HƯỚNG DẪN GIẢI CODE TOUR 2024
Code Challenge #2

BÀI A: BÀI TẬP VỀ NHÀ

Solution:

C++	https://ideone.com/GUs9Nb

Tóm tắt đề:

Cho ***N*** bài tập về nhà, để làm bài tập thứ ***i*** tốn A_i thời gian và cần có năng lực ít nhất là B_i để hoàn thành.Có ***Q*** học sinh, học sinh thứ ***i*** có năng lực là X_i.

Yêu cầu: Tìm xem tổng thời gian để hoàn thành tất cả các bài tập mà mỗi học sinh có thể làm

Input:

Dòng đầu gồm hai số nguyên dương ***N***, ***Q*** ($1 \le N, Q \le 10^5$) - thể hiện số bài tập về nhà và số học sinh trong lớp.

N dòng tiếp theo gồm các cặp số A_i và B_i($1 \le A_i, B_i \le 10^9$) - thời gian để hoàn thành và độ khó của bài tập thứ ***i***.

Q dòng tiếp theo, mỗi dòng gồm một số nguyên ***X*** ($1 \le X \le 10^9$) - năng lực của học sinh thứ ***i***.

Output:

Gồm Q dòng, mỗi dòng gồm một số nguyên là tổng thời gian hoàn thành hết bài tập có khả năng làm được.

Ví dụ:

7 5 10 7 3 2 9 3 2 5 7 3 1 9 6 6	3 0 21 37 19

2 1 5 8 4	

Giải thích:

Với trường hợp ví dụ, gồm 7 bài tập và 5 học sinh:

- Học sinh thứ nhất có năng lực là 2, có thể làm được duy nhất bài tập thứ hai nên tổng thời gian là 3.
- Học sinh thứ hai có năng lực là 1 nên không thể làm bất kì bài tập nào được giao.
- Học sinh thứ ba có năng lực là 5, có thể làm được các bài 2, 3, 4, 5 nên tổng thời gian là 3 + 9 + 2 + 7 = 21.
- Học sinh thứ tư có năng lực là 8, có thể làm được các bài 1, 2, 3, 4, 5, 7 nên tổng thời gian là 10 + 3 + 9 + 2 + 7 + 6 = 37.
- Học sinh thứ năm có năng lực là 4, có thể làm được các bài 2, 3, 5 nên tổng thời gian là 3 + 9 + 7 = 19.

Hướng dẫn giải:

Với mỗi học sinh, ta cần duyệt qua tất cả bài tập để tính tổng thời gian của các bài tập mà học sinh đó có thể làm. Tuy nhiên, do dữ liệu rất lớn và cách làm như vậy có thể lên đến ***O(N * Q)***.

Có nhận xét rằng với bài tập ***i*** mà học sinh không thể làm thì với mọi bài tập ***j*** với $B_i \leq B_j$ cũng không làm được. Tương tự như vậy, nếu học sinh có thể làm bài tập ***i*** thì mọi bài tập ***j*** với $B_j \leq B_i$ cũng có thể làm được.

Với nhận xét trên ta có thể tiến hành sắp xếp lại danh sách bài tập theo thứ tự tăng dần độ khó của bài tập. Với mỗi học sinh có thể dùng ***tìm kiếm nhị phân*** để tìm ra bài tập có độ khó bé nhất mà học sinh không thể hoàn thành, sau đó lấy tổng thời gian của các bài tập trước đó. Để lấy tổng nhanh ta có thể dùng tổng tiền tố.

Độ phức tạp: ***O((N + Q)log(N))***

BÀI B: THÔNG ĐIỆP VŨ TRỤ

Solution:

Solution	https://ideone.com/QfRYwQ

Tóm tắt đề:

Cho số nguyên dương M, và một bảng chữ cái latinh in hoa tuần hoàn dài vô tận được đánh số từ 0, dựng một chuỗi có thứ tự từ điển nhỏ nhất được tạo bởi các kí tự có vị trí là các ước số nguyên tố của M trên bảng chữ cái latinh đã cho.

Input:

Một dòng duy nhất là 1 số nguyên dương M

Điều kiện:

- $2 \leq M \leq 10^{12}$

Output:

Một dòng duy nhất là thông điệp vũ trụ có thứ tự từ điển nhỏ nhất.

Ví dụ:

Input	Output
18	CD
1450	CDF

Hướng dẫn giải:

Dựng một mảng $S[]$ là bảng chữ cái từ 'A' đến 'Z' được đánh số từ 0.

Phân tích thừa số nguyên tố của số M, với mỗi thừa số nguyên tố x là kí tự $S[x \ mod \ 26]$, ta thêm kí tự này vào mảng kết quả.

Sau đó sắp xếp mảng kết quả theo thứ tự tăng dần.

Độ phức tạp: $O(N * log(N))$

BÀI C: BRACKET CIRCLE

Solution:

C++	https://ideone.com/bAPWqA
Python	https://ideone.com/f4uvZN

Tóm tắt đề:

Cho một dãy ngoặc đúng S bất kì, đếm số lượng S' phân biệt thu được bằng cách xoay xâu S sao cho S' vẫn là một dãy ngoặc đúng.

Ví dụ:

6 ( ) (( ) ( ))	2

Giải thích:

Có 2 dãy ngoặc đúng thu được bằng xoay xâu trên là ()(()()) và (()())().

Hướng dẫn giải:

Subtask 1: |S| <= 2e3

Có thể mô phỏng việc xoay vòng bằng hàng đợi 2 đầu (deque); hoặc chèn thêm một copy của xâu S vào sau và kiểm tra từng xâu con độ dài N trên xâu đó. Với mỗi xoay vòng, để kiểm tra dãy ngoặc hợp lệ, ta xét từng prefix của xâu S sao cho không tồn tại một ví trí mà ngoặc đóng ‘)’ nhiều hơn ngoặc mở ‘(‘. Dùng set/map để đếm số dãy ngoặc đúng phân biệt.

Subtask 2: |S| <= 2e6

Đầu tiên tìm một xâu con T có độ dài ngắn nhất sao cho từ xâu T có thể tạo ra được xâu S bằng các copy của xâu T, xâu T này còn được gọi là border của xâu S. Ví dụ S = ()(()())(()())(()()) thì border của xâu S sẽ là ()(()()) do S được ghép từ 3 lần ()(()()). Để tìm border của S ta có thể dùng Z-function hoặc KMP trong O(N) hay thử từng độ dài là các ước của S để kiểm tra border với độ phức tạp là **O(NlogN).**

Đáp án cuối cùng là số lượng prefix của border là dãy ngoặc đúng. Ví dụ ở trường hợp S = ()(()())(()())(()()) thì đáp án là 2 do border ()(()()) có 2 prefix là dãy ngoặc đúng là () và ()(()()).

Ngoài ra bài này còn có thể giải bằng cách xoay xâu S từng bước một, nếu gặp một dãy ngoặc hợp lệ thì lưu lại giá trị hash vào set/map như subtask 1 để đếm số lượng xâu phân biệt.

.

BÀI D: EXPLOIT THE BUG

Solution:

C++	https://ideone.com/aZb09z

Tóm tắt đề:

Cho 2 dãy kí tự A và B có độ dài bằng nhau và bằng N, cả 2 dãy chỉ chứa 3 kí tự sau: P, S, R.

Biết rằng P > R > S > P.

Sắp xếp lại A theo 1 cách bất kì.

A_{score} = số phần tử $a_i > b_i$

B_{score} = số phần tử $b_i > a_i$

In ra $max(A_{score} - B_{score})$

Input:

- Dòng đầu tiên là số nguyên N ($1 \leq N \leq 10^6$)
- Dòng thứ hai là N kí tự cách nhau bởi 1 khoảng trắng tương ứng N lá bài phát cho A
- Dòng thứ ba là N kí tự cách nhau bởi 1 khoảng trắng tương ứng N lá bài phát cho B
- Dòng thứ 2 và thứ 3 chỉ chứa các kí tự R, S, P tương ứng búa, kéo và bao

Output:

In ra duy nhất 1 số nguyên

Ví dụ:

3 R P S P P P	1

Hướng dẫn giải:

Ta chứng minh rằng cách sau là tối ưu nhất (gọi là cách 1): (A thắng nhiều nhất) $*$ 1 + (A hòa nhiều nhất sau khi thắng)$*$ 0 + (còn lại)$*$ (-1)

Gọi tập index mà A thắng là W_1, tập index mà A hòa là D_1, và tập thua là L_1

Ta giả sử rằng có 1 cách khác tối ưu hơn (gọi là cách 2): gọi tập index lần lượt là W_2, D_2, L_2 bởi vì W_1 là các A thắng nhiều nhất có thể, nên W_2 là tập con của W_1.

Gọi số lượng phần tử tập các index thuộc W_1 không thuộc W_2 là k.

Ta chú ý rằng: sau khi ta cố định các lá bài của A thắng, thì để tối ưu thì A luôn phải hòa nhiều nhất có thể (giảm thiểu số lá bài thua).

Vì vậy nên D_1 lại là tập con của D_2 (bởi vì k index kia là lấy ra từ A, nói cách khác là không ảnh hưởng gì tới tập hòa D_1, và cả 2 tập D là tối ưu hòa)

Gọi số lượng phần tử tập các index thuộc D_2 không thuộc D_1 là $2m$ $(2m \leq 2k)$ (vì có k index mới, ta có $D1$ là tập con $D2$, nên số index hòa chỉ sinh ra bằng cách swap Thua và Thắng cho nhau, có tối đa k lần swap như vậy, và mỗi lần swap thì $D \mathrel{+}= 2$ nên $2m \leq 2k$)

Vậy $|L_2| = |L_1| - 2m + k.$ $(W + L + D = N(const))$

Ta có cách tính điểm cách 1 = $|W_1| - |L_1|$

Cách tính điểm cách 2 = $|W_2| - |L_2| = |W_1| - k - (|L_1| + k - 2m) = |W_1| - |L_1| + 2m - 2k \leq |W_1| - |L_1|$

Điều trên là vô lí, hay nói cách khác cách 1 là tối ưu nhất.

Cách code:

Bởi vì A có thể xếp tùy ý nên ta quan tâm tới frequency của các R, S, P của cả 2.

Vậy ta tính điểm cao nhất của A = min(bao của A, búa của B) + min(búa của A, kéo của B) + min(kéo của A, bao của B)

Ta loại bỏ các trường hợp đã dùng ở trên: trừ fre của A và B tương ứng với các min ở trên.

Ta tiếp tục loại bỏ đi các trường hợp được sử dụng cho trường hợp A Hòa (nhiều hòa nhất có thể dựa trên fre đã lượt bỏ): trừ fre của búa A và búa B cho min(búa A, búa B), tương tự kéo và bao.

Tính điểm còn lại, là điểm A thua: tổng fre còn lại của A.

In ra A thắng - A thua.

Độ phức tạp: O(n)

BÀI E: CHIA HẾT

Solution:

Python	https://ideone.com/R1tqfL
C++	https://ideone.com/k4r6IR
Java	https://ideone.com/79J7oP

Tóm tắt đề:

Cho một dãy số nguyên $a_1, a_2, \dots, a_N$ $(1 \le N, a_i \le 10^6)$, số A được tạo ra bằng cách tính tích các số trong dãy trên.

Cho T $(1 \le T \le 5 * 10^5)$ truy vấn, mỗi truy vấn là một cặp số (X, P) $(1 \le X, P \le 5 * 10^5)$. Hãy cho biết A có chia hết cho X^P không?

Ví dụ:

5 1 2 3 4 5 4 5 1 2 3 3 4 15 2	1100

Giải thích:

Trong ví dụ trên, $A = 1 * 2 * 3 * 4 * 5 = 120$, có tất cả 4 truy vấn:

- Truy vấn 1, X = 5, P = 1 và A = 120 chia hết cho $5^1 = 5$
- Truy vấn 2, X = 2, P = 3 và A = 120 chia hết cho $2^3 = 8$
- Truy vấn 3, X = 3, P = 4 và A = 120 không chia hết cho $3^4 = 81$
- Truy vấn 4, X = 15, P = 2 và A = 120 không chia hết cho $15^2 = 225$

Hướng dẫn giải:

1. **Subtask 1:** $A \leq 10^9, X, P \leq 10^5$
   Vì A nhỏ, nên có thể tính trực tiếp số A bằng cách nhân các số trong mảng lại, với mỗi truy vấn tính X^P bằng cách dung vòng lặp, tuy nhiên cần lưu ý P khá lớn có thể dẫn đến tràn số, do đó trong lúc tính X^P nếu thấy kết quả lớn hơn A thì dừng và kết luận X^P không chia hết bởi A.

2. **Subtask 2:** $P = 1$
   Vì $P = 1$ nên chỉ cần kiểm tra xem A có chia hết cho X không. Tuy nhiên số A là ***rất lớn***, nên không thể tính tích một cách thông thường được.
   Mặc dù A rất lớn, nhưng vì $a_i \leq 10^6$ nên ta có thể lưu A dưới dạng tích của các thừa số nguyên tố trong mảng $cnt[10\text{^}6]$. Với $cnt[x]$ là số mũ của thừa số x.
   Ví dụ $A = 120 = 2^3 * 3 * 5$, lúc này $cnt[2] = 3, cnt[3] = 1$ và $cnt[5] = 1$.

   Vậy làm sao để xây dựng mảng cnt này? Với tính chất toán học của phép nhân luỹ thừa cùng cơ số, ta có $x^a * x^b = x^{a+b}$. Nên ta chỉ cần phân tích thừa số nguyên tố lần lượt từng số a_i trong dãy rồi cộng dồn số mũ vào mảng $cnt[]$.

   Vậy làm sao để kiểm tra số A trên có chia hết cho một số X nào đó không? Tương tự, ta cũng phân tích X thành các thừa số nguyên tố, nếu A chia hết cho X thì phải thoả mãn điều kiện: *số mũ của mọi thừa số nguyên tố của X phải không vượt quá A.*
   Ví dụ với $A = 120$ và $X = 12 = 2^2 * 3$ thì A chia hết cho X, tuy nhiên với $X = 7$ thì A không chia hết cho X, vì số mũ của 7 là 1, trong khi A không có thừa số 7 (hay số mũ của 7 trong A bằng 0).

   Bây giờ, ta đã biết cách kiểm tra một số X có là ước của số A không bằng cách phân tích thừa số nguyên tố. Tuy nhiên vì giới hạn bài toán là khá lớn ($N \leq 10^6, T \leq 5 * 10^5$), nên ta cần một thuật toán đủ nhanh để phân tích thừa số nguyên tố.

3. **Subtask 3**: $N, A, X, P \leq 10^5$
   Đối với subtask này, ta có thể thực hiện phân tích thừa số nguyên tố với độ phức tạp là $O(\sqrt{X})$, với nhận xét rằng một số nguyên X chỉ có tối đa một thừa số nguyên tố lớn hơn $\sqrt{X}$, vì nếu tồn tại 2 thừa số nguyên tố lớn hơn, giả sử gọi đó là U và V thì $U * V > X$ (vô lý).
   Do đó, ta chỉ cần thực hiện một vòng lặp từ 2 đến $\sqrt{X}$ để tìm các thừa số nguyên tố.
   Thử giải quyết lại bài toán ở Subtask 2, với *P = 1*, trong quá trình phân tích X của truy vấn, so sánh số mũ của các thừa số x nếu tất cả thừa số đều bé hơn hoặc bằng $cnt[x]$ thì kết quả là chia hết.
   Vậy nếu $P > 1$ thì sao? Có thể thấy X^P thì số mũ của các thừa số nguyên tố đều gấp P lần lên. Ví dụ $X = 12 = 2^2 * 3$, thì $X^2 = (2^2 * 3)^2 = 2^4 * 3^2$.

Độ phức tạp cho thuật toán trên là $O(N\sqrt{10^6} + T\sqrt{X})$, đủ để qua subtask này.

4. **Subtask 4:** $N, A \leq 10^6, 1 \leq T, X, P \leq 5 * 10^5$
   Với subtask cuối này, ta cần một thuật toán phân tích thừa số nguyên tố nhanh hơn nữa. Dưới đây mô tả thuật toán phân tích cho mỗi truy vấn với độ phức tạp $O(logX)$, ứng dụng thuật toán Sàng nguyên tố (có thể tham khảo ở đây).
   Thay vì cài đặt giống sàng nguyên tố thông thường, chỉ lưu $prime[x] = true/false$ cho biết x có phải số nguyên tố không. Thì ta sẽ lưu $prime[x] = y$, với y là một thừa số nguyên tố của x.

```
for (int i = 2; i <= MAX_A; i++)
  if (!prime[i])
    for (int j = i; j <= MAX_A; j += i)
      prime[j] = i;
```

Khi có mảng $prime[]$, thì việc tìm một ước số nguyên tố của X chỉ trong $O(1)$, và tổng thời gian để phân tích một số là $(logX)$,

```
while (x > 1) {
  int y = prime[x];
  while (x % y == 0) {
    x /= y;
    cnt[y]++;
  }
}
```

Độ phức tạp: $O(Nlog(10^6) + Tlog(X))$

---------- HẾT ----------