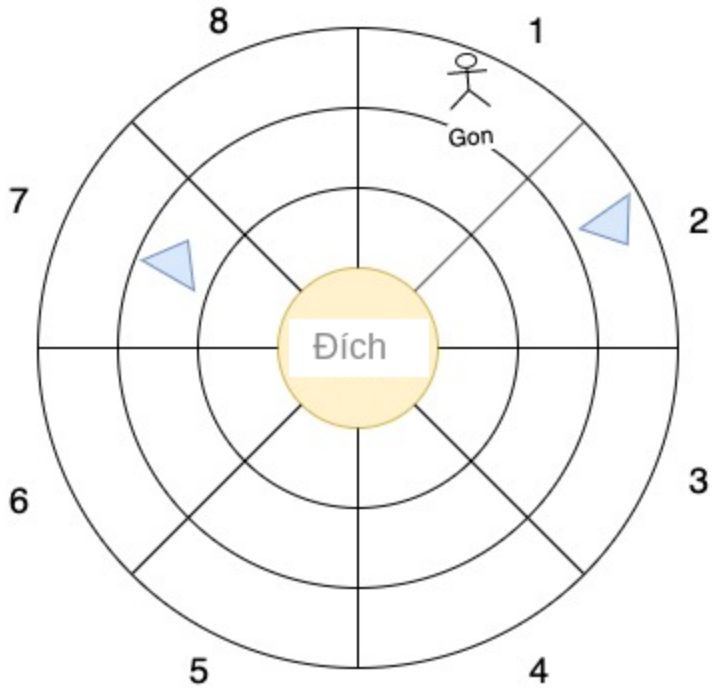
Tổng quan

	Số đội AC	Độ khó	Tác giả	Lời giải
A	32	**	Nguyễn Thành Trung (RR)	Phạm Đức Tú
B	57	**	Lăng Trung Hiếu	Trần Tấn Phát
C	53	*	Lăng Trung Hiếu	Trần Quỳnh Mai
D	10	**	Nguyễn Vương Linh	Lê Duy Thức
E	0	***	Lăng Trung Hiếu	Đặng Đoàn Đức Trung
F	2	***	Lăng Trung Hiếu	Đặng Đoàn Đức Trung
G	142	*	Lê Minh Hoàng	Trần Quỳnh Mai
H	1	****	Lăng Trung Hiếu	Đặng Đoàn Đức Trung
I	0	****	Lăng Trung Hiếu	Lê Duy Thức
J	0	***	Phan Đức Nhật Minh	Lê Quang Tuấn
K	9	**	Lăng Trung Hiếu	Đặng Đoàn Đức Trung
L	3	***	Phạm Văn Hạnh	Lê Duy Thức

A. A New Adventure

Tóm tắt đề bài

Ta cần đi đến đích; bao quanh đích là C đường tròn lần lượt đánh số từ 1 đến C từ ngoài vào trong, mỗi đường tròn có A ô đánh số từ 1 đến A theo chiều kim đồng hồ. Một số ô có xe máy, các xe máy này sẽ di chuyển với tốc độ 1 ô/s theo chiều kim đồng hồ. Gon xuất phát từ ô (1, 1), tìm thời gian ngắn nhất để di chuyển đến ô đích mà không bị xe máy tông. Ví dụ với C = 3, A = 8:



Lời giải

- Nhận xét trạng thái tất cả xe máy di chuyển sang 1 ô theo chiều kim đồng hồ tương đương với trạng thái Gon di chuyển 1 ô ngược chiều kim đồng hồ.
- Do đó ta có thuật toán: Cho Gon di chuyển 1 ô ngược chiều kim đồng hồ thay vì để tất cả xe máy di chuyển, lúc này ta có thể dễ dàng BFS để tìm thời gian ngắn nhất để Gon tới đích.
- Chú ý xét lại điều kiện để Gon không bị tông với trạng thái cho Gon di chuyển 1 ô ngược chiều kim đồng hồ.

Độ phức tạp: O(C * A).
Code: https://ideone.com/2GzNSq

B. Breaking Cake

Tóm tắt đề bài.

Cho m điểm phân biệt trên hệ tọa độ 3 chiều nằm trong hình hộp chữ nhật có 2 góc đối tọa độ (1, 1, 1) và $(R_0,\ R_1,\ R_2)$. Cần chia hình hộp ra m phần sao cho mỗi phần chứa đúng 1 điểm, 2 phần bất kỳ không giao nhau, hợp các phần đúng bằng hình hộp ban đầu.

Bài toán tổng quát + Chứng minh

Để cài đặt đơn giản, ta sẽ giải bài toán tổng quát cho k chiều, sau đó áp dụng k = 3.

Giả sử tọa độ của 1 điểm x trên trục k chiều có dạng $(x_0, x_1, ...,\ x_{k-1})$.

Đề bài đưa về ta có 1 tập điểm $\{x_0,\ x_1, ..., x_{m-1}\}$ và 1 hình hộp chữ nhật k chiều S, sao cho các điểm đều nằm trong hình hộp. Giả sử S có dạng $((L_0, R_0), (L_1, R_1), ..., (L_{k-1}, R_{k-1}))$ ($L_i \le R_i$ là khoảng đóng mà S phủ trên trục thứ i).

Gọi tập các tọa độ phân biệt có được từ $x_{0,\,k-1}, x_{1,\,k-1}, ..., x_{m-1,\,k-1}$ là $c_0 < c_1 < ... < c_{t-1}$ (t là số tọa độ phân biệt). Gán lại $c_0 = L_{k-1}$, $c_{t-1} = R_{k-1} + 1$.

Sử dụng dãy c, ta có thể cắt hình S ra làm t phần không giao nhau trên trục k, từ đấy quy về bài toán k - 1 chiều. Cụ thể:

Ta sẽ chia hình S thành t phần có dạng $S'_i = ((L_0, R_0), (L_1, R_1), ..., (c_i, c_{i+1} - 1))\ (0 \le i < t)$. Dễ thấy rằng $\bigcap_{i=0}^{t-1} S'_i = S$.

Với mỗi điểm $x_i\ (0 \le i < k)$ dễ dàng tìm được j sao cho $x_i \in S'_j$

Nếu k = 1, ta đã tìm ra được 1 cách chia đúng vì các điểm có tọa độ phân biệt.

Nếu k > 1, với mỗi S'_i, ta có 1 tập điểm thuộc hình hộp đấy có dạng $x'_i = \{x'_{i,\,0},\ x'_{i,\,1}, ..., x'_{i,\,m'-1}\}\ (m' < m)$.

Ta có $S'_i = ((L_0, R_0), (L_1, R_1), ..., (L_{k-2}, R_{k-2}),\ (c_i, c_{i+1} - 1))$.

$$x'_{i,j} = (x'_{i,j,\,0}, x'_{i,j,\,1}, ...,\ x'_{i,j,\,k-2},\ x'_{i,j,\,k-1})$$

Đặt $S''_i = ((L_0, R_0), (L_1, R_1), ..., (L_{k-2}, R_{k-2}))$.

$$x''_{i,j} = (x'_{i,j,\,0}, x'_{i,j,\,1}, ...,\ x'_{i,j,\,k-2}).$$

Nói đơn giản, ta bỏ qua tọa độ trục thứ k - 1.

Với 1 cách chia tập hình hộp S''_j cùng tập điểm x''_j thỏa điều kiện đề bài ,ta có 1 cách chia tập hình hộp S'_j cùng tập điểm x'_j tương ứng bằng cách khôi phục tọa độ trục thứ k - 1.

Sử dụng đệ quy, giải tương tự cho tập hình hộp S''_j cùng tập điểm x''_j

Tham khảo cài đặt cụ thể tại: https://ideone.com/xdtz5c

ĐPT: O(kmlog(m)) với m là số điểm, k là số chiều.

C. Checking Break

Tóm tắt đề bài

Cho m miếng bánh và m viên socola trong hệ tọa độ 3 chiều. Kiểm tra xem cách phân chia này có hợp lệ (miếng bánh thứ i chứa viên socola thứ i, không có phần nào của miếng bánh ban đầu bị bỏ thừa, không có 2 miếng bánh nào trùng nhau, tọa độ miếng bánh hợp lệ)

Lời giải

Với mỗi test case, ta kiểm tra theo đúng yêu cầu đề bài:

- Miếng bánh thứ i chứa viên socola thứ i: Duyệt từng miếng bánh, viên socola thứ i được coi là nằm trong miếng bánh thứ i nếu tọa độ cả 3 chiều của viên socola đều nằm trong khoảng tọa độ của chiều tương ứng của miếng bánh.
- Không có phần nào của cái bánh ban đầu bị bỏ thừa: Tính tổng thể tích tất cả các miếng bánh, nếu tổng thể tích cái bánh bằng tổng thể tích tất cả các miếng bánh thì hợp lệ.
- Không có 2 miếng bánh nào trùng nhau: Lồng 2 vòng lặp duyệt 2 miếng bánh đôi 1, kiểm tra xem 2 miếng bánh có giao nhau không. 2 miếng bánh được coi là giao nhau khi $\max(x_i, x_j) \leq \min(u_i, u_j)$, $\max(y_i, y_j) \leq \min(v_i, v_j)$, $\max(z_i, z_j) \leq \min(w_i, w_j)$.
- Tọa độ miếng bánh hợp lệ: Kiểm tra như phương trình của đề bài.

Độ phức tạp: $O(M^2)$ * số test case

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

D. Dropping Ball

Tóm tắt đề bài

Có K ma trận $R + 1 * C$ "tương tự" nhau được xếp chồng lên nhau. Hàng cuối cùng (hàng R + 1) của mỗi ma trận có chứa C số tượng trưng cho điểm của từng ô (được thể hiện bởi mảng **a**), khi quả bóng đi qua các ô này thì tổng điểm sẽ tăng lên bằng đúng giá trị của ô đó. Ta cần thả một quả bóng tại ô đầu tiên của cột bất kỳ, sao cho khi quả bóng rơi xuống thì tổng điểm đạt được là lớn nhất.

Trong R hàng đầu tiên của mỗi ma trận, mỗi hàng có C ô thuộc 1 trong 5 loại sau:

- Ô trống: quả bóng khi đi qua ô này sẽ tiếp tục rơi xuống ô bên dưới nó.
- Ô chứa ký tự 'L': quả bóng sẽ đi chuyển sang trái đúng 1 ô.
- Ô chứa ký tự 'R': quả bóng sẽ di chuyển sang phải đúng 1 ô.
- Ô cấm: khi quả bóng đi vào ô này, trò chơi kết thúc.
- Ô chứa ký tự '?': Ta có thể thay thế ô này thành ký tự 'L' hoặc 'R'. Lưu ý là các ký tự thay thế cho '?' trong K ma trận không cần phải giống nhau.

Trò chơi sẽ kết thúc khi quả bóng:

- Rơi khỏi bảng (rơi hết toàn bộ K ma trận)
- Đi ra ngoài bảng
- Đi qua ô cấm
- Quả bóng di chuyển quá 10^{20} lần (trường hợp này xảy ra khi rơi vào cặp ô RL).

Lời giải

Đầu tiên, ta thử tính kết quả trong trường hợp K = 1 (chỉ có một ma trận duy nhất): Ta có thể thử hết tất cả các ô bắt đầu, sau đó simulate lại quá trình rơi của quả bóng, khi gặp ký tự '?' ta thử thay nó bằng 'L' hoặc 'R'. Cách này có thể làm đơn giản bằng thuật toán BFS.

Để có thể giải được với các K lớn hơn, ta cần dựng mảng $D[i][j]$ sao cho, $D[i][j] = a[j]$ khi ta bắt đầu ở ô $(1, i)$ và có thể đi tới ô $(R + 1, j)$, ngược lại ta xem $D[i][j] = \infty$.

Để xử lý trưởng hợp trò chơi kết thúc "sớm" (kết thúc khi chưa rơi hết K ma trận), ta có thể dựng thêm một cột ảo thứ $C + 1$ ($D[i][C + 1] = 0$ nếu ta có thể kết thúc trò chơi (đi vào ô cấm hoặc lặp vô hạn)). $D[C + 1][i] = -\infty$ với mọi $i \neq m + 1$.

Ta có thể tính mảng D bằng cách BFS từ từng cột, khi gặp các ô '?' thì ta có thể xem như đó vừa là 'L' và vừa là 'R'. Mỗi lần BFS ta tốn độ phức tạp là $O(R * C)$, vậy tổng độ phức tạp để khởi tạo mảng D là $O(C * R * C)$.

Nếu $K = 1$, kết quả đơn giản chỉ là số lớn nhất trong mảng D. Với các K lớn hơn, ta có thuật quy hoạch động như sau:

- Gọi $F[i][j]$ là tổng điểm lớn nhất có thể nhận được khi ta xét j ma trận đầu tiên, và ta đang đứng ở ô $(R + 1, i)$ của ma trận j.
- Khởi tạo $F[1][j] = max(D[1, 2, 3, \ldots n][j])$

- $F[i][j] = max(F[x][j-1] + D[x][i]),\ \forall\ x \in [1,\ C+1]$. Lý giải công thức này đơn giản là khi ta muốn kết thúc ở ô $(R+1,\ i)$ của ma trận thứ j, ta có thể duyệt qua mọi cách kết thúc ở ma trận $j-1$, sau đó lấy max tất cả các cách.
- DPT của cách làm này là $O(C * k * C)$ (ta có $O(C * k)$ trạng thái F[i][j], mỗi trạng thái chi phí chuyển là $O(C)$).

Với cách làm trên thì chưa thể AC được vì K lên đến 10^9, nhưng ta nhận xét để tính $F[?][j]$ thì ta chỉ cần $F[?][j-1]$, và mảng D giữa các ma trận là không thay đổi, vì thế nên ta có thể dùng nhân ma trận để tăng tốc thuật toán. Đây là một kỹ thuật khá thường dùng, các bạn có thể tham khảo ở VNOI wiki;

Cụ thể hơn, ở bài này ta cần thay đổi operator "nhân" trong phép nhân ma trận là phép lấy tổng, phép "cộng" là phép lấy max (xem thêm ví dụ 3 của nhân ma trận)

Sau đó ta có phép nhân như sau:

$$\begin{bmatrix} D[1][1] & D[2][1] & \dots & D[C+1][1] \\ D[1][2] & D[2][2] & \dots & D[C+1][2] \\ \dots & \dots & \ddots & \dots \\ D[1][C+1] & D[2][C+1] & \dots & D[C+1][C+1]] \end{bmatrix} * \begin{bmatrix} F[1][j] \\ F[2][j] \\ \vdots \\ F[C+1][j] \end{bmatrix} = \begin{bmatrix} F[1][j+1] \\ F[2][j+1] \\ \vdots \\ F[C+1][j+1] \end{bmatrix}$$

Các bạn có thể tham khảo code của đội mình ở đây: https://ideone.com/wc9nYd

E. Easy Probability

Tóm tắt đề bài

Có 2 người chơi, người chơi đầu tiên có xâu g, người chơi thứ hai có xâu k (cả 2 xâu chỉ gồm kí tự H và T). Ta xét một xâu s bất kì có 10^100 kí tự chỉ gồm H và T. Định nghĩa hàm pos(x, y) nhận 2 xâu x, y chỉ gồm kí tự H và T trả về giá trị w, nếu w là vị trí nhỏ nhất mà tiền tố độ dài w của x nhận y là xâu con. Người chơi đầu tiên thắng khi pos(s, g) < pos(s, k); người chơi thứ hai thắng khi pos(s, g) > pos(s, k); hai người chơi hoà khi pos(s, g) = pos(s, k). Tính xác suất người chơi đầu tiên thắng với tất cả các trường hợp của xâu s.

Lời giải

Ta sẽ giải bài toán con sau: Biết được xâu biểu diễn một ván chơi, kiểm tra xem ở ván chơi này Gon thắng/thua/hoà. Ta có thể giải bài toán này bằng cách dựng Aho-Corasick automaton của xâu g và k, rồi duyệt qua từng kí tự của xâu ván chơi. Khi duyệt qua từng kí tự, ta nhảy tới node tiếp theo trên automaton, rồi kiểm tra xem lúc ấy xâu g hoặc xâu k có xuất hiện hay chưa.
Quay lại bài toán ban đầu, ta có thể định nghĩa một xâu ván chơi với 2 giá trị: trạng thái của ván chơi (thắng/hoà/thua/vô định) và node hiện tại của ván chơi trên automaton (vô định ở đây tức là cả g và k đều chưa xuất hiện). Khi thêm kí tự H hoặc T vào, ta sẽ nhảy tới một trạng thái mới, và vì ta có thể tính sẵn trước ở mỗi trạng thái, khi thêm kí tự H/T sẽ nhảy tới trạng thái nào với khả năng bao nhiêu, ta có thể làm quy hoạch động pr[i][j], với i là số lượng kí tự hiện tại của ván đấu và j là trạng thái của ván đấu tới kí tự i.

Đến đây ta có 2 cách để tính khả năng xảy ra của mỗi trạng thái khi i = 10^100:

1. Ta có thể sử dụng nhân ma trận. Độ phức tạp là O((|g| + |k|)^3 * log(10^100)).
2. Vì 10^100 rất lớn, nên ta có thể coi như là vô cực. Khi đó, ta có thể coi như khả năng xảy ra của mỗi trạng thái là một hằng số (tức là khi thêm 1 kí tự thì khả năng xảy ra của mỗi trạng thái giữ nguyên). Lúc này, coi như mỗi cách chuyển tiếp trạng thái là một phương trình, cùng với phương trình "tổng khả năng của mọi trạng thái = 1", ta có thể giải hệ phương trình này bằng khử Gauss. Độ phức tạp là O((|g| + |k|)^3).

Cuối cùng, đáp án cho bài sẽ là (tổng khả năng các trạng thái thắng) / (1 - tổng khả năng các trạng thái vô định). Nhận thấy khi s có độ dài rất lớn thì khả năng s vô định là rất nhỏ, nên ta chỉ cần lấy tổng khả năng các trạng thái thắng là được.

Cách 2 sẽ tạo ra sai số, nên cần cẩn thận khi cài đặt (có thể thay thế floating point bằng phân số big num, hoặc dùng thư viên BigDecimal của Java và Python, …)

F. Final Exam

Tóm tắt đề bài

Cho 3 mảng số A, B, C mỗi mảng có n phần tử. Đếm số bộ số (i, j, k) mà a[i], b[j], c[k] lần lượt là 3 cạnh của 1 tam giác vuông, và c[k] là cạnh huyền của tam giác này.

Lời giải

Với 2 số nguyên m và n sao cho gcd(m, n) = 1 và có đúng 1 số trong m và n chia hết cho 2, ta có thể tạo ra một bộ số Pytago nguyên thuỷ (bộ số Pytago có gcd = 1) như sau:

$$a = m^2 - n^2$$
$$b = 2mn$$
$$c = m^2 + n^2$$

Nếu chạy trâu, ta có thể thấy số lượng bộ số Pytago không quá lớn (mình chạy trâu thì được cỡ 150 triệu bộ số có cạnh huyền không quá 30 triệu). Với mỗi bộ số Pytago nguyên thuỷ, trước tiên ta kiểm tra xem có tồn tại phần tử trong mảng A chia hết cho a, phần tử trong mảng B chia hết cho b, phần tử trong mảng C chia hết cho c không (thật ra chỉ cần kiểm tra 1 trong 3 điều kiện này là đủ). Nếu có tồn tại, ta duyệt qua các k để tạo ra một bộ số Pytago mới có dạng

a' = ka
b' = kb
c' = kc

rồi ta thêm (số lượng phần tử trong A bằng a') * (số lượng phần tử trong B bằng b') * (số lượng phần tử trong C bằng c') vào đáp án.

Chú ý

Ta cần phải duyệt qua m, n để tạo ra các bộ số Pytago nguyên thuỷ. Khi duyệt, ta cần phải kiểm tra xem gcd(m, n) = 1 hay không. Để giảm độ phức tạp đoạn này đi log lần, ta có thể xây một mảng quy hoạch động f[i][j] trả về giá trị là gcd(i, j).

G. Gathering In Yorknew

Tóm tắt đề bài

Có n điểm 1 đường thẳng. Hãy tính tổng khoảng cách nhỏ nhất từ n điểm đó tới 0 giả sử có 1 đường tắt giữa 2 điểm mà khoảng cách giữa chúng bằng 0.

Lời giải

Vì vị trí của các điểm trải dài từ -10^9 đến 10^9:

- Đường tắt nối 1 điểm âm đến 0: một số điểm bên âm được lợi, các điểm bên dương không được lợi gì vì muốn sang điểm bên âm đã phải đi qua 0 (tương tự cách nối 1 vị trí dương đến 0)
- Đường tắt nối 1 điểm âm đến 1 điểm dương:
    - Nếu khoảng cách ban đầu từ điểm bên âm đến 0 > khoảng cách ban đầu từ điểm bên dương đến 0: một số điểm bên âm được lợi nhưng các điểm bên dương sẽ không được lợi gì, và nếu như thế thì để đường tắt nối 1 điểm bên âm đến 0 sẽ có lợi hơn cho các điểm bên âm (tương tự nếu khoảng cách ban đầu từ điểm bên âm đến 0 < khoảng cách ban đầu từ điểm bên dương đến 0)
    - Nếu khoảng cách ban đầu đến 0 từ điểm bên âm = khoảng cách ban đầu đến 0 từ điểm bên dương, không bên nào có lợi.
- Đường tắt nối 1 điểm âm đến 1 điểm âm (hoặc dương tới dương): tương tự, nối trực tiếp đến 0 sẽ tối ưu hơn

Như vậy ta sẽ chỉ cần quan tâm đặt đường tắt ở 1 vị trí nào đó và vị trí 0. Ta sẽ giải bài toán với bên dương trước (bên âm làm tương tự).

- Chia mảng ban đầu thành 2 mảng âm dương riêng biệt. Gọi số lượng phần tử trong mảng dương là numPos. Sắp xếp lại các mảng dương tăng dần.
- Dựng 1 mảng tổng cộng dồn khoảng cách đến 0 của các điểm, gọi là sumPos[]. Duyệt từng phần tử i ở mảng dương, với định nghĩa là những điểm từ i trở về trước khoảng cách ngắn nhất đến 0 giữ nguyên, từ i + 1 trở về sau thì khoảng cách ngắn nhất sẽ là đi đến điểm trung vị của đoạn đó và có đường tắt từ trung vị tới 0.
- Kết quả tương ứng với mỗi phần tử gồm tổng của 4 thành phần sau:
    - Tổng khoảng cách từ 0 đến các điểm dương từ 1 đến i: *sumPos[i]*
    - Tổng khoảng cách từ 0 đến các điểm dương từ i+1 đến trung vị: (*số lượng phần tử từ (i + 1) đến trung vị) * pos[trung vị] – (sumPos[trung vị] – sumPos[i])*
        - Vì tổng khoảng cách của các điểm trước trung vị đến trung vị là (pos[trung vị] – pos[i + 1]) + (pos[trung vị] – pos[i + 2]) + …
    - Tổng khoảng cách từ 0 đến các điểm âm từ (trung vị + 1) đến n: *(sumPos[numPos] – sumPos[trung vị]) – (số lượng phần tử từ (trung vị + 1) đến numPos) * pos[trung vị]*

        - Tương tự như trên
    - Tổng khoảng cách từ 0 đến các điểm âm
- Kết quả của bài là tổng nhỏ nhất tính theo công thức trên.

Độ phức tạp: O(N * log N) (do sort mảng)
Code tham khảo: https://ideone.com/ksdS58

H. Hunter x Communication

Tóm tắt đề bài

Ta có thao tác biến đổi một số A bằng cách lấy hai chữ số kề nhau rồi cùng tăng hay giảm 2 chữ số đó 1 đơn vị (không được tăng 9, không được giảm 0, không được có chữ số 0 đứng đầu). Đếm số lượng cặp có thứ tự (a, b), sao cho cả a và b đều có N kí tự, và từ a có thể dùng một lượng hữu hạn các thao tác trên để biến đổi thành b. Đáp án lấy dưới modulo 998244353.

Hướng giải

Nhận xét: Khi ta làm bất kì thao tác nào, tổng các chữ số ở vị trí chẵn - tổng các chữ số ở vị trí lẻ không thay đổi (mình sẽ gọi hiệu trên là trạng thái của một số). Thêm nữa, với hai số bất kì có trạng thái, ta có thể biến đổi số này thành số kia. Vì thế, với mỗi giá trị x, ta có thể tìm A là số lượng số có trạng thái = x, thì đáp án của chúng ta là tổng A^2 với mọi giá trị x.

Ta có thể tìm các giá trị A trên bằng quy hoạch động. Gọi $f[i][j]$ là số lượng các số có i chữ số và trạng thái là j, thì với vị trí i chẵn, $f[i][j] = \sum_{k=0}^{9} f[i-1][j-k]$; với i lẻ, $f[i][j] = \sum_{k=0}^{9} f[i-1][j+k]$. Ta quy hoạch động đến chữ số thứ N - 1, rồi ta tiếp tục quy hoạch động ở chữ số thứ N, nhưng ta không thêm chữ số 0 ở dòng này. Độ phức tạp là O(N * MAX), với N = 10^5 và MAX là số lượng trạng thái tối đa = 10^5 * 9, nên ta không thể áp dụng trực tiếp cách quy hoạch động này được. Tuy nhiên, ta có thể tối ưu cách làm này.

Lời giải

Nếu ta biểu diễn dòng thứ i của mảng quy hoạch động bằng một đa thức, mà hệ số của x^j trong đa thức này bằng f[i][j], thì ta nhận thấy rằng đa thức biểu diễn dòng thứ i + 1 của mảng quy hoạch động trên bằng tích của đa thức dòng i với một đa thức P. Cụ thể, với i chẵn, $P_{chẵn} = \sum_{k=0}^{9} x^k$, và với i lẻ, $P_{lẻ} = \sum_{k=0}^{9} x^{-k}$. Ta có thể chứng minh được ở dòng thứ N - 1, đa thức biểu diễn dòng này là $(P_{chẵn})^{(N-1)\ div\ 2} \times (P_{lẻ})^{N\ div\ 2}$. Ta có thể tính nhanh bằng FFT (NTT vì mod là 998244353) và luỹ thừa nhanh.
Với cách làm trên sẽ thu được thuật toán có độ phức tạp O(N log^2 N). Ta có 2 cách chính để giảm độ phức tạp xuống:

1. Ta chỉ cần FFT hai đa thức trên một lần, rồi đem lũy thừa/nhân các đa thức ở dạng FFT. Cuối cùng, ta làm FFT inverse một lần duy nhất.
2. Để làm đơn giản hoá bài toán thì ở mỗi vị trí lẻ, thay vì x^{-k}, ta lấy x^{9-k}, vì vậy đa thức cuối cùng cần tính là

$(x^1 + x^2 + ... + x^9)(1 + x^1 + x^2 + ... + x^9)^{N-1} = x\frac{x^9-1}{x-1}(\frac{x^{10}-1}{x-1})^{N-1} = \frac{x(x^9-1)(x^{10}-1)^{N-1}}{(x-1)^N}$ Có thể dùng tổ hợp để tính các giá trị trong khai triển $(x^{10}-1)^{N-1}$ và $(x-1)^N$ sau đó dùng FFT inverse mẫu và tính tích.

Độ phức tạp: O(N log N). Bạn có thể tham khảo code của mình ở đây.

I. IMO Harder Problem

Tóm tắt đề bài

Cho một xâu A chỉ gồm các ký tự ‘H’, ‘T’, ta biến đổi xâu A như sau:

- Gọi K là số ký tự ‘H’ trong xâu A.
- Đổi A[K] từ ‘H’ sang ‘T’ hoặc ngược lại.

Lặp lại đến khi A không còn ký tự ‘H’ nào.
Ta gọi $L(A)$ là số bước thực hiện để biến đổi toàn bộ xâu A thành ký tự ‘T’.
Đề bài cho một xâu S gồm các ký tự ‘H’, ‘T’, ‘?’. Từ S, ta có thể dựng tổng cộng 2^M xâu chỉ gồm ‘H’ và ‘T’ bằng cách thay các ký tự‘?’ thành 2 ký tự này (M là số dấu ‘?’ của xâu S). Ta cần tính giá trị trung bình $L(A)$ của 2^M xâu kia.

Lời giải:

Bài này quan trọng nhất là ta phải tìm được công thức đếm được số bước nếu cho trước 1 trạng thái.

Nếu cho trước một trạng thái xâu A chỉ gồm H, T. Thì số bước để biến đổi cả xâu S về trạng thái TTTT...TTT được tính bởi công thức sau đây:
$L(A) = 2 * (a[1] + a[2] + a[3] + ... + a[k]) - k^2$, với k là số lượng ký tự H, a[i] là vị trí của ký tự H thứ i. Chứng minh: (https://artofproblemsolving.com/community/c6h1876772p12756367)

Giờ ta đi vào bài toán chính.
Gọi nH là số lượng ký tự H của xâu đề bài, nU là số lượng ký tự '?'. Đề yêu cầu ta tính giá trị trung bình của mọi L(S) nên ta có thể tính tổng giá trị của mọi L(S) sau đó chia cho 2^{nU};
Ta sẽ chia L(S) ra làm 2 phần:

1. Tổng $sum = 2 * (a[1] + a[2] + ... + a[k])$:
   - Nên nhớ ta có thể viết tổng trên dưới dạng:
     - $sum = 2 * (1 * (S[1] == 'H') + 2 * (S[2] == 'H') + ... + n * (S[n] == 'H'))$
   - Do đó, tổng sum của 2^{nU} trạng thái có thể viết dưới dạng:
     - $2 * (1 * F(1) + 2 * F(2) + 3 * F(3) + ... + n * F(n))$. Với F(i) là số lượng trạng thái mà ký tự tại vị trí i bằng 'H'.
2. Tổng tất cả k^2, với k là số lượng ký tự 'H'. Ta có thể tính trâu tổng này bằng cách:
   - $\sum_{i=0}^{nU} (nH + i)^2 * C(i, nU)$ ($C(i, nU)$ là số lượng tổ hợp chập i của nU). Về mặt lý thuyết, ta có thể tính tổng này trong O(N) nhưng sẽ xảy ra vấn đề sai số vì ta cần chia kết quả cho 2^{nU}.

- Ta biến đổi như sau:
$$\sum_{i=0}^{nU}(nH + i)^2 * C(i, nU) = \sum_{i=0}^{nU}(nH^2 + i^2 + 2 * nH * i) * C(i, nU)$$

Ta chia ra làm 3 tổng:

$$S1 = nH^2 * \sum_{i=0}^{nU} C(i, nU);\ S2 = 2 * nH * \sum_{i=0}^{nU} i * C(i, nU);\ S3 = \sum_{i=0}^{nU} i^2 * C(i, nU);$$

- Tổng đầu tiên $S1 = nH^2 * 2^{nU}$.
- Tổng thứ 2: $S2 = 2 * nH * nU * 2^{nU-1}$ (Do dài nên mình không đề cập cách tính chi tiết ở đây, các bạn có thể tính bằng cách khai triển nhị thức newton của (X+ 1)^ N và đạo hàm 2 vế, sau đó thế X = 1).
- Tổng thứ 3: $S3 = nU * (nU + 1) * 2^{nU-2}$ (Do dài nên mình không đề cập cách tính chi tiết ở đây, cũng dùng đạo hàm khai triển nhị thức newton).

Thuật toán.

Chứng minh toán thì dài dòng nhưng code bài này khá đơn giản:

1. Tính $2 * (a[1] + a[2] + a[3] + ... + a[k])$:
   - Xác suất để có 'H' ở vị trí 'H' và '?' lần lượt là 1.0 và 0.5 nên ở mỗi vị trí i là 'H' ta sẽ cộng thêm giá trị 2i và i là '?' sẽ cộng thêm giá trị i
2. Tính k^2:
   - Để tính $E(k^2)$ ta phân tích k^2 thành tổng $(x[1] + x[2] + ... + x[n])^2$ với $x[i] = 1$ nếu giá trị i là 'H' và $x[i] = 0$ nếu giá trị i là 'T' từ đây ta cần tính tổng của các $x[i]^2$ và $x[i] * x[j]$ với i khác j
     - Tính tổng các $x[i]^2$: với i là 'H' ta sẽ cộng thêm 1, với i là '?' cộng thêm 0.5
     - Tính tổng các $x[i] * x[j]$: với i và j đều là 'H' ta sẽ cộng thêm 1, với i và j là 'H' và '?' sẽ cộng thêm 0.5 với i và j đều là '?' sẽ cộng thêm 0.25.
   - Gọi nH là số lượng 'H' và nU là số lượng '?' kết quả cuối cùng sẽ là:
     $nH * nH + nH * nU + nU * (nU + 1) * \frac{1}{4}$

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

J. Just Enough Bridges

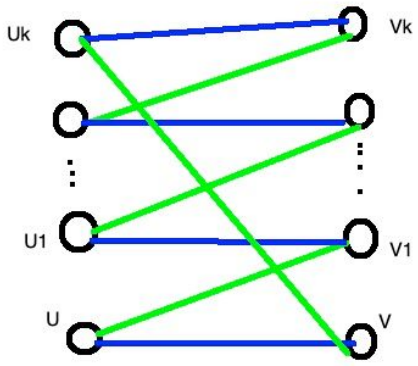
Lời giải: Lê Quang Tuấn (unsigned - Đại học công nghệ đại học Quốc gia Hà Nội).

Tóm tắt đề bài

Cho một đồ thị hai phía, mỗi phía có N đỉnh, và M cạnh nối các đỉnh từ trái qua phải. Đề bài đảm bảo tồn tại bộ ghép sao cho mỗi một đỉnh bên trái được ghép với một đỉnh bên phải (và ngược lại). Hỏi có thể xoá những cạnh nào trong M cạnh đã cho để đồ thị hai phía vẫn tồn tại bộ ghép hoàn hảo (bộ ghép có độ lớn N).

Lời giải

- Nhận xét: Xét một bộ ghép hoàn bảo bất kì. Dễ nhận thấy các cạnh nào mà không nằm trong bộ ghép thì ta có thể xoá đi mà đồ thị còn lại vẫn còn bộ ghép hoàn hảo. Do đó ta chỉ quan tâm xem liệu có thể xoá các cạnh ở trong bộ ghép đang có hay không.
- Xét cạnh (u -> v) trên bộ ghép. Giờ ta thử xem nếu xoá cạnh (u, v) này đi thì có tồn tại một bộ ghép hoàn hảo nào khác hay không.
    - Thử xoá (u, v), do đó u cần phải được ghép với một đỉnh mới, gọi đỉnh này là v1 sao cho tồn tại một cạnh không thuộc bộ ghép nối từ u đến v1. Ta sẽ thử ghép u với v1 này.
    - Đỉnh v1 này ở bộ ghép cũ chắc chắn đang được ghép với đỉnh u1 nào đó.
    - Vì v1 sẽ được chuyển sang ghép cho u, nên u1 phải được nối đến đỉnh v2 khác sao cho tồn tại một cạnh chưa nằm trong bộ ghép nối từ u1 đến v2…
    - Cứ thế cứ thế … cho đến khi ta xét đến đỉnh uk sao cho có cạnh nối từ uk đến v, vì đỉnh v đang ghép với u nên đỉnh v cần được ghép bởi đỉnh uk mới.
    - Nếu tồn tại một “đường tăng" như thế thì sẽ tồn tại một bộ ghép mới mà không ghép (u, v) nữa. Ngược lại, thì u sẽ luôn luôn phải ghép với v trên bộ ghép.



- Hình minh hoạ: Cạnh màu xanh lá cây là cạnh chưa thuộc bộ ghép, cạnh màu xanh nước biển là cạnh đang thuộc bộ ghép. Bộ ghép cũ ta có u ghép với v, u1 ghép với v1, … uk ghép với vk. Sau khi tìm được đường tăng, thì u sẽ ghép với v1, u1 sẽ ghép với v2…. uk quay lại ghép với v.

Thuật toán

- Đường tăng ở trên có dạng như sau: Bắt đầu từ u, đi qua một cạnh không thuộc bộ ghép từ trái qua phải, đi qua một cạnh thuộc bộ ghép từ phải qua trái, đi qua một cạnh không thuộc bộ ghép từ trái qua phải … cứ thế cho đến khi đến được v.
- Ta sẽ xây đồ thị mới G', với mỗi cạnh (u, v), nếu (u, v) không trong cặp ghép thì ta nối cạnh có hướng (u -> v) trong G', còn không ta nối cạnh có hướng (v -> u) trong G'.
- Công việc của ta bây giờ là mỗi một cạnh (u, v) nằm trong bộ ghép, kiểm tra có tồn tại một đường đi có hướng từ u -> v hay không. Rõ ràng (v -> u) thuộc G'. Do đó nếu tồn tại đường đi từ u đến v, thì u và v phải nằm trong cùng một thành phần liên thông mạnh thuộc G'.
- Đến đây ta chỉ cần cài thuật toán Tarjan liệt kê ra tất cả các thành phần liên thông mạnh của G', và kiểm tra!

Chú ý

- Đồ thị có thể là đa đồ thì, có cạnh trùng, cần xét riêng những cạnh trùng. Nếu tồn tại 3 cạnh (u, v) trong đó u được ghép với v, thì ta thêm 2 cạnh (u -> v) và thêm 1 cạnh từ (v -> u) trên G' là được.
- Nhiều bạn có thuật toán chỉ xét những cạnh vô hướng và kiểm tra hai đỉnh (u, v) có thuộc cùng một thành phần song liên thông hay không. Thuật toán này là sai vì đường tăng phải là một đường có hướng trên “mạng thặng dư". (Khái niệm mạng thặng dư các bạn có thể đọc cuốn Cấu trúc dữ liệu và giải thuật của thầy Lê Minh Hoàng để hiểu thêm về luồng và bộ ghép).

Độ phức tạp O(chạy cặp ghép + M + N). O(chạy cặp ghép) = Msqrt(N).
Code: https://ideone.com/4N3S48.

Tản mạn

Bài này là một bài rất cổ điển của bộ ghép, để làm được những bài thế này bạn cần thực sự hiểu thuật toán tìm bộ ghép cực đại làm như thế nào, thế nào là một “đường tăng", thế nào là “mạng thặng dư", và mối liên hệ giữa bài toán bộ ghép cực đại và bài toán tìm luồng cực đại. Khi hiểu được đường tăng là gì, bạn sẽ dễ dàng tìm được cách thay đổi bộ ghép (tương đương với một đường tăng, hay một đường tăng luồng trên mạng thặng dư).

Một phiên bản khác của bài này là: Hãy tìm các đỉnh nằm trong TẤT CẢ bộ ghép cực đại. Các bạn có thể đọc ở đây.
Chúc các bạn có một ngày vui vẻ và hạnh phúc.

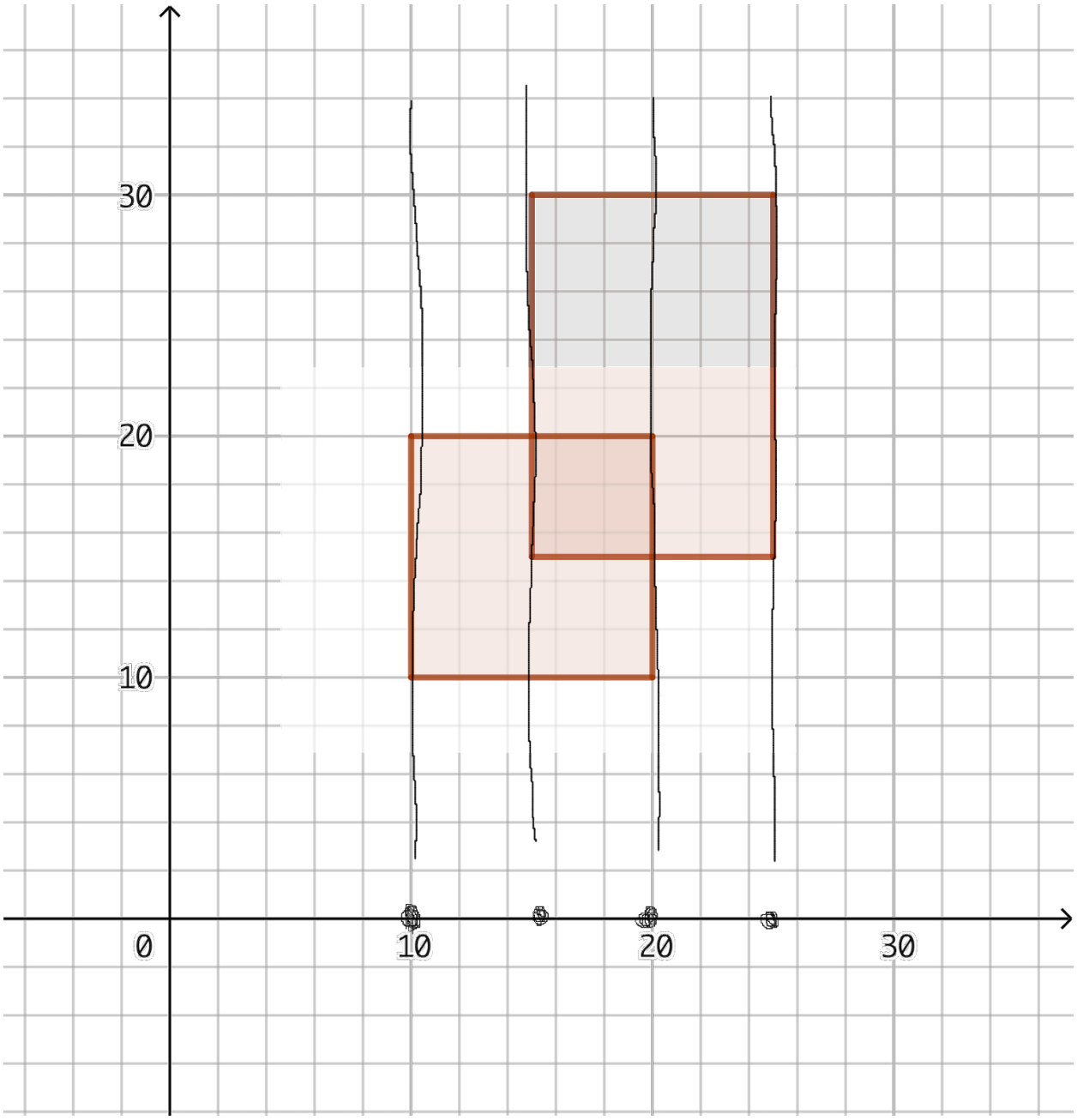
K. Kingdom of Ants

Tóm tắt đề bài

Cho N hình chữ nhật có đỉnh nguyên và các cạnh song song với trục toạ độ. Tính tổng diện tích của các vùng mà có một số lượng chẵn dương các hình chữ nhật đè lên.

Lời giải

Xét các "sự kiện" dọc theo trục hoành, mà mỗi sự kiện là một toạ độ x của một cạnh của một hình chữ nhật nào đó. Ta nhận thấy, giữa hai sự kiện liên tiếp nhau (mình sẽ gọi là dải sự kiện), "hình dáng" của các hình chữ nhật và "trạng thái" của các ô vuông dọc theo trục tung giữ nguyên (trạng thái ở đây nghĩa là với mỗi ô vuông, có bao nhiêu hình chữ nhật đè lên nó). Ví dụ như test đề bài, ta có hình minh hoạ dưới đây.



Nếu ta có được trạng thái của các ô vuông theo trục tung ở đầu 1 điểm sự kiện, thì vì ta biết được trạng thái của các ô vuông không thay đổi cho đến điểm sự kiện tiếp theo, ta có thể đem nhân số lượng ô vuông có trạng thái chẵn dương với độ dài của dải sự kiện trên để có được số lượng ô vuông trong dải sự kiện thoả mãn đề bài.

Bây giờ ta tìm cách cập nhật mỗi lúc đến một điểm sự kiện. Ta nhận thấy, mỗi điểm sự kiện tương ứng với một hình chữ nhật mới xuất hiện - nghĩa là với mỗi ô vuông nằm giữa 2 toạ độ y của hình chữ nhật mới này, số lượng hình chữ nhật đè lên ô vuông này tăng lên 1, hoặc là một hình chữ nhật biến mất - tương tự, số lượng hình chữ nhật đè lên các ô vuông trong đoạn giảm

đi 1. Nhận thấy rằng, tăng/giảm 1 đồng nghĩa với việc tính chẵn/lẻ thay đổi. Vì thế, ta có thể dựng segment tree, mà mỗi nút trong segment tree lưu 2 giá trị: số lượng ô vuông trong đoạn mà nút này quản lý có trạng thái lẻ/chẵn. Khi ta update một đoạn, ta chỉ cần tráo đổi 2 giá trị trên. Đương nhiên, segment tree này sẽ có lazy propagation - ta chỉ cần lưu lại đoạn này tăng/giảm bao nhiêu hình chữ nhật đè lên nó, rồi cập nhật 2 giá trị trên tương ứng. Tuy nhiên, bài còn yêu cầu ta cần phải lấy đáp án là trạng thái chẵn dương, nhưng cách trên chỉ đáp ứng được các trạng thái chẵn. Ta có hai cách tiếp cận:

1. Ta chỉ cần đếm số lượng ô vuông có trạng thái là 0. Để làm được điều này, ta có thể lưu thêm 2 giá trị trong nút segment tree: trạng thái nhỏ nhất của một ô vuông trong đoạn, và số lượng ô vuông trong đoạn có trạng thái bằng trạng thái nhỏ nhất này. Khi lấy đáp án, ta lấy được số lượng ô vuông có trạng thái chẵn dương = số lượng ô vuông có trạng thái chẵn - số lượng ô vuông có trạng thái 0.
2. Ta có thể tính diện tích của các vùng có ít nhất một hình chữ nhật đè lên, rồi đem đáp án trừ đi diện tích của các vùng có một số lẻ các hình chữ nhật đè lên. Để tính diện tích các vùng có ít nhất một hình chữ nhật đè lên, với mỗi dải sự kiện, ta lấy diện tích của dải sự kiện đó - diện tích của những ô vuông trống trong dải sự kiện.

Độ phức tạp của thuật toán là O(N log N). Các bạn có thể tham khảo code của mình ở đây.

L. Learning to code

Thật ra bài này chỉ là đề kêu gì làm nấy, nhưng chứa 2 cái trap khá to. Tạm thời bỏ qua 2 cái trap, cách làm bài này là ta lưu các biến vào một cái map, sau khi tính xong thì lưu nó vào map để lần sau không phải tính lại.
Cách tính các biến ta có 3 trường hợp:

- Gán một biến bằng 1 xâu, cái này khá đơn giản nên bỏ qua
- Gán một biến bằng 1 biến, cái này cũng đơn giản
- Gán một biến bằng một biểu thức, ta có thể dùng đệ quy để tính phần này: Mã giả phần này có thể viết như sau

```
string Solve(string s):
    Nếu s là xâu cơ bản:
        return s
    Nếu s là tên biến
        return giá trị biến s
    Ngược lại,
```

```
gọi l là vị trí dấu { đầu tiên
r là vị trí dấu } khớp với vị trí l để tạo ra dãy ngoặc đúng
ta chia xâu s ra làm 3 phần:
    từ đầu tới l
    từ l tới r
    từ r tới hết xâu
ta gọi đệ quy tính toán từng phần và trả về kết quả
```

Trong đoạn code trên, ta phải tìm một cặp ngoặc khớp với nhau để gọi đệ quy xuống, ta có thể dùng stack để tìm như sau:

- Ban đầu stack rỗng, ta duyệt qua các ký tự của xâu s:
    - Nếu ký tự đó là dấu '{' thì ta push vào stack
    - Nếu ký tự đó là dấu '}' thì ta pop một phần tử ra khỏi stack, nếu sau bước này stack rỗng thì cặp ngoặc vừa tạo được chính là cặp cần tìm.

Trap

Bài này có 2 trap như sau:

- Ký tự '$' có thể tồn tại nhưng không đi kèm ký tự '{'. Vì vậy nên cẩn thận khi xử lý '$'. Ví dụ:
    - var a = "1";
    - var b = `${a}$${a}`;
- Đề chỉ đảm bảo độ dài input và output không lớn hơn 1e4, chứ đề không đảm bảo trong lúc tính toán kết quả không vượt quá 1e4. Ví dụ: test như sau:
    - var a = "1";
    - var b = `${a}${a}`;
    - var c = `${b}${b}`;
    - var d = `${c}${c}`;
    - …
    - var z = `${y}${y}`;
    - print a;
    - end.
- Như test này vẫn đảm bảo input và output số ký tự không quá 1e4, nhưng trong lúc tính toán giá trị z lên tới 2^25, do đó sẽ gây TLE/MLE/RE. Để tránh trường hợp này thì trong lúc tính toán nếu thấy kết quả quá dài ta có thể thoát ra và không tính toán nữa.

Các bạn có thể tham khảo code của mình ở đây: https://ideone.com/IrU14J