Tổng quan

	Số đội AC	Độ khó	Lời giải
A	16	3	Lê Duy Thức
B	4	4	Hoàng Xuân Nhật
C	2	3	Nguyễn Diệp Xuân Quang
D	73	1	Lê Duy Thức
E	0	4	Trần Tấn Phát
F	3	3	Nguyễn Vương Linh
G	32	2	Thầy Nguyễn Thanh Tùng
H	38	2	Lê Duy Thức
I	47	2	Tô Tuấn Dũng
J	0	5	Nghĩa Phạm
K	13	3	Lê Duy Thức
L	13	3	Nguyễn Diệp Xuân Quang
M	111	1	Lê Duy Thức

A. Abstract Painting

Tóm tắt đề bài

Ta có 3 màu: xanh, đỏ, vàng. Cần đếm số cách tô màu tất cả các cạnh của một bảng ô vuông có R hàng, C cột sao cho:

- Trong mỗi ô, 4 cạnh của nó có đúng 2 màu, và mỗi màu xuất hiện đúng 2 lần.

Giới hạn: $1 \le R \le 14, 1 \le C \le 2000$

Lời giải

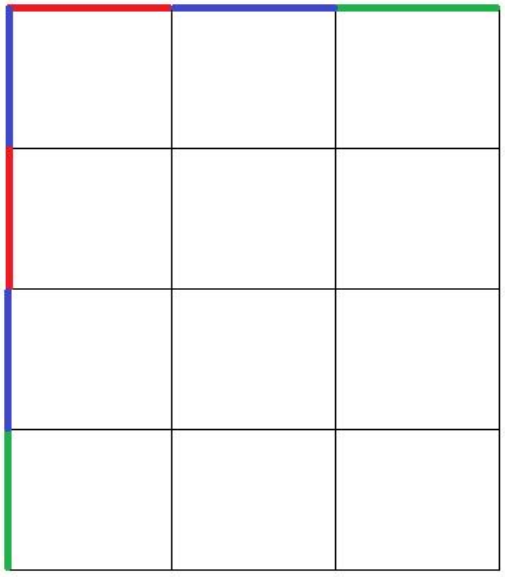
Khi nhìn vào giới hạn nhỏ của bài này, nhiều bạn đã nghĩ tới một thuật DP bitmask gì gì đó (trong đó có cả mình), nếu cứ lún sâu vào cách nghĩ đó thì khả năng rất lớn là sẽ không nghĩ ra cách dễ hơn cho bài này. Bài này thật ra có một cách đơn giản hơn nhiều:

Ta xét 1 ô đã được tô trước 2 cạnh, có 2 trường hợp như sau:

- 2 cạnh đó đã được tô cùng 1 màu: do đó, 2 cạnh còn lại chỉ có thể tô cùng 1 màu, vậy số cách tô ở trường hợp này là 2.
- 2 cạnh đó đã được tô khác màu: do đó, ở 2 cạnh còn lại đều phải dùng màu đã được tô ở 2 cạnh trước, do đó số cách tô ở trường hợp này cũng là 2 (ví dụ ta đã tô 2 màu đỏ xanh, thì 2 cạnh còn lại có thể tô (xanh, đỏ) hoặc (đỏ, xanh)).

Ta suy ra **nhận xét rất quan trọng** sau: Khi một ô đã được cố định trước 2 cạnh, số cách tô phần còn lại của nó luôn là 2, không quan trọng tới cách tô 2 cạnh trước

Do đó, nếu ta cố định trước màu ở cột đầu và hàng đầu của bảng như hình bên. Thì lần lượt ta thấy: ô (1, 1) sẽ có 2 cách tô, sau đó ô (1, 2) sẽ bị cố định bởi màu của ô (1, 1) và cạnh ở trên, do đó nó cũng có 2 cách tô. Cứ tương tự như thế ta có thể suy ta toàn bộ ô trong bảng đều có 2 cách tô.



Suy ra: Khi cố định trước màu cột và hàng đầu tiên, số cách tô phần còn lại của bảng chính bằng 2^{R*C} cách. Ta có tổng cộng R cạnh cần cố định ở cột đầu, C cạnh cần cố định ở hàng đầu, mỗi cạnh lại có 3 cách chọn màu, vậy số cách cố định hàng và cột đầu là: 3^{R+C}. **Vậy suy ra số cách tô màu một bảng ô vuông** $R * C$ **là:** $3^{R+C} * 2^{R*C}$.

B. Banana problem

Tóm tắt đề bài

Có N cái thang, sắp theo 1 hàng ngang. Mỗi cái thang có độ dài H. Có R cái dây, mỗi dây có độ dài và nối 2 thang tại một độ cao nhất định. **Tại một độ cao của một thang, chỉ có tối đa một dây nối vào đó.** Có B quả chuối xuất hiện tại các vị trí đặc biệt, quả thứ i có vị trí tại độ cao h_i của thang l_i. Mỗi quả sẽ xuất hiện trong x_i giây rồi biến mất trong y_i giây rồi lặp lại. **Không có dây nào nối vào một vị trí đặc biệt.**
Có N con khỉ, mỗi con khỉ xuất phát tại bậc dưới cùng của thang. Con khỉ thứ i mất c_i giây để trèo lên một bậc thang, và mất d_i giây để đi qua một độ dài của giây. Mỗi con khỉ sẽ đi lên, khi gặp dây sẽ đi qua và không đi ngược lại. Khỉ sẽ ăn chuối nếu thời điểm khỉ bước vào một vị trí đặc biệt, vị trí đó có chuối. Hỏi tổng số chuối các con khỉ ăn là bao nhiêu.

Lời giải

Bài này các bạn phải đọc kĩ đề và vượt qua các dữ kiện mang tính hù dọa của BTC.
Nếu các bạn đọc kĩ điều kiện của bài toán, bài này có thể hình dung như sau: các con khỉ sẽ đi lên, khi gặp dây thì đi ngang, xong lại đi lên. Khi gặp chuối thì ăn chuối nếu có.
Đầu tiên, nếu các bạn nhìn kĩ điều kiện của thời gian xuất hiện các quả chuối, có thể thấy tại một vị trí, không bao giờ có nhiều hơn 1 quả chuối. Hơn nữa, các con khỉ luôn đi lên, vì vậy dữ kiện một con khỉ nếu đến một vị trí nhiều lần có thể ăn nhiều chuối là thừa.
Sau đó, ta nhận thấy các con khỉ luôn đi lên và đi ngang. Hơn nữa, chúng bắt đầu từ các thang khác nhau. Ta hãy tưởng tượng: ban đầu ta có dãy N con khỉ, đánh số từ 1 -> N. Mỗi con khỉ sẽ ở trên một cái thang. Khi gặp một dây nối 2 thang A và B, 2 con khỉ đang trên 2 thang này sẽ đổi chỗ cho nhau. Từ đó ta có một nhận xét vô cùng quan trọng: **đường đi của các con khỉ trên các thang là không giao nhau, trừ tại các dây nối các thang.** Ta lại quay lại một điều kiện quan trọng khác: **tại các đầu dây và trên dây, không có chuối.** Như vậy, sẽ không có quả chuối nào có thể bị hai con khỉ đi qua.
Từ hai dữ kiện trên, ta có thể tính chính xác, nếu một con khỉ tới vị trí đặc biệt tại thời điểm *t* thì nó có ăn được chuối hay không trong bằng phép chia. Nếu con khỉ tới được vị trí đặc biệt tại thời điểm chuối còn tồn tại, thì chắc chắn sẽ ăn được chuối.
Cũng từ đó, ta có thể thấy là một vị trí đặc biệt có đúng 1 con khỉ đi qua. Một sợi dây cũng chỉ có đúng 2 con khỉ đi qua. Ta coi đây là các "sự kiện": một con khỉ sẽ đi lên liên tục, cho tới khi gặp chuối hoặc dây. Vì số sự kiện là O(n), ta chỉ cần mô phỏng lại các sự kiện là xong.
Code: https://ideone.com/yaY8Cp

C. Counting Palindromes

Tóm tắt đề bài

Đếm số lượng số đối xứng x gồm n chữ số sao cho $x \equiv k \ (mod \ p)$ (tức là $x \ mod \ p = k$).

Lời giải

Thuật toán với n đủ nhỏ

Để tiện giải thích, giả sử n là số chẵn. Khi đó, một số đối xứng sẽ có dạng:

$$x = \overline{a_1 a_2 a_3 ... a_{n/2} a_{n/2} ... a_3 a_2 a_1}$$
$$= 10^{n-1} a_1 + 10^{n-2} a_2 + 10^{n-3} a_3 + ... + 10^{n/2} a_{n/2} + 10^{n/2-1} a_{n/2} + ... + 10^2 a_3 + 10 a_2 + a_1$$
$$= \sum_{i=1}^{n/2} 10^{n-i} \ a_i + \sum_{i=1}^{n/2} 10^{i-1} \ a_i$$
$$= \sum_{i=1}^{n/2} s_i a_i \ (\text{với } \ s_i = (10^{n-i} + 10^{i-1}) \ mod \ p)$$

Bài toán trở thành: đếm số bộ $a_1, a_2, ..., a_{n/2}$ ($1 \le a_1 \le 9$ và $0 \le a_i \le 9$ với $2 \le i \le n/2$) sao cho $\sum_{j=1}^{i} s_j a_j \equiv r \ (mod \ p)$. Nếu n đủ nhỏ, ta có thể giải bài toán trên bằng thuật toán quy hoạch động sau:

Gọi $dp[i][r]$ là số bộ a_1, a_2, …, a_i sao cho $\sum_{j=1}^{i} s_j a_j \equiv r \ (mod \ p)$. Khi đó, đáp án cần tìm là $dp[n/2][k]$. Cơ sở quy hoạch động: $dp[0][0] = 1$, $dp[0][r] = 0$ với $1 \le r < p$.

Với mỗi trạng thái $dp[i-1][r]$, ta duyệt lần lượt các chữ số trong phạm vi cho phép của a_i (gọi chữ số đang duyệt là d) và thử đặt chữ số này vào vị trí i. Với mỗi d, ta có $\sum_{j=1}^{i} s_j a_j \equiv \sum_{j=1}^{i-1} s_j a_j + s_i d \equiv r + s_i d \ (mod \ p)$, do đó ta cập nhật $dp[i][(r + s_i d) \ mod \ p] \ += \ dp[i-1][r]$.

Cách làm trên có độ phức tạp $O(np)$. Hiển nhiên, với giới hạn $n \le 10^{18}$, cách làm này không đủ nhanh.

Biểu diễn đa thức

Để cải tiến, ta sẽ biểu diễn mỗi dòng của bảng quy hoạch động trên dưới dạng một đa thức. Cụ thể:

$$P_i = \sum_{r=0}^{p-1} dp[i][r] * x^r = dp[i][p-1] * x^{p-1} + dp[i][p-2] * x^{p-2} + ... + dp[i][2] * x^2 + dp[i][1] * x + dp[i][0]$$

Đáp án cần tìm chính là hệ số của x^k trong $P_{n/2}$.

Với hai đa thức $A = a_p x^p + ... + a_2 x^2 + a_1 x + a_0$ và $B = b_p x^p + ... + b_2 x^2 + b_1 x + b_0$, ta định nghĩa tích của chúng như sau:

$$C = A * B = \sum_{i=0}^{p-1} \sum_{j=0}^{p-1} a_i b_j x^{(i+j) \bmod p}$$

Định nghĩa này khác với định nghĩa phép nhân đa thức thông thường ở chỗ, bậc của x ở đa thức kết quả sẽ được mod p. Các tính chất giao hoán, kết hợp của phép nhân thông thường vẫn sẽ đúng với định nghĩa mới này.

Cải tiến thuật toán

Ý tưởng của cải tiến này là, ta sẽ biểu diễn việc cập nhật trạng thái từ dòng $i-1$ sang dòng i của bảng dp dưới dạng một phép nhân đa thức. Cụ thể, xét hai đa thức P_{i-1} và P_i tương ứng với hai dòng $i-1$ và i của bảng dp. Nhắc lại về quá trình cập nhật từ dòng $i-1$ sang dòng i như sau:

- Với mỗi d từ $lo(i)$ đến 9 (với $lo(i) = 1$ nếu $i = 1$ và $lo(i) = 0$ nếu $2 \le i \le n/2$), ta cập nhật $dp[i][(r + s_i d) \bmod p] \mathrel{+}= dp[i-1][r]$.

Từ đó, ta có:

$$P_i = \sum_{r=0}^{p-1} \sum_{d=lo(i)}^{9} dp[i-1][r] * x^{(r+s_i d) \bmod p} = (\sum_{r=0}^{p-1} dp[i-1][r] * x^r) * (\sum_{d=lo(i)}^{9} x^{(s_i d) \bmod p})$$

Ta định nghĩa đa thức $F_i = \sum_{d=lo(i)}^{9} x^{(s_i d) \bmod p}$. Từ đó: $P_i = P_{i-1} * F_i$

Đa thức $P_{n/2}$ có thể được khai triển như sau:

$$P_{n/2} = P_{n/2-1} * F_{n/2}$$
$$= P_{n/2-2} * F_{n/2-1} * F_{n/2}$$
$$= ...$$
$$= P_0 * F_1 * F_2 * ... * F_{n/2-1} * F_{n/2}$$
$$= F_1 * F_2 * ... * F_{n/2-1} * F_{n/2}$$

Ta nhận xét rằng, nếu p và 10 nguyên tố cùng nhau (tức là $gcd(p, 10) = 1$) thì theo định lý Fermat nhỏ, ta có: $10^{p-1} \equiv 1 \ (mod\ p)$. Do đó, với $i > p$, ta có:

$s_i \equiv 10^{n-i} + 10^{i-1} \pmod p$
$\equiv 10^{n-i} * 10^{p-1} + 10^{i-1-(p-1)} * 10^{p-1} \pmod p$
$\equiv 10^{n-i+(p-1)} + 10^{i-1-(p-1)} \pmod p$
$\equiv 10^{n-(i-(p-1))} + 10^{(i-(p-1))-1} \pmod p$
$\equiv s_{i-(p-1)} \pmod p$

Với $i \geq 2$ thì đa thức F_i được định nghĩa chỉ phụ thuộc vào s_i. Từ đó, ta có $F_i = F_{i-(p-1)}$ với mọi $i \geq p$.

Từ nhận xét trên, ta có thể tính nhanh $P_{n/2}$ như sau:
$P_{n/2} = F_1 * (F_2 * F_3 * ... * F_p) * (F_2 * F_3 * ... * F_p) * ... * F_2 * F_3 * ... * F_{(n/2) \bmod (p-1)}$
$= (F_2 * F_3 * ... * F_p)^{(n/2-1)/(p-1)} * F_1 * F_2 * F_3 * ... * F_{(n/2) \bmod (p-1)}$

Ta cần tính $A = F_2 * F_3 * ... * F_p$ trong và $B = F_1 * F_2 * F_3 * ... * F_{(n/2) \bmod (p-1)}$. Nếu ta tính A bằng cách thực hiện $p-2$ phép nhân đa thức, độ phức tạp sẽ là $O(p^3)$, không đủ nhanh với giới hạn $p \leq 2000$. Thay vào đó, ta sẽ dùng thuật toán quy hoạch động (tương tự lời giải với n đủ nhỏ) để tính A và B trong $O(p^2)$.

Khi đó, ta có thể dùng thuật toán luỹ thừa nhanh để tính $A^{(n/2-1)/(p-1)}$ trong $O(p^2 \log n)$. Kết quả cần tìm chính là hệ số của x^k trong $A^{(n/2-1)/(p-1)} * B$.

Với n lẻ, các số đối xứng sẽ có dạng: $\overline{a_1 a_2 a_3 ... a_{n/2} a_{n/2+1} a_{n/2} ... a_3 a_2 a_1}$. Lời giải đối với trường hợp này không quá khác biệt so với trường hợp n chẵn, và xem như bài tập cho bạn đọc. (Ta cần xử lí riêng trường hợp $n = 1$).

Lưu ý rằng, trường hợp $p = 2$ và $p = 5$ thì p và 10 không nguyên tố cùng nhau, nên ta không thể áp dụng được định lý Fermat nhỏ. Do đó, ta cũng cần xử lý riêng hai trường hợp này.

Độ phức tạp: $O(p^2 \log n)$

D. Dating time

Tóm tắt đề bài

Đếm số lần mà kim giờ và kim phút tạo thành một góc $alpha$ trong khoảng thời gian $[hL : mL; hR : mR]$, với $0 <= hL, hR <= 23; 0 <= mL, mR <= 59; alpha \in \{0, 90, 180\}$.

Lời giải

Do giới hạn khá nhỏ, ta có thể nghĩ tới cách tạo mảng hằng trước mọi kết quả có thể.

Nhận xét 1: khi với mỗi trạng thái kim giờ và kim phút tạo thành một góc bất kì, ta luôn có thể xem giờ hiện tại đang là số nguyên, và phút hiện tại là số thực. Do đó ta có thể duyệt qua 24 giờ, tìm những phút thoả mãn.

Cho đơn giản, gọi giờ hiện tại ta đang xét là $h : m \ (h \in N, 0 <= h <= 23; m \in R, 0 <= m < 60)$.

Tính chất 1: sau mỗi giờ, kim giờ sẽ di chuyển một góc 30 độ, sau mỗi phút, kim giờ sẽ di chuyển một góc 0.5 độ. Vậy tại giờ hiện tại, kim giờ đang tạo thành 1 góc bằng $(30 * h + 0.5m)$ so với vị trí của nó tại 0 giờ.

Tính chất 2: sau mỗi phút, kim phút sẽ di chuyển một góc 6 độ. Vậy tại giờ hiện tại, kim phút đang tạo thành 1 góc bằng $6m$ so với vị trí của nó tại 0 giờ.

Vậy góc tạo giữa kim giờ và kim phút hiện tại là:
$|30 * h + 0.5m - 6m| = |30 * h - 5.5m| = alpha$
Tới đây ta có phương trình 2 ẩn, nhưng nhận thấy ta có thể duyệt qua mọi h, do đó ta có thể tìm m bằng cách phá dấu trị tuyệt đối đi:

- $|30 * h - 5.5m| = alpha \Leftrightarrow 30 * h - 5.5m = alpha \cap 30 * h - 5.5m = -alpha$.
- Lưu ý rằng -90 = 270 trên modulo 360.

Giới hạn bài toán này khá nhỏ nên có 1 cách khác là tính tay tất cả các trường hợp ra rồi sau đó dùng mảng hằng (chỉ có 88 trạng thái thoả mãn).
Code tham khảo:

- Mảng hằng: https://ideone.com/0ibR6e
- Trường hợp alpha đặc biệt như đề bài: https://ideone.com/pD17iE
- Trường hợp với alpha bất kì nằm trong đoạn $[0, 180]$: https://ideone.com/uuwnBY

E. Easy Query

Bài toán phụ 1:

Cho 1 dãy A[1..n] (1 <= A[i] <= 10 ^ 5), cho các truy vấn L, R , tính tổng các số phân biệt trong đoạn L, R. Để đơn giản lời giải, coi A[i] là màu của số thứ i.
Để giải bài toán trên, cần xử lý offline các truy vấn theo thứ tự tăng dần về R. Khi tăng R lên 1 đơn vị, ta thấy rằng chỉ cần giữ lại lần xuất hiện cuối cùng của 1 số, từ đấy ta có thể dùng 1 cấu trúc dữ liệu bất kì (BIT/Segment Tree) hỗ trợ truy vấn thêm và lấy tổng.

Bài toán phụ 1.1:

Cho 1 dãy A[1..n] (1 <= A[i] <= 10 ^ 5), cho các truy vấn L, R, low, high, tính tổng các số phân biệt v trong đoạn L, R và low <= v <= high.

Ta vẫn xử lý offline các truy vấn nhưng giờ đây ta phải dùng segmentTree 2D để truy vấn nhanh 1 đoạn màu liên tiếp. Mỗi nút segment tree có thể sử dụng 1 cấu trúc như đã nói ở bài toán 1.

Bài toán phụ 1.2:
Cho 1 dãy A[1..n] (1 <= A[i] <= 10 ^ 5), cho các truy vấn L, R, low, high, k hỏi có bao nhiêu số phân biệt v trong đoạn L, R xuất hiện ít nhất k lần và low <= v <= high. (k <= 3)

Duy trì k segment tree phân biệt, với mỗi màu duy trì lần xuất hiện gần thứ k, giải tương tự bài toán phụ 1.1.

Đến đây, ta thấy rằng với đề bài ban đầu, nếu ta tìm ra được và bỏ đi màu nhỏ nhất và màu lớn nhất trong truy vấn thì đề bài đưa về bài toán phụ 1.2. Để xử lý nhanh phần này, có thể sử dụng persistent segment tree để tìm và xử lý trong O(nlog^2n)

Code tham khảo: https://ideone.com/KewOac

F. Fair Bandwidth Sharing

Lời giải bằng tiếng Anh

Code Lăng Trung Hiếu

G. Generating Numbers

Tóm tắt đề bài

Cho hai loại phép biến đổi một số nguyên dương x:

- Tăng x lên 1
- Tạo số mới không bắt đầu bằng chữ số 0 bằng cách đổi chỗ tùy ý các chữ số trong x.

Cho biết số phép biến đổi ít nhất để nhận được số nguyên dương y từ $x = 1$.

Giới hạn: $1 \le y \le 10^9$, có $t \le 1000$ bộ dữ liệu vào.

Lời giải

Ta kí hiệu $z_n = 10^n = 100...0$ ($n-1$ chữ số 0).

Gọi n là số chữ số có nghĩa của y.
Để tạo được y trước hết cần tạo z_n.

- Xuất phát từ $z_1 = 1$ tạo được $z_2 = 10$ với $9 = 10 - 1$ phép biến đổi.
- Từ $z_2 = 10$ tạo được $z_3 = 100$ với $19 = 20 - 1$ phép biến đổi:
  - $10 \to 19 \to 91 \to 99 \to 100$
    (số phép biến đổi lần lượt là 9, 1, 8, 1)
- Từ $z_3 = 100$ tạo được $z_4 = 1000$ với $29 = 30 - 1$ phép biến đổi:
  - $100 \to 109 \to 190 \to 199 \to 991 \to 999 \to 1000$
    (số phép biến đổi lần lượt là 9, 1, 9, 1, 8, 1)
- ...
- Bằng phương pháp quy nạp dễ dàng chứng minh được rằng từ z_{n-1} để dẫn xuất z_n cần $10 * (n-1) - 1$ phép biến đổi.

Như vậy, số phép biến đổi cần thiết để từ 1 dẫn xuất ra z_n là

$$\sum_{i=1}^{n-1} (10 * i - 1)$$

Giả thiết $y = \overline{a_0 a_1 a_2 ... a_{n-1}}$ trong đó $0 \le a_i \le 9$, $a_0 > 0$.

Từ z_n, nếu $a_0 \ne 1$ thì ta cần $a_0 + 1$ phép biến đổi để đưa chữ số trái nhất về a_0:

- $100...00 \to 100...0a_0 \to a_0 00...01$
  (số phép biến đổi lần lượt là a_0, 1)

Nếu $a_0 = 1$ thì không cần thực hiện biến đổi

Với mỗi $a_i \neq 0$ tiếp theo *chưa phải là chữ số cuối cùng*:

- Trường hợp $a_0 \neq 1$:
    - Để nhận được chữ số $a_i \neq 0$ tiếp theo cần thực hiện $(a_i - 1) + 1 = a_i$ phép biến đổi
    - Với các $a_i \neq 0$ còn lại: Cần thực hiện $a_i + 1$ phép biến đổi.
- Trường hợp $a_0 = 1$: Cần thực hiện ai+1 phép biến đổi.

Với *chữ số cuối cùng*: xét tương tự, nhưng bớt 1 phép biến đổi vì không cần đổi vị trí.

Trường hợp đặc biệt: y chỉ chứa một chữ số khác không (ví dụ, $y = 600$), khi đó cần tính số phép biến đổi để nhận được $y - 1$ và thêm 1 nữa để nhận được y.

Các số cần xử lý nên lưu ở dạng xâu để dễ dàng tính độ dài (số lượng chữ số) cũng như lấy ra từng chữ số.

Với mỗi số, độ phức tạp của giải thuật là $O(n)$, trong đó n là độ dài của số cần xử lý. Do $n \leq 9$, ta có thể coi độ phức tạp xử lý mỗi số là $O(1)$.
Như độ phức tạp của toàn bài toán là $O(t)$.

H. Hanjie

Tóm tắt đề bài

Ta cần đếm số cách tô màu 1 bảng có kích thước $r * c$ thành 2 màu trắng, đen, với yêu cầu như sau:

- Với mỗi hàng/cột, có một mảng manh mối là $a_1, a_2, ..., a_k$, ta cần tô hàng/cột đó bằng màu trắng, đen sao cho:
    - Bắt đầu bởi một số ô trắng liên tiếp (có thể không có)
    - Tiếp theo là a_1 ô đen liên tiếp
    - Tiếp theo là ít nhất 1 ô trắng liên tiếp
    - Tiếp theo là a_2 ô đen liên tiếp
    - Tiếp theo là ít nhất 1 ô trắng liên tiếp
    - …
    - Tiếp theo là a_k ô đen liên tiếp
    - Cuối cùng là một số ô trắng liên tiếp (có thể không có)

Giới hạn: $1 \le r,\ c\ \le 6$

Lời giải

Giới hạn bài này khá nhỏ, ta nghĩ tới việc backtrack, nhưng nếu backtrack một cách ngây thơ, độ phức tạp có thể lên tới $O(2^{r*c} * (r+c))$. Ta có thể giảm độ phức tạp bằng cách:

- Với mỗi hàng, tính trước các trạng thái thỏa mãn các manh mối. Mỗi hàng có nhiều nhất 10 trạng thái thỏa mãn manh mối. (Trường hợp $r = 6$ và manh mối có duy nhất 2 số 1).
- Sau đó, khi backtrack, thay vì chỉ duyệt qua từng trạng thái của 1 ô, ta duyệt qua trạng thái của cả 1 hàng thỏa mãn các manh mối (như trên, nhiều nhất 10 trạng thái).
- Sau khi hoàn thành các hàng, ta chỉ việc kiểm tra xem các cột có thoả mãn manh mối hay không.

Độ phức tạp: $O(cnt[1] * cnt[2] * cnt[3] ... * cnt[r] * cost)$, với $cnt[i]$ là số trạng thái thỏa mãn của hàng i $(cnt[i] \le 10)$, $cost$ là chi phí kiểm tra các cột có thoả mãn hay không, việc kiểm tra có thể thực hiện dễ dàng trong $O(r * c)$, nên độ phức tạp sẽ xấp xỉ $10^6 * 36$

Code tham khảo: https://ideone.com/9yAJ18

I. Inspecting Illumination

Tóm tắt đề bài

Đây là dạng bài interactive
Có n bóng đèn, nối với n công tắc khác nhau. Mỗi lần truy vấn, ta có thể bật 1 số công tắc, và biết được những bóng đèn nào được bật sáng. Được sử dụng nhiều nhất 32 truy vấn như trên, cần tìm ra mỗi công tắc điều khiển bóng đèn nào tương ứng.
Giới hạn: $n \leq 1000$

Lời giải

Dễ dàng nhận thấy từ giới hạn của đề bài, chúng ta cần phải nghĩ ra phương pháp truy vấn hợp lý chỉ trong log2(n) thao tác. Các phương pháp tương ứng gồm có tìm kiếm nhị phân (như rất nhiều bài interactive khác, lol), hay chia để trị, xử lý bit,... Nhưng tìm kiếm nhị phân là một điều mơ hồ trong bài toán này, nên chúng ta hãy thử nhìn vào các phương pháp còn lại.
Vậy chúng ta bắt đầu từ đâu?

- Hãy thử chia tập các công tắc làm đôi: một bên từ 1...n/2, một bên từ n/2+1...n, sau đó dùng truy vấn #1 để hỏi theo tập hợp trên. Vậy, chúng ta đã phân biệt được tập hợp n bóng đèn thành 2 đặc tính: một bên sáng trong truy vấn #1, và 1 bên tối trong truy vấn #1.
- Với truy vấn 2, chúng ta sẽ tìm cách chia đôi tiếp. Giờ chúng ta chẳng lẽ hỏi đánh vào từng tập hợp đã chia ra sao: hỏi từ 1...n/4 á? Cách này sẽ thiếu hiệu quả và khi n đủ lớn thì nó đủ để khiến thuật toán vượt quá giới hạn. Thay vì thế, chúng ta sẽ hỏi sao cho để chia đôi được cả 2 tập hợp đã được chia ra cùng lúc (Cụ thể, hãy hỏi các công tắc từ 1...n/4, n/2+1...3n/4). Sau bước này, chúng ta sẽ phân biệt được 4 loại đèn ứng với các công tắc theo các truy vấn đã hỏi: các loại sáng ở cả 2 truy vấn, các loại tối ở cả 2 truy vấn, các loại sáng ở truy vấn #1, tối ở truy vấn #2, và các loại tối ở truy vấn #2, sáng ở truy vấn #1.
- Với truy vấn #3, làm tương tự, chúng ta sẽ chia được tập hợp các bóng đèn thành 8 tập hợp khác nhau dựa vào kết quả 3 truy vấn đã hỏi.
- Cứ chia như vậy, tương tự, sau log2(n) truy vấn, chúng ta biết được n bóng đèn ứng với n công tắc như thế nào.

Với tư tưởng chia tập hợp và hỏi gộp truy vấn như trên, bài toán này có thể giải quyết dễ dàng chỉ sau ceil(log2(1000)) = 10 truy vấn.
Về phần trình bày, các bạn hoàn toàn có thể chia n rồi cắt đôi từng đoạn như trên. Một cách đẹp mắt hơn là bạn có thể chia theo biểu diễn nhị phân của các chỉ số, kết hợp cùng với các toán tử bitmask để cho bài toán bớt cồng kềnh phần xử lý toán học. Cụ thể:
Truy vấn #1 bật các công tắc: 1, 3, 5, 7, … (các chỉ số có bit 0 bật)
Truy vấn #2 bật các công tắc: 2, 3, 6, 7, 10,... (các chỉ số có bit 1 bật)

Truy vấn #3 bật các công tắc: 4, 5, 6, 7, 12, … (các chỉ số có bit 2 bật)
…
Truy vấn #i bật các công tắc có bit (i-1) bật.

Khi đó, bạn nhận biết được công tắc nào nối với bóng đèn nào dựa vào trạng thái bit của chỉ số trên công tắc đó. Chẳng hạn, bạn biết công tắc 13 (nhị phân: 1101) nối với bóng đèn x nào đó khi bóng đèn x đấy bật trong truy vấn 1, tắt trong truy vấn 2, bật trong truy vấn 3, 4 và tắt trong tất cả các truy vấn sau đó).
Độ phức tạp: O(n*log2(n))
Code: https://pastebin.com/VyVUenr4

J. Justice for Ants

Tóm tắt đề bài

Cho một cây (đồ thị vô hướng không chu trình) với N đỉnh, mỗi đỉnh ban đầu được gán một giá trị. Có Q truy vấn, mỗi truy vấn là một cặp hai đường đi độ dài bằng nhau trên cây, yêu cầu các cặp đỉnh theo thứ tự xuất hiện trên hai đường đi cần phải có giá trị bằng nhau. Tại một bước, ta có thể thay đổi giá trị của một đỉnh một lượng $t * k + 1$ (cộng hoặc trừ) với t là một số nguyên bất kì và k là một số nguyên cho trước. Cần thay đổi giá trị của các nút theo cách trên để thỏa mãn các yêu cầu với số bước ít nhất có thể.

Giới hạn: $1 <= N <= 5 * 10^5$, $1 <= Q <= 2 * 10^5$, $1 <= k <= 10^6$.

Lời giải

Để đơn giản, ta sẽ chia bài toán ra làm hai phần chính:

- Cho một dãy các giá trị $a_1, a_2, ..., a_n,$ tính số thao tác nhỏ nhất để đưa các giá trị này về bằng nhau.
- Cho các truy vấn dạng hai đường đi như đề bài, nhóm tất cả các đỉnh cần có giá trị bằng nhau lại thành nhiều tập phân biệt.

Trước hết ta sẽ đi giải phần khó nhất của bài toán, bài toán con thứ nhất.

Đầu tiền quan sát các trường hợp nhỏ với ta có:

- $k = 0$ thì mỗi bước, ta có thể cộng hoặc trừ 1 một giá trị trong dãy, nên cách làm tối ưu là cộng trừ để đưa tất cả các số trong dãy về trung vị.
- $k = 1$ thì mỗi bước, ta có thể cộng hoặc trừ một giá trị trong dãy một lượng bất kì, nên cách làm tối ưu là đưa tất cả giá trị về giá trị xuất hiện nhiều lần nhất.

Với cá trường hợp k lớn, nhận thấy mỗi bước thay đổi, ta có thể thay đổi một giá trị lên một lượng $t * k$ bất kì, kèm theo $modulo\ k$ của giá trị đó thay đổi 1 hoặc -1.

Giả sử cần đưa số x trong dãy về một giá trị v nào đó, ta có thể làm như sau:

- Nếu $x = v$ số thao tác là 0.
- Nếu $x \bmod k \neq v \bmod k$, vì mỗi bước ta có thể nhảy lên một lượng $t * k$ bất kì nên chỉ duy nhất khoảng cách về modulo là quan trọng. Trong trường hợp này, số bước nhỏ nhất là $min(abs(x \bmod k - v \bmod k), k - abs(x \bmod k - v \bmod k))$.

- Nếu $x \bmod k = v \bmod k$ thì ta tốn đúng 2 bước để đưa x về v. (Bạn đọc tự c/m).

Vậy với mỗi $modulo\ k$ bất kì, ta có thể chia vòng tròn modulo thành các khoảng có công thức khoảng cách giống nhau, tính tổng số bước để di chuyển mỗi khoảng. Để duy trì các khoảng, ta có thể nhân đôi mảng modulo và sử dụng kĩ thuật two-pointer. Với các số có cùng $modulo\ k$, rõ ràng ta sẽ chọn số có số lần xuất hiện nhiều nhất để làm đích.

Các giá trị modulo quan trọng

Rõ ràng trong trường hợp xấu nhất, ta có thể có đến N tập phân biệt và độ phức tạp tính toán có thể lên tới $O(N * K)$.

Nhưng quan sát rằng với một dãy có M phần tử, ta có $O(M)$ giá trị modulo cần quan tâm. Với mỗi giá trị x trong dãy, đặt $f(v)$ là khoảng cách nhỏ nhất để đi từ $x \bmod k$ đến $v \bmod k$. Ta hoàn toàn có thể chia hàm $f(v)$ thành các khoảng đồng biến, lưu lại các chốt cực trị của $f(v)$ và **2 điểm xung quanh đó**. Dễ dàng chứng minh được với mọi x thì $f(v)$ có không quá 3 điểm cực trị phân biệt, do đó với mỗi x, ta có tối đa **9** giá trị modulo cần quan tâm. (Bạn đọc tự c/m).

Bài toán con thứ hai là từ các truy vấn cặp đường đi, đưa các số cần có giá trị bằng nhau về cùng một tập.

Trước hết, ta sẽ tìm cách giải quyết cho trường hợp dễ, với một cây bamboo (đoạn thẳng với $p_i = i - 1).$ Với trường hợp này, ta sẽ sử dụng kĩ thuật chia để trị, cụ thể như sau:

Xét $log_2(N)$ đồ thị phân biệt, hai đỉnh u và v liên thông ở đồ thị thứ i thì tất cả các cặp giá trị $(u, v), (u + 1, v + 1), ..., (u + 2^i - 1, v + 2^i - 1)$ cần phải bằng nhau.

Đến đây, rõ ràng cách truy vấn trong đề bài ta hoàn toàn có thể tách ra làm log cạnh trên log đồ thị.

Lần lượt xét các đồ thị từ $log_2(N)$ về 0, nếu hai đỉnh u, v liên thông với nhau ở đồ thị i thì ta cần thêm hai cạnh xuống đồ thị $i - 1$: $(u, v), (u + 2^{i-1}, v + 2^{i-1})$.

Rõ ràng, nếu ta duyệt qua tất cả các cặp đỉnh liên thông tại một đồ thị, thì độ phức tạp không khác gì so với thuật toán duyệt trâu. Nhưng có một nhận xét là tính liên thông có tính chất bắc cầu, do đó với mỗi thành phần liên thông trong đồ thị i ta chỉ cần chọn ra một đỉnh đại diện và thêm cạnh ở $i - 1$ từ đỉnh đó đến các đỉnh khác trong thành phần liên thông, như vậy, tính liên thông của tất cả các cặp đỉnh ở đồ thị $i - 1$ không thay đổi.

Quay lại bài toán với hai đường đi trên cây, ta nhận thấy rằng việc chia thành $log_2(N)$ đồ thị hoàn toàn có thể làm tương tự trên cây, chỉ thay đổi ở định nghĩa của các đỉnh. Bây giờ nếu hai đỉnh u và v liên thông với nhau ở đồ thị i thì tất cả các cặp giá trị sau cần phải bằng nhau: $(u, v), (p_u(1), p_v(1)), ..., (p_u(2^i - 1), p_v(2^i - 1))$ với $p_x(y)$ là tổ tiên thứ y của x.

Bài toán bây giờ trở nên phức tạp hơn, do mỗi đường đi có thể đổi hướng một lần tại LCA của hai mút. Do đó tại mỗi đồ thị, ta cần dựng thêm một đỉnh ngược của đỉnh u (đánh số là $u + N$) với ý nghĩa tương tự u nhưng theo chiều ngược lại.

Cụ thể việc chia đường đi vào các đồ thị thế nào thì rất dài nên mình xin nhường phần nháp + casework lại cho bạn đọc (hoặc xem code của mình ở đây).

Để practice kỹ thuật chia để trị ở trên các bạn có thể submit bài (hoặc xem editorial) ở đây.

K. Keep It Sorted

Tóm tắt đề bài

Có một hoán vị $a_1, a_2, ..., a_n$ $(n \leq 100)$. Mỗi phép biến đổi ta có thể thực hiện như sau:

- Chọn một đoạn con liên tiếp $[L, R]$ của hoán vị a sao cho đoạn con này đã được sắp tăng dần hoặc giảm dần.
- Đảo ngược đoạn con đó.

Cần thực hiện nhiều nhất 191 phép biến đổi để biến hoán vị a thành sắp xếp tăng dần.

Lời giải 1: Thuật chưa chuẩn

Với bài này có tương đối nhiều lời giải, đầu tiên hãy xem xét lời giải dùng $2n - 2$ phép biến đổi sau đây:

- Bắt đầu với 1 vị trí bất kì trong hoán vị a. Tạm gọi đó là vị trí i, tư tưởng là ta sẽ nở rộng phần tử i này dần dần ra hết hoán vị (nghĩa là lần lượt thêm các phần tử của hoán vị a. Đầu tiên, lần lượt thêm những vị trí $(i + 1), (i + 2), (i + 3), ..., n$ vào như sau: Giả sử hiện tại ta đã có dãy con $a_i, a_{i+1}, a_{i+2}, ..., a_l$ đã được sắp xếp tăng dần, ta cần thêm phần tử a_{l+1} vào dãy con kia nhưng vẫn giữ tính sắp xếp của nó.
    - Trường hợp $a_l < a_{l+1}$. Trường hợp này ta không tốn phép biến đổi nào, thêm phần tử $l + 1$ vào thì dãy con vẫn giữ tính sắp tăng dần.
    - Trường hợp $a_{l+1} < a_i$. Trường hợp này ta có thể đảo ngược dãy $[i, l]$ sau đó đảo ngược dãy $[i, l + 1]$.
    - Ngược lại, tìm k lớn nhất thuộc đoạn $[i, l]$ thoả $a_{l+1} < a_k$. Sau đó đảo ngược dãy $[k, l]$ rồi tiếp tục đảo ngược dãy $[k, l + 1]$.
- Việc thêm các phần tử từ vị trí $1, 2, ..., i - 1$ cũng có thể làm tương tự. Ta thấy mỗi lần thêm 1 phần tử vào dãy con đang xét thì mất nhiều nhất là 2 phép biến đổi, vậy trường hợp xấu nhất ta mất $2 * (n - 1) = 2n - 2$ phép biến đổi.

Nếu áp dụng thẳng thuật toán này thì ta không chắc sẽ dùng ít hơn 191 phép biến đổi được, tới đây ta có thể "cải tiến" nó như sau:

- Thay vì thêm toàn bộ phần bên phải $i + 1, i + 2, ..., n$ rồi sau đó mới thêm phần còn lại từ $1, 2, ..., i - 1$ thì ta có thể luân phiên làm xen kẽ việc thêm này để tận dụng những trường hợp không tốn phép biến đổi nào. Ngoài ra ta có thể thử chọn các vị trí bắt đầu khác nhau. Cách này không phải là chuẩn nhưng rất khó để sinh test giết, và thuật này AC hết bộ test.

Code AC có thể tham khảo ở đây: https://ideone.com/QfNGgI

Lời giải 2: Thuật thoả mãn số bước yêu cầu.

Thay vì biến đổi từ hoán vị cho trước về hoán vị chuẩn, ta sẽ tìm cách biến đổi hoán vị chuẩn về hoán vị của đề bài.
Có một cách làm rất đơn giản (tốn $2n - 2$ bước) như sau:

- Với mỗi 2 bước, ta sẽ cố gắng làm tăng độ dài prefix trùng nhau của 2 hoán vị lên 1.
- Ví dụ ta đang có hoán vị $(1, 2, 3, 4, 5, 6, ..., n)$ và hoán vị đề bài có dạng $(5, ?, ?, ?, ?, ..., ?)$, ta sẽ tìm cách đưa số 5 lên đầu, hay nói các khác là biến đổi hoán vị đang có thành: $(5, 1, 2, 3, 4, 6, ..., n)$. Việc này có thể thực hiện bằng 2 truy vấn $(1, R), (2, R)$, với R là vị trí của số đang cần đưa về đầu. Sau đó do 2 hoán vị đã có prefix trùng nhau nên ta có thể xem như ta đã xoá phần tử đầu tiên đi và làm lại từ đầu.
- Code tham khảo cho thuật này: https://ideone.com/GMrxGo

Một số nhận xét để cải tiến thuật ở trên:

- Sau mỗi 2 bước, prefix trùng nhau lại tăng lên 1, nói cách khác, phần khác nhau là phần suffix.
- Với n nhỏ ($n \leq 8$) ta có thể viết thuật toán BFS để tìm đường biến đổi ngắn nhất giữa các trạng thái. Sau khi khảo sát thì ta có một nhận xét sau: với hoán vị độ dài n, luôn có cách biến đổi nó về một hoán vị độ dài n khác với số bước nhiều nhất là $(n - 1)$.

Kết hợp 2 nhận xét trên ta có thuật toán như sau:

- Ban đầu dùng thuật toán $2(n-8)$ bước để tạo ra một hoán vị có prefix độ dài $(n-8)$ trùng với hoán vị đề bài. Sau đó 8 phần tử còn lại dùng BFS để biến đổi tiếp.
- Số bước cần dùng là: $2(n-8) + (8 - 1) = 2n - 9 \leq 2 * 100 - 9 = 191.$

Code tham khảo cho thuật trên: https://ideone.com/GlDAKQ

Câu hỏi

1. **Hỏi**: tại sao không dùng BFS cho lời giải 1, vì số bước cũng tương đương.
   Trả lời: Nên nhớ là ở lời giải 2, 8 phần tử khác nhau luôn nằm ở cuối cùng, vì vậy số trạng thái là 8! (vì ta chỉ tác động lên 8 phần tử đó). Còn ở lời giải 2, dù có 8 phần tử khác nhau nhưng nó vừa có thể nằm ở đầu, vừa có thể nằm ở cuối, vì vậy số trạng thái cần quan tâm là lớn hơn.

L. Latin Square

Tóm tắt đề bài

Một bảng ô vuông được gọi là bảng đẹp nếu mỗi số từ 1 đến n xuất hiện trong mỗi dòng và mỗi cột đúng 1 lần.

Cho một bảng ô vuông n*n, trong đó đã điền sẵn n*k số (gồm k số nguyên phân biệt, mỗi số xuất hiện đúng n lần) sao cho không tồn tại hai số nguyên cùng dòng hoặc cùng cột có giá trị giống nhau. Hãy điền số vào các ô còn lại để tạo thành một hình vuông đẹp.

Lời giải

Không mất tính tổng quát, giả sử n*k số đã điền sẵn gồm n số 1, n số 2, …, n số k. Ta gọi một bảng bất kì là bảng tốt nếu không tồn tại hai số nguyên cùng dòng hoặc cùng cột có giá trị giống nhau. Bảng từ dữ liệu vào ban đầu sẽ luôn là bảng tốt.

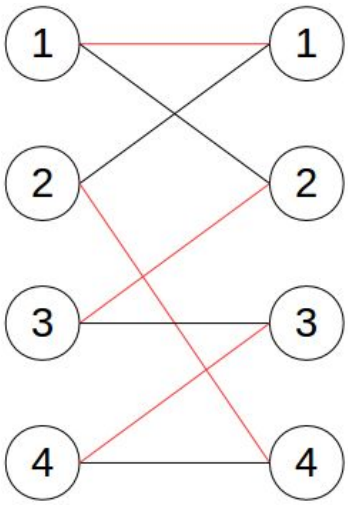
Trước hết, ta xét bài toán đơn giản hơn, chỉ yêu cầu điền tiếp n số k+1 vào bảng sao cho bảng vẫn tốt. Để thực hiện, ta cần chọn n ô trống trong bảng, sao cho không có hai ô nào cùng hàng hoặc cùng cột với nhau.

Bài toán nói trên có thể giải bằng thuật toán tìm bộ ghép cực đại. Ta tạo một đồ thị hai phía, phía bên trái gồm n đỉnh (đỉnh thứ i đại diện cho dòng thứ i), phía bên phải gồm n đỉnh (đỉnh thứ i đại diện cho cột thứ i). Nếu ô (i, j) là một ô trống, ta thêm cạnh nối giữa đỉnh i bên trái và đỉnh j bên phải. Khi đó, một bộ ghép hoàn hảo (một cách chọn n cạnh sao cho không có đỉnh nào kề quá 2 cạnh trong số các cạnh được chọn) sẽ tương đương với một cách chọn n ô trống trên bảng sao cho không có hai ô nào cùng hàng và cùng cột với nhau.

Ví dụ, với bảng:

		1	2
	1	2	
2			1
1	2		

Thì đồ thị hai phía tương ứng với bảng trên như sau:



Một bộ ghép hoàn hảo tìm được là (1, 1), (2, 4), (3, 2) và (4, 3). Tương ứng với bộ ghép trên là các ô (1, 1), (2, 4), (3, 2) và (4, 3) trong bảng.

3		1	2
	1	2	3
2	3		1
1	2	3	

Bài toán tìm bộ ghép cực đại trên đồ thị hai phía là một bài toán cổ điển, có thể được giải bằng thuật toán đường mở (Augmenting Path) trong O(VE) hoặc thuật toán Hopcroft-Karp trong $O(E\sqrt{V})$ với V là số đỉnh, E là số cạnh.

Đồng thời, ta cũng có nhận xét rằng, việc chọn n ô trống nói trên luôn có thể diễn ra. Với một bảng tốt bất kỳ mà mỗi số từ 1 đến k xuất hiện đúng n lần thì luôn tồn tại cách chọn n ô trống sao cho không có hai ô nào cùng hàng hoặc cùng cột với nhau. Nói cách khác, đồ thị hai phía tương ứng với bảng này luôn tồn tại bộ ghép hoàn hảo.

Ta sẽ chứng minh điều này bằng cách áp dụng định lý Hall (Hall's Marriage Theorem). Định lý được phát biểu như sau:

Gọi G là một đồ thị hai phía, với hai tập đỉnh của hai phía lần lượt là X và Y (không mất tính tổng quát, giả sử $|X| \le |Y|$). Với một tập con W của X, gọi $N_G(W)$ là tập các đỉnh thuộc Y kề với ít nhất một đỉnh thuộc W. Khi đó, tồn tại một bộ ghép bao phủ tất cả các đỉnh của X khi và chỉ khi với mọi tập con W của X, ta có $|W| \le |N_G(W)|$.

Đối với bài toán trên, tập X là tập đỉnh tương ứng với các dòng và tập Y là tập đỉnh tương ứng với các cột. Tập X và Y đều có kích thước N, do đó bộ ghép bao phủ tất cả các đỉnh của X cũng chính là bộ ghép hoàn hảo. Điều kiện của định lý Hall sẽ tương đương với điều kiện: với mọi tập S bất kì gồm p dòng trong bảng, sẽ luôn có ít nhất p cột có ít nhất một ô trống nằm tại một dòng thuộc tập S.

Ta xét một bảng tốt gồm các số từ 1 đến k (mỗi số xuất hiện đúng n lần). Trước hết, ta nhận xét rằng, mỗi dòng và mỗi cột của bảng sẽ luôn có đúng N - k ô trống (do mỗi dòng và mỗi cột đều chứa các số từ 1 đến k, mỗi số đúng 1 lần).

Xét một tập S gồm p dòng trong bảng. Giả sử có nhiều nhất p-1 cột có ít nhất một ô trống nằm tại một dòng thuộc tập S. Khi đó, số ô trống của các dòng thuộc tập S nhiều nhất sẽ là (N - k) * (p - 1). Mặt khác, do mỗi dòng có N - k ô trống, các dòng trong tập S sẽ có tổng cộng (N - k) * p ô trống (mâu thuẫn với việc các dòng thuộc tập S có nhiều nhất (N - k) * (p - 1) ô trống). Do đó, có ít nhất p cột có ít nhất một ô trống nằm tại một dòng thuộc tập S, và điều kiện của định lý Hall được đảm bảo.

Trở về với bài toán ban đầu, ta cần điền tiếp các số từ k+1 đến n vào bảng, mỗi số đúng n lần, sao cho bảng kết quả là bảng tốt. Để giải bài toán này, với mỗi x từ k+1 đến n, ta sẽ tìm tập n ô còn trống trên bảng sao cho không có hai ô nào cùng hàng hoặc cùng cột với nhau, rồi điền số x vào n ô tìm được. (ta cũng nhận thấy rằng không có trường hợp nào in ra ‘NO’).

Độ phức tạp: $O(n^3)$ với n là kích thước của bảng.

M. Moscow Dream

Tóm tắt đề bài

Một bộ đề được xem là hay nếu như nó có đúng n bài, trong đó có ít nhất 1 bài dễ, 1 bài trung bình, 1 bài khó. Hiện có a bài dễ, b bài trung bình, c bài khó. Hãy xác định xem có thể tạo được một bộ đề đẹp hay không.

Lời giải

Giới hạn bài này khá nhỏ, ta có thể làm với độ phức tạp $O(a * b * c)$ như sau:

- Duyệt x, y, z lần lượt là số bài dễ, trung bình, khó, kiểm tra $x + y + z = n$ có đúng hay không.

Hoặc cách làm $O(1)$:

- Để tồn tại ít nhất 1 bài dễ, trung bình, khó, hiển nhiên phải thoả điều kiện: $(a >= 1 \&\& b >= 1 \&\& c >= 1)$, xong ta cần kiểm tra có thể tạo được n bài, nên cần kiểm tra tổng $(a + b + c >= n)$. Chưa hết, để tạo được đề thì ta cần thêm điều kiện n phải lớn hơn hoặc bằng 3, mình nghĩ rất nhiều đội sai bài này vì lỗi không kiểm tra điều kiện đó. Tóm lại, ta chỉ cần kiểm tra: $(a >= 1 \&\& b >= 1 \&\& c >= 1 \&\& a + b + c >= \&\& n >= 3)$.

Code tham khảo: https://ideone.com/xB527H