OLP khối không chuyên 2019

Bài 1: Cột bò

Link nộp bài: https://oj.vnoi.info/problem/olp_kc19_cow

Giải thích:

Đây là một bài tập chung vào khả năng lập trình của các bạn. Ý tưởng rất đơn giản là với mỗi đống rơm bạn cần tính khoảng cách từ tâm đống rơm đến chỗ cột bò, gọi là d_i. Khi đó độ dài l_i không được lớn hơn giá trị $d_i - r_i$. Có hai trường hợp:

- Nếu $d_i - r_i$ là số nguyên thì $l_i = d_i - r_i - 1$
- Nếu không phải trường hợp trên thì $l_i = d_i - r_i$

Cuối cùng thì $l = min(l_i)$. Tuy nhiên cần chú ý một số điều như sau:

- l là số nguyên nên phải chú ý khi in ra.
- Giá trị l sau khi chuyển sang số nguyên không được lớn hơn giá trị l thập phân nên phải sử dụng hàm floor(l).

Tag: Cài đặt, tính toán cơ bản

Bài 2: Nhân ma trận

Link nộp bài: https://oj.vnoi.info/problem/olp_kc19_mat

Giải thích:

Đây cũng là một bài đánh vào khả năng lập trình đó là khả năng xử lý số lớn của các bạn. Với giới hạn biến như này thì độ phức tạp của nhân ma trận n x n là k x n^3(chắc chắn TLE).

Do đề bài chỉ yêu cầu tính giá trị ở hàng i cột j của ma trận cuối cùng nên ta cần rút ra một số các nhận xét như sau:

- Giả sử t lấy hàng i của ma trận đầu, nếu ta đem nhân với từng cột của ma trận thứ hai thì ta được một ma trận 1 x n chính là hàng thứ i của kết quả trong phép nhân ma trận giữa hai ma trận.
- Nếu thực hiện việc này k lần thì ta sẽ được cột i của ma trận cuối cùng và lúc này chỉ cần lấy phần tử thứ j ra.

- Độ phức tạp lúc này sẽ được giảm xuống còn O(k x n^2) là có thể chạy được với giới hạn bài toán.

Tuy nhiên cần phải chú ý rằng với việc thực hiện rất nhiều các phép nhân và cộng như vậy thì giá trị hoàn toàn có thể vượt quá khả năng lưu trữ. Do vậy bài này sẽ cần cài đặt số lớn trong C++ vì long long là không đủ. Vì vậy lời khuyên của mình là sử dụng python trong trường hợp này.

Tag: big num, nhân ma trận.

Bài 3: Khớp dữ liệu

Link nộp bài: https://oj.vnoi.info/problem/olp_kc19_seq

Bài này là một bài cần cài đặt cấu trúc dữ liệu phức tạp hơn là Segment Tree. Đầu tiên nhận thấy một nhận xét như sau:

$$a\%m = b\%m <=> (a-b)\%m = 0$$

Do đó với mỗi cặp a_i và b_i ta có $c_i = |a_i - b_i|$. Vậy với n cặp c_i đáp án của bài toán là giá trị m lớn nhất sao cho $c_i\%m == 0$. Tức là $m = gcd(c_i)$. Đến đây bạn có thể lập trình để với mỗi cặp (l, r) sẽ tính toán $gcd(c_{i \to j})$ và độ phức tạp sẽ là $\sum j - i$. Tuy nhiên hiển nhiên với giới hạn của bài toán thì cách này sẽ bị TLE. Đây là lý do chúng ta cần sử dụng một segment tree để có thể tính toán $gcd(c_{i \to j})$ với độ phức tạp logn.

Tag: toán học, segment tree

Ngoài segment tree thì cấu trúc Fenwick tree cũng có thể được dùng trong bài này nhé.

Bài 4: Tam giác

Link nộp bài: https://oj.vnoi.info/problem/olp_kc19_tri

Giải thích:

Đối với bài này cách đơn giản nhất là sử dụng kỹ thuật min-max trên đoạn tịnh tiến, các bạn có thể đọc trong bài viết https://wiki.vnoi.info/algo/data-structures/deque-min-maxđể duyệt qua các đoạn k phần tử liên tiếp. Mới mỗi k phần tử liên tiếp xác định giá trị lớn nhất trong k phần tử này.

Chúng ta có một nhận xét rằng để tạo ra một tam giác thì cạnh lớn nhất nhân 2 phải luôn luôn nhỏ hơn hoặc bằng tổng 2 cạnh còn lại. Do đó một tập các cạnh để nối tạo thành một tam giác thì cạnh lớn nhất trong các thanh gỗ phải là một cạnh và không nối dài. Mặt khác cạnh này nhân 2 phải nhỏ hơn hoặc bằng tổng k-1 thanh gỗ còn lại để có thể tạo thành 1 tam giác. Bước này có thể được tính toán bằng cách sử dụng một mảng tiền tố để giảm độ phức tạp cho mỗi test.

Tag: prefix sum, stl