Overview

Bài	First AC	#AC	Editorialist
A	116 P_not_equal_NP	10	Phạm Bá Thái P_not_equal_NP
B	57 998244353	5	Nguyễn Đinh Quang Minh Sends Bobs to Alice
C	14 array	121	Vương Hoàng Long UET-Map
D	56 -----CENSORED-----	15	Vương Hoàng Long UET-Map
E		0	Phạm Văn Hạnh judge
F		0	Nguyễn Đinh Quang Minh Sends Bobs to Alice
G	165 bitset	1	Trần Tấn Phát Sends Bob to Alice
H	13 PTIT.BlackKnights	121	Đinh Nguyên Khôi HCMUS - Intimidate
I	27 Supir Tayo	21	Nguyễn Diệp Xuân Quang HCMUS - Intimidate
J	87 Teeeeeeeeeeeeemo needs blue buff	10	
K		0	Nguyễn Khánh UET-Map
L	85 Supir Tayo	15	Nguyễn RR Thành Trung judge

Tổng quan

Theo đánh giá của ban tổ chức, đề năm 2018 khó hơn so với đề năm 2016 và 2017. So với năm 2017, 2 bài dễ năm nay (C và H) có cài đặt khá đơn giản, các đội có thể làm xong trong 1h đầu và còn 4h để tập trung vào các bài trung bình:

- Bài A: Mặc dù là 1 bài cơ bản trên VOJ, tuy nhiên với cách viết đề là tìm 1 đường đi từ A đến B qua C (thay vì tìm 2 đường đi từ C đến A và B), đã làm cho các đội rối trí và không nhận ra đây chính là 1 bài luồng mincost cơ bản. Tuy nhiên bài này cũng có cách làm rất đơn giản chỉ cần dùng BFS hoặc DFS thông thường -- sẽ được trình bày chi tiết hơn trong phần lời giải.
- Bài D: Nghĩ ra cách truy vấn với $O(logN)$ truy vấn có lẽ không khó với rất nhiều đội -- tuy nhiên giới hạn 30 truy vấn đã khiến cho nhiều đội phải dừng bước với bài này. Trong lúc chuẩn bị đề ban tổ chức đã cân nhắc việc cho các đội thêm 1 truy vấn (giúp bài toán trở nên đơn giản hơn rất nhiều) -- tuy nhiên sau khi cân nhắc cẩn thận, ban tổ chức nhận thấy rằng nếu tăng giới hạn lên sẽ có rất nhiều đội không biết làm gì nữa sau khi đã AC thêm bài D.
- Bài I: Đây là 1 bài tập khá cơ bản tuy nhiên đòi hỏi các bạn phải nắm chắc các kiến thức như KMP và Aho-Corasick. Ngoài ra các bạn cũng cần đọc kĩ đề ;).

Ngoài ra trong đề có 2 bài khó giả dạng bài dễ:

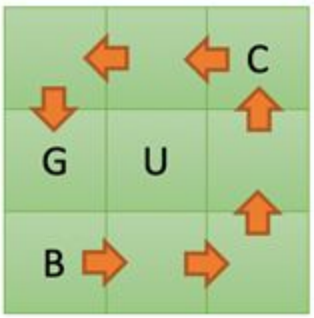
- Bài J: Hầu hết các đội chỉ tìm được 2 dạng đồ thị thoả mãn đề bài. Rất tiếc trên thực tế có 3 dạng đồ thị.
- Bài K: Với đề bài ngắn, hình vẽ bắt mắt cộng với tam giác chỉ có 3 cạnh nghe có vẻ rất đơn giản, một số đội đã lao vào bài K và dẫn đến kết quả đau buồn.

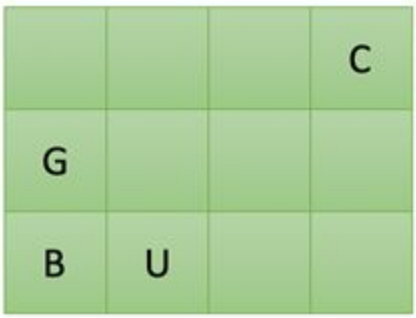
Ban tổ chức hi vọng các đội sẽ rút được kinh nghiệm cho những lần thi sau -- trước khi làm bài cần tỉ mỉ chứng minh thuật toán và xét trường hợp cẩn thận hơn.

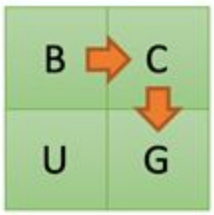
A - Amazing Adventures

Phạm Bá Thái - P_not_equal_NP

Bài này khá giống bài *HIWAY* trên spoj, nhưng "khù khoằm" hơn.
<u>Cách 1:</u> Làm tương tự *HIWAY*.







- Khác biệt thứ nhất với *HIWAY* là bài này hai đường đi đó chỉ chung nhau 1 đầu. Giải quyết bằng cách thêm 1 đỉnh phát s, nối 2 cạnh lưu lượng 1 từ s vào B và từ s vào G, xem C là đỉnh thu.
- Khác biệt thứ hai là bài này yêu cầu mỗi đỉnh không cho quá 1 lưu lượng luồng đi qua. Ta tách đỉnh x thành x_in và x_out, nối x_in với x_out bởi 1 cạnh có lưu lượng 1, cạnh (x, y) trên đồ thị được xem là (x_out, y_in) và (y_out, x_in).
- Chi phí của các cạnh ban đầu là 1, của các cạnh được thêm vào là 0. Đến đây bài toán trở thành *HIWAY* (từ s đến C).
- Tìm luồng cực đại chi phí cực tiểu trên đồ thị, những cung có luồng tạo thành 2 đường đi cần tìm.
- Do chỉ có 2 lần tăng luồng nên độ phức tạp là $O(M \times N) = O(N^2)$ với $N = n^2$ là số đỉnh của đồ thị và $M = 4N$ là số cạnh. Nhân thêm số test thì đpt này rất lớn, có thể đặt cận cho ford-bellman do số cạnh âm chỉ khoảng $O(n)$.

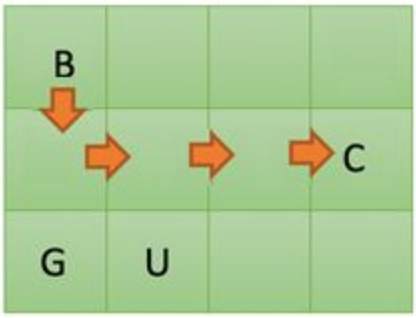
(Đồ thị thu được là vô hướng nên sẽ hơi lạm dụng nếu nói luồng. Tuy nhiên mỗi cạnh có thể xem là 2 cung ngược của nhau.)

<u>Cách 2:</u> Có thêm một nhận xét quan trọng, nên bài này team mình làm khá nhanh. Sau đây là cách làm đó (dễ cài đặt hơn nhiều so với cách 1).
Gọi đường đi tốt nhất từ $B \to C \to G$ là $f(B, G)$. $f(X, Y)$ tính bởi:

- Tìm một đường đi ngắn nhất từ $X \to C$.
- Xóa hết các đỉnh trên đường đi vừa tìm được (ngoại trừ C).
- Tìm một đường đi ngắn nhất từ $C \to Y$.
- Nếu một trong hai đường trên không tồn tại thì $f(X, Y) = null$. Ngược lại, $f(X, Y)$ là hợp của hai đường trên.

Kết quả là đường tốt hơn trong hai đường $f(B,G)$ và $f(G,B)$ (đường $null$ là xấu nhất). Độ phức tạp của cách này là $O(n^2)$.

Nếu không có vật cản thì thuật trên là đúng vì đây là đồ thị lưới với các cạnh có trọng số bằng nhau. Nếu có nhiều vật cản thì phải quay về cách một mà thôi. Khi có một vật cản, sở dĩ $f(B,G)$ bị sai vì có thể đường đi từ $B \to C$ đã cắt đứt tính liên thông của G và C (xem hình sau).



Tuy nhiên nếu ta ưu tiên G trước thì không xảy ra trường hợp này. Nhận xét quan trọng ở đây là nếu tồn tại kết quả thì một trong hai đường $f(B,G)$ và $f(G,B)$ phải tìm ra, vì U không thể chặn cùng lúc cả hai được (ở hình trên, U đã chặn mất 1 hướng đi từ $G \to C$).

Fun fact:

Ban tổ chức đã sinh khoảng $100,000$ test cho bài này, gồm tất cả các bảng 4×4, các loại bảng $1 \times N,\ 2 \times N$ và nhiều loại bảng ngẫu nhiên kích thước to nhỏ khác nhau. Do đó chỉ có thuật chuẩn mới có thể AC.

B - Bipartite Battle

Nguyễn Đinh Quang Minh - khán giả

Trò chơi kết thúc khi đồ thị hiện tại có 0 đỉnh và 0 cạnh. Hiển nhiên đây là một trạng thái thua. Tương tự như các trò chơi tổ hợp 2 người thông thường, một trạng thái là thắng khi và chỉ khi từ trạng thái này có thể đi đến một trạng thái thua. Vậy công việc của chúng ta là tìm các trạng thái thua.

Để giải quyết bài toán này một cách hiệu quả nhất, ta có thể vẽ thử một số đồ thị nhỏ rồi quan sát, cuối cùng chứng minh chặt chẽ. Trên thực tế, người chữa đã nhận ra đặc điểm của các trạng thái thua và chứng minh sau khi vẽ các đồ thị 2 phía có không quá 4 đỉnh. Việc này tuy đơn giản nhưng tốn mực nên xin nhường lại cho các bạn đọc.

Sau khi vẽ xong, không khó để nhận ra rằng: một trạng thái là thua khi và chỉ khi số đỉnh và số cạnh chẵn (*). Ta sẽ chứng minh điều này bằng quy nạp.

1. Trạng thái kết thúc trò chơi có số đỉnh và số cạnh chẵn (0 đỉnh, 0 cạnh).
2. Xét một trạng thái không phải kết thúc, gồm n đỉnh và m cạnh. Theo giả thiết quy nạp, tất cả các trạng thái mà nó đến được đều thỏa mãn (*). Xét tính chẵn lẻ của n và m:
    - n chẵn, m chẵn: nếu ta xóa một đỉnh thì số đỉnh nhận được sẽ là lẻ, ngược lại nếu xóa một cạnh thì số cạnh nhận được sẽ là lẻ. Theo giả thiết quy nạp, các trạng thái đi đến được đều là trạng thái thắng. Do đó, đây là trạng thái thua.
    - n chẵn, m lẻ: nếu ta xóa một cạnh, trạng thái mới sẽ có n đỉnh (chẵn) và $m-1$ cạnh (chẵn), là một trạng thái thua. Do đó, n chẵn m lẻ là trạng thái thắng.
    - n lẻ, m chẵn: nếu ta xóa một đỉnh có bậc chẵn, trạng thái mới sẽ có chẵn đỉnh và chẵn cạnh. Ta cần chứng minh đỉnh này luôn tồn tại. Thật vậy: do đồ thị là hai phía, tồn tại một phía có lẻ đỉnh và một phía có chẵn đỉnh. Xét phía có lẻ đỉnh, nếu tất cả các đỉnh của phía này có bậc lẻ thì tổng số cạnh của đồ thị sẽ là tổng của lẻ số lẻ, tức là một số lẻ - trái với giả thiết m chẵn. Do đó, tồn tại ít nhất một đỉnh có bậc chẵn. Như vậy, n lẻ m chẵn là trạng thái thắng.
    - n lẻ, m lẻ: tương tự như trên, ta cần chứng minh sự tồn tại của một đỉnh bậc lẻ. Xét phía có chẵn đỉnh, nếu tất cả các đỉnh đều có bậc chẵn thì m sẽ bằng tổng của các số chẵn - trái với giả thiết m lẻ. Vậy n lẻ m lẻ là trạng thái thắng.

Từ những điều trên, ta có thể suy ra (*) đúng với mọi trạng thái.

Vấn đề còn lại là đếm số cách thêm các cạnh để trạng thái nhận được là trạng thái thua. Việc này vô cùng đơn giản: nếu tổng số đỉnh ban đầu là lẻ, đáp án là 0. Ngược lại, đáp án là 2^{p-1}, với p là tổng số cạnh có thể thêm, trong bài toán này chính là tổng các $a_i \times b_i$.

C - Conquest Campaign

Vương Hoàng Long - UET Map

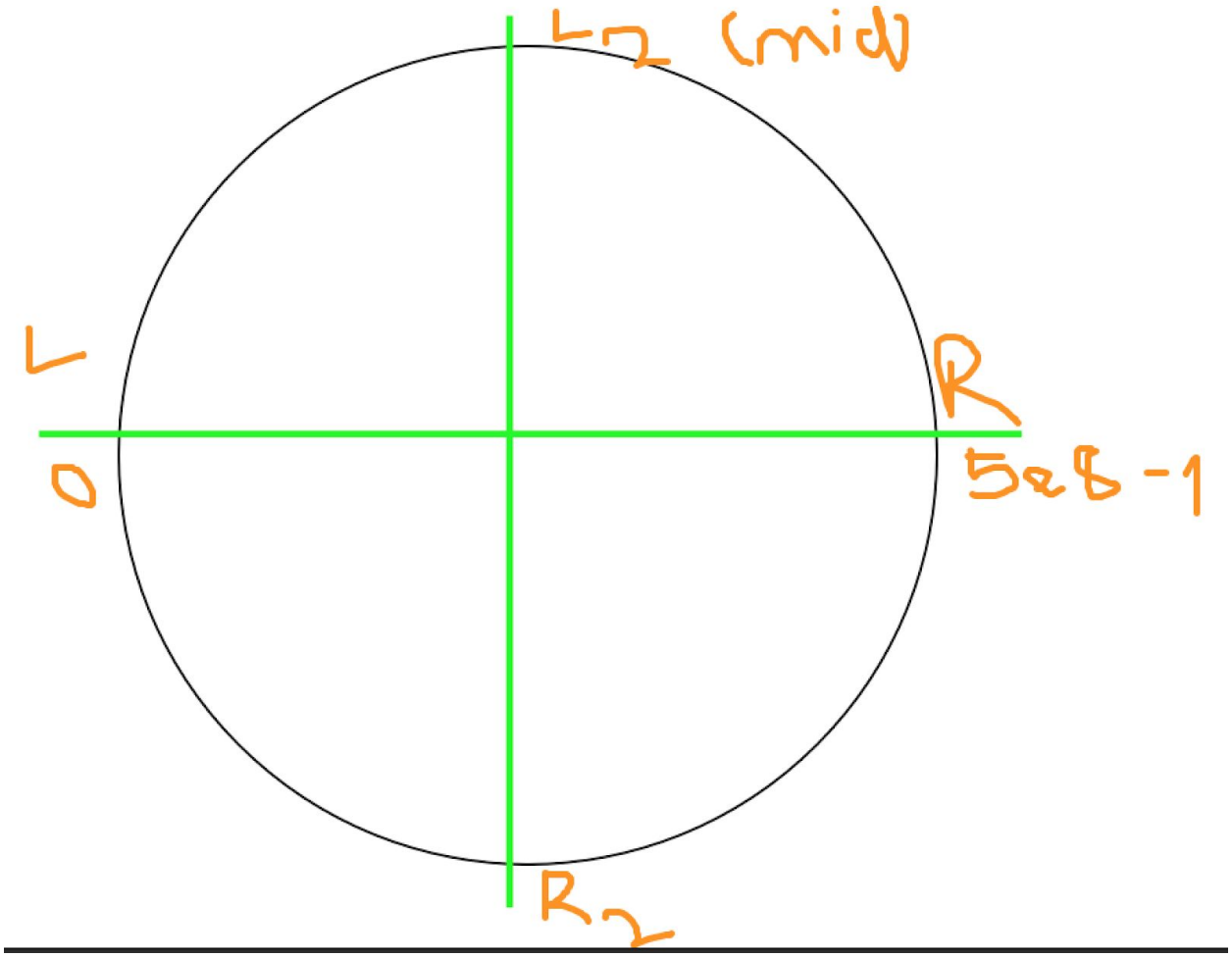
Bài này không có gì để viết cả, các bạn cứ *BFS* từ các điểm đã cho dần ra thôi, với mỗi lượt *BFS* lấy $ans = max(ans, distance)$ là xong.

D - Dividing Doughnut

Vương Hoàng Long - UET Map

Foreword:
Chào các bạn, lại là mình viết bài interactive đây =))).
Bài interactive lần này thực ra là một bài mà cũng chỉ chơi tay một hồi sẽ nhận ra cách làm, chứ ban đầu đọc đề thấy cũng hư cấu vãi. Tuy vậy, nó không nhiều AC như bài interactive tại National (dù dễ hơn?) do ban tổ chức có một cái bẫy khá hiểm (Và mình trong giờ đã vô cùng ẩu khi không chịu test cái Grader). Anyway Sau đây là lời giải của mình.

Solution:
***** note: trong các lần + thì mọi người tự thêm mod 10^9 vào để cho đúng nhé *****
Mình gọi "Sprinkle" trong bài là kẹo nhé.
Vì mỗi người sẽ lấy 5×10^8 phần bánh liên tiếp, nên sẽ có 1 đường cắt chia bánh giữa 2 người. Mình định nghĩa hàm $F(X) = (số\ kẹo\ trong\ đoạn\ X \rightarrow X + 5 \times 10^8 - 1) - \frac{n}{2}$. Đầu tiên mình tính $F(0)$, nếu $F(0) = 0$, xong. Nếu $F(0) > 0$, đặt $l = 0, r = 5 \times 10^8 - 1$, nếu không thì đặt $l = 5 \times 10^8$ và $r = 10^9 - 1$. Thêm nữa, vì 2 phần bánh đều có kích thước là 5×10^8, nên chắc chắn có đúng 1 đầu mút của đường cắt nằm trong mỗi đoạn độ dài 5×10^8. Mục tiêu của chúng ta chỉ cần tìm được 1 đầu mút sẽ suy ra đầu mút còn lại. Gọi vị trí đầu mút cần tìm là P.



Ok, không mất tính tổng quát, giả sử $F(0) > 0$ và $L = 0, R = 5 \times 10^8 - 1$. Đương nhiên $L+1 \le P \le R$ (P != L do $F(L) > 0$).

Ta có: $F(L) > 0, F(R+1) < 0$ (Theo giả sử). Giờ $mid = \frac{L+R}{2}$. Ta xét 3 TH:

1) Nếu $F(mid) = 0$, xong, mid chính là đầu mút.
2) Nếu $F(mid) > 0$, kết hợp với $F(R+1) < \frac{n}{2}$ ta có $mid + 1 \le P \le R$.
3) Nếu $F(mid) < 0$, kết hợp với $F(L) > \frac{n}{2}$ ta có $L + 1 \le P \le mid - 1$.

Mình sẽ chứng minh cho các bạn kết luận của điều thứ 2, cái thứ 3 tương tự nhé:
Ta thấy rằng $F(R+1) < 0$, mà $F(mid) > 0$ suy ra tồn tại 1 vị trí P mà $mid + 1 \le P \le R + 1$ sao cho $F(P) = 0$. Lý do: vì giá trị 0 bị kẹp giữa $F(R+1)$ và $F(mid)$, $abs(F(i) - F(i+1)) \le 1$.

Nói rõ hơn, hàm $F(X)$ đang Dương, bây giờ muốn chuyển sang âm, mà hiệu 2 hàm liên tiếp lại ≤ 1. Vậy nên chắc chắn phải đi qua giá trị 0. Suy ra tồn tại $F(P) = 0$. (Gần giống như Lagrange Mean Value Theorem).

Chứng minh tương tự ta sẽ chứng minh được kết luận số 3.

Vậy với điều này, bài toán có thể giải đơn giản với hàm solve chính như sau:

```
Solve(L,R){
    If (L==R) => Done
    Mid = (L+R)/2;
    If (F(Mid) == 0) => Done
    Else if (F(Mid) > 0) Solve(mid+1,R)
    Else Solve(L,mid-1)
}
```

Và ta gọi `Solve(0,1e9-1)` đầu tiên.
Vì sau mỗi lần độ dài của đoạn đều giảm đi ít nhất một nửa + 1 đơn vị (do bỏ vị trí mid) nên số lần gọi hàm solve chỉ khoảng 29 lần, + 1 lần trả lời $F(0) = 30$ lần đúng bằng giới hạn đề bài nên solution dễ dàng AC. (Chứng minh tại sao lại là chạy 29 lần xin challenge các bạn 😁)

Ok vậy bài toán kết thúc tại đây, trong giờ có khá nhiều bạn do cài hơi ẩu một chút nên đã không AC được bài này (thừa 1 2 truy vấn), kinh nghiệm là các bạn nên viết Grader cẩn thận cho các bài interactive nếu bị WA, vì viết Grader cũng không tốn nhiều công sức lắm, bù lại bạn có thể test bài Interactive vô cùng dễ dàng. Trong giờ do chán đời nên mình đã viết 1 cái Grader rất vớ vẩn dẫn tới không AC. Ở dưới đây là code mình, có cả Grader chuẩn luôn sau khi về nhà, các bạn có thể đọc tham khảo. Hẹn gặp các bạn ở các bài viết Editorial khác 😂 From Map with love 😭.

Code:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/regional/codes/ideone_X0LNJF.cpp

E - Enjoying Elderberry

Bài toán gốc: lấy ý tưởng từ việc những đoạn code được viết bởi *ECMA Script* 2015++ cần đc downgrade xuống các phiên bản JavaScript (*ECMA Script*) cũ hơn để có thể chạy được ở các môi trường ko support JavaScript bản cao. Một trong những bài toán đó là làm sao để "thay" toàn bộ các biến let thành var mà ý nghĩa của code không thay đổi.

Bạn có thể đọc thêm ở đây và demo ở đây.

Ví dụ đoạn code sau:

```
if (1 < 2) {
  let a = 1;
  console.log(a);
} else {
  let a = 2;
  console.log(a);
}
```

Hai biến a ở trường hợp trên là khác nhau, nhưng nếu thay let thành var thì hai biến thành giống nhau, nên một trong hai biến phải đổi tên đi.

Bài này có sự thay đổi, đơn giản hóa cách hoạt động của let và var trong *JS*. Ý tưởng chính là: nếu khai báo "$var\ x$" thì x có tầm hoạt động là cái regular-function block gần nhất. Ngược lại, nếu khai báo "$let\ x$" thì x có tầm hoạt động là cái block gần nhất.

Tất nhiên, scope tracking của *JS* còn phức tạp hơn nhiều, bài toán này cũng đã lược bỏ rất nhiều vấn đề rất lớn của block-scoping transform, bao gồm Closure-problem và Temporal Dead Zone.

Bài này được chuyển thể như sau:

- Cái cây chính là cái *Abstract-Syntax-Tree* của đoạn code.
- Mỗi interval node (node không phải lá) là một block.
- "BIG BRANCH" đại diện cho function body.
- "SMALL BRANCH" đại diện cho một block bình thường (một block giữa hai dấu { })
- Nút gốc là 1 "BIG BRANCH" vì nó đại diện cho toàn bộ cái code (có thể gói trong function body).
- “GIANT BIRD” đại diện cho một biến được khai báo bằng var.
- “TINY BIRD” đại diện cho một biến được khai báo bằng let.
- “BERRY” đại diện cho việc một biến được sử dụng ở đâu đó.

- Giả sử mọi “TINY BIRD" đều thành giant bird: giả sử thay mọi let thành var.
- “Controlled area” tượng trưng cho block/function-body tương ứng mà một biến được khai báo.
- “BERRY” được ăn bởi một “BIRD” có cùng label, và “BERRY” phải nằm trong “controlled area” của “BIRD”. Đồng thời nếu có nhiều “BIRD” như vậy, “BIRD” có tầm kiểm soát nhỏ nhất chiếm đc “BERRY” $\Rightarrow$ một biến luôn refer đến cái localize gần nhất.

Ngoại trừ ràng buộc không có 8 bird cùng label (chỉ có để backtrack), các ràng buộc còn lại mang ý nghĩa:

- Mỗi “BERRY” nằm trong controlled area của ít nhất 1 “BIRD” cùng label -> biến phải khai báo mới dùng được (dù cái này không hẳn đúng trong *JavaScript*).
- Không có 2 bird cùng label và cùng controlled area $\Rightarrow$ ko thể có hai biến cùng tên trong một scope.

Giờ ta quay lại bài toán này, ở đây mình sẽ trình bày lời giải trên văn cảnh bài toán *JavaScript*:

- Đầu tiên, ta cần tìm xem khi một biến được đề cập đến, nó tương ứng với lệnh khai báo biến nào. Ta duyệt AST theo chiều sâu hai lần: Lần một sẽ lưu lại với mỗi scope (block/function-block) những biến được khai báo trong scope đó. Lần hai, ta duyệt cây theo chiều sâu để tìm những nơi có một biến được sử dụng, tìm scope gần nhất mà tên biến đó được khai báo để xác định lệnh khai báo biến được sử dụng đó.
- Giờ ta sẽ chia các nút lá (các lệnh khai báo biến và các nơi sử dụng biến) thành các nhóm, mỗi nhóm ứng với một biến. Ta biết rằng, khi ta đổi tên một biến thì cả nhóm này phải thay đổi cùng nhau.
- Nhận xét thêm rằng, kho tên biến hợp lệ (xâu kí tự gồm không quá 5 ký tự là rất lớn ($26^5 > 11 \times 10^6$), nên mỗi khi đổi tên, ta luôn đổi sang một tên chưa dùng đến, để tránh những rắc rối về việc trùng tên biến không đáng có.
- Như vậy, ta sẽ thấy rằng các nhóm biến cùng tên sẽ được xử lý độc lập và lần lượt. Do chỉ có tối đa 8 biến cùng tên, ta có thể duyệt qua toàn bộ tập con các biến không được đổi tên. Do mỗi biến được đổi tên đều mang tên độc lập, nó sẽ không gây ra rắc rối về việc trùng tên biến. Do đó ta cần kiểm tra xem các với các biến giữ nguyên tên cũ, nó có gây ra mâu thuẫn hay lỗi gì không.

F - Fun with Fibonacci

Nguyễn Đinh Quang Minh - khán giả

Nhận xét: mặc dù đề bài vô cùng đơn giản, đây là một bài toán đòi hỏi nhiều tầng lớp suy nghĩ, vận dụng nhiều kiến thức toán, và cả một chút "niềm tin" (nếu bạn không chứng minh được cụ thể). Do đó, không ngạc nhiên khi không đội nào giải được bài F trong giờ thi.

Trước hết, ta cần tìm cách giải bài toán mà không cần dùng đến số nguyên lớn. Để ý rằng, với modulo M bất kì, dãy Fibonacci modulo M luôn lặp lại sau một chu kì nào đó. Dãy lặp lại sau chu kì d khi và chỉ khi $F(d) \% M \equiv 0$ và $F(d+1) \% M \equiv 1$. Ta kí hiệu $\Psi(M)$ là độ dài chu kì *bất kì* của dãy Fibonacci modulo M. Hiển nhiên, mọi bội của $\Psi(M)$ đều là độ dài chu kì thỏa mãn.

Theo định nghĩa, ta có:

$$\begin{aligned} & G(K, N) \% M \\ = \; & F(G(K-1, N)) \% M \\ = \; & F(G(K-1, N) \% \Psi(M)) \% M \\ = \; & F(F(G(K-2, N)) \% \Psi(M)) \% M \\ = \; & F(F(G(K-2, N) \% \Psi(\Psi(M))) \% \Psi(M)) \% M \\ = \; & \dots \end{aligned}$$

Đến cuối cùng, ta sẽ phải tính $F(N)$ theo modulo $\Psi(\Psi(\Psi(...\Psi(M)...)))$. Bạn có thể ngạc nhiên, nhưng ta có thể dựng hàm Ψ sao cho $\Psi(\Psi(\Psi(...\Psi(M)...)))$ không quá lớn. Hãy chứng minh tại sao.

Trước hết, ta có một số bổ đề sau:

1) Nếu $gcd(p, q) = 1$, $\Psi(pq) = lcm(\Psi(p), \Psi(q))$.
2) Nếu p là số nguyên tố, $\Psi(p^k) \mid p^{k-1}\Psi(p)$.

Việc chứng minh xin nhường lại cho bạn đọc (gợi ý: định lý thặng dư Trung Hoa).

Giờ ta chú ý tới $\Psi(p)$ với p nguyên tố. Trước hết, dễ dàng tìm được $\Psi(2) = 3$ và $\Psi(5) = 20$. Ta chứng minh 2 điều sau:

1) Nếu $p \equiv \pm 1 \pmod 5$ thì $F(p-1) \equiv 0 \pmod p$ và $F(p) \equiv 1 \pmod p$, do đó $\Psi(p) \mid (p-1)$.
2) Nếu $p \equiv \pm 2 \pmod 5$ thì $F(2p+2) \equiv 0 \pmod p$ và $F(2p+3) \equiv 1 \pmod p$, do đó $\Psi(p) \mid (2p+2)$.

Để chứng minh (1) và (2), ta sử dụng đẳng thức quen thuộc đúng với mọi $n > 0$:

$$F(n) = 2^{1-n} * \sum_{k=0}^{\infty} \binom{n}{2k+1} 5^k$$

Ngoài ra, ta có: $5^{\frac{p-1}{2}} \equiv \left(\frac{p}{5}\right) (mod\, p)$ là kí hiệu *Legendre* của p theo $modulo\, 5$, do đó:
$p \equiv \pm 1 (mod\, 5) \Rightarrow 5^{\frac{p-1}{2}} \equiv 1 (mod\, p)$, và $p \equiv \pm 2 (mod\, 5) \Rightarrow 5^{\frac{p-1}{2}} \equiv -1 (mod\, p)$.

Với $p \equiv \pm 1 (mod\, 5)$, ta có:
$$\binom{p-1}{2k+1} \equiv -1 \pmod{p}$$
(với mọi k thỏa mãn $2k+1 \leq p-1$)

Suy ra:
$$F(p-1) \equiv -2^{2-p} * \sum_{k=0}^{\frac{p-3}{2}} 5^k \equiv -2^{2-p} * \frac{5^{\frac{p-1}{2}} - 1}{4} \equiv 0 \pmod{p}$$

Mặt khác:
$$\binom{p}{k} \equiv 0 \pmod{p} \forall k \notin \{0, p\}$$
(theo định lý Lucas)

Suy ra: $F(p) \equiv 2^{1-p} \times 5^{\frac{p-1}{2}} \equiv 1 (mod\, p)$.
Vậy (1) được chứng minh.

Tương tự, với $p \equiv \pm 2 (mod\, 5)$, ta có:
$F(p) \equiv 2^{1-p} \times 5^{\frac{p-1}{2}} \equiv -1 (mod\, p)$

Mặt khác:
$$\binom{p+1}{k} \equiv 0 \pmod{p} \forall 1 < k < p$$
(cũng theo định lý Lucas)

Suy ra:
$$F(p+1) \equiv 2^{-p} * \left(\binom{p+1}{1} * 5^0 + \binom{p+1}{p} * 5^{\frac{p-1}{2}}\right)$$
$$\equiv 2^{-p} \times ((p+1) \times 1 + (p+1) \times -1) \equiv 0 \, (mod\, p)$$

$\Rightarrow F(p+2) \equiv F(p) + F(p+1) \equiv -1 \, (mod\, p)$

Từ đó ta có:
$F(2p+2) \equiv F(p+2)^2 - F(p)^2 \equiv 0 \, (mod\, p)$
$F(2p+3) \equiv F(p+1)^2 + F(p+2)^2 \equiv 1 \, (mod\, p)$

Vậy (2) được chứng minh.

Từ các chứng minh trên, ta định nghĩa hàm $\Psi(p)$ như sau:
$\Psi(2) = 3, \Psi(5) = 20$,
$\Psi(p) = p - 1$ *với* $p \equiv \pm 1 \ (mod\ 5)$ và p nguyên tố,
$\Psi(p) = 2(p+1)$ với $p \equiv \pm 2 \ (mod\ 5)$ và p nguyên tố,
$\Psi(pq) = lcm(\Psi(p), \Psi(q))$ nếu p, q nguyên tố cùng nhau,
$\Psi(p^k) = p^{k-1}\Psi(p)$ nếu p nguyên tố.

Xét một số $M = {p_1}^{a_1} \times {p_2}^{a_2} \times ... \times {p_k}^{a_k}$ với $a_1, a_2, ..., a_k > 0$ và $p_1 < p_2 < ... < p_k$. Khi đó ta có:
$\Psi(M) = lcm({p_1}^{a_1} \times \Psi(p_1), {p_2}^{a_2} \times \Psi(p2), \ldots, {p_k}^{a_k} \times \Psi(p_k))$.
Gọi $P(x)$ là thừa số nguyên tố lớn nhất của x. Giả sử $P(M) = p_k \geq 7$, ta dễ dàng chứng minh được $P(\Psi(p_i)) < p_k$ với mọi i. Do đó $P(\Psi(M)) < P(M)$. Điều này có nghĩa là, nếu ta thay M thành $\Psi(M)$ thì sau một số hữu hạn bước (ít hơn số thừa số nguyên tố của M), số M chỉ gồm các thừa số 2, 3 và 5. Giả sử ở bước đó ta có $M = 2^a 3^b 5^c$, thì
$\Psi(M) = lcm(2^{a-1} \times 3, 3^{b-1} \times 8, 5^{c-1} \times 20) = 2^x 3^y 5^z$, với
$x = max(3, a-1), y = max(1, b-1), z = c$. Như vậy, sau một ít bước (cụ thể là $max(a-3, b-1)$ bước), M sẽ có dạng $2^3 \times 3 \times 5^c = 24 \times 5^c$, thỏa mãn $\Psi(M) = M$.
Nói tóm lại, ta sẽ tính $\Psi(M)$ theo định nghĩa cho đến khi M có dạng 24×5^c, khi đó bài toán của chúng ta sẽ trở thành:

Tính $F(F(...F(N) \% M...) \% M) \% M$, với $M = 24 \times 5^c$

Trước hết, ta nhận thấy với giới hạn của đề bài, c không thể vượt quá 8, do đó M không vượt quá $24 \times 5^8 = 9375000$. Ngoài ra, chỉ có 9 giá trị có thể của M. Do vậy, ta sẽ tìm cách chuẩn bị trước cho cả 9 giá trị này trước khi trả lời các truy vấn.
Xét một giá trị M cố định, nếu ta dựng một đồ thị gồm M đỉnh được đánh số từ 0 đến $M-1$, và các cạnh có hướng từ đỉnh u đến đỉnh $(F(u) \ mod \ M)$, thì ta cần tìm đỉnh thứ K trên đường đi xuất phát từ N. Hiển nhiên, đồ thị này gồm một vài chu trình và các đỉnh không thuộc chu trình thì có cạnh hướng về phía chu trình. Nếu ta có thể tính được với mỗi đỉnh u các giá trị sau:

a) Đỉnh đầu tiên thuộc chu trình trên đường đi từ u
b) Số bước từ u đến đỉnh đó
c) Độ dài chu trình

Thì ta sẽ tính được đáp án **nếu như số bước từ u đến chu trình không lớn hơn k**.
Tất cả các đại lượng trên đều có thể tính rất dễ dàng.
Vậy nếu k nhỏ hơn số bước từ u đến chu trình thì sao? Thoạt nhìn, ta có thể nghĩ rằng số bước từ u đến chu trình có thể lên đến M bước, tuy nhiên với $M = 24 \times 5^c$, ta hoàn toàn có thể chứng minh được giá trị này thực ra chỉ lên đến 4 mà thôi!

Phần chứng minh khá dài dòng, người chữa xin phép lược bỏ ở đây và coi như một bài tập cho bạn đọc.

Tóm tắt hướng chứng minh:

- Gọi $d(n, M)$ là khoảng cách từ đỉnh n đến chu trình trong đồ thị với modulo M.
- Với $c = 0, M = 24$, tìm được max các $d(n, 24)$ bằng 4.
- Chứng minh $F(24p + r) \equiv F(24q + r) \pmod{5^c} \Leftrightarrow p \equiv q \pmod{5^c}$.
- Từ đó suy ra $d(n, M) = d(n \bmod 24, 24) \Rightarrow \max d(n, M) = 4$ với mọi $M = 24 \times 5^c$.

Tóm lại, xuất phát từ đỉnh N, ta chỉ cần nhảy từng bước cho đến khi đỉnh hiện tại thuộc chu trình, sau đó trả lời trong $O(1)$ với đáp án được chuẩn bị sẵn.

Code đã AC của người chữa có thể tìm thấy ở https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/regional/codes/funwithfibonacci.cpp.

G - Grab a Graph

Lời giải : Trần Tấn Phát (Send Bobs to Alice)

Bài toán 1: $N \leq 72, M \leq 2525$.
Dựng đồ thị đầy đủ n đỉnh với cạnh $i \to j$ có trọng số $j - i$, sẽ có 2^{n-i-1} đường đi ngắn nhất khác nhau từ i đến n. Sau đó thêm 1 đỉnh bắt đầu, dựa vào dạng nhị phân của A để thêm vào các cạnh phù hợp.
Số đỉnh sử dụng: $n = 61, m \leq 60 \times 59 / 2 + log_2(A) = 1830$.

Bài toán 2: $N \leq 88, M \leq 214$.
Zeckendorf's theorem: Mỗi số nguyên đều có thể được biểu diễn dưới dạng tổng của 1 số số Fibonacci.

Đầu tiên tạo cạnh từ $2 \to 1$ với trọng số 1, sau đó với mỗi $i \geq 3$, tạo cạnh từ $i \to i - 2$ với trọng số 2, và cạnh từ $i \to i - 1$ với trọng số 1. Dễ thấy từ i đến 1 có $fib(i)$ đường đi khác nhau với $fib(1) = fib(2) = 1$. Dựa vào kết quả phía trên, thêm 1 đỉnh bắt đầu và 1 số các cạnh phù hợp (giới hạn bài toán 2 chặt, cần dùng 87 số fibonacci đầu tiên nên chỉ có thể dùng thêm 1 đỉnh bắt đầu)
Số đỉnh sử dụng: $n = 88, m \leq 86 + 87 + 86 \; div \; 2 = 214$.

H - Hydra's Heads

Lời giải: Đinh Nguyên Khôi (HCMUS - Intimidate)

Bài H này ta hoàn toàn có rất nhiều hướng tiếp cận. Cách tiếp cận “khỏe” nhất đó chính là sử dụng BFS. Xuất phát từ trạng thái ban đầu là con rồng chỉ có H đầu và T đuôi, ta BFS trạng thái đó theo đề bài và dừng cho đến khi $H = 0$ và $T = 0$. Tuy nhiên, nếu như cứ với mỗi test ta đều BFS như vậy thì khả năng cao sẽ bị Time limit nếu cài đặt không khéo. Vì vậy, một cách an toàn để ta có thể nghĩ đến đó là ta duyệt mọi trạng thái (H, T), mỗi biến chạy từ 1 đến 100, với mỗi trạng thái như vậy ta sử dụng BFS và lưu các trạng thái đó vào mảng hằng. Việc duyệt như vậy không lâu (chỉ mất của bạn tầm 15 giây chờ đợi là đã có được bảng hằng rồi), sau đó copy mảng hằng này vào source code chính của bạn. Như vậy cứ với mỗi bộ test bạn in ra giá trị trong mảng hằng tương ứng thôi. Độ phức tạp $O(T)$ với T là số lượng bộ test.

Fun fact: Trường hợp -1 sẽ không bao giờ xảy ra.

Đánh giá bài H: Bài H này thực tế ta còn một cách dùng toán học các kiểu, tuy nhiên việc dùng toán học để tìm ra lời giải cũng không phải là chuyện dễ như ăn kẹo và ta sẽ mất năng lượng vào việc phải mày mò công thức. Thay vào đó, chỉ cần duyệt BFS, không cần dùng não, code cũng không dài là đã có thể giải được bài H. Đối với những bài dễ nhất đề như bài C hoặc bài H, thì việc tốn năng lượng vào chúng là thực sự không cần thiết.

I - Insider's Identity

Nguyễn Diệp Xuân Quang (đội HCMUS-Intimidate)

Kí hiệu trong lời giải

- $|S|$: độ dài xâu S
- $S[i, j]$: xâu con từ vị trí i đến vị trí j của S

Phân tích ban đầu

Trước hết, ta cần chú ý một giới hạn quan trọng trong đề bài: số lượng kí tự 1 chiếm ít nhất một nửa xâu P . Nói cách khác, gọi K là số lượng kí tự * trong xâu P . Ta có $K \leq |P| / 2$. Mà theo đề bài, $|P| \leq 30$, nên $K \leq 15$.

Với giới hạn K nhỏ như vậy, ta có thể sinh tất cả 2^K xâu thỏa mãn xâu P và có độ dài là $|P|$. Gọi tập xâu này là T_P . Bài toán bây giờ trở thành: Đếm xem có bao nhiêu xâu S có độ dài n và ít nhất một trong các xâu của tập T_P là xâu con của S .

Bài toán con với $K = 0$

Trước hết, ta sẽ giải quyết một bài toán con của bài toán trên: giả sử $K = 0$. Bài toán sẽ trở nên đơn giản hơn: Đếm xem có bao nhiêu xâu S có độ dài n và nhận xâu P làm xâu con.

Để giải quyết bài toán này, ta cần hiểu cách mà thuật toán KMP hoạt động. Nếu bạn chưa biết về KMP, các bạn có thể tham khảo bài viết tại: http://vnoi.info/wiki/translate/wcipeg/kmp.

Ta sẽ sử dụng thuật toán quy hoạch động với ý tưởng tương tự thuật toán KMP.

Trước hết, ta cần xây dựng bảng $nxt[i][c]$ như sau:

- Nếu $i < |P|$ và $P[i] = c$ thì $nxt[i][c] = i + 1$
- Ngược lại, gọi T là xâu P sau khi thêm kí tự c ở đuôi. Khi đó, $nxt[i][c]$ là số nguyên x lớn nhất sao cho hậu tố thứ x của T trùng với tiền tố thứ x của P (nói cách khác, $P[0, x-1] = T[i-x+1, i]$).

Bảng nxt có ý nghĩa tương tự hàm π trong bài viết trên

Khi đó, gọi $dp[i][j][contain]$ là số lượng xâu S :

- Có độ dài i
- Đã khớp được j vị trí đầu tiên với xâu P
- $contain = 1$ có nghĩa là: xâu S đã nhận xâu P làm xâu con. $contain = 0$ có nghĩa là: xâu S chưa nhận xâu P làm xâu con.

Với mỗi trạng thái $dp[i][j][contain]$, ta duyệt lần lượt kí tự 0 và 1 (gọi kí tự đang duyệt là c) và thử đặt kí tự này vào vị trí $i + 1$. Ta cần cập nhật trạng thái quy hoạch động như sau.

- Gọi $x = nxt[i][c]$.
- Gọi $newContain$ với ý nghĩa: $newContain = 1$ nếu xâu $S + c$ sẽ nhận P làm xâu con, ngược lại thì $newContain = 0$.
- Nếu $x = |P|$ thì có nghĩa là xâu P đã xuất hiện trong xâu S từ vị trí $i - |P| + 1$ đến vị trí i. Khi đó, $newContain = 1$. Ngược lại, $newContain = contain$.
- Ta cập nhật $dp[i+1][x][newContain]$ += $dp[i][x][contain]$.

Đáp án cần tìm là tổng $dp[n][j][1]$ với mọi j từ 0 đến $|P|$.

Độ phức tạp của thuật toán này là $O(n \times |P|)$.

Trở lại bài toán ban đầu

Ta sẽ trở lại bài toán ban đầu: Đếm xem có bao nhiêu xâu S có độ dài n và ít nhất một trong các xâu của tập T_P là xâu con của S.

Trước hết, ta có thể mở rộng thuật toán quy hoạch động trên bằng cách lưu trữ số kí tự đã khớp cho cả 2^K xâu trong tập T_P. Ví dụ, với $K = 2$ thì trạng thái quy hoạch động sẽ là $dp[i][j_0][j_1][j_2][j_3][contain]$, với j_0, j_1, j_2, j_3 lần lượt là số kí tự đã khớp với xâu $T_P[0], T_P[1], T_P[2], T_P[3]$, và $contain$ nhận giá trị 1 nếu xâu S chứa ít nhất một trong các xâu thuộc tập T_P hoặc 0 nếu ngược lại.

Tuy nhiên, nếu làm như vậy thì số lượng trạng thái quy hoạch động sẽ là $2 \times n \times |P|^{2^K}$, quá lớn để có thể chấp nhận được. Do đó, ta cần tìm một cách biểu diễn khác thay thế cho dãy chỉ số trong trạng thái quy hoạch động trên.

Đến đây, ta sẽ sử dụng một cấu trúc *automaton* (*finite state machine* - máy trạng thái hữu hạn). Về bản chất, *automaton* này là một cây tiền tố (*trie*) biểu diễn một tập xâu T, mỗi đỉnh sẽ tương ứng với một xâu là tiền tố của ít nhất một trong các xâu thuộc tập T (và cũng tương ứng một dãy chỉ số j hợp lệ). Ngoài ra, tại mỗi đỉnh còn lưu trữ thêm các thông tin sau:

- $suffix\_link$ của đỉnh u là cạnh nối đến hậu tố hợp lệ dài nhất (*longest proper suffix*) của u.

- $nxt[c]$ của đỉnh u là cạnh nối đến hậu tố dài nhất (*longest suffix*) của $u + c$ (ý nghĩa của dãy nxt này giống với bảng nxt được đề cập trong mục trước).

Automation có thể được xây dựng bằng thuật toán *Aho-Corasick* trong độ phức tạp $O(L)$ với L là tổng độ dài các xâu trong tập. Các bạn có thể tham khảo thêm về trie, về automaton, thuật toán *Aho-Corasick* trong bài viết: https://cp-algorithms.com/string/aho_corasick.html

Khi đã xây dựng được *automaton*, trạng thái quy hoạch động sẽ trở thành $dp[i][u][contain]$ với u là chỉ số của đỉnh trong *automation* (i và $contain$ vẫn giữ nguyên ý nghĩa).

Với mỗi trạng thái $dp[i][u][contain]$, ta cũng duyệt c lần lượt là kí tự 0 và 1 và cập nhật trạng thái quy hoạch động như sau.

- Gọi $x = nxt[c]$ (với mảng nxt thuộc đỉnh u).
- Gọi $newContain$ với ý nghĩa: $newContain = 1$ nếu xâu $S + c$ sẽ nhận ít nhất một trong các xong thuộc tập T_P làm xâu con, ngược lại thì $newContain = 0$.
- Nếu đỉnh u biểu diễn một xâu thuộc tập T_P thì $newContain = 1$. Ngược lại, $newContain = contain$.
- Ta cập nhật $dp[i+1][x][newContain]$ += $dp[i][u][contain]$.

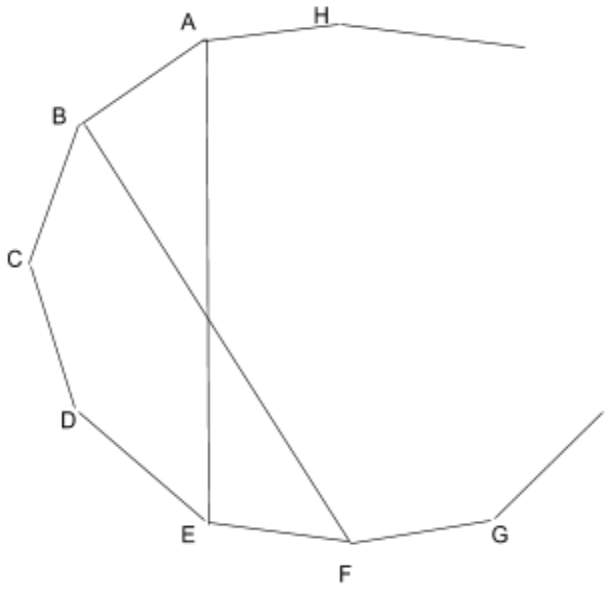
Đáp án cần tìm là tổng $dp[n][u][1]$ với mọi u từ 0 đến số lượng đỉnh trong *automation*.

Độ phức tạp: $O(N * 2^K * |P|)$.

J - Jurassic Jungle

Trước hết ta cần tìm những loại đồ thị nào thoả mãn điều kiện đề bài.

Xét đồ thị sau:



Tìm 1 chu trình Hamilton **H** bất kỳ: $ABCDEFG....H$
Xét 1 cạnh bất kỳ không thuộc chu trình H. Đặt d là khoảng cách giữa 2 đỉnh trên H.

Trong hình vẽ trên ta có cạnh $A-E$ với $d = 4$.
Ta thấy có chu trình $BCDEAH.....GF$ nên có cạnh BF, do đó tất cả các cạnh d đều tồn tại

Có $BFEAH....GCD \Rightarrow BD$ tồn tại $\Rightarrow$ tất cả các cạnh $d-2$ đều tồn tại
Có $AEFBCDH.....G \Rightarrow AG$ tồn tại $\Rightarrow$ tất cả các cạnh $d+2$ đều tồn tại

Nếu d chẵn thì tất cả các cạnh chẵn đều tồn tại $\Rightarrow$ 2 tồn tại $\Rightarrow AEDCBH....GF \Rightarrow$ tồn tại hay $d+1$ tồn tại hay tất cả các cạnh lẻ tồn tại.

Nếu d lẻ n lẻ thì ta có tất cả các cạnh $1,3,.....n-2$ tồn tại hay $n-(n-2) = 2$ tồn tại $\Rightarrow$ tất cả các cạnh chẵn cũng tồn tại.

Vậy kết luận được là ngoại trừ n chẵn d lẻ thì nếu đồ thị có thêm 1 cạnh mà $d! = 1$ thì luôn sinh được full graph. Trường hợp n chẵn d lẻ chính là full đồ thị 2 phía $n/2$ đỉnh mỗi bên.

K - Kingdom of Kittens

Lời giải: Nguyễn Khánh (UET - Map)

Bài này có rất nhiều lời giải, trong đó lời giải mà viết hết trường hợp cần xét ra là dễ tiếp cận nhất.

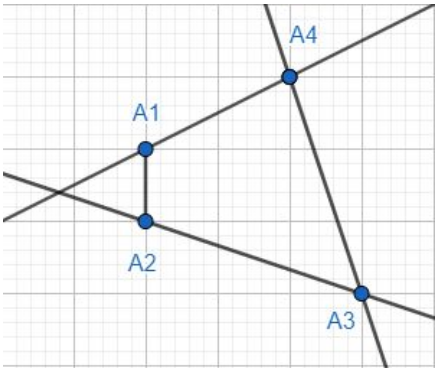
Trước hết, đề bài không nói gì đến việc các điểm đã cho có phân biệt hay không. Vì vậy, việc đầu tiên cần làm là loại đi các điểm bị trùng.

Lời giải này sẽ dựa trên một vài nhận xét mà dễ dàng quan sát được:

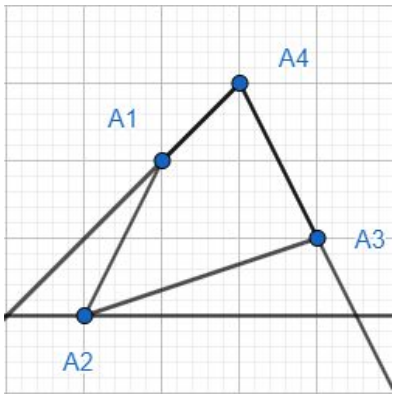
- Tập điểm đã cho phải tạo thành một đa giác lồi nghiêm ngặt không quá 6 đỉnh
- Mọi điểm trong tập điểm đã cho phải nằm trên cạnh của bao lồi nói trên

Gọi A là bao lồi nói trên và n là số đỉnh của A. Ta sẽ đi tìm các cách vẽ tam giác chứa tất cả đỉnh của hình A; các điểm mà không phải đỉnh của A sẽ được xử lý riêng.

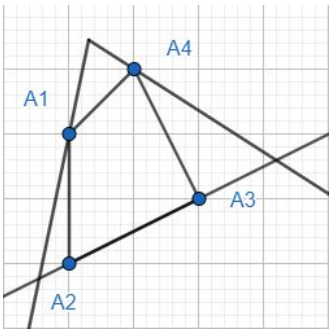
- Nếu $n \leq 3$, hiển nhiên vẽ được một tam giác đi qua tất cả đỉnh của A
- Với $n = 4$, có 3 cách để vẽ một tam giác đi qua 4 đỉnh:
    - Cả ba cạnh của tam giác đều đi qua 2 điểm (không mất tính tổng quát, giả sử đi qua A_1A_2, A_2A_3 và A_3A_4):



    - Có 2 cạnh đi qua 2 điểm và cạnh còn lại đi qua 1 điểm:

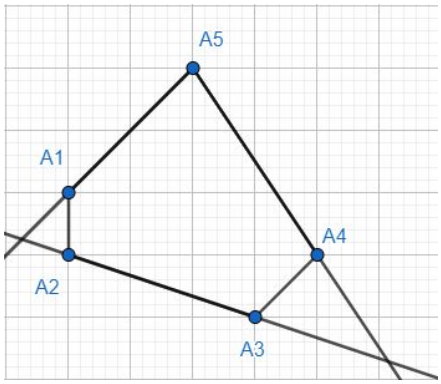


- Có 1 cạnh đi qua 2 điểm và 2 cạnh đi qua 1 điểm:

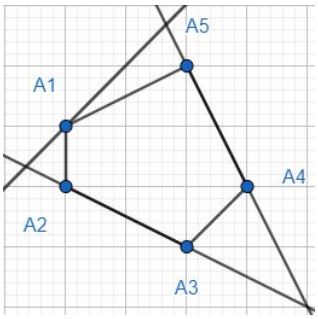


Có thể thấy rằng, ta luôn vẽ được cách 2 và 3 và cách 2 luôn tốt hơn cách 3.

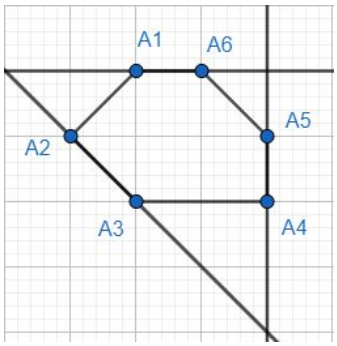
- Khi $n = 5$, có 2 cách vẽ tam giác:
  - Có 3 cạnh đi qua 2 điểm:



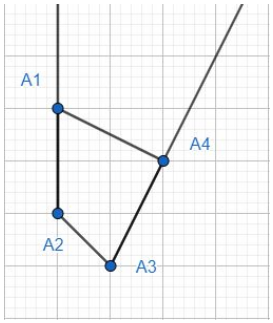
  - Có 2 cạnh đi qua 2 điểm và 1 cạnh đi qua một điểm:



- Trong trường hợp $n = 6$, chỉ có một cách vẽ, trong đó 3 cạnh của tam giác đều đi qua 2 đỉnh của A:



Lưu ý có nhiều trường hợp vẽ mà không tạo thành tam giác, ví dụ trong cách vẽ 1 của $n = 4$:



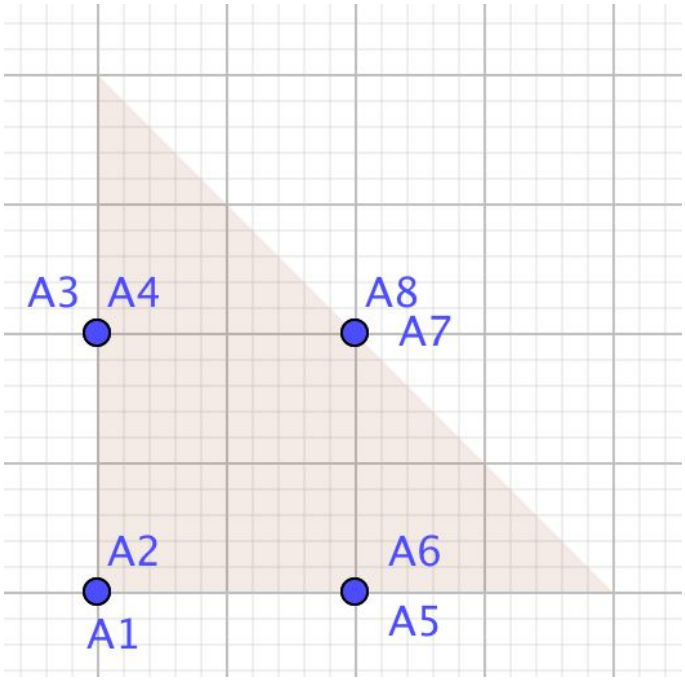
Ta cần phải kiểm tra xem với cách chọn cạnh đa giác hiện tại, có thể vẽ được tam giác không. Việc kiểm tra vô cùng dễ, để dành cho bạn đọc.

Fun fact

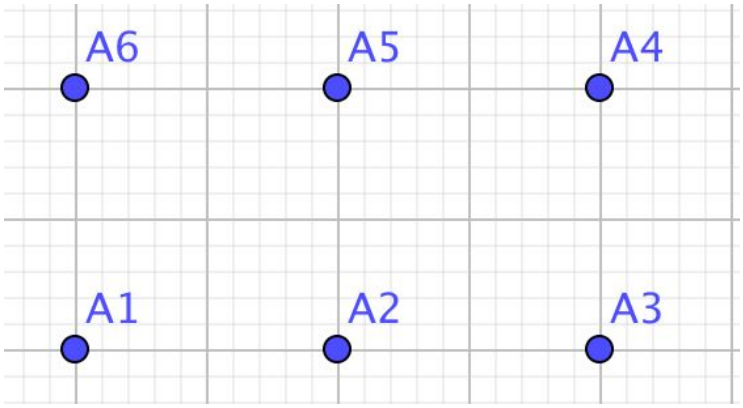
Đây là một bài toán mới nhìn qua thì rất đơn giản nhưng trên thực tế lại khá khó do có rất nhiều trường hợp. Ban tổ chức đã sinh khoảng $1,000,000$ test vô cùng đa dạng cho bài này (10^6 test, mỗi file input có chứa rất nhiều test).

Dưới đây là 1 số test mà các team bị sai:

- Test có 8 điểm tạo thành hình vuông (mỗi đỉnh hình vuông có 2 điểm):



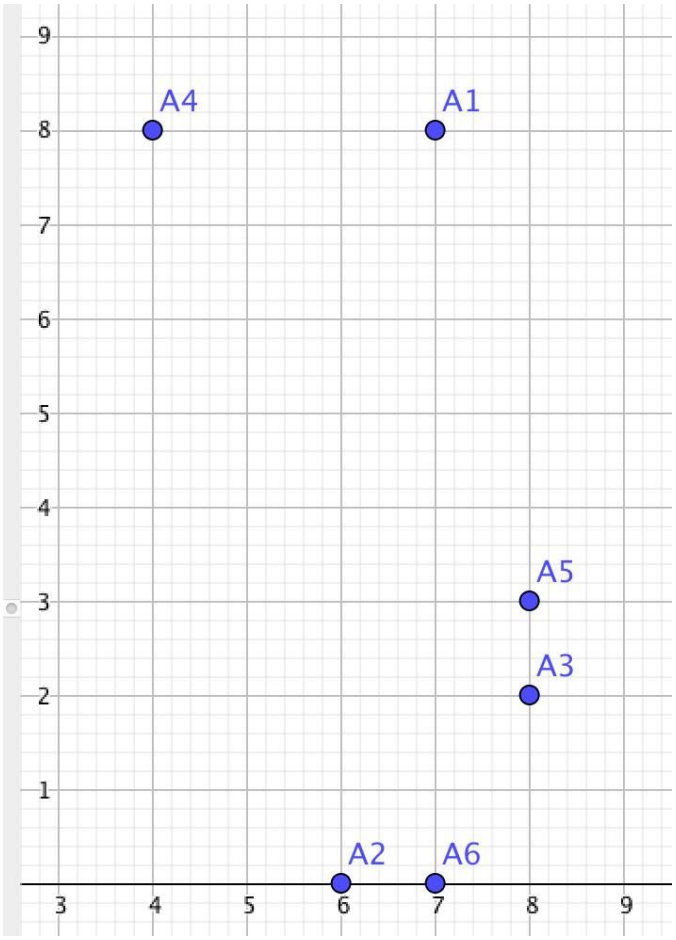
- Test 6 điểm tạo thành bàn bi-a. Đây là 1 trong những test khó nhất do rất dễ xét nhầm thành trường hợp trên:



- Test giết đội Hàn Quốc "Teeeeeeeeeeeeemo needs blue buff" -- đội đã đúng vài trăm nghìn test cho đến test này:

Point

- A1 = (7, 8)
- A2 = (6, 0)
- A3 = (8, 2)
- A4 = (4, 8)
- A5 = (8, 3)
- A6 = (7, 0)



L - Lazy Learner

Bài toán mới nhìn qua rất lằng nhằng phức tạp, tuy nhiên nếu ta tỉ mỉ suy nghĩ từng bước sẽ thấy rất đơn giản.

Đầu tiên ta nhận thấy nếu xâu P là subsequence của substring $S_l...S_r$, thì nó cũng là subsequence của tất cả các substring $S_l...S_{r'}$ với $r' > r$. Đây cũng chính là tư tưởng chính của bài toán. Khi ta cố định L, ta có thể kiểm tra subsequence và trả lời các truy vấn $(L,\ R)$. Cụ thể:

- Xét các truy vấn có cùng L.
    - Khi cố định L, với mỗi xâu con, tính $f(i)$ là vị trí R nhỏ nhất mà xâu con i là subsequence của $[L,\ R]$.
    - Xét các sự kiện:
        - Truy vấn ở R.
        - Có 1 xâu con i là subsequence của $[L,\ R]$
    - Xử lý lần lượt các event này theo R tăng dần. Ta cần thực hiện 2 loại truy vấn:
        - Thêm 1 số x vào cấu trúc dữ liệu.
            - Chú ý ta có thể sắp xếp lại tất cả các xâu, rồi dùng chỉ số trong mảng đã sắp xếp, thay vì sử dụng xâu -- như vậy cấu trúc dữ liệu của chúng ta chỉ cần lưu số (thay vì xâu) → giảm độ phức tạp thuật toán cũng như cài đặt đơn giản hơn.
        - Tìm số lớn thứ K trong cấu trúc dữ liệu.
- → Đpt: $Q \times log\ +\ |S| \times N \times log\ +\ S\ \times\ (tổng\ độ\ dài\ xâu\ con)$

Trong bài này ta có thể sử dụng nhiều cấu trúc dữ liệu khác nhau:

- **Fenwick tree**: Code chemthan, Code RR.
- **Segment tree**: Code của team trie.
- **C++ ordered set**: Code của bạn Nguyễn Xuân Tùng.