HƯỚNG DẪN GIẢI ĐỀ THI CODE TOUR ONLINE CODE CHALLENGE #1 (TÁI ĐẤU)

A. LỚP HỌC CÔ THÚY

Solution:

C++	https://ideone.com/vEXQJC

Tóm tắt đề:

Cho hai mảng số thực a[n], b[n] lần lượt là bảng điểm trung bình của các bạn nam và các bạn nữ trong lớp theo thứ tự ID từ 1 đến N. Biết mỗi cặp nam nữ được chọn theo quy tắc bạn nam có điểm GPA thấp nhất sẽ được xếp chung nhóm với bạn nữ có điểm GPA cao nhất, bạn nam có điểm GPA thấp nhì sẽ được xếp chung nhóm với bạn nữ có điểm GPA cao nhì,..., bạn nam có điểm GPA cao nhất sẽ được xếp chung nhóm với bạn nữ có điểm GPA thấp nhất. Tìm ID và GPA của bạn nữ chung nhóm với bạn nam có ID k.

Input

- Dòng đầu tiên gồm hai số nguyên N và k.

- Hai dòng tiếp theo: dòng đầu tiên lần lượt là N số thực tương ứng với điểm GPA của N bạn nam có ID từ 1 đến N, tương tự, dòng còn lại là N điểm GPA của các bạn nữ.

Giới hạn:

- $1 \le N \le 10^5$
- $1 \le k \le N$
- $0.0 \le a[i], b[i] \le 10.0$

Output

Một dòng duy nhất gồm ID và GPA (được hiển thị giống như ở input) của bạn nữ tương ứng.

Ví dụ

4 1 8 5.4 2.5 4.9	1 3.8

3.8 9.4 4.2 10	

Giải thích ví dụ:

Từ input ta có các (ID, **GPA**) của các bạn nam lần lượt là {(1, **8**); (2, **5.4**); (3, **2.5**); (4, **4.9**)} và của các bạn nữ lần lượt là {(1, **3.8**); (2, **9.4**); (3, **4.2**); (4, **10**)}. Vậy theo đề bài, các bạn sẽ được sắp xếp theo cặp nam – nữ như sau:

- Nam (3, **2.5**) – Nữ (4, **10**)
- Nam (4, **4.9**) – Nữ (2, **9.4**)
- Nam (2, **5.4**) – Nữ (3, **4.2**)
- Nam (1, **8**) – Nữ (1, **3.8**)

Vậy bạn nam có ID 1 sẽ được xếp chung nhóm với bạn nữ có ID 1 và điểm GPA 3.8.

Hướng dẫn giải:

Ta lần lượt đọc các điểm GPA từ input, đồng thời lưu ID tương ứng vào hai mảng có phần tử là <ID,GPA> cho nam và nữ.

Sau đó sắp xếp lại mảng theo phần tử GPA: với mảng nam, ta sắp xếp theo GPA tăng dần, với mảng nữ, ta sắp xếp theo GPA giảm dần, lưu ý nếu GPA bằng nhau, ta ưu tiên ID nhỏ hơn cho cả hai mảng. Ở đây ta có thể sử dụng hàm sort có sẵn của thư viện (độ phức tạp O(nlogn)). Tuy nhiên để thuận tiện cho việc cài đặt, ta sẽ nhân ID của nữ với -1 trước khi sort để đảm bảo khi sắp xếp giảm dần, bạn nữ có ID thấp hơn vẫn được ưu tiên.

Ta duyệt lần lượt từng phần tử trong mảng nam đã sắp xếp để tìm thứ tự của bạn nam có ID k (O(n)), bạn nữ ở vị trí tương ứng trong mảng nữ đã sắp xếp là bạn nữ cần tìm.

Độ phức tạp: O(NlogN) với N là số học sinh nam/nữ.

B. LỚP HỌC VẼ

Solution:

C++	https://ideone.com/fIXO7d
Java	https://ideone.com/UInK7A
Python	https://ideone.com/ScKTkG

Tóm tắt đề:

Cho N là số đường thẳng cần kẻ. Vẽ một bảng hình chữ nhật gồm nhiều ô, mỗi ô là một hình chữ nhật từ N đường thẳng, sao cho số lượng ô được tạo ra của bảng là lớn nhất.

Input

- Dòng đầu tiên gồm một số nguyên dương T - số lượng test.

- Mỗi test chứa một số nguyên dương N - số lượng đường thẳng.

Giới hạn:

- $1 \le T \le 1000$
- $1 \le N \le 10^9$

Output

Mỗi test in trên một dòng - một số nguyên duy nhất là số lượng ô lớn nhất được tạo ra.

Ví dụ

3 4 5 6	1 2 4

Hướng dẫn giải:

Mục tiêu của bài toán là cần tìm số lượng đơn vị ô lớn nhất có thể.

- Gọi row, column lần lượt là số hàng và số cột có thể tạo ra của bảng. Khi đó, số đơn vị ô của bảng sẽ là row x column.

- Đầu tiên, để kẻ viền ngoài cùng của bảng thì ta cần 4 đường thẳng. Nên số đường thẳng còn lại là M = N - 4. Nếu N < 4 thì kết quả sẽ là 0.

- Cứ x đường kẻ ngang thì sẽ tạo ra được row = x + 1 hàng và cứ y đường kẻ dọc thì sẽ tạo ra được column = y + 1 cột.

- Kết quả bài toán sẽ là max(row x column), tương đương với (x + 1)(y + 1) = xy + x + y + 1 là lớn nhất.

- Do đó, ta cần tìm hai số nguyên x, y sao cho xy lớn nhất và x + y = M

- Theo bất đẳng thức Cauchy thì xy lớn nhất khi và chỉ khi x = y
- Mà x, y nguyên nên ta có:

$$x = \lfloor \frac{M}{2} \rfloor$$

$$y = M - x$$

- Vậy

$$column = M - \lfloor \frac{M}{2} \rfloor + 1$$

$$row = \lfloor \frac{M}{2} \rfloor + 1$$

- Kết quả: row x column**.**

Độ phức tạp: O(T) với T là số lượng test..

C. MÓN QUÀ 20/11

Solution:

C++	https://ideone.com/BPPDGD
Python	https://ideone.com/X5ADHD
Java	https://ideone.com/bLXm6Q

Tóm tắt đề:

Cho một ma trận gồm n hàng, m cột, và một số nguyên T, mỗi ô trên ma trận chứa một số nguyên trong đoạn từ 1 tới n * m, không có 2 ô nào mang giá trị giống nhau. Mỗi lượt chơi được quyền chọn từ mỗi cột một ô, số T sẽ bị giảm đi một lượng không vượt quá tổng trên m ô được chọn. Hãy tìm ra số lượt chơi ít nhất để số T trở về 0, với điều kiện là các cách chọn không được phép trùng nhau.

Input

- Dòng đầu tiên chứa 3 số nguyên n, m và T, tương ứng là số hàng, số cột của phím bấm và số T ($1 \le n, m \le 1000, 0 \le T \le 10^{15}$).

- n dòng tiếp theo, mỗi dòng chứa m số nguyên là ma trận các phím bấm trên ổ khóa ($a_{ij} \le n * m$).

Output

- Gồm một dòng, là số lượt ít nhất cần để mở khóa chiếc rương. Dữ liệu đầu vào đảm bảo luôn tồn tại kết quả và không vượt quá 10^4.

Ví dụ

3 3 24 1 2 3 4 5 6 7 8 9	1

Giải thích ví dụ

Thấy rằng chỉ cần một lượt để chọn 3 ô trong 3 cột lần lượt là 7 8 9 để đưa số T = 24 trở về 0.

Hướng dẫn giải:

Giả sử K là số lượt ít nhất để đưa số T về 0, do đó cách tối ưu nhất là chọn K cách có tổng lớn nhất mà tổng của K cách chọn này lớn hơn hoặc bằng T.

Cách đơn giản nhất để chọn được K phần tử lớn nhất đó là liệt kê hết tất cả các cách chọn, rồi sau đó sắp xếp các cách chọn giảm dần theo tổng. Tiếp theo chỉ cần lựa chọn K cách chọn có tổng lớn nhất.

Độ phức tạp: $O(n^m * m)$ với n, m là kích thước của ma trận.

Để cải tiến cách làm trên, ta thấy là, cách chọn thứ K sẽ là cách chọn có tổng bé nhất trong cả K cách chọn, tạm gọi tổng đó là sum, do đó thay vì liệt kê tất cả n^m cách chọn, ta chỉ cần liệt kê ra tất cả cách chọn có tổng lớn hơn hoặc bằng sum, và sau đó lấy ra K cách chọn lớn nhất là được.

Vấn đề đặt ra là làm sao biết được giá trị sum đó, thì thấy rằng, số lượng cách chọn có tổng không bé hơn sum sẽ tỉ lệ nghịch với sum. Tức là sum càng tăng, thì ta chọn được càng ít cách, nên sẽ cần tìm một giá trị sum lớn nhất mà tổng của cách chọn ≥ sum sẽ lớn hơn hoặc bằng T, do đó có thể áp dụng Binary Search để giải quyết vấn đề này.

Trong quá trình chặt nhị phân, với mỗi giá trị Mid, để biết tổng của các cách chọn ≥ sum có vượt quá T hay không, chỉ cần liệt kê các cách chọn đó ra, tuy nhiên thay vì liệt kê tất cả cách chọn như trên, ta chỉ liệt kê những cách chọn có tổng ≥ sum. Để làm được điều này, ta sẽ thêm một cận vào trong quá trình chọn, giả sử ta đã chọn được i số trong i cột đầu, ta sẽ kiểm tra xem nếu tiếp tục lựa chọn nếu như tiếp tục chọn sẽ được một tổng ≥ sum. Ta có thể kiểm tra bằng cách kiểm tra xem tổng các ô được chọn trong i cột đầu và các ô mang giá trị lớn nhất trong các cột còn lại có lớn hơn hoặc bằng sum hay không.

Để thuận tiện trong quá trình cài đặt, nên sắp xếp các phần tử trong các cột theo giảm dần theo giá trị. Và để kiểm tra việc có nên tiếp tục xây dựng khi chọn được một số cột đầu, có thể sử dụng mảng cộng dồn để tối ưu việc xử lý.

Độ phức tạp: $O(\log(\text{Sum}) * n^m * m)$ với n, m là kích thước của ma trận. Sum là tổng các phần tử lớn nhất trong m cột.

Tuy nhiên thấy là mỗi lần ta sẽ không cần phải liệt kê hết tất cả các cách chọn, và đề cho là kết quả chắc chắn không vượt quá 10^4 nên mỗi lần kiểm tra sẽ không cần liệt kê ra quá nhiều (có thể số lượng cách được liệt kê sẽ lớn hơn 10^4 vì có những cách chọn có tổng bằng nhau, nhưng sẽ không quá nhiều)

D. TRUNDER

Solution:

C++	https://ideone.com/OxzpJh

Có N sự kiện → Có tối đa N phần tử trong hàng đợi.

Việc xóa phần tử trong hàng đợi có thể thực hiện bằng cách đánh dấu phần tử đó đã bị loại (isValid = false). → Vì không trực tiếp xóa phần tử, thay vì xem xét đúng K phần tử cuối, ta xem xét K phần tử valid → Có thể dùng binary search + segment tree ($O(\log(N)^2)$ / query) hoặc binary search on segment tree ($O(\log(N))$ / query) để vừa đếm số lượng valid + vừa lấy kết quả.

Như thế, việc chuyển phần tử tại vị trí X xuống cuối hàng có thể tách thành 2 bước:

1. Xóa phần tử tại vị trí X
2. Thêm một phần tử mới là phần tử X vào hàng đợi.

Cách làm này với truy vấn 3 có thể khiến vị trí của user trong hàng đợi không còn giống đề bài. Có thể xử lý việc này = 1 thao tác binary search để lấy position đúng của user trước khi update trên segment tree.

Note: theo cách làm trên, mảng ta sẽ luôn tăng → để tiện thì dùng curLen ($curLen <= N$) để keep track the size.

Tóm tắt, với mỗi query, ta làm như sau:

- Query loại 1:
    1. Tăng $curLen \mathrel{+}= 1$
    2. Update giá trị $attractiveness$ trong segment tree tại vị trí curLen
- Query loại 2:
    1. Binary search từ cuối về đến khi có K phần tử valid → lấy kết quả
- Query loại 3:
    1. Binary search để biến pos thành $newPos$ - vị trí thật của user cần tìm trong mảng.
    2. Đánh dấu & update đã xóa phần tử tại $newPos$
    3. Thêm phần tử tại $newPos$ vào cuối mảng (tương tự query 1)

Complexity:

- Binary search + Segment tree: $\boldsymbol{O(N * \log(N)^2)}$
- Binary search on segment tree: $\boldsymbol{O(N * log(N))}$