HƯỚNG DẪN GIẢI CODE TOUR 2024
Code Challenge #1

BÀI A: VNGGames de Fan

Solution:

Python	https://ideone.com/zIbvp0
C++	Solution bằng cách dùng hàm toán: https://ideone.com/yyLn0u Solution bằng Binary Search: https://ideone.com/9IYq0N

Tóm tắt đề:

Cho một số N rất lớn, được biểu diễn dưới dạng là tích của các số nguyên tố p_i.

Cho giá trị $C \equiv X^3 \ (mod\ N)$. Tìm giá trị X ban đầu.

Input:

Dòng đầu là một số nguyên $\boldsymbol{K}$ thể hiện số lượng số nguyên tố.

Dòng tiếp theo có $\boldsymbol{K}$ số nguyên tố p_i các nhau bằng khoảng trắng.

Dòng cuối cùng là số $\boldsymbol{C}\ (\boldsymbol{mod}\ \boldsymbol{N})$.

Output:

Gồm một dòng duy nhất chứa số X tìm được.

Ví dụ:

3 1108819 1108823 1108867 1881365963625	12345

Giải thích:

- $N = 1108819 \times 1108823 \times 1108867 = 1363334245757698079$

- Ta thấy được với $X = 12345$ thỏa được $C \equiv X\text{^}3 \ (mod\ N)$.

Hướng dẫn giải:

Đọc kỹ phần ràng buộc input của bài toán, chúng ta thấy được rằng $C < N$.

Như vậy $C \equiv X^3 \ (mod\ N) \leftrightarrow C = X^3$.

Chúng ta chỉ cần tính giá trị căn bậc 3 của C là được, giá trị của N không quan trọng trong bài toán này.

BÀI B: SWIMMING

Solution:

C++	https://www.ideone.com/Sg6KOC

Tóm tắt đề:

Cho hồ bơi dài N , mỗi sải bài dài K, khi thực hiện sải bơi thứ i thì độ mệt mỏi cộng thêm là i. Tính tổng độ mệt mỏi sau khi bơi được đến đích.

Input:

- Một dòng duy nhất chứa 2 số nguyên dương N và K

Điều kiện:

- $1 \leq N \leq 10^9$
- $1 \leq K \leq N$

Output:

Một dòng duy nhất là tổng độ mệt mỏi sau khi bơi đến đích

Ví dụ:

10 3	10

Hướng dẫn giải:

Ta nhận thấy rằng chỉ cần biết được số sải bơi cần thiết để có thể bơi đến đích thì mới có thể tính được tổng độ mệt mỏi, nên ta chỉ cần đi qua 2 bước:

Bước 1: Tính số sải bơi $x = \left\lfloor \frac{N}{K} \right\rfloor + ((N \ mod \ K) \ \neq 0 \ )$

Bước 2: Tính tổng độ mệt mỏi $ans = \frac{x*(x+1)}{2}$

Độ phức tạp: O(1)

BÀI C: Xếp hàng

Solution:

C+	https://ideone.com/Vttg6T

Tóm tắt đề:

Cho một mảng ***A*** gồm ***N*** phần tử được sắp xếp tăng dần. Bài toán gồm ***Q*** truy vấn, mỗi truy vấn gồm một số nguyên dương ***X***.

Yêu cầu: Với mỗi truy vấn, cần xếp ***X*** vào mảng ***A*** sao cho các số lớn hơn thì được xếp sau nó, và cho biết vị trí của ***X*** trong mảng ***A*** sau khi được xếp vào.

Input:

Dòng đầu gồm hai số nguyên dương ***N***, ***Q*** ($1 \le N, Q \le 5 \cdot 10^5$) - thể hiện số phần tử ban đầu và số phần tử được xếp vào sau

Dòng thứ hai gồm ***N*** số nguyên dương $A_1, A_2, ..., A_N$ ($1 \le A_i \le 10^9$)

Q dòng tiếp theo, mỗi dòng gồm một số nguyên ***X*** ($1 \le X \le 10^9$) - giá trị của số cần xếp vào ***A***

Output:

Gồm Q dòng, mỗi dòng gồm một số nguyên là vị trí của ***X*** sau khi xếp vào ***A***.

Ví dụ:

8 5 2 3 3 5 5 7 7 9 9 5 2 5 3	9 6 2 8 5

Giải thích:

Mảng ban đầu như sau: 2 3 3 5 5 7 7 9

Lần lượt các các số được thêm vào là:

- Xếp lần thứ nhất: 2 3 3 5 5 7 7 9 **9**. Vị trí thứ 9.
- Xếp lần thứ hai: 2 3 3 5 5 **5** 7 7 9 9. Vị trí thứ 6.
- Xếp lần thứ ba: 2 **2** 3 3 5 5 5 7 7 9 9. Vị trí thứ 2.
- Xếp lần thứ tư: 2 2 3 3 5 5 5 **5** 7 7 9 9. Vị trí thứ 8.

- Xếp lần thứ năm: 2 2 3 3 **3** 5 5 5 5 7 7 9 9. Vị trí thứ 5.

Hướng dẫn giải:

Do mảng ban đầu đã được sắp xếp theo thứ tự tăng dần, với mỗi truy vấn ta thực hiện tìm vị trí thỏa điều kiện đề bài của ***X*** (có thể dùng tìm kiếm nhị phân), sau đó thực hiện chèn ***X*** vào mảng.

Tuy nhiên, do thao tác chèn trên mảng có thể lên đến độ phức tạp *O*(kích thước mảng ***A*** hiện tại)

Nhận xét rằng ta chỉ cần quan tâm vị trí của ***X***, không cần thiết phải chèn trực tiếp vào mảng ***A***.

Để tìm vị trí mà không cần phải chèn trực tiếp vào ***A***, ta có thể tạo một mảng *f* thể hiện số lần xuất hiện của các giá trị trong ***A***. Khi đó vị trí của X là tổng từ *f[1]* đến *f[X]*

Có thể dùng tổng tiền tố để lấy tổng nhanh, nhưng mỗi khi thêm ***X*** vào ***A*** ta cần cập nhật lại toàn bộ mảng *f*. Để tối ưu thao tác cập nhật, có thể sử dụng Segment Tree hoặc Fenwick Tree

Ngoài ra, vì ta không cần quan tâm giá trị của các số trong mảng A, chỉ cần giữ được tính chất thứ tự giữa các số là được. Do đó ta thực hiện nén các số lại để dễ dàng thực hiện truy xuất trên mảng.

Độ phức tạp: ***O((N + Q)log(N + Q))***.

BÀI D: HOMEWORK

Solution:

C++	https://ideone.com/yG99cE

Tóm tắt đề:

Có n công việc được chia cho 2 người và được mô tả như sau:

- n, k, d: lần lượt là số bài, giới hạn số loại và số lượng bài được nghỉ liên tiếp nhiều nhất.
- $t[i]$, loại của bài thứ i.
- $a[i]$ nếu bài loại i không được giải liền trước.
- $b[i]$ nếu bài loại i đã được giải liền trước.

Yêu cầu in ra thời gian ngắn nhất để 2 người hoàn thành công việc.

Input:

- Dòng đầu tiên chứa số nguyên dương T là số lượng testcase.
- Đối với mỗi testcase:

  - Dòng đầu tiên là số nguyên dương n, k, d.

  - 3 dòng tiếp theo là mảng số nguyên $t[i], a[i], b[i]$.

Điều kiện:

- $1 \leq T \leq 10$
- $1 \leq n, d, k \leq 5000$
- $1 \leq t_i \leq k$
- $1 \leq a_i \leq 10^9$
- $1 \leq b_i \leq a_i$

Output:

Mỗi dòng in ra một số nguyên dương là thời gian ít nhất để hoàn thành công việc.

Ví dụ:

1 4 3 4 1 3 1 3 4 7 8 2 7 7	21

Giải thích test:

2 người A và B sẽ làm thứ tự như sau: $ABAB$

Thời gian để hoàn thành là: $4 + 8 + 2 + 7 = 21$.

Hướng dẫn giải:

Sub1 $n \leq 20$:

Vì n bé nên chúng ta duyệt tất cả những cách chia việc có thể xảy ra và tìm ra thời gian hoàn thành nhỏ nhất.

Độ phức tạp: $\mathbf{O(2^n)}$

Sub2 $d = n$:

Ý tưởng đầu tiên ta có thể gọi: $dp[i][n_1][n_2]$ là thời gian ngắn nhất để hoàn thành i công việc đầu tiên và công việc gần nhất mà người 1 và người 2 hoàn thành là: n_1, n_2. Ở đây, ta có một nhận xét rằng, n_1 và n_2 luôn có một biến có giá trị bằng i nên ta không cần quản lý chiều i. Như vậy, công thức quy hoạch động cho bài này là: $dp[i][j]$ là thời gian ngắn nhất để hoàn thành i công việc đầu với người làm gần nhất hoàn thành công việc i và người còn lại hoàn thành công việc j. Ngoài ra ta có một nhận xét $d = n$ nên chúng ta không cần kiểm tra điều kiện: không có người nào nghỉ quá d công việc.

Ta có công thức chuyển trang thái như sau:

- Trường hợp người làm công việc i tiếp tục làm công việc $i + 1$:
  - Nếu $t[i] = t[i+1]$: $dp[i][j] + b[i] \rightarrow dp[i+1][j]$
  - Nếu $t[i] \neq t[i+1]$: $dp[i][j] + a[i] \rightarrow dp[i+1][j]$
- Trường hợp người làm công việc j làm công việc $i + 1$:
  - Nếu $t[i] = t[i+1]$: $dp[i][j] + b[i] \rightarrow dp[i+1][i]$
  - Nếu $t[i] \neq t[i+1]$: $dp[i][j] + a[i] \rightarrow dp[i+1][i]$

Độ phức tạp: **O(T*N*N)**

***Sub3*:**

Nếu chúng ta đã làm được sub2 thì sub này khá đơn giản, chúng ta phải quản lý thêm điều kiện của d. Trong công thức quy hoạch động của chúng ta như sub2, có thể thấy:

- Trường hợp 1: khi chuyển từ trạng thái $(i, j) \rightarrow (i + 1, j)$ ta phải kiểm tra $j \geq i + 1 - d$.
- Trường hợp 2: dễ thấy luôn thỏa điều kiện d.

Do đó, công thức chuyển trạng thái cho bài này là:

- Trường hợp người làm công việc i tiếp tục làm công việc $i+1$. Nếu thỏa điều kiện $j \geq i+1-d$:
  - Nếu $t[i] = t[i+1]: dp[i][j] + b[i] \rightarrow dp[i+1][j]$
  - Nếu $t[i] \neq t[i+1]: dp[i][j] + a[i] \rightarrow dp[i+1][j]$
- Trường hợp người làm công việc j làm công việc $i+1$:
  - Nếu $t[i] = t[i+1]: dp[i][j] + b[i] \rightarrow dp[i+1][i]$
  - Nếu $t[i] \neq t[i+1]: dp[i][j] + a[i] \rightarrow dp[i+1][i]$

Độ phức tạp: **O(T*N*N)**

BÀI E: EATING

Solution:

C++	https://ideone.com/D2vYDp

Tóm tắt đề:

Đề bài gồm nhiều testcase. Ở mỗi testcase, cho mảng A các số nguyên dương gồm N phần tử. Xác định xem liệu có tồn tại một vị trí i $(i \leq N)$ thỏa mãn 3 yêu cầu:

- Điều kiện 1: $A_i = 1$

- Điều kiện 2: Với mọi j $(i < j \leq N)$:

$$\sum_{i \leq x < j} A_x + 1 \geq A_j$$

- Điều kiện 3: Với mọi j $(1 \leq j < i)$:

$$\sum_{i \leq x \leq N} A_x + \sum_{j < y < i} A_y + 1 \geq A_j$$

Input:

- Dòng đầu tiên chứa số nguyên dương T là số lượng testcase.
- Đối với mỗi testcase:

  - Dòng đầu tiên là số nguyên dương N.

  - Dòng tiếp theo chứa N số nguyên dương A_i.

Điều kiện:

- $1 \leq T \leq 10$
- $1 \leq N \leq 10^5$
- $1 \leq A_i \leq 10^{13}$

Output:

Một dòng duy nhất là "YES" nếu tồn tại cách chọn thỏa mãn, "NO" nếu không.

Ví dụ:

2 5 10 7 1 2 4 5 10 1 1 2 4	YES NO

Hướng dẫn giải:

Biến đổi điều kiện 2, ta có:

$$\sum_{i \le x < j} A_x + 1 \ge A_j \ \ \& \ \sum_{i \le x < j-1} A_x + 1 \ge A_{j-1}$$

$$\rightarrow \sum_{i \le x < j-1} A_x + 1 \ge A_j - A_{j-1} \ \ \& \ \sum_{i \le x < j-1} A_x + 1 \ge A_{j-1}$$

$$\rightarrow \sum_{i \le x < j-1} A_x + 1 \ge \max(A_j - A_{j-1}, A_{j-1})$$

$$\rightarrow A_i + 1 \ge \max\left(A_{i+1}, A_{i+2} - A_{i+1}, A_{i+3} - A_{i+1} - A_{i+2}, \ldots, A_N - \sum_{i < x < N} A_x\right)$$

Gọi $MAX_i = \max(A_i, A_{i+1} - A_i, A_{i+2} - A_i - A_{i+1}, \ldots, A_N - \sum_{i<x<N} A_x)$

➔ $MAX_i = \max(A_i, MAX_{i+1})$

➔ Vị trí i được xem là thỏa mãn điều kiện 1 và 2 khi và chỉ khi:

$$A_i + 1 \ge MAX_{i+1} \ \& \ A_i = 1$$

Hay

$$MAX_i = 1$$

Ta còn nhận thấy rằng, vị trí thỏa mãn điều kiện 1 và 2 càng nhỏ (càng gần 1) thì khả năng thỏa mãn điều kiện 3 của vị trí ấy càng cao. (Vì tổng điểm càng lớn)

Ta duyệt các vị trí từ N về 1 để tìm vị trí i nhỏ nhất có $MAX_i = 1$.

Duyệt trâu để kiểm tra xem vị trí vừa tìm được có thỏa điều kiện 3 hay không. Nếu thỏa mãn điều kiện 3, xuất "YES", ngược lại xuất "NO"

Độ phức tạp: O(T*N)

---------- **HẾT** ----------