ACM ICPC HCMC 2017 Editorial

Overview

Tên bài	Số AC	Độ khó	Lời giải nhanh nhất
A - Arranging Wine	1	5	Phút 207 sleep 18000 - **The University of Tokyo**
B - Barcode	46	2	Phút 27 sleep 18000 - **The University of Tokyo**
C - Cu Chi Tunnels	124	1	Phút 12 sleep 18000 - **The University of Tokyo**
D - Dropping Ball	5	4	Phút 166 Aizu_Dragon - **University of Aizu**
E - Engaging with Loyal Customers	16	2	Phút 18 Reboom - **Peking University**
F - Famous Pagoda	18	3	Phút 51 Cxiv - Dxiv - **The University of Tokyo**
G - Game of Divisibility	3	3	Phút 163 Cxiv - Dxiv - **The University of Tokyo**
H - Height Preservation	47	2	Phút 59 Reboom - **Peking University**
I - ICPC Awards	147	0	Phút 5 Kizza Manau - **Universitas Indonesia**
J - Joining Network	5	3	Phút 126 unsigned - **UET VNU**
K - K-rotating	5	3	Phút 241 DoWF - **Korea University**
L - Land Inheritance	0	4	N/A

Bộ đề được chuẩn bị bởi:

- Lăng Trung Hiếu
- Phạm Văn Hạnh
- Nguyễn Thành Trung - RR
- Lê Đôn Khuê
- và 1 số thầy giáo dạy ở các trường đại học, cấp 3 trong cả nước.

Lời giải được đóng góp bởi:

- Lê Quang Tuấn, Nguyễn Đinh Quang Minh, Phạm Cao Nguyên - team VNU - unsigned.
- Nguyễn Diệp Xuân Quang - team HCMUS Illuminate
- Phạm Văn Hạnh
- Nguyễn Thành Trung - RR
- Lê Đôn Khuê
- Lăng Trung Hiếu

Nếu có thắc mắc về lời giải, bạn có thể hỏi ở: group VNOI.

Trong các lời giải có sử dụng 1 số bài viết trên VNOI wiki:

- Sàng số nguyên tố
- Tính nghịch đảo modulo
- Tối ưu hoá QHĐ
- Kĩ thuật bao lồi
- Các chủ đề về đồ thi

Các bạn có thể nộp thử bài ở Kattis.

Code mẫu:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/tree/master/2017/regional

Bảng xếp hạng cuối cùng: https://hochiminh17.kattis.com/standings

First to solve problem | Solved problem | Attempted problem | Pending judgement

RK	TEAM	SLV.	TIME	A	B	C	D	E	F	G	H	I	J	K	L
1	unsigned	9	1423		5 42	2 24	1 263	2 153	3 227		1 60	1 12	4 126	1 296	
2	Cxiv-Dxiv	9	1618		11 250	1 31	2 181		1 51	1 163	2 89	1 7	4 221	3 285	
3	sleep 18000	8	1034	1 207	2 27	1 12		1 --	2 99		1 68	1 16	2 224	3 281	
4	Reboom	8	1177		3 91	1 32	1 175	1 18	5 297		2 59	2 6		3 299	
5	DomiNUS	6	660		4 34	1 57		1 174	1 158		2 139	1 18	6 --		
6	HCMUS-Illuminate	6	667		2 57	1 55		1 217	1 107	1 --	2 184	1 7			
7	semoga ayas juara	6	760		6 115	1 39		1 60	2 173		2 218	1 15			
8	\OAO/	6	892		6 96	1 27		1 50	11 282		1 124	1 13			
9	BK.CS	6	938		4 64	1 56		1 92			7 231	1 5	2 290		
10	FPT_HAN_ThreeFrogs	6	940		7 86	2 46		6 --	3 119		2 177	1 9	2 283		

A. Arranging Wine (5*)

Tác giả: Lăng Trung Hiếu

Lời giải 1: Phạm Văn Hạnh

Thể loại: Tổ hợp, Inclusion-Exclusion

Nhận xét: Giả sử ta xếp các thùng rượu thành một hàng ngang gồm R + W thùng rượu, sau đó xếp các thùng rượu liên tiếp cùng màu vào một chồng, ta sẽ được một dãy các chồng rượu vang đỏ trắng xen kẽ (mỗi chồng chứa đúng một loại rượu và hai chồng cạnh nhau có rượu khác màu). Do đó, bài toán có thể phát biểu lại như sau:

Đếm số dãy nhị phân gồm R bit 1 và W bit 0 sao cho có không quá D bit 1 liên tiếp.

Ta xếp W bit 0 lên dãy và lần lượt đặt R bit 1 vào đầu dãy, cuối dãy hoặc xen giữa các bit 0. Bài toán đó được chuyển về bài toán tổng quát sau đây:

Đếm số dãy số nguyên không âm $(x_1, x_2, ..., x_n)$ có $\sum_{i=1}^{n} x_i = S$ và $0 \le x_i \le k \ \forall 1 \le i \le n$.

Để giải quyết bài toán này, ta sử dụng nguyên lý bao hàm loại trừ với mục đích đưa về bài toán "dễ giải hơn" - bài toán không có ràng buộc $x_i \le k$. Đáp số bài toán khi đó là C_{S+n-1}^{n-1}. Phần chứng minh xin nhường lại cho bạn đọc.

Bài toán sau đây cũng có lời giải đơn giản, là hệ quả của "bài toán đơn giản" phía trên: Cho $A \subset \{1, 2, ..., n\}$, đếm số dãy $(x_1, x_2, ..., x_n)$ có $\sum_{i=1}^{n} x_i = S$ và $0 \le x_i \forall i \notin A$, $k < x_i \forall i \in A$.

Vai trò của tập hợp A trong dãy trên là: với mọi $i \in A$, ta biết chắc chắn x_i là "phần tử xấu" $(x_i > k)$. Ngược lại, nếu $i \notin A$, x_i có thể xấu cũng có thể tốt (có thể $x_i \le k$ hoặc $x_i > k$, ta chưa rõ).

Gọi $f(A)$ là đáp số của bài toán trên, với tập hợp A là tập hợp các phần tử "chắc chắn xấu". Để tìm đáp số của bài toán ban đầu (tìm số dãy $(x_1, x_2, ..., x_n)$ không chứa "phần từ xấu"), ta nghĩ tới ý tưởng sau: Gọi $result$ là đáp số của bài toán.

0. Đầu tiên ta giả sử đáp số bài toán là số dãy mà các phần tử có thể "tốt hoặc xấu tùy ý". Sau đó ta tìm cách loại đi những đáp án xấu.

```
int result = f(∅)
```

1. Đầu tiên, ta sẽ loại bỏ các dãy có **chính xác** một phần tử xấu, nghĩa là:

```
for (int i = 1; i <= n; i++)
    result -= f({i})
```

2. Sau bước vừa rồi, ta thấy trong result không chứa các dãy có **duy nhất** một phần tử xấu. Tuy nhiên, những dãy có **chính xác** hai phần tử xấu vừa bị trừ 2 lần ở bước trên, nghĩa là đang bị trừ thừa 1 lần, nên giờ phải cộng ngược lại:

```
for (int i = 1; i <= n; i++)
```

```
for (int j = i + 1; j <= n; j++)
      result += f({i, j})
```

3. Sau bước vừa rồi, ta đã hoàn toàn loại bỏ khỏi `result` những dãy có **chính xác** một hoặc hai phần tử xấu. Giờ ta muốn loại bỏ khỏi `result` những dãy có **chính xác** ba phần tử xấu. Hãy quan sát những dãy này đã bị tính bao nhiêu lần ở `result` trong ba bước trên: Ở bước 0, mỗi dãy được cộng 1 lần. Ở bước 1, mỗi dãy bị trừ 3 lần. (Một dãy có ba vị trí xấu x,y,z sẽ được tính trong $f(\{x\}),f(\{y\}),f(\{z\})$). Tới bước 2, mỗi dãy lại được cộng lại ba lần. (Một dãy có ba vị trí xấu x,y,z sẽ được tính trong $f(\{x,y\}),f(\{y,z\}),f(\{z,x\})$). Như vậy, hiện tại mỗi dãy có **chính xác** ba phần tử xấu hiện đang được tính 1 lần trong `result.` Công việc của ta bây giờ là trừ nó đi:

```
for (int i = 1; i <= n; i++)
      for (int j = i + 1; j <= n; j++)
            for (int k = j + 1; k <= n; k++)
                  result -= f({i, j, k}).
```

…

Tiếp tục lập luận tương tự, ta đi tới quy tắc tính sau đây: **Xét mọi tập con A của** $\{1,2,...,n\}$**.** Nếu A có **chẵn** phần tử, ta **cộng thêm** vào result $f(A)$. Ngược lại, nếu A có **lẻ** phần tử, ta **trừ đi** ở result $f(A)$.

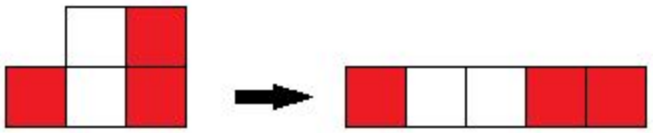
Dễ thấy giá trị của $f(A)$ chỉ phụ thuộc vào lực lượng của tập hợp A, ta ký hiệu $g(k)=f(A)$ với tập hợp A tùy ý có K phần tử. Thuật toán chuẩn của bài toán là:

```
int result = 0;
for (int k = 0; k <= n; k++) {
      int tmp = C(n,k) × f(k)
      if (k % 2 == 0) result += tmp; else result -= tmp;
}
```

Lời kết: Trên đây là phần trình bày lời giải của mình. Lời giải này giành phần lớn dung lượng để giải thích cách sử dụng nguyên lý bao hàm loại trừ. Với những bạn đã làm quen với nguyên lý này từ lâu, các bạn chỉ cần đọc lời giải đến phần "Đếm số dãy nhị phân R bit 1..." là đủ. Tuy nhiên, có nhiều bạn ở Việt Nam dù học chuyên tin nhưng lại ít học toán và mới nghe qua khái niệm này lần đầu, sẽ cảm thấy xa lạ với cách giải này. Cũng xin nói thêm rằng, cách suy luận mình đã trình bày ở trên mang đậm tính suy luận trực giác và có phần thiếu chặt chẽ, với mục đích giúp các bạn ít tiếp xúc với toán dễ cảm thụ hơn. Dù mình đã chứng minh chặt chẽ được bằng toán học mọi kết quả của bài toán này, mình xin phép không trình bày tại đây. Hơn nữa, ngày đầu tiên học nguyên lý bao hàm và loại trừ, mình đã sử dụng cách suy luận ở trên để khiến những dòng suy luận thêm phần gần gũi.

Lời giải 2: Nguyễn Đinh Quang Minh (team VNU - unsigned)

Trước hết, vì hai chồng rượu liên tiếp phải khác màu nhau nên ta có thể “trải” các hộp rượu thành hàng ngang. Nói cách khác, mỗi cấu hình thỏa mãn tương đương với một cách xếp tất cả R + W hộp rượu thành hàng ngang, sao cho không có d hộp rượu vang đỏ liên tiếp.



Vì không có ràng buộc nào liên quan tới rượu trắng, ta sẽ xây dựng một cấu hình thỏa mãn như sau:

+ Đặt W hộp rượu trắng vào trước, tạo thành W+1 “khoảng” để đặt rượu đỏ vào.
+ Đặt R hộp rượu đỏ vào các khoảng, sao cho không có khoảng nào chứa nhiều hơn d hộp.

Nói cách khác, nếu coi $(a_1, a_2, \ldots, a_{W+1})$ là số hộp rượu đỏ nằm trong từng khoảng thì số cấu hình thỏa mãn là số bộ $(a_1, a_2, \ldots, a_{W+1})$ thỏa mãn $\sum_{i=1}^{W+1} a_i = R$ và $0 \leq a_i \leq d, \forall i = 1 \ldots W+1$.

Xét đa thức $f(x) = (1 + x + x^2 + \cdots + x^d)^{W+1} = \sum_{i=0}^{d(W+1)} c_i x^i$, ta thấy hệ số c_R chính là số cấu hình thỏa mãn.

Biến đổi đa thức f:

$$f(x) = (\frac{1 - x^{d+1}}{1 - x})^{W+1}$$

Theo khai triển Newton, ta có:

$$(1 - x^{d+1})^{W+1} = \sum_{i=0}^{W+1} (-1)^i \binom{W+1}{i} x^{(d+1)i}$$

Áp dụng công thức Taylor:

$$\frac{1}{(1-x)^{W+1}} = \sum_{i=0}^{\infty} \binom{W+i}{i} x^i$$

Nhân vào được:

$$f(x) = \sum_{i=0}^{W+1} \sum_{j=0}^{\infty} (-1)^i \binom{W+1}{i} \binom{W+j}{j} x^{di+i+j}$$

Thay $k = di + i + j \implies j = k - di - i$ ta được:

$$f(x) = \sum_{k=0}^{\infty} \sum_{i=0}^{W+1} (-1)^i \binom{W+1}{i} \binom{W+k-di-i}{W} x^k$$

Vậy:

$$c_R = \sum_{i=0}^{W+1} (-1)^i \binom{W+1}{i} \binom{W+R-di-i}{W}$$

Ta có thể tính tổng trên trong độ phức tạp O(R+W).

B - Barcode (2*)

Lời giải: Nguyễn Diệp Xuân Quang (team HCMUS - Illuminate)

Thể loại: Tổ hợp, Tính nghịch đảo modulo, Sàng số nguyên tố

Ta xem một barcode tương đương một xâu có độ dài N gồm các kí tự 'B' (màu xanh) hoặc 'R' (màu đỏ).

Gọi S_1 là số chuỗi thỏa mãn yêu cầu thứ nhất (số kí tự 'B' bằng số kí tự 'R'), S_2 là số chuỗi thỏa mãn yêu cầu thứ hai (không có hai kí tự 'B' liên tiếp), S_{12} là số chuỗi thỏa mãn cả hai yêu cầu trên. Đáp số bài toán là: $S_1 + S_2 - S_{12}$

Tính S_1:
Nếu N lẻ, hiển nhiên $S_1 = 0$. Ngược lại, S_1 là số cách chọn N/2 vị trí để gán cho kí tự 'B' trong số N vị trí của xâu. Nói cách khác, $S_1 = C(N, N/2)$.

Tính S_2:
Gọi dp[i][0] là số lượng xâu độ dài i kết thúc bởi kí tự 'B' thỏa mãn yêu cầu thứ hai, dp[i][1] là số lượng xâu độ dài i kết thúc bởi kí tự 'R' thỏa mãn yêu cầu thứ hai, F[i] = dp[i][0]+dp[i][1].
Ta có:

- dp[i][0] = dp[i-1][1]
- dp[i][1] = dp[i-1][0]+dp[i-1][1]

⇒ F[i] = dp[i][0] + dp[i][1]
= dp[i-1][1] + dp[i-1][0] + dp[i-1][1]
= dp[i-2][0] + dp[i-2][1] + dp[i-1][0] + dp[i-1][1]
= F[i-2] + F[i-1]
Vậy F[i] chính là số Fibonacci thứ i ⇒ S_2 chính là số Fibonacci thứ N.

Tính S_{12}:
Nếu N lẻ, hiển nhiên $S_{12} = 0$. Ngược lại, ta nhận xét là sẽ có không quá 1 vị trí có hai kí tự 'R' liên tiếp nhau. Ta có các trường hợp sau

- "RBRB...RB"
- "BRBR...BR"
- "RB...RBRRBR..BR" (có n/2 - 1 xâu)

Như vậy, $S_{12} = n/2 + 1$

Vấn đề còn lại là tính C(N, N/2) modulo M.

Ta phân tích C(N, N/2) thành tích các thừa số nguyên tố.

C(N, N/2) = tích(pi ^ ki), với pi là số nguyên tố trong khoảng [1, N].

Xét lần lượt các số nguyên tố trong khoảng [1, N] (có thể dùng sàng để liệt kê các số nguyên tố), để tính ki, ta làm như sau:

- Đặt f(N, p) = x, với x là luỹ thừa lớn nhất của p mà N! chia hết cho p^x.

- Vì C(N, N/2) = N! / (N/2)! / (N/2)!, nên ki = f(N, p) - f(N/2, p) - f(N/2, p)
- Để tính f(N, p) ta lần lượt đếm số lần xuất hiện của p trong khoảng [1, N], rồi đếm số lần xuất hiện của p^2 trong khoảng [1, N].... Ta có thể code đệ quy rất đơn giản như sau:

```
int f(int N, int p) {
   if (N < p) return 0;
   return N / p + f(N/p, p);
}
```

Vì số lượng số nguyên tố trong khoảng [1, N] là O(N/logN) và độ phức tạp mỗi lần gọi f() là O(logN), nên ta có độ phức tạp với mỗi test là O(N).

Lưu ý

Có rất nhiều team làm sai bài này, do tầng tầng lớp lớp các bẫy mà người ra đề đặt ra:

- Nếu "mắt nhanh hơn não" mà nhìn vào test ví dụ, ta rất dễ lầm tưởng đáp án là số Fibonacci thứ n. Điều này chỉ đúng với n lẻ (và vô tình đúng với n = 2).
- Trong quá trình tính C(n, n/2) = thì cần chú ý là ta không thể áp dụng công thức $a^{-1} \equiv a^{M-2} \pmod M$ (hệ quả của Định lí Euler) được, do với giới hạn M đề bài cho thì (n/2)! Và M có thể không nguyên tố cùng nhau. Người ra đề còn cố tình cho dữ kiện M là số nguyên tố nhằm gài bẫy các thí sinh không hiểu rõ bản chất của định lí Euler.
    - Những bạn nào bị sai lỗi này hãy đọc lại bài viết trên VNOI 10 lần: Tính nghịch đảo modulo.

C - Cu Chi Tunnel (1*)

Tác giả: Lê Đôn Khuê

Lời giải: Nguyễn Diệp Xuân Quang (team HCMUS - Illuminate)

Thể loại: Đồ thi, Tham lam

Ta sẽ tiến hành gắn các đỉnh vào cây theo thứ tự từ 2 đến N. Khi đó, với đỉnh thứ i, ta có thể gắn nó vào bất kì đỉnh nào bé hơn i mà chưa đủ bậc. Nếu tất cả các đỉnh đều đã đủ bậc thì câu trả lời là “NO”. Cuối cùng, nếu đã gắn xong đỉnh N mà vẫn còn đỉnh không đủ bậc thì cũng in ra “NO”.

Ví dụ:

```
8
3 2 2 1 1 3 1 1
```

Đỉnh 2: gắn với đỉnh 1. Mảng bậc hiện tại là: {1, 1, 0, 0, 0, 0, 0, 0}
Đỉnh 3: gắn với đỉnh 1. Mảng bậc hiện tại là: {2, 1, 1, 0, 0, 0, 0, 0}
Đỉnh 4: gắn với đỉnh 1. Mảng bậc hiện tại là: {3, 1, 1, 1, 0, 0, 0, 0}
Đỉnh 5: gắn với đỉnh 2. Mảng bậc hiện tại là: {3, 2, 1, 1, 1, 0, 0, 0}
Đỉnh 6: gắn với đỉnh 3. Mảng bậc hiện tại là: {3, 2, 2, 1, 1, 1, 0, 0}
Đỉnh 7: gắn với đỉnh 6. Mảng bậc hiện tại là: {3, 2, 2, 1, 1, 2, 1, 0}
Đỉnh 8: gắn với đỉnh 6. Mảng bậc hiện tại là: {3, 2, 2, 1, 1, 3, 1, 1}

Ta có thuật toán độ phức tạp O(N^2), đủ để AC bài này. Nhưng ta có thể cải tiến thêm bằng cách nhận xét rằng ta chỉ cần quan tâm đến tổng bậc còn thiếu của các đỉnh đã được gắn vào cây mà không cần quan tâm đến bậc còn thiếu của từng đỉnh. Do đó, trong quá trình duyệt, ta chỉ cần lưu một biến cnt là tổng bậc còn thiếu hiện tại. Độ phức tạp: O(N).

D - Dropping Ball (4*)

Tác giả: Nguyễn Thành Trung (RR)

Lời giải: Lê Quang Tuấn (team VNU - unsigned)

Tóm tắt đề bài: Cho một bảng N x M với mỗi ô trong bảng có một trong 2 loại thanh '/' hoặc '\'.
Ta có Q truy vấn:

- Đổi tường của một ô (i,j).
- Hỏi xem nếu thả một quả bóng vào cột i, thì quả bóng có thể xuống hàng cuối cùng rồi ra khỏi bảng không?

Lời giải trâu :

Để giải được toàn bộ bài này, trước tiên ta phải nghĩ ra một lời giải trâu nào đó rồi tìm cách cải tiến nếu có thể, tuỳ vào lời giải trâu như nào mà cải tiến khác nhau.
Có thể coi một ô (i,j) bằng 2 nửa : nửa trên và nửa dưới. Coi mỗi một nửa là một đỉnh trên đồ thị. Có thể dễ dàng nhận thấy mỗi một đỉnh này chỉ có một đỉnh ra duy nhất tuỳ vào loại thanh của các ô xung quanh. Nếu là nửa dưới thì nó chỉ có cạnh đi xuống dưới, còn nếu là nửa trên, thì nó chỉ có cạnh đi sang một trong 2 ô bên cạnh cùng hàng nó.
Vậy với lời giải này, chỉ cần bfs lại toàn bảng mỗi khi có truy vấn hỏi quả bóng đi đâu.

Lời giải tối ưu

Lấy ý tưởng bài **http://codeforces.com/problemset/problem/13/E**, cùng với việc nhận ra N x M <= 100 000. Mình đã có ý tưởng chia bảng thành những block các hàng liên tiếp nhau , mỗi block chứa xấp xỉ sqrt(N x M) ô. Với mỗi block, ta lưu lại được nếu thả quả bóng vào cột j ở đầu block, thì đến cuối block, quả bóng sẽ ở cột nào, sử dụng lời giải trâu ở trên.
Do vậy:

- Truy vấn loại update, ta thực hiện lại lời giải trâu với tất cả các ô thuộc block chứa ô (i,j) cần update, thực hiện lại việc bfs trên bảng. Độ phức tạp thực hiện công việc này là O(số ô của block) = O(sqrt(N x M)). Sau khi update, với mỗi block, ta có mảng nxt[j] = ô mà quả bóng sẽ rơi xuống ở cuối block, nếu quả bóng được thả ở cột j của block.
- Truy vấn loại yêu cầu trả lời: Đi từ block 1, nhảy đến hàng đầu tiên của block 2 sử dụng thông tin trong block 1, tiếp tục… cho đến hết bảng. Độ phức tạp (N x M/ (độ lớn mỗi block)) = O(sqrt(N x M)).

Thế nhưng với trường hợp số hàng rất ít, số cột rất to, ta không thể chia làm các block sao cho mỗi block có ít hơn sqrt(N x M) phần tử được. Do vậy với trường hợp N <= sqrt(N x M), ta phải giải trâu, tức là đi từ trên xuống dưới, mỗi truy vấn mất đpt O(N).

Thuật toán này bọn mình cảm thấy rất dễ code. Với sự tinh tế của Nguyên, team Unsigned đã hoàn thành việc code trong chỉ 15 phút, nộp phát AC luôn trong sự thăng hoa của team.

E - Engaging with Loyal Customers (2*)

Thể loại: Đồ thị, Cặp ghép

Lời giải:
Đây là bài toán cặp ghép có trọng số cực đại.

Để làm được bài này các bạn cần nắm vững Cặp ghép có trọng số trong sách thầy Lê Minh Hoàng (link).

Cài đặt thuật toán Hungary để giải Cặp ghép có trọng số: code

Một thuật toán khác cũng có thể dùng để giải cặp ghép có trọng số là luồng mincost (code), tuy nhiên không thể chạy đủ nhanh vì đồ thị bài này có 10^6 cạnh.

F - Famous Pagoda (3*)

Lời giải: Nguyễn Thành Trung - RR

Thể loại: Tối ưu hoá Quy hoạch động - Chia để trị.

Đặt cost(i, j) là chi phí để xây cầu thang từ điểm i đến j.
Đặt f(k, i) = chi phí nhỏ nhất để k nhóm thợ xây tất cả cầu thang từ 1 đến i. Ta có công thức QHĐ:

- f(k, i) = min(f(k-1, j) + cost(j+1, i), với j = 1..i-1).

Kết quả của bài toán chính là f(G, N).

Có 2 điểm mấu chốt của bài này:
- Tính cost(i, j)
- Tính nhanh bảng f(k, i). Nếu tính trực tiếp theo công thức trên, ta mất độ phức tạp O(G*N^2), không đủ để AC bài này.

1. Tính cost(i, j)

Với k = 1:
- Khi k = 1, điểm v chính là median của dãy A(i)..A(j).
- Do dãy A đã được sắp xếp tăng dần, v = A[i + (j - i + 1) / 2]
- Với mỗi (i, j), ta có thể tính nhanh cost(i, j) trong O(1) bằng mảng cộng dồn.

Với k = 2:
- Đặt L = số phần tử của đoạn A(i)..A(j) = j - i + 1
- cost(i, j) = sum((a(s) - v)^2)
    - = sum(a(s)^2) + L*v^2 + 2*sum(a(s))*v
- Đặt B = 2*sum(a(s)), C = sum(a(s)^2). Ta tính được B và C trong O(1) bằng mảng cộng dồn.
- cost(i, j) = L*v^2 + B*v + C, là một hàm bậc 2 của v. Do đó điểm v để làm cost(i, j) nhỏ nhất chính là v = -B / L. Tuy nhiên (-B / L) có thể là số thực, còn trong bài toán này v phải là số nguyên, nên ta cần xét 2 số nguyên v gần nhất với (-B / L) bởi vì đồ thị hàm số lồi

2. Tính f(k, i)

Để tính f(k, i), ta cần dùng kĩ thuật QHĐ chia để trị.

Bạn có thể đọc lý thuyết ở VNOI wiki và code ở đây. Trong lời giải này mình chỉ tập trung vào việc chứng minh tính đúng đắn của thuật toán.

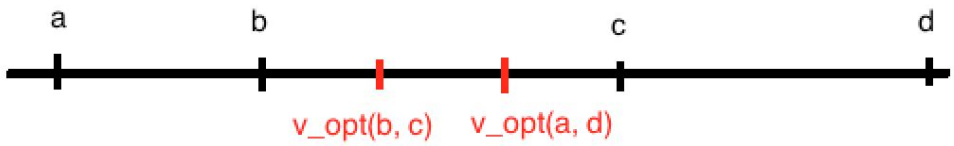
Nhắc lại, ta có thể sử dụng được QHĐ chia để trị nếu:

- cost(a, d) + cost(b, c) >= cost(a, c) + cost(b, d) với mọi a < b < c < d

Đặt:

- f(i, j, v) = (sum (|a(s) - v|^k) với s = i..j)
- v_opt(a, b) = v để f(a, b, v) đạt giá trị nhỏ nhất
- cost(a, b) = f(a, b, v_opt(a, b)) <= f(a, b, v) với mọi v.

Không làm mất tính tổng quát, giả sử v_opt(b, c) <= v_opt(a, d). Ngoài ra ta cũng biết rằng v_opt(b, c) nằm trong khoảng [b, c].



Ta có:

```
    cost(a, d)
  = f(a, d, v_opt(a, d))
  = f(a, b-1, v_opt(a, d)) + f(b, d, v_opt(a, d))
>= f(a, b-1, v_opt(a, d)) + cost(b, d)
```

Mặt khác, do tất cả đoạn [a, b-1] ở bên trái v_opt(b, c) và v_opt(b, c) < v_opt(a, d) nên:

```
    f(a, b-1, v_opt(a, d)) + cost(b, c)
>= f(a, b-1, v_opt(b, c)) + cost(b, c)
  = f(a, b-1, v_opt(b, c)) + f(b, c, v_opt(b, c))
  = f(a, c, v_opt(b, c))
>= cost(a, c)
```

Kết hợp 2 bất đẳng thức trên:

```
    cost(a, d) + cost(b, c)
>= f(a, b-1, v_opt(a, d)) + cost(b, d) + cost(b, c)
  = f(a, b-1, v_opt(a, d)) + cost(b, c) + cost(b, d)
>= cost(a, c) + cost(b, d)
```

Chứng minh hoàn tất. Kết luận ta có thể dùng QHĐ chia để trị để giải bài toán với độ phức tạp O(G*N*log(G))

G - Game of Divisibility (3*)

Tác giả: thầy Hiếu (Hưng Yên) + Lăng Trung Hiếu

Lời giải: Đại Ca Đi Học

Giả sử người đi trước là A, người đi sau là B. Gọi s[i] là số lượng số chia k dư i (0≤i<k). X là số lượng giá trị i có s_i lẻ.

X=0: Trận đấu hòa. Cả A và B đều có chiến lược để mình không thể thua. Do đó trận đấu hòa.

X = 1: Nếu người lấy số dư lẻ ra (t) thắng thì A luôn là người lấy được. Nếu người lấy phần dư lẻ ra thua thì B luôn bắt A phải lấy phần đó. Do đó A luôn là người lấy phần số dư lẻ ra. Số dư A nhận được là (t+(sum-t)/2), số dư B nhận được là (sum-t)/2.

X=2: số dư lẻ 2 là p,q.A được quyền quyết định lấy số dư p hoặc q lẻ. Sau đó trở về trạng thái X=1 như trên. B sẽ bắt buộc lấy số dư còn lại. Trong trường hợp này, A chọn 1 trong 2, A sẽ chọn số dư có khả năng thắng. Ở trường hợp này nếu cả 2 đều không có khả năng thắng, A sẽ chọn số dư để hòa. (Chắc chắn không thua).

X=3: Sau khi chọn 1 số dư lẻ, sẽ về trạng thái X=2. Người nhận trạng thái này, không bao giờ thua. Nên người ở trạng thái X=3 cũng có quyền tự quyết chọn để người ở trạng thái X=2 không thể thắng. Do đó X=3 là trạng thái 2 người không thể thua. HÒA.

Tương tự các trường hợp X>2 tất cả sẽ đều hòa.

Ghi chú:

Đây là bài toán 3* cực khó chứ không dễ như sau khi nhìn lướt qua, lời giải trên chỉ dành cho những bạn đã cực kỳ xuất sắc về dạng này, muốn hiểu rõ thêm bạn cần đọc đoạn giải thích, lưu ý là có thể đọc xong giải thích càng khó hiểu thêm.

Bài B với bài J có chứa nhiều trap nhưng bản thân bài G lại là một cái trap cực lớn, các dữ kiện như n <= 1000, k <= 32 chỉ mang tính chất hoa mỹ không liên quan gì đến lời giải đặc biệt là a[i] <= 2^31 làm tỉ lệ WA tăng lên đáng kể. Chính vì thế để giải được bài này các bạn cần phải tư duy thuần tuý mạnh mẽ chứ không được bám theo lối mòn suy nghĩ như kiểu có thể nhìn được công thức hay dùng giới hạn để đoán thuật toán.

Giải thích:

X = 0, cả A và B đều có thể có chiến lược để bằng điểm đối phương. Đối với B chỉ cần chọn số có số dư giống A vừa chọn. Đối với A có thể start bằng số bất kỳ xong chơi giống hệt B, nếu không tồn tại số có số dư bằng số B vừa chọn thì lại chọn số bất kỳ đến hết. Ta thấy sau mỗi lượt A chơi thì đều chỉ có tối đa 1 tập số dư có lẻ số.

Với X = 1, nếu số dư lẻ ra có lợi cho A, A sẽ chọn số đấy rồi chơi chiến thuật chọn giống B. Còn ngược lại B sẽ chọn chiến lược chọn giống A ép A phải chọn hơn B 1 lần chính là số dư lẻ ra đó. Các bạn có thể dùng suy nghĩ như vậy để áp dụng với X >= 2 và những bài toán tương tự liên quan đến trò chơi giữa hai người và số dư.

H - Height Preservation (2*)

Lời giải: Nguyễn Diệp Xuân Quang (team HCMUS - Illuminate)

Thể loại: Đồ thị, Topo sort.

Trước hết, ta sẽ gộp các ô có giá trị giống nhau ở cùng một dòng hoặc cùng một cột vào một thành phần liên thông (ta gọi là siêu đỉnh). Việc tìm các cặp ô để gộp có thể thực hiện bằng cách duyệt từng dòng, sắp xếp các ô trên từng dòng theo thứ tự tăng dần, và thêm cạnh giữa hai ô trên cột j và cột j+1 của dòng hiện tại nếu số trên hai ô này bằng nhau (Với từng cột cũng làm tương tự). Việc gộp ô có thể làm bằng DSU hoặc DFS.

Sau đó, ta duyệt từng dòng, sắp xếp các ô trên dòng theo thứ tự tăng dần, rồi thêm cạnh một chiều nối từ siêu đỉnh chứa ô trên cột j đến siêu đỉnh chứa ô trên cột j+1 của dòng hiện tại nếu số trên cột j nhỏ hơn số trên côt j+1. Ta làm tương tự như vậy với từng cột.

Cuối cùng, ta sẽ được một DAG trên các siêu đỉnh. Đáp số cần tìm chính là số đỉnh trên đường đi dài nhất của DAG này. Bài toán tìm độ dài đường đi dài nhất trên DAG là một bài toán quy hoạch động kinh điển, có thể giải được trong O(N+M) (với N là số đỉnh trong DAG, M là số cạnh trong DAG).

Độ phức tạp: O(N*M*log(N+M)).

I - ICPC Awards (0*)

Tác giả: Lê Đôn Khuê

Bài này được ra với mục đích tất cả các đội tham gia cuộc thi đều giải được ít nhất 1 bài. Đáng tiếc là vẫn còn một đội không giải được.

Bài toán này chỉ đòi hỏi biết sử dụng C++ STL.

J - Joining Networks (3*)

Tác giả: Lăng Trung Hiếu

Lời giải: Phạm Cao Nguyên (team VNU - unsigned)

Thể loại: Quy hoạch động bao lồi.

Tóm tắt đề bài

Cho 2 cây <= 50000 đỉnh, tìm cách nối 1 cạnh giữa 2 cây để tổng bình phương khoảng cách giữa mọi cặp đỉnh là nhỏ nhất.

Lời giải

Đây là một bài thuộc loại cổ điển, viết kĩ công thức rồi tính từng phần một.

Kiến thức cần biết: Quy hoạch động trên cây, Bao lồi đường thẳng, Kiên trì phá ngoặc

Phân tích công thức

Ta định nghĩa f(T, a, b) là khoảng cách giữa 2 nút a và b trên cây T.
Xét 2 cây A (N nút) và B (M nút), khi nối đỉnh u của cây A với đỉnh v của cây B, ta có khoảng cách giữa mọi cặp đỉnh như sau:

$$C = \sum_{i=1}^{N}\sum_{j=1}^{i-1} f(A, i, j)^2 + \sum_{i=1}^{M}\sum_{j=1}^{i-1} f(B, i, j)^2 + \sum_{i=1}^{N}\sum_{j=1}^{M} (f(A, u, i) + f(B, v, j) + 1)^2$$

2 tổng đầu tiên khá dễ hiểu: chúng chỉ là tổng bình phương giữa mọi cặp đỉnh trong cây. Nó không quá khó tính (qhđ trên cây cơ bản thôi), và không đổi với mọi cách chọn u, v nên ta tạm thời bỏ ra khỏi phương trình. Phần còn lại, ta phân tích sẽ ra

$$\sum_{i=1}^{N}\sum_{j=1}^{M} (f(A, u, i) + f(B, v, j) + 1)^2$$

$$= \sum_{i=1}^{N} f(A, u, i)^2 * M + \sum_{j=1}^{M} f(B, v, j)^2 * N + N * M + 2(\sum_{i=1}^{N} f(A, u, i) * M + \sum_{j=1}^{M} f(B, v, j) * N + \sum_{i=1}^{N} f(A, u, i) * \sum_{j=1}^{M} f(B, v, j))$$

Gọi S(T, u) là tổng khoảng cách từ đỉnh u đến tất cả các nút còn lại của cây T, Sq(T, u) là tổng bình phương khoảng cách từ đỉnh u đến tất cả các nút còn lại của cây T. Ta có thể viết lại công thức thành:

$$M * Sq(A, u) + N * Sq(B, v) + 2(M * S(A, u) + N * S(B, v) + S(A, u) * S(B, v))$$

Gom lại các phần độc lập theo u, v cho dễ nhìn, ta có:

$$M * Sq(A, u) + 2M * S(A, u) + N * Sq(B, v) + 2N * S(B, v) + 2S(A, u) * S(B, v)$$

Như vậy, chỉ có phần 2S(A, u)*S(B, v) là phụ thuộc vào cả 2 đại lượng u và v. Tuy nhiên, không khó để nhìn ra đây là đại lượng tuyến tính, nên khi ta coi S(A, u) là hằng số, S(B, v) là biến, công thức trở thành công thức đường thẳng. Đối với các bạn đã quen thuộc với việc sử dụng thủ thuật bao lồi trong quy hoạch động, đây là một dạng tương tự: Ta sẽ dựng bao lồi các đường thẳng của u, sau đó truy vấn v trên bao lồi đó.
Tóm lại, các phần cần tính bao gồm:

- S(A, u) và S(B, v) cho mọi u, v
- Sq(A, u) và Sq(B, v) cho mọi u, v

Tính S(A, u)

Trước tiên, vì 2 cây là độc lập, nên trong phần này ta sẽ gọi S(A, u) là S(u) cho ngắn gọn.

Đây là bài toán qhđ cơ bản, đã xuất hiện trong 1 số bài tập trên VNOI (http://vnoi.info/problems/show/NTTREE/). Cách làm như sau:

- Đầu tiên, tính size[v] là số đỉnh trong cây con gốc v.
- Thực hiện qhđ trên cây để tính down[u]: tổng khoảng cách giữa u và các đỉnh trong cây con gốc u. Công thức là $down[u] = \sum_{v \varepsilon child[u]} child[v] + size[v]$. Nôm na là, vì tất cả khoảng cách từ u đến các các đỉnh thuộc cây con gốc v đều hơn 1 so với khoảng cách từ v, nên tổng của chúng sẽ cộng thêm một lượng là số đỉnh trong cây con gốc v. Ta có thể tính các giá trị size[v] bằng 1 lần dfs.
- Giờ ta sẽ tính up[u]: tổng khoảng cách từ u đến các đỉnh không nằm trong cây con gốc u. Không khó để nhìn thấy up[u] gồm 2 phần:
    - Các đỉnh không thuộc cây con gốc parent(u). Phần này được tính bằng up[parent(u)] + (N -size[parent(u)]).
    - Các đỉnh thuộc cây con gốc parent(u) nhưng không thuộc cây con gốc u. Phần này được tính bằng down[parent(u)] + size[parent(u)] - (down[u] - size[u]).
    - Tổng lại, ta có up[u] = up[parent(u)] + down[parent(u)] - down[u] + (N - size[u]).
    - Phần này cũng có thể được tính bằng một phép dfs. Mặc định, nếu 1 là gốc, ta có up[1] = 0.
- Hiển nhiên, S[u] = down[u] + up[u], theo định nghĩa. Ta tính thành công S[u] trong O(n).

Tính Sq(A, u)

Tương tự phần trên, ta gọi Sq(A, u) là Sq(u) cho đơn giản.
Việc tính Sq(A, u) yêu cầu có các giá trị size[u], down[u], up[u] đã tính ở trên.
Ta thực hiện như sau:

- Đầu tiên, cũng như bài trên, ta tính downsq[u] là tổng **bình phương** khoảng cách từ u đến các đỉnh thuộc cây con gốc u. Vậy với mỗi v là con của u, downsq[u] tăng thêm bao nhiêu? Ta có downsq[v] = tổng các d(v, i)^2 với i thuộc cây con gốc v. Vậy downsq[u] += tổng (d(v, i) + 1)^2, hay downsq[u] += tổng d(v, i)^2 + 2 * tổng d(v, i) + size[v] = downsq[v] + 2 * down[v] + size[v]. Việc tính downsq[u], vẫn có thể tính bằng 1 lần dfs.
- Tiếp tục tính upsq[u]. Với cách phân tích tương tự, ta sẽ có:
    - Phần không thuộc cây con gốc parent(u): upsq[parent(u)] + 2*up[parent(u)] + (N - size[parent(u)]).

    - Phần thuộc cây con gốc parent(u):
      downsq[parent(u)] - (downsq[u] + 2 * down[u] + size[u]) + 2 * (down[parent(u)] - (down[u] + size[u])) + (size[parent(u)] - size[u]).
    - upsq[u] sẽ là tổng của 2 giá trị trên.
- Sq[u] = downsq[u] + upsq[u].

Như vậy việc tính Sq(A, u) cũng có thể thực hiện trong O(N).

Dựng bao lồi

Còn lại chỉ là việc dựng bao lồi và tính giá trị nhỏ nhất.
Theo như công thức trên, đỉnh u sẽ được biểu diễn dưới dạng đường thẳng có a = 2S(A, u), b = $M * Sq(A, u) + 2M * S(A, u)$.
Khi truy vấn đỉnh v, ta truy vấn tọa độ x = S(B, v), và cộng thêm $N * Sq(B, v) + 2N * S(B, v)$ để ra giá trị tối ưu khi chọn đỉnh v.
Đáp số sẽ là min của các giá trị đối với các lựa chọn v.
Để ý ta hoàn toàn có thể thêm tất cả đường thẳng vào trước khi truy vấn, nên ta nên sort lại các đỉnh u theo slope (a = 2S(A, u)) giảm dần để dễ dàng dựng bao lồi.
Ta cũng không cần thực hiện truy vấn theo đúng thứ tự: sort lại các truy vấn theo tọa độ tăng dần để không cần chặt nhị phân khi truy vấn trên bao lồi, giảm thời gian cài đặt.
Việc dựng bao lồi có thể tham khảo trên http://vnoi.info/wiki/translate/wcipeg/Convex-Hull-Trick

Độ phức tạp của cả bài toán là O(N log N).

Lưu ý!

- Khi so sánh 2 đường thẳng trên bao lồi, không thể dùng phép nhân chéo vì sẽ bị tràn số. Sử dụng chia bằng long double đủ qua.
- Đáp số lớn hơn 1e18, hãy sử dụng hằng số vô tận lớn hơn.
- Có khá nhiều đội để N, M là kiểu int nên bị tràn số khi tính tích N * M.

K - K-rotating (3*)

Lời giải: Lê Quang Tuấn (team VNU - unsigned)

Tóm tắt đề bài:

Cho N cô giáo và N lớp, mỗi cô giáo dạy ở một lớp. Ban đầu cô giáo thứ i dạy cho lớp thứ j. Và mỗi tuần có thể có một số sự thay đổi ở vị trí của các cô giáo (lưu ý là số sự thay đổi mỗi tuần <= 10).

Có 2 loại truy vấn :

- Thêm một sự thay đổi về lớp của các cô giáo.
- Hỏi xem ở tuần thứ x, sau khi đã thực hiện xong việc thay đổi ở tuần thứ x, thì cô giáo thứ y dạy ở lớp nào ???

Lời giải

Với bài này, mình nghĩ đến ngay phải chia căn, lại phải nhảy nhảy như bài D. Thế nhưng chia căn như nào, tính như nào thì mới chia căn được mới là vấn đề.

Để nghĩ ra một lời giải trâu thì không khó, thế nhưng nghĩ ra một lời giải trâu để chia căn được thì lại rất khó. Ở bài này, lời giải trâu như sau:

- Tính hàm f[x][y] : được trả về ở ngày x, cô giáo thứ y đang dạy lớp của cô giáo nào trong tuần x - 1. Tức là f[x][y] trả về k, có nghĩa là lớp của cô giáo k ở tuần x - 1sẽ được chuyển thành cô lớp của cô giáo y ở tuần x.
- Tính hàm f[x][y] như nào ? Ban đầu f[x][y] = y. Nếu ở tuần x, ta phải swap lớp của 2 cô p và cô q. thì ta cũng swap luôn f[x][p] và f[x][q].
- Khi có hàm f rồi, để truy vấn ta chỉ cần đi từ tuần x về tuần 1,y = f[x][y]. Thì đến cuối cùng y sẽ là lớp cần tìm !

Theo mình thấy, thì những lời giải trâu đi từ tuần 1 đến tuần x đều không khả thi cho việc chia căn, chỉ có cách đi từ tuần x về tuần 1 là khả thi !

Giờ phải chia căn như nào ? Tất nhiên là chia làm sqrt(M) block, mỗi block gồm xấp xỉ sqrt(M) tuần. Với mỗi block, ta lưu lại f[y] là ở tuần cuối của block, cô giáo y sẽ dạy lớp của cô giáo nào ở đầu block !

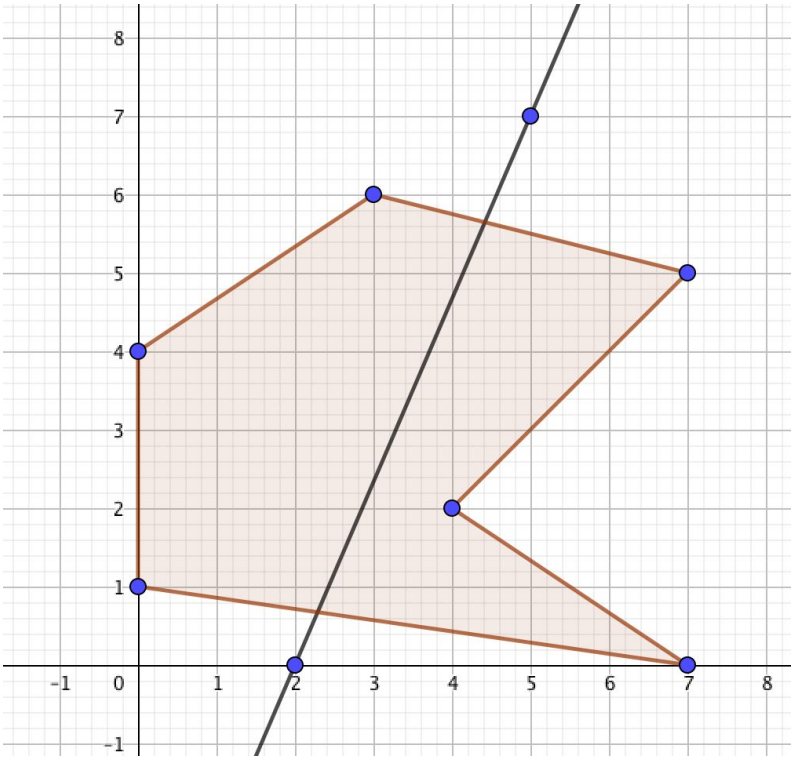
- Update : Với mỗi truy vấn update, tính lại toàn bộ hàm f[] của block. Độ phức tạp O(sqrt(M) * 10). Do tối ta chỉ có 10 phép swap mỗi tuần.
- Truy vấn : Đi trâu từ tuần x về cuối của một block, rồi sau đó dùng hàm f[] của mỗi block để nhảy về block trước như thuật trâu chỉ ở trên. Đpt (sqrt(M) * 10) .

Bộ nhớ O(sqrt(M) * N) vì mỗi block ta phải lưu lại mảng f[] gồm N phần tử.
Để tối ưu các bạn nên chia lớn hơn sqrt(M) block và mỗi block có độ dài nhỏ hơn sqrt(M).

L - Land Inheritance (4*)

Tác giả: Lăng Trung Hiếu

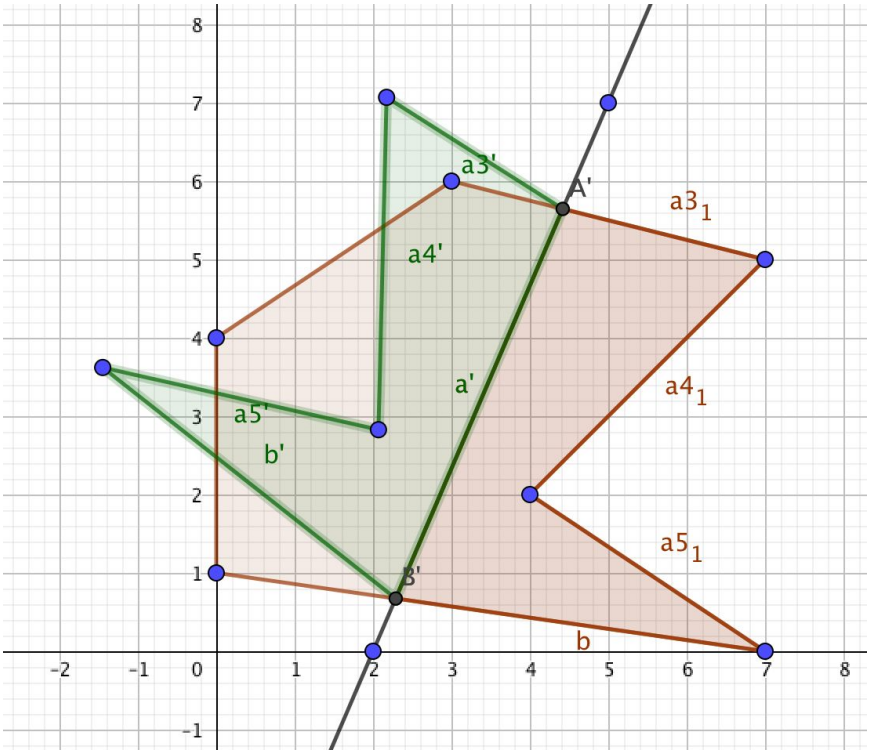
Lời giải: Nguyễn Thành Trung - RR



Đặt P là mảnh đất.

Bước 1:
Dùng đường thẳng **d** cắt P thành 2 phần A và B.
Bạn có thể tham khảo cài đặt ở Notebook của RR.

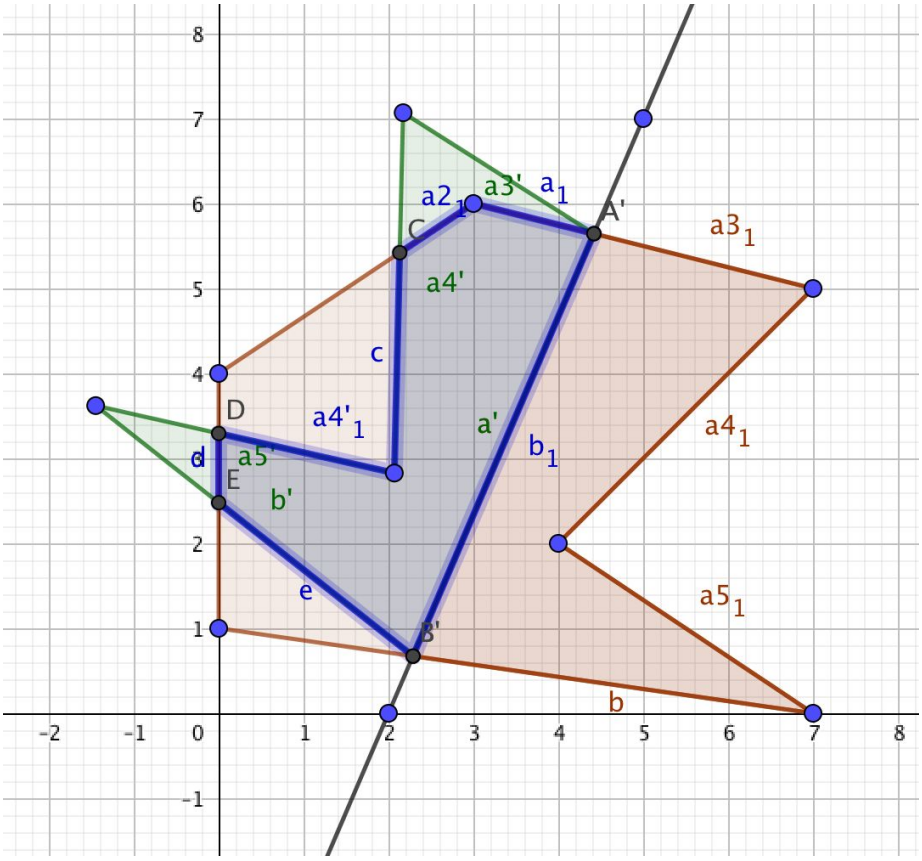
Bước 2:
Đặt A' là đối xứng của A qua đường thẳng d (màu xanh lá).



Bạn có thể tham khảo cài đặt ở [Notebook của RR](#).

Bước 3:

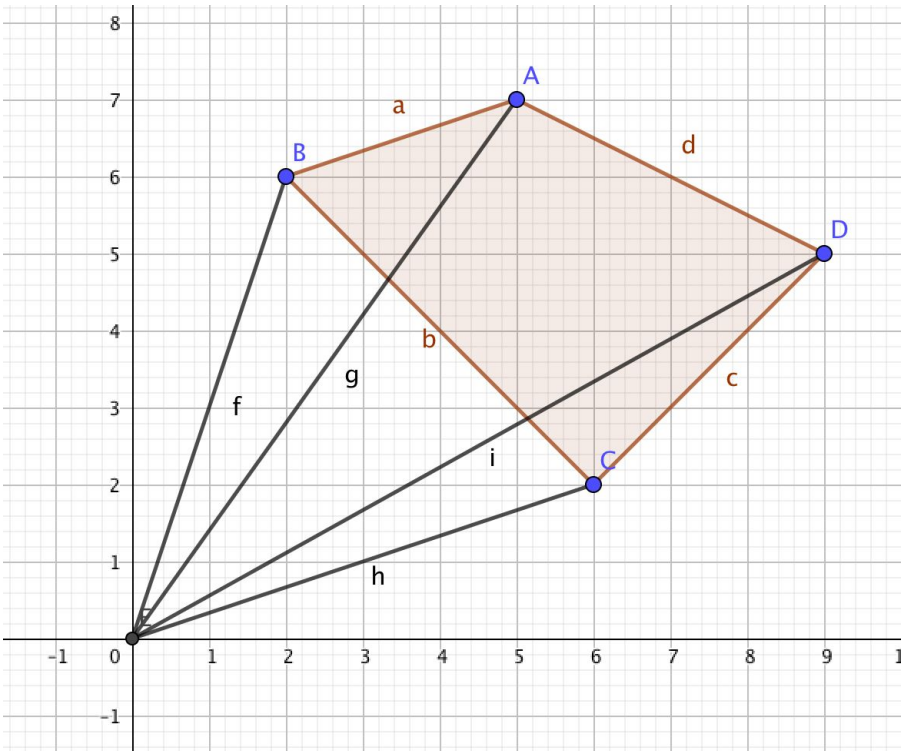
Dễ thấy kết quả bài toán chính là diện tích của phần giao của A' và B (màu xanh da trời).



Do A' và B không phải đa giác lồi, nên việc tìm phần giao của A' và B rất khó, tuy nhiên nếu chỉ muốn tính diện tích thì đơn giản hơn rất nhiều.

Trước hết, ta hãy xem cách tính diện tích **có dấu** của một đa giác bất kỳ:

- Đặt O là một điểm bất kỳ trên mặt phẳng (thông thường ta lấy O là gốc toạ độ để cài đặt đơn giản).
- Diện tích có dấu của đa giác $A = A_1A_2...A_n$ = tổng diện tích có dấu của n tam giác $OA_iA_{(i+1)}$. Ta coi $A_{(n+1)} = A_1$.
- Đây chính là lý do ta có thể tính diện tích đa giác bất kỳ theo công thức 0.5 * tổng (A(i).x*A(i+1).y - A(i+1).x*A(i).y).z



Ví dụ:

- Diện tích có dấu tứ giác ABCD (màu đỏ)
- = (Diện tích OAB) + (Diện tích OBC) + (Diện tích OCD) + (Diện tích ODA)

Ở đây O -> A -> B, O -> C -> D và O -> D -> A có các đỉnh ngược chiều kim đồng hồ, nên 3 tam giác OAB, OCD và ODA có diện tích dương.

O -> B -> C có các đỉnh theo chiều kim đồng hồ, nên tam giác OBC có diện tích âm.

ABCD có các đỉnh ngược chiều kim đồng hồ nên có diện tích dương.

Quay trở lại bài toán, để tính diện tích phần giao của 2 đa giác A' và B, ta cũng làm tương tự:

- Lấy 1 điểm O bất kỳ.
- Xét các tam giác $OA_iA_{(i+1)}$ và các tam giác $OB_jB_{(j+1)}$.
- Thay vì tính phần giao của 2 đa giác, ta sẽ tính phần giao của các tam giác $OA_iA_{(i+1)}$ với các tam giác $OB_jB_{(j+1)}$. Sau đó cộng tổng các diện tích các phần giao lại.

    - Chú ý: Ta đang làm việc với cả các tam giác âm và dương. Khi tính tổng diện tính phần giao:
        - Nếu 2 tam giác **dương**, ta coi phần giao có diện tích **dương**.
        - Nếu 2 tam giác **âm**, ta coi phần giao có diện tích **dương**.
        - Nếu 1 tam giác **âm** và 1 tam giác **dương**, phần giao có diện tích âm.
- Để tìm giao của 2 tam giác ABC và XYZ, ta lần lượt dùng 3 cạnh của XYZ cắt tam giác ABC:
    - Lấy đường thẳng XY cắt ABC thành 2 phần, giữ lại phần cùng phía với Z, gọi là S1.
    - Lấy đường thẳng YZ cắt S1 thành 2 phần, giữ lại phần cùng phía với X, gọi là S2.
    - Lấy đường thẳng ZX cắt S2 thành 2 phần, giữ lại phần cùng phía với Y, gọi là S3.
    - S3 chính là phần cần tìm.
    - (Phần này ta sử dụng lại code lấy 1 đường thẳng cắt 1 đa giác: Notebook của RR).

Đến đây bài toán đã được giải quyết với độ phức tạp O(N^2).