CODEFORCES DAILY PRACTICE SUMMARY

Date: 12-01-2026
Thien Nhan

F. Jumping Man

Đề bài tóm tắt

Cho cây n nút ($n \le 5000$). Mỗi nút có 1 ký tự.
Từ nút u trong cây con của i, nhảy tùy ý xuống con cháu tạo thành chuỗi S.
Tính $\sum(\text{count}(S))^2$ cho mỗi nút i.

Lời giải chi tiết (Step-by-step)

Ý tưởng: Biến đổi bài toán $\sum count^2$ thành đếm số cặp đường đi (P_1, P_2) có nội dung chuỗi giống hệt nhau.

Bước 1: Chuẩn bị dữ liệu (DFS Order)

- Duyệt DFS để phẳng hóa cây thành mảng.
- Lưu `tin[u]` (thời điểm vào) và `tout[u]` (thời điểm ra).
- Cây con của nút u tương ứng với đoạn chỉ số $[tin[u], tout[u]]$ trên mảng phẳng.

Bước 2: Quy hoạch động (DP) trên ma trận 2D
Ta xây dựng ma trận $DP[N][N]$. $DP[i][j]$ lưu số cặp đường đi giống nhau bắt đầu tại nút có chỉ số DFS là i và nút có chỉ số DFS là j.

- Chúng ta duyệt ngược từ N về 1 (duyệt từ lá lên gốc theo thứ tự DFS) để khi tính nút cha, dữ liệu nút con đã có sẵn.

Bước 3: Xử lý logic tại mỗi cặp (u, v)

Với mọi cặp nút (u, v) trong cây:

1. Nếu ký tự `char[u] != char[v]`:
   - $Ways(u, v) = 0$. Hai nút khác ký tự không thể bắt đầu 2 chuỗi giống nhau.
2. Nếu ký tự `char[u] == char[v]`:
   - $Ways(u, v) = 1 + \sum Ways(child\_u, child\_v)$.
   - Nghĩa là: 1 cặp là chính (u, v), cộng thêm tất cả các cặp đường đi xuất phát từ con cháu của chúng.
   - Trên ma trận DP (đã phẳng hóa), tổng của con cháu chính là tổng hình chữ nhật:
     - Vùng cây con u: $[tin[u] + 1, tout[u]]$
     - Vùng cây con v: $[tin[v] + 1, tout[v]]$
   - Sử dụng **Suffix Sum 2D** để lấy tổng hình chữ nhật này trong $O(1)$.

Bước 4: Tính đáp án

- Sau khi điền xong bảng DP.
- Đáp án cho nút u = Tổng hình vuông trên ma trận giới hạn bởi cây con u với chính nó:
  - Hình vuông: Góc trái trên $(tin[u], tin[u])$, góc phải dưới $(tout[u], tout[u])$.

G. Snake Instructions (Interactive)

Đề bài tóm tắt

n rắn, vị trí a_i, tốc độ $s_i \in \{0, 1, 2\}$ chưa biết.
Dùng tối đa 3 lệnh (L, R) để tìm s_i. Rắn nhanh hơn đâm chết rắn chậm hơn.

Lời giải chi tiết

Sử dụng đúng 3 truy vấn cố định: `? L`, `? LR`, `? R`.

Bước 1: Truy vấn `? L` và `? LR`

- Gọi lệnh `? L`: Nhận về danh sách vị trí P_1 (các rắn sống sót sau khi dồn sang trái).
- Gọi lệnh `? LR`: Nhận về danh sách vị trí P_2.
- **Mấu chốt:** Tập hợp rắn sống sót sau lệnh `L` và lệnh `LR` là **giống hệt nhau**.
  - Lệnh `LR` đưa rắn về đúng vị trí ban đầu của nó (a_i).
  - Vậy, nếu con rắn thứ k trong danh sách P_2 có vị trí là X, thì đó chính là con rắn ban đầu có $a_{idx} = X$.
  - Ta biết được con nào sống, con nào chết.
- **Tính tốc độ rắn sống:**
  - Với rắn sống, ta so sánh vị trí sau lệnh `L` (từ P_1) với vị trí gốc a_i.
  - $s_i = a_i - \text{Vị trí trong } P_1$.

Bước 2: Suy luận rắn đã chết (Dead deduction)

Với những con rắn biến mất sau lệnh `L`, chúng chết do bị con liền kề bên phải đâm vào. Duyệt từ phải sang trái:

- Xét rắn i (chết) và rắn $i+1$ (sống hoặc đã biết tốc độ).
- Khoảng cách $d = a_{i+1} - a_i$.
- Để va chạm xảy ra khi đi sang trái: Rắn $i+1$ phải đuổi kịp rắn i.
  - Nếu $s_{i+1} = 0$: Không thể giết ai (vì nó đứng im). Vô lý (trừ khi rắn i tự đâm vào rắn $i-1$, xét sau).
  - Nếu $s_{i+1} = 1$: Nó chỉ giết được rắn i nếu $s_i = 0$ và khoảng cách d đủ nhỏ (cụ thể $d = 1$).
  - Nếu $s_{i+1} = 2$: Nó giết được $s_i = 0$ hoặc $s_i = 1$.
- Dùng logic này để điền các giá trị s_i còn thiếu.

Bước 3: Truy vấn `? R`

- Thực hiện tương tự bước 1 & 2 nhưng theo hướng phải để tìm nốt các con rắn chưa xác định được từ hướng trái.

Bước 4: Kiểm tra tính đúng đắn

- Sau khi điền hết bảng s. Giả lập lại quá trình chạy. Nếu kết quả mâu thuẫn $\rightarrow$ In `-1`.
- Đặc biệt: Nếu dãy s chứa đoạn `0 1 0` hoặc `0 2 0`, in `-1` ngay (trường hợp không thể phân biệt).

H. Minimise Cost

Đề bài tóm tắt

Chia mảng a thành **đúng** k đoạn con liên tiếp (sau khi sort).
Chi phí = $\sum(\text{độ dài đoạn} \times \text{tổng phần tử đoạn})$.
Tìm chi phí nhỏ nhất.

Lời giải chi tiết

Bước 1: Sắp xếp và Quan sát

- Sort mảng a tăng dần.
- Các số âm cần nhân với hệ số lớn nhất (độ dài lớn nhất) $\to$ Gom tất cả số âm vào đoạn đầu tiên.
- Các số dương cần nhân hệ số nhỏ $\to$ Chia nhỏ ra.

Bước 2: Alien's Trick (WQS Binary Search)

Ta không thể DP trực tiếp $O(N \cdot K)$ vì N, K lớn.

- Bỏ ràng buộc "đúng k đoạn". Thay vào đó, mỗi lần tạo một đoạn mới, ta phải trả phí phạt λ.
- Hàm mục tiêu mới: Tìm cách chia để $MinCost(\lambda) = \text{Tổng chi phí} + \lambda \times \text{Số đoạn}$.
- Ta "chat nhị phân" giá trị λ (từ 0 đến 10^{15}).
  - Nếu với phí λ, số đoạn tối ưu $> k \to$ Tăng phí phạt λ (để giảm số đoạn).
  - Nếu số đoạn tối ưu $< k \to$ Giảm phí phạt λ.

Bước 3: Hàm DP + Monotonic Queue (Deque)

Với một λ cố định, ta cần tính DP nhanh nhất.

- $DP[i]$: Chi phí nhỏ nhất để chia i phần tử đầu tiên.
- Công thức chuyển:

$$DP[i] = \min_{j<i}\{DP[j] + (i-j) \times (Sum[i] - Sum[j]) + \lambda\}$$

- Đặt $Cost(j, i) = (i-j) \times (S_i - S_j)$. Đây là hàm lồi.

- Sử dụng **Deque** để lưu các ứng viên j.
  - Mỗi phần tử trong Deque lưu `(index, start_valid_pos)`.
  - Khi xét i, loại bỏ các ứng viên ở đầu Deque nếu `start_valid_pos` của ứng viên tiếp theo $\le i$.
  - Khi thêm i vào cuối Deque, ta loại bỏ các ứng viên ở cuối nếu i tốt hơn họ ngay từ điểm bắt đầu của họ. Dùng tìm kiếm nhị phân để tìm giao điểm cắt nhau giữa đường cong chi phí của i và ứng viên cuối hàng đợi.

Độ phức tạp: $O(N \log N \cdot \log(\text{Max\_Cost}))$.