The 2018 ICPC Vietnam Southern Provincial Contest Editorial

Overview

Bài	First AC	#AC	Writer
A	128 ONE	8	Hoàng Xuân Nhật Ascension
B	18 SACCAROZO	37	Phạm Bá Thái P_not_equal_NP
C	176 Unstoppable	6	Nguyễn Diệp Xuân Quang HCMUS - Intimidate
D	:(	0	Nguyễn Ngọc Trung
E	38 HCMUS-Chicken & map	43	Nguyễn Thành Vinh bitset
F	292 Send Bobs to Alice	2	Vương Hoàng Long + Phạm Đức Thắng map
G	159 trie	7	Lê Anh Đức amazingbamboo_with_coccoc
H	36 CSP_BeztDonkey	76	Phạm Bá Thái P_not_equal_NP
I	43 HCMIU & ONE	58	Nguyễn Khánh map
J	21 multimap	95	Đinh Nguyên Khôi HCMUS - Intimidate
K	12 ONE	204	Lê Quang Tuấn Send Bobs to Alice
L	88 map	21	Nguyễn Đinh Quang Minh Send Bobs to Alice

A

Lời giải: Hoàng Xuân Nhật (HCMUS-Ascension)

Dễ thấy kết quả là một biểu thức có dạng $A = x_1 + x_2 + ... + x_R$ với x_i là một chữ số hoặc một biểu thức khác có dạng $B = y_1 * y_2 * ... * y_S$ với y_j là một chữ số. Vì vậy ta gọi `dp[i][j]` là giá trị lớn nhất của biểu thức dạng *A* kết thúc tại ô `(i, j)` (ô `(i, j)` phải không chứa dấu nhân). Với mỗi ô `(p, q)` chứa dấu cộng, ta tính `f[p][q][x][y]` là giá trị lớn nhất biểu thức dạng B bắt đầu sau ô `(p, q)` và kết thúc tại ô `(x, y)`. Sau đó với ô `(i, j)` chứa số ta cập nhật

```
dp[i][j] = max(dp[i][j], dp[p][q] + f[p][q][i][j]).
```

Bài này test kĩ năng cài đặt của các team: xét trường hợp kĩ, biết code số lớn (hoặc Python, Java).
Độ phức tạp: O($n^2 m^2 * C$) với C là độ phức tạp của tính toán số lớn ($C \simeq 3$).

B

Lời giải: Phạm Bá Thái (P_not_equal_NP)

Bài toán tìm đường đi dài nhất trên đồ thị tổng quát là bài toán thuộc lớp NP-Hard. Tuy nhiên đồ thị trong bài là một đồ thị xương rồng (cactus graph – hai chu trình đơn khác nhau thì có không quá một đỉnh chung) nên bài toán là giải được, như sau:

- Tìm các thành phần song liên thông (đỉnh) của đồ thị
- Tìm một đường đi đơn bất kỳ từ A đến B. Đường đi này sẽ đi qua một số thành phần song liên thông, gọi x1, x2, …, xk là các đỉnh phân cách các vùng song liên thông đã đi qua, theo đúng thứ tự trên đường
- Tìm đường đi đơn dài nhất từ A đến x1, từ x1 đến x2, …, từ xk đến B
- Hợp các đường trên lại ta được đường đi đơn dài nhất từ A đến B

Việc tìm các đường đi dài nhất ở bước 3 có thể làm được vì hai đỉnh đó cùng thuộc một thành phần song liên thông, mà thành phần song liên thông của đồ thị xương rồng chỉ có thể là một chu trình đơn hoặc là một cạnh đơn.

Thực ra đồ thị trong bài là một dạng đặc biệt của đồ thị xương rồng: Mỗi đỉnh đều thuộc ít nhất một chu trình đơn. Do đó ta có cách dễ cài đặt hơn:

- Tìm đường đi ngắn nhất từ A đến B
- Tính tổng số lượng cạnh của tất cả đa giác (thành phần song liên thông) mà có ít nhất một cạnh thuộc vào đường đi đó
- Lấy tổng trên trừ đi đường đi ngắn nhất ta được kết quả

Độ phức tạp của cả hai cách trên đều là O(m+n)

C

Tác giả: Nguyễn Diệp Xuân Quang (team HCMUS - Intimidate)

Gọi L là đường thẳng đi qua hai điểm (x_1, y_1) và (x_2, y_2). Trước hết, ta cần tìm cách mô tả các điểm nguyên có tọa độ nguyên nằm trên L.

Gọi d = gcd($|x_2 - x_1|$, $|y_2 - y_1|$). Khi đó, phương trình biểu diễn tất cả các điểm nguyên trên L là:

$$P(k) = (x_1 + k * (x_2 - x_1)/d,\ y_1 + k * (y_2 - y_1)/d)$$

với k là số nguyên

Gọi dist(k) là khoảng cách từ P(k) đến tâm đường tròn - điểm (x_0, y_0). Ta nhận xét rằng, dist(k) là một hàm lồi, nên ta có thể dùng kĩ thuật chặt tam phân để tìm điểm nguyên trên L nằm gần tâm (x_0, y_0) nhất. Ta gọi điểm này là điểm P(m).

Nếu dist(m) > R thì đáp án là 0. Ngược lại, ta cần tìm:

- *hi*: số nguyên x lớn nhất sao cho dist(x) ≤ R
- *lo*: số nguyên x nhỏ nhất sao cho dist(x) ≤ R

Khi đó, các điểm P(k) với lo ≤ k ≤ hi đều nằm trong đường tròn nên đáp án là hi - lo + 1.

Việc tìm lo và hi có thể thực hiện bằng kĩ thuật chặt nhị phân.

Một số đều cần lưu ý khi cài đặt bài này:

- Do tọa độ có thể lên đến 10^9 nên việc tính toán bằng số thực có thể sẽ dẫn đến sai số làm tròn. Ta nên thực hiện các tính toán bằng số nguyên. Cụ thể, thay vì kiểm tra $\sqrt{x} \le R$ thì ta nên kiểm tra $x \le R^2$ và có thể tránh được việc sử dụng số thực.
- Cũng do tọa độ lên đến 10^9 nếu sử dụng số nguyên 64 bit thì cần phải rất cẩn thận trong việc đặt giới hạn chặt tam phân và chặt nhị phân (nếu không thì kết quả khi tính bình phương khoảng cách từ một điểm đến tâm đường tròn có thể vượt quá phạm vi long long).

Theo ý kiến chủ quan của mình thì để triển khai bài này dễ dàng nhất thì nên xử lí số nguyên lớn (bignum), hoặc là sử dụng ngôn ngữ có hỗ trợ sẵn bignum như Java, Python, …

Độ phức tạp: O(log(X + R)) với X là giới hạn tọa độ của các điểm được cho trong đề bài

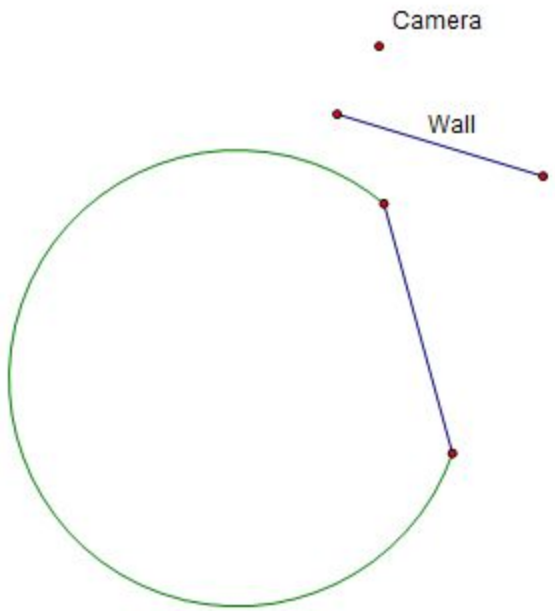
Code python: https://ideone.com/hFwSFX

D

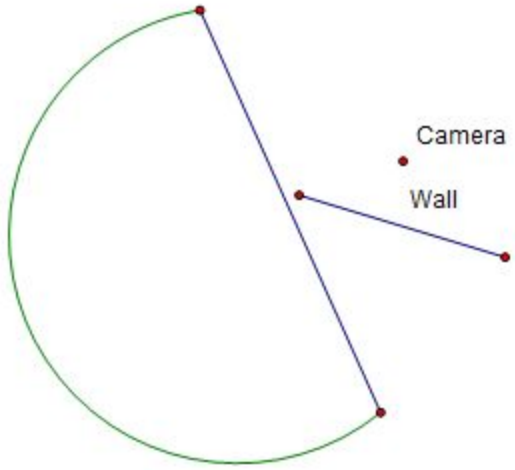
Lời giải: Nguyễn Ngọc Trung (chemthan)

Bài toán không có gì đặc sắc ngoài implementation. Có 2 trường hợp xảy ra:

1. Vị trí camera nằm ngoài đường tròn chứa sân vận động.



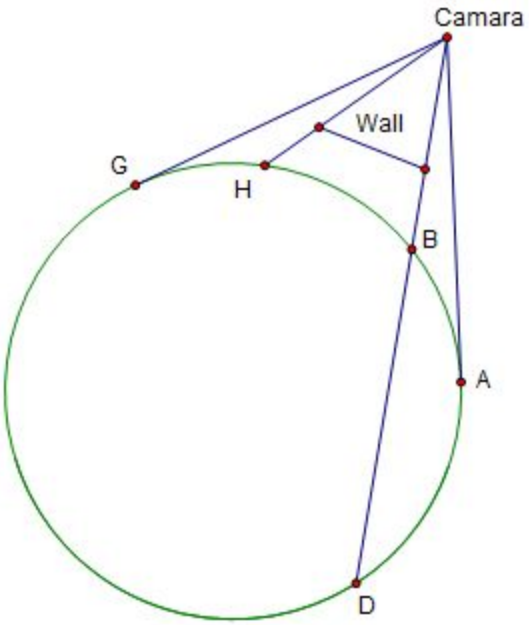
2. Vị trí camera nằm trong hoặc trên biên đường tròn chứa sân vận động.



Trước hết, ta sẽ giả sử cả sân vận động là 1 hình tròn kín. Ta sẽ tìm các cung có thể nhìn thấy từ vị trí của camera.

- Trường hợp 1

Ta tiếp tục giải bài dễ hơn khi bỏ qua bức tường. Khi đó rõ ràng cung ta nhìn thấy từ vị trí camera là cung tạo thành từ 2 tiếp tuyến từ vị trí camera đến hình tròn. (Cung nhỏ AG ở hình dưới)

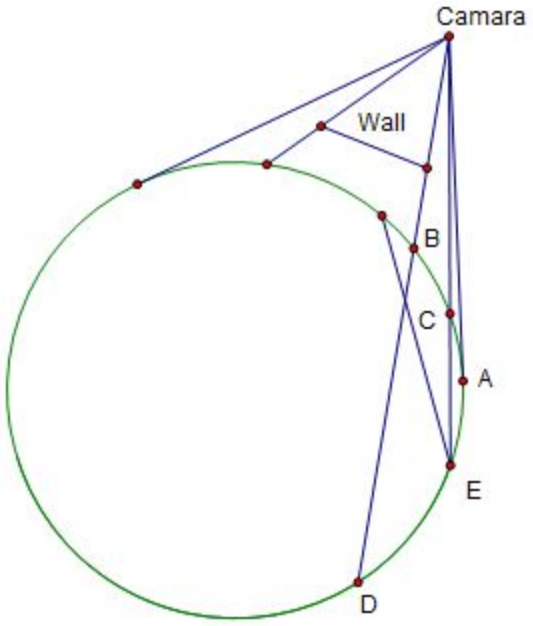


Bây giờ ta phải bỏ đi những phần bị che bởi bức tường.
(Như ở hình trên thì cung BH bị che khuất, nên ta sẽ phải bỏ qua cung này, và tập hợp những cung nhìn được từ vị trí camera là cung AB và HG)

- Trường hợp 2
  Nếu bỏ qua bức tường thì toàn bộ cung tròn là có thể nhìn được từ vị trí camera. Cũng như trên, ta cũng phải loại bỏ những phần bị che khuất bởi bức tường.

Sau khi tìm được những cung của đường tròn có thể nhìn thấy từ vị trí camera, ta sẽ xét đến phần đường tròn không thuộc sân vận động.

- Trường hợp 1
  Đối với từng cung của đường tròn thu được từ bước 1, thì phần nằm trên sân vận động sẽ được thêm trực tiếp vào kết quả. Đối với phần không nằm trên sân vận động, ta tiếp tục tìm cung nằm sau nó. Chú ý là chỉ những phần của cung này nằm trên sân vận động mới được tính vào kết quả.



(Như ở ví dụ trên, thì cung CB không nằm trên sân vận động nên ta sẽ xét tiếp phần nằm sau nó là DE, và phần của cung này nằm trên sân vận động sẽ được tính vào kết quả, trường hợp này thì toàn bộ cung DE nằm trên, nhưng không phải lúc nào cũng vậy)

- Trường hợp 2
  Tương tự nhưng đơn giản hơn trường hợp 1, khi những phần của cung không nằm trên sân vận động ta bỏ quả luôn.

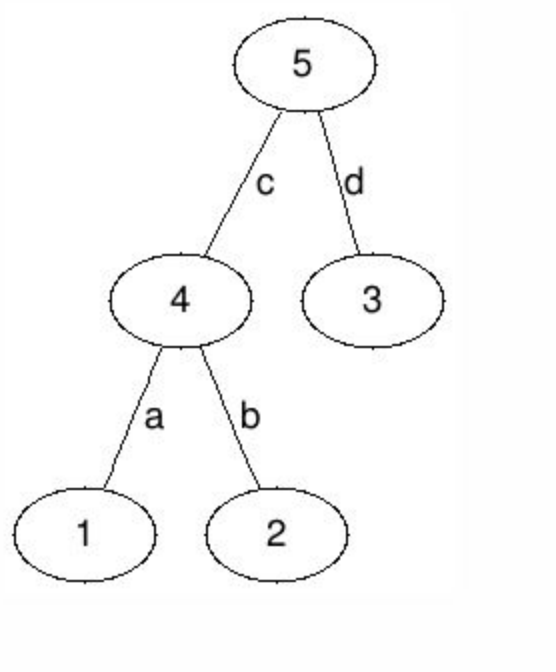
Accepted code: https://ideone.com/yCrW8n.

E

Lời giải: Nguyễn Thành Vinh (bitset)

Xét 3 đỉnh bất kì trên cây (giả sử 3 đỉnh đó là 1, 2, 3 theo thứ tự DFS). Theo đề bài ta có đường đi từ 1 đến 2 = đường đi từ 2 đến 3 = đường đi từ 3 đến 1.

Gọi LCA(1, 2) = 4, LCA(2, 3) = 5. Ta sẽ có một đồ thị như hình sau đây:



Ở đây: a + b = b + c + d = a + c + d <=> a = b = c + d.

Dễ dàng nhận thấy đường đi từ 4 đến 3 đỉnh sẽ bằng nhau.

Vậy ta có thể chứng minh được rằng một bộ 3 đỉnh bất kì thỏa mãn yêu cầu đề bài khi và chỉ khi tồn tại duy nhất một đỉnh trên cây (gọi là đỉnh gốc) sao cho độ dài đường đi từ đỉnh gốc đến 3 đỉnh là bằng nhau.

Đến đây bài toán trở nên rất dễ dàng, ta chỉ cần chọn đỉnh gốc, DFS từ đỉnh đó, và đếm số bộ ba đỉnh cùng độ cao sao cho chúng thuộc ba cây con khác nhau từ đỉnh gốc nối ra. Độ phức tạp thuật toán: *O(N^2).*

F

Lời giải: Vương Hoàng Long + Phạm Đức Thắng (map)

Bài này là một bài giải bằng nhân ma trận, với một số tối ưu đặc biệt để pass được time limit.
Đầu tiên ta có lời giải trâu:
Gọi f[u][len] là số đường đi kết thúc tại U mà có thời gian là len.Chuyển trạng thái thì với mỗi (U,len) ta chuyển sang (V,len+W) với V là 1 đỉnh mà U có cạnh nối đến, và W là trọng số của cạnh đó. Gọi Total[len] là tổng tất cả các DP[u][len] với u từ 1->n.
Vậy dễ nhận ra là ta cứ DP như vậy, đến khi nào tồn tại số X sao cho tổng các Total[len] với len từ 1->X >= số ngày thì X chính là đáp án.

Tối ưu xuống nhân ma trận:
Dễ nhận thấy với hàm DP này, ta chuyển trạng thái với công thức cố định, nên ta có thể sử dụng nhân ma trận để tính. Để ý hơn nữa thì vì ta cần tính tổng tất cả các total[len] với time từ 1->X nên ta có thể add thêm 1 cột là tổng vào cuối ma trận.
dựng ma trận, tại độ dài len ta xét:
+ ma trận base (1 x 3n) -> tương trưng cho f(i, j) là số đường đi với i là nút hiện tại và len + j là độ dài.
+ ta muốn update f(i, j) với độ dài len thành f'(i, j) với độ dài len + 1, ta sử dụng:
- f(i, 0), f(i, 1), f(i, 2) để update cho f'(j, 2) sử dụng cạnh (i, j) với từng trọng số 3, 2, 1 tương ứng.
- f(i, 1) để update cho f'(i, 0).
- f(i, 2) để update cho f'(i, 1).

-> như vậy ta có thể xây dựng mảng ma trận mul (3n x 3n) để chuyển trạng thái hay nhân ma trận.
Và vì ta thêm 1 cột vào , nên ô 3*n+1 trong bảng mỗi lần chuyển sẽ được cộng thêm tổng tất cả các f[i][0] trong ma trận hiện tại. Vậy ta sẽ có ma trận 1*(3n+1) và ma trận chuyển là (3n+1)*(3n+1).

Đó là ta đã tính được tổng Total[len] với len từ 1->X với X là một số tuỳ thích. Vậy để tìm X nhỏ nhất thoả mãn, ta sẽ nghĩ ngay đến việc chặt nhị phân. Ta chặt nhị phân X và nhân ma trận như bình thường, thì ta sẽ có lời giải độ phức tạp (3*n)^3*log(k)^2 ~ 6e9, quá chậm để AC.

(Note cho nhân ma trận) Các bạn nhận ra số cách có thể cực lớn, và dễ dàng vượt qua LONG_LONG_MAX, nhưng để ý ta chỉ quan tâm đến tính chất >=k hay chưa, nên ở mỗi bước nhân ta có thể lấy min với k để tránh tràn số.

Tối ưu chặt nhị phân:

Ta nhận thấy chặt nhị phân rồi lại mũ log(khi mũ ma trận) có vẻ hơi thừa, thay vào đó ta sẽ lưu lại mat[i] là ma trận chuyển sau khi đã mũ lên (2^i) lần. Gọi ma trận hiện tại là cur. Vậy thay vì chặt nhị phân, ta chỉ cần for từ 60 về 0, nếu tổng số cách (nói cách khác là ô 3*n+1) của ma trận cur * mat[i] mà < k thì ta nhân vào, X|=(1<<i), nếu không thì bỏ qua.

Vậy ta đã tìm được X lớn nhất sao cho tổng các total[len] với len từ 1->X là nhỏ hơn HẲN k, vậy đáp án sẽ là X+1. Độ phức tạp lúc này là (3*n)^3*log(k) ~ 1e8, có vẻ khá đủ nhưng vẫn hơi chậm để AC, do khi nhân ma trận ta phải thực hiện phép mod.

Tối ưu đặt cận:

Để ý khi nhân ma trận, ta for tất cả cặp hàng và cột, lấy tích và cộng vào ô hiện tại. Để ý ở bước lấy min với k, ta có thể tối ưu sâu hơn nữa, bằng việc nếu ô hiện tại đã = k rồi thì ta sẽ ngừng việc for trên lại. Điều này sẽ làm giảm thời gian chạy rất nhiều, bởi lẽ tốc độ tăng của nhân ma trận là rất nhanh , cộng với việc một khi đã = k thì tất cả các phép nhân với ô đó trong tương lai cũng sẽ đều >=k => số lần phải duyệt sẽ giảm rất nhanh.

Với tối ưu cuối cùng này thì các bạn đã có thể AC =))). Với mình thì tối ưu này là một tối ưu khá hiển nhiên? nhưng trong giờ thi lại hơi khó nghĩ ra =))).

G

Phát biểu lại bài toán:
- Ta cần chia các ô trong bảng thành 2 loại A và B
- Giá trị của 1 cách chia là: (tổng giá trị các ô loại A) + (tổng giá trị các ô loại B) - (tổng chi phí các cặp ô A - B kề nhau)
- Cần tìm cách chia có giá trị lớn nhất

Ta sẽ giải bài này bằng cách đưa về lát cắt cực tiểu, dựng mạng như sau:
- Mạng có N * M + 2 đỉnh, gồm: đỉnh phát S, đỉnh thu T, tập V với N * M đỉnh ứng mới N * M ô trong bảng
- Từ đỉnh S có cạnh nối đến các đỉnh u trong V với thông lượng A(u)
- Từ các đỉnh u trong V nối đến T với thông lượng B(u)
- Giữa 2 đỉnh kề u, v trong V có cạnh nối đến nhau với thông lượng C(u, v) = max(c(u), c(v))

Một lát cắt S-T chính là một cách phân hoạch tập V + {S, T} thành 2 tập X, Y sao cho:
- Union(X, Y) = V + {S, T}
- Intersection(X, Y) = {}
- S thuộc X
- T thuộc Y

Ta có ánh xạ 1-1 giữa một lát cắt và một cách chia bảng:
- Những đỉnh u trong X có nhãn A, tính vào chi phí lát cắt một lượng capacity(u, T) = B(u)
- Những đỉnh v trong Y có nhãn B, tính vào chi phí lát cắt một lượng capacity(S, v) = A(v)
- Những cạnh nối 2 ô kề u, v có nhãn khác nhau sẽ được tính vào chi phí của lát cắt một lượng capacity(u, v) = C(u, v)
- Giá trị của cách chia bảng = (tổng giá trị A) + (tổng giá trị B) - (chi phí lát cắt)

Vậy lát cắt S-T nhỏ nhất ứng mới cách chia bảng có giá trị lớn nhất.
Với giới hạn của đề bài, sử dụng thuật toán luồng Dinic chạy đủ nhanh.

H

Lời giải: Phạm Bá Thái (P_not_equal_NP)

Gọi hai bảng đã cho là A và B. Đề bài (nếu đọc kỹ cả phần giải thích test ví dụ) yêu cầu tìm một đường đi mà vừa hợp lệ trên cả A và B, vừa là đường đi ngắn nhất trên cả A và B. Xét C là bảng hợp thành từ A và B (C(i,j) có vật cản nếu A(i,j) có vật cản hoặc B(i,j) có vật cản)

Giả sử tồn tại L là đường đi ngắn nhất trên cả A và B, khi đó L cũng là một đường đi ngắn nhất trên C. Mặt khác, một đường đi hợp lệ bất kỳ trên C thì đều hợp lệ trên A, và cả trên B. Vậy kết quả bài toán (nếu có) sẽ là một đường đi ngắn nhất bất kỳ trên C. Ta có thuật toán sau:

- Tìm L là một đường đi ngắn nhất trên C. Nếu không tồn tại L, output -1
- Tìm L1 là một đường đi ngắn nhất trên A
- Tìm L2 là một đường đi ngắn nhất trên B
- Nếu |L|=|L1|=|L2| thì output |L|, ngược lại output -1

(Các đường đi xét đến trong bài đều bắt đầu tại S và kết thúc tại F)

I

Lời giải: Nguyễn Khánh (map)

Trước hết, ta tính đáp án cho từng cạnh, sau đó mới trả lời các truy vấn.

Dễ dàng thấy được, với các cạnh không phải là **cầu** thì đáp án của cạnh đó bằng 0. Do đó, ta chỉ xét các cạnh là cầu.

Để tìm cầu, có nhiều loại thuật toán. Giải thuật này sẽ dùng thuật toán trong sách Giải thuật và lập trình của thầy Lê Minh Hoàng để bạn đọc dễ hiểu. Các bạn có thể tìm đọc ở đây.

Giả sử (u, v) là cầu và u là cha của v trong lúc DFS. Ta có thể tính được số đỉnh nằm ở dưới v trên cây DFS (tạm gọi là x), từ đó xác định được số cặp đỉnh phải đi qua cạnh (u, v) là x * (n - x) với n là số đỉnh nằm trong thành phần liên thông chứa đỉnh u, v.

Độ phức tạp của thuật toán là O(n).

J

Lời giải - Đinh Nguyên Khôi (HCMUS Intimidate)

Ta thấy rằng giả sử ta cố định kim đồng hồ đầu tiên chỉ vào một vị trí nào đó, thì công việc còn lại của ta đối với lại N chiếc đồng hồ này chỉ là cô gắng chỉnh làm sao cho toàn bộ cây kim chỉ vào vị trí 12 giờ và phải chỉnh theo lần lượt từ trái sang phải. Vậy ta sẽ thực hiện như sau:
Ta vét cạn hết 12 trạng thái mà cây kim đồng hồ thứ 1 có thể điểm. Với chiếc đồng hồ thứ i, ta chọn nhấn nút sao cho số thao tác để cây kim trên đồng hồ thứ i chỉ về vị trí 12 là ít nhất có thể. Vì nhấn nút giữa i và i + 1 thì sau khi ta chỉnh cây kim ở đồng hồ i về 12 giờ, thì cây kim của đồng hồ i + 1 sẽ nằm ở một vị trí nào đó. Sau đó ta lại tiếp tục nhấn nút ở giữa đồng hồ i + 1 và i + 2, như thế cho đến khi hoàn thành trạng thái cho N đồng hồ.

Source code: https://ideone.com/DY94NP

K

Lời giải: Lê Quang Tuấn.

- Nếu $N \leq 9$ thì đáp án có thể là 6 hoặc 8.
- Nếu N có K chữ số, hiển nhiên ta có số T = 888..888 (K - 1 chữ số 8) thoả mãn.
- Xét trường hợp đáp án có K chữ số. Dễ dàng nhận thấy một số T may mắn và lớn nhất nếu :
    - T = N và N là số may mắn.
    - T có chung prefix với L càng dài càng tốt.
    - Gọi i là vị trí đầu tiên mà T[i] khác N[i] (T[i] là chữ số thứ i của T, tính từ trái sang), i là lớn nhất có thể, trong các số T có i lớn nhất có thể, chọn số có T[i + 1] lớn nhất có thể và T[i + 1] < S[i + 1]. Phần từ T[i + 2] đến hết chỉ toàn các chữ số 8.

L

Lời giải: Nguyễn Đinh Quang Minh

Trước hết, gọi P(x) là xác suất ô thứ x được thăm trong trò chơi. Ta có:

+ P(x) = 0 với x < 0
+ P(0) = 1
+ P(x) = (P(x-1) + P(x-2) + … + P(x-n)) / n với x > 0

Từ công thức trên, không khó để chứng minh dãy P hội tụ khi x tiến đến dương vô cùng. Với x lên đến 10^G với $G = 10^{100}$, ta có thể giả sử P(x) = P(x + 1) = … = p.

Để tính xác suất không có ô xấu (ô trong khoảng 10^G+1 đến 10^G+m) nào được thăm, ta xét bước nhảy từ ô x đến ô y *vượt qua* tất cả các ô cấm này. Nói cách khác, xét mọi cặp (x, y) thỏa mãn x < y ≤ x+n và x < 10^G+1 < 10^G+m < y, gọi E(x, y) là xác suất tồn tại một bước nhảy từ x đến y trong trò chơi, kết quả bài toán sẽ là tổng của các E(x, y).

Rõ ràng E(x, y) = P(x) * 1/n = p/n (vì x đủ lớn). Mặt khác, nếu m ≥ n, có 0 cặp (x, y) thỏa mãn, ngược lại có (n - m)(n - m + 1) / 2 cặp. Vậy vấn đề còn lại là tìm p.

Để tìm p, ta có thể tìm giới hạn của dãy P đã nhắc đến ở trên. Tuy nhiên, có một cách giải đơn giản hơn: 1-p thực ra chính là đáp án bài toán khi m=1. Do đó, p/n * n(n-1) / 2 = 1-p. Giải ra được p = 2/(n+1). Như vậy, đáp án bài toán khi m < n sẽ là (n - m)(n - m + 1) / (n(n+1)).