Hướng dẫn giải Code Challenge #3

Big-O & VNG - 08/08/2022

BÀI A: NGÀY LẬP TRÌNH VIÊN

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Vì mỗi nội dung thi đấu độc lập nhau, nên ta có thể tính số lượng thí sinh thi đấu của mỗi nội dung, sau đó tìm số trận tối thiểu cho mỗi nội dung thi đấu và tính tổng lại.

Để tính được số lượng thí sinh tham gia mỗi nội dung, ta có thể dùng đếm phân phối với độ phức tạp $O(N)$ hoặc sắp xếp lại rồi đếm số lượng phần tử bằng nhau liên tiếp với $O(NlogN)$.

Với mỗi nội dung, vì số thí sinh trong vòng tiếp theo sẽ bằng số trận đấu ở vòng hiện tại, nên càng ít trận đấu càng tốt, cách tốt nhất để giảm số trận đấu là cho mỗi trận có số thí sinh tối đa (là K).

Khi đó, gọi X là số thí sinh ở vòng hiện tại, số trận đấu ở vòng hiện tại sẽ là:

- X/K nếu $XmodK \leq 1$ (nếu trận cuối chỉ có 0 hoặc 1 thí sinh thì không cần tổ chức)
- $X/K + 1$ nếu $XmodK > 1$

Số thí sinh ở vòng tiếp theo:

- X/K nếu X mod K = 0
- X/K + 1 nếu X mod K > 0

Hay có thể tóm gọn bằng một công thức: (X - 1) / K + 1. Vì sau mỗi vòng thi, số thí sinh giảm K lần nên độ phức tạp là $O(log_k N)$.

Các bước giải

- Bước 1: đọc N, M.
- Bước 2: khởi tạo mảng đếm phân phối cnt: $cnt[i] = 0, \forall 1 \leq i \leq M$.
- Bước 3: đọc từng phần tử x là nội dung thi đấu của từng người, tăng cnt[x] tương ứng lên 1 đơn vị.
- Bước 4: duyệt qua mảng cnt, nếu $cnt[i] > 0$ thì tính số trận cần thi đấu cho nội dung i và lưu vào tổng.
- Bước 5: in kết quả.

Độ phức tạp: $O(N + MlogN)$

Mã nguồn tham khảo

- C++: https://ideone.com/GArtME
- Python: https://ideone.com/ONazZJ
- Java: https://ideone.com/Va83le

BÀI B: NGÀY LẬP TRÌNH VIÊN

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

- Không cần phải biến đổi từng số trong mảng với X cho mỗi truy vấn, chỉ cần giữ 1 biến $diff$ để cộng dồn sự thay đổi sau mỗi truy vấn rồi tính toán trên mảng gốc ban đầu.
- Có thể tính prefix sum để giữ lại tổng của các mảng con trong mảng ban đầu.
- Sắp xếp mảng ban đầu theo thứ tự tăng dần. Đi tìm phân cách của miền kết quả dương và miền kết quả âm sau mỗi truy vấn bằng cách dùng chặt nhị phân đi tìm vị trí pos của giá trị $\geq -diff$ trong mảng ban đầu. Sau đó tính toán giá trị trên 2 miền:
  - Miền âm: $|\sum_{i=1}^{pos-1} a_i + (pos - 1) \times diff|$
  - Miền dương: $\sum_{i=pos}^{N} a_i + (N - pos + 1) \times diff$
- Cộng 2 miền giá trị lại sẽ ra kết quả.

Độ phức tạp: $O(max(NlogN, QlogN))$ với N là số lượng mảng và Q là số truy vấn.

Mã nguồn tham khảo

- C++: https://ideone.com/KWrCb2

BÀI C: TRUEID

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Bản chất của bài toán là đi tìm giá trị lớn nhất của tổng giá trị lớn nhất và nhỏ nhất của một mảng con liên tiếp của mảng a. Nghĩa là với một mảng con liên tiếp $a_i, a_{i+1}, ..., a_j$ ta sẽ tính được giá trị $k = min(a_i, a_{i+1}, ..., a_j) + max(a_i, a_{i+1}, ..., a_j)$ và ta cần tính giá trị k lớn nhất là bao nhiêu.

Nhận xét: giá trị k lớn nhất cần tìm chính là giá trị lớn nhất của tổng hai số liên tiếp trong mảng.

Chứng minh: Gọi k là giá trị lớn nhất cần tìm. Giả sử không tồn tại bất kỳ hai số liên tiếp nào có tổng bằng k, nghĩa là giữa hai số lớn nhất và nhỏ nhất của mảng con thỏa mãn tồn tại những số khác, gọi mảng đó là $a_i, a_{i+1}, ..., a_j$ với a_i là giá trị nhỏ nhất của mảng con này và a_j là giá trị lớn nhất (giả sử ngược lại vẫn đảm bảo tính tổng quát). Vì $a_i = min => a_i \leq a_{j-1}$ $=> k = a_i + a_j \leq a_{j-1} + a_j$ Vì k là giá trị lớn nhất nên $a_{j-1} + a_j = k$, trái với giả thuyết. Suy ra điều giả sử là sai hay có thể kết luận rằng nhận xét ban đầu là đúng.

Từ nhận xét trên ta dễ dàng giải quyết bài toán với độ phức tạp $O(n)$.

Mã nguồn tham khảo

- C++: https://ideone.com/sz6qCF

BÀI D: LEFT OR RIGHT

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Subtask 1

Subtask này có thể giải quyết đơn giản bằng cách sinh hết toàn bộ tập sinh

LR. Duyệt lần lượt từng kí tự dãy S từ trái qua phải, và thử xem kí tự đó sẽ được chèn vào trái hay phải. Sau đó ta kiểm tra các số với B để cập nhật kết quả.

Độ phức tạp: $O(T * 2^n * n)$

Subtask 2
Đầu tiên, ta sẽ duyệt ngược dãy S, khi này, việc xây dựng một số bằng quy tắc tập sinh LR đã được thay đổi thành:

Đã sinh được $left$ kí tự bên trái và $right$ kí tự bên phải, kí tự tiếp theo của dãy S sẽ được chèn vào ngay kề phải của $left$ kí tự đầu tiên hoặc kề trái của $right$ kí tự cuối cùng.

Từ đây, ta có thể thực hiện việc quy hoạch động theo chữ số với cấu hình $cnt[left][right][ok1][ok2]$ là số lượng phần tử của tập LR có thể sinh tiếp khi ta đã chèn $left$ kí tự bên trái và $right$ kí tự bên phải, hai biên $ok1$ và $ok2$ để kiểm soát hai đầu của số đang sinh, đang bé hơn, bằng hay lớn hơn so với phần tương ứng của số B.

Mảng $sum[left][right][ok1][ok2]$ cũng được phát biểu tương tự, chỉ khác là nó lưu tổng các phần tử không vượt quá B.

Khi này, việc chèn thêm kí tự tiếp theo (được xác định bởi $|S|, left, right$) sẽ đưa cần đến hai cấu hình mới là $(left + 1, right, new\ ok1, ok2)$ hoặc $(left, right + 1, ok1, new\ ok2)$, tương ứng là chèn kí tự này vào bên trái hoặc phải của số đang xây dựng.

Gọi số mới chèn vào là $digit$, và vị trí của nó trong số cuối cùng (tức là vị trí của nó sau khi đã chèn hết toàn bộ $|S|$ số) là pos (có thể xác định dựa vào $left$ hoặc $right$, tùy trường hợp), ta có công thức cập nhật mảng quy hoạch động là:

sum_{cấu \ hình \ hiện \ tại} += sum_{cấu \ hình \ mới} + digit * 10^{pos} * cnt_{cấu \ hình \ mới}

Về chi tiết cài đặt, bạn có thể xem trong đoạn code đính kèm.

Độ phức tạp: $O(T * |S| * |S|)$

Mã nguồn tham khảo

- C++: https://ideone.com/f51sjc