nhà trong nhóm có A_i lớn nhất, còn lại đều gặt được. Khi có nhiều nhà cùng A_{max}, ta nên bỏ nhà có B_i **nhỏ nhất** trong nhóm đó (tối ưu).

Khi có ràng buộc C. Ta vẫn muốn đi theo *giảm dần* theo A, nhưng dãy bắt buộc $C_1 \to C_2 \to \cdots \to C_M$ có thể buộc ta *nhảy lên* (từ A nhỏ hơn sang A lớn hơn), và mỗi lần nhảy như vậy sẽ *phải bỏ* một nhà ở phía trước để hạ mức L tối ưu. Định nghĩa **chỉ số xấu** (bad) trong C như sau:

$$C_i \text{ là bad nếu } i = 1 \text{ hoặc } A_{C_{i-1}} > A_{C_i}.$$

Trực giác: mỗi C_i "bad" tiêu tốn 1 "suất bỏ" để không làm mất tối ưu khi buộc phải đi theo chuỗi C.

Tập ứng viên để bỏ. Chỉ những vị trí sau mới có thể/bắt buộc được dùng để "bỏ" tối ưu:

$$\mathcal{P} = \{\text{các } C_i \text{ bad}\} \cup \{\text{các chỉ số } j \notin C\}.$$

Ta sẽ sắp xếp các chỉ số trong $\mathcal{P}$ theo **giảm dần** A_i, và khi hoà:

(i) Ưu tiên *không thuộc* C trước *thuộc* C;

(ii) Nếu còn hoà nữa, ưu tiên B_i *nhỏ hơn* trước.

Thuật toán với cấu trúc dữ liệu hàng đợi ưu tiên (min-heap). Duyệt các chỉ số $i \in \mathcal{P}$ theo thứ tự đã sắp:

- Chèn B_i vào một **min-heap** S (hoặc multiset).
- Nếu i là *bad* (tức $i \in \{C_i \text{ bad}\}$), ta **pop** phần tử *nhỏ nhất* khỏi S và **đánh dấu bỏ** (nhà tương ứng sẽ không gặt).

Lý do: mỗi "suất bỏ" nên trả giá rẻ nhất có thể tại ngưỡng A hiện tại, do đó lấy B nhỏ nhất là tối ưu (đối sánh tham lam theo ngưỡng giảm dần của A).

Bảo đảm phải bỏ một nhà có A_{max}. Như trường hợp $M = 0$, luôn phải bỏ ít nhất *một* nhà trong nhóm A_{max}. Nếu sau khi duyệt xong mà **chưa** bỏ ai trong nhóm A_{max}, ta làm:

- Bỏ nhà có B *nhỏ nhất* trong nhóm A_{max};
- "Hoàn lại" (un-skip) một trong các nhà đã bỏ trước đó có B *lớn nhất* để *giảm thiểu* tổn thất tổng B.

Kết quả & hiện thực. Tổng số hồn gặt được bằng $\sum_i B_i$ trừ đi $\sum_{\text{các nhà bị bỏ}} B_i$. Ta có thể lưu song song danh sách những nhà bị bỏ để thực hiện bước điều chỉnh ở A_{max}.

Độ phức tạp. Sắp xếp $\mathcal{P}$: $O(N \log N)$; mỗi thao tác trên heap S là $O(\log N)$. Tổng cộng $\boxed{O(N \log N)}$.