THE ICPC 2018
VIETNAM NATIONAL CONTEST

Posts and Telecommunications Institute of Technology

NOVEMBER 4, 2018

Editorial

Tổng quan

Bài	First AC	#AC	Editorialist	Tác giả
A	ONE 67	14	Nguyễn Diệp Xuân Quang HCMUS - Intimidate	Lăng Trung Hiếu
B	CHY.KB 51	28	Phạm Văn Hạnh judge	Phạm Văn Hạnh
C	THREE 149	13	Vương Hoàng Long UET - Map	Lăng Trung Hiếu
D	map 16	174	Lê Minh Phúc NUS - Illumina	Đại học Khoa Học Tự Nhiên HCM
E	P_Not_Equal_NP 95	6	Phạm Văn Hạnh judge	Phạm Văn Hạnh
F	map 277	1	Nguyễn Ngọc Trung judge	Nguyễn Ngọc Trung
G	map 106	6	Phạm Văn Hạnh judge	Phạm Văn Hạnh
H	THREE 99	1	Vương Linh judge	Nguyễn Vương Linh
I	Trie 6	216	Vương Hoàng Long UET - Map	Nguyễn Thành Trung
J	**EPU_KDH** 36	179	Nguyễn Vũ Hoàng Vương	Đại học An Ninh
K	Bitset 55	13	Đinh Nguyên Khôi HCMUS - Intimidate	Nguyễn Ngọc Trung
L	Bitset 157	4	Nguyễn Vũ Hoàng Vương	Lăng Trung Hiếu

Bảng xếp hạng chung cuộc

RK	TEAM	SLV.	TIME	A	B	C	D	E	F	G	H	I	J	K	L
1	map	11	1902	2 117	3 138	1 228	1 16	3 213	3 277	3 106		1 23	1 88	5 240	1 196
2	amazingbamboo_with_coccoc	9	1620	3 192	1 159	1 180	3 200			3 218		2 9	1 145	1 89	1 288
3	ONE	8	1459	5 67		1 214	1 62			8 276		2 10	1 191	1 115	2 264
4	bitset	7	858	3 --	2 207	1 187	1 18			8 --		2 25	1 109	1 55	4 157
5	P_not_equal_NP	7	940	2 69	1 203	1 255	2 59	2 95				2 18	1 161	5 --	
6	HCMUS-Intimidate	7	1140	1 96	4 291	1 244	2 61	5 --				2 14	2 109	2 185	
7	HSG-Team02	7	1227	5 --	4 281		2 45	4 98		3 160		2 25	2 176	2 202	
8	THREE	6	585		1 --	1 149	1 64		3 --	3 --	1 99	1 25	1 37	1 211	
9	array	6	992	2 279	2 158		1 33			4 --		2 10	4 126	4 206	
10	HCMUS-TheCows	5	516	3 --	2 101	2 262	1 49					1 7	1 57		
11	HCMUS-Chicken	5	627	4 276	3 102		1 38					1 10	2 81		
12	vector	5	632	4 160	7 --	2 202	1 28					2 10	1 132		

A

Lời giải: Nguyễn Diệp Xuân Quang (HCMUS - Intimidate)

Kí hiệu

- $E(X)$ là giá trị kì vọng của biến ngẫu nhiên X
- $Pr\{X\}$ là xác suất để xảy ra biến cố X

Ý tưởng

Ở một thời điểm bất kì, giả sử ta đã đặt K viên đá (tức là đã xử lí K truy vấn loại 1). Gọi b_x là chỉ số hộp mà viên đá thứ x được đặt vào.

Trước hết, ta xem các truy vấn "1 u v" đều có u = v (tức là b_x sẽ luôn nhận được một giá trị cố định, thay vì nhận một giá trị ngẫu nhiên). Khi đó, với chiếc hộp thứ i, giá trị a_i^2 (bình phương số viên đá được đặt vào hộp i) chính là số cặp chỉ số (x, y) với 1 ≤ x, y ≤ K sao cho cả hai viên đá x và y đều nằm trong hộp i. Do đó, giá trị $A = \sum_{i=1}^{N} a_i^2$ chính là số cặp viên đá (x, y) nằm cùng một hộp.

Trở lại với bài toán ban đầu. Để tính giá trị kì vọng của A, ta sẽ tính với mỗi cặp chỉ số (x, y) (1 ≤ x, y ≤ K) xác suất để hai viên đá x và y nằm trong cùng một hộp rồi tính tổng của chúng.

Gọi $c_{x,y}$ là xác suất để hai viên đá x và y nằm trong cùng một hộp. Nói cách khác, $c_{x,y} = Pr\{b_x = b_y\}$. Hiển nhiên, $c_{x,x} = 1$. Ta có:

$$E(A) = \sum_{x=1}^{K} \sum_{y=1}^{K} c_{x,y}$$

Khi gặp truy vấn "1 u v", giả sử lúc đó đã đặt K viên đá, và đáp án hiện tại là:

$$S = \sum_{x=1}^{K} \sum_{y=1}^{K} c_{x,y}$$

Ta cần đặt viên đá K+1 một cách ngẫu nhiên vào một trong các chiếc hộp từ u đến v. Đáp án sau khi đặt thêm viên đá này là:

$$S' = \sum_{x=1}^{K+1} \sum_{y=1}^{K+1} c_{x,y}$$

Ta nhận xét rằng, thay vì tính lại giá trị S' từ đầu, ta có thể tính sử dụng giá trị S để tính S' theo công thức:

$$S' = S + \sum_{x=1}^{K} c_{x,y} + \sum_{y=1}^{K} c_{x,y} + c_{K+1,K+1}$$

$$= S + 2\sum_{x=1}^{K} c_{x,K+1} + 1 \text{ (do } c_{x,y} = c_{y,x})$$

	1	2	3	...	K	K+1
1	$c_{1,1}$	$c_{1,2}$	$c_{1,3}$	...	$c_{1,K}$	$c_{1,K+1}$
2	$c_{2,1}$	$c_{2,2}$	$c_{2,3}$	...	$c_{2,K}$	$c_{2,K+1}$
	$c_{3,1}$	$c_{3,2}$	$c_{3,3}$	...	$c_{3,K}$	$c_{3,K+1}$
...	...	...	...	...	...	...
K	$c_{K,1}$	$c_{K,2}$	$c_{K,3}$	...	$c_{K,K}$	$c_{K,K+1}$
K+1	$c_{K+1,1}$	$c_{K+1,2}$	$c_{K+1,3}$	...	$c_{K+1,K}$	$c_{K+1,K+1}$

Làm thế nào để tính $\sum_{x=1}^{K} c_{x,K+1}$?

Trước hết, với một viên đá x bất kỳ, để tính $c_{x,K+1}$, ta có thể xét từng chiếc hộp i, tính xác suất để viên đá x và K+1 đều nằm trong hộp i rồi tính tổng của các xác suất trên. Cụ thể:

$$c_{x,K+1} = \sum_{i=1}^{N} Pr\{b_x = i \cap b_{K+1} = i\}$$

Với một chiếc hộp i bất kì, hai biến cố $b_x = i$ và $b_{K+1} = i$ độc lập với nhau nên ta có:

$$c_{x,K+1} = \sum_{i=1}^{N} Pr\{b_{K+1} = i\} * Pr\{b_x = i\}$$

$$\sum_{x=1}^{K} c_{x,K+1} = \sum_{x=1}^{K}\sum_{i=1}^{N} Pr\{b_{K+1} = i\} * Pr\{b_x = i\} = \sum_{i=1}^{N} Pr\{b_{K+1} = i\} * \sum_{x=1}^{K} Pr\{b_x = i\}$$

Theo tính chất tuyến tính của giá trị kì vọng, ta có:

$$E(a_i) = \sum_{x=1}^{K} Pr\{b_x = i\}$$

Từ đó:

$$\sum_{x=1}^{K} c_{x,K+1} = \sum_{i=1}^{N} Pr\{b_{K+1} = i\} * E(a_i)$$

Viên đá K + 1 sẽ được đặt ngẫu nhiên vào một trong các chiếc hộp từ u đến v với xác suất $\frac{1}{v-u+1}$, do đó:

$$\sum_{x=1}^{K} c_{x,\,K+1} = \sum_{i=u}^{v} \frac{1}{v-u+1} * E(a_i) = \frac{1}{v-u+1} \sum_{i=u}^{v} E(a_i)$$

Thay vào công thức tính S' từ S:

$$S' = S + 2 \sum_{x=1}^{K} c_{x,\,K+1} + 1 = S + \frac{2}{v-u+1} \sum_{i=u}^{v} E(a_i) + 1$$

Gọi q là nghịch đảo modulo $10^9 + 7$ của $v - u + 1$. Tức là, ta cần tìm số nguyên q sao cho $q * (v - u + 1) \equiv 1 \ (mod\ 10^9 + 7)$. Các bạn có thể tham khảo cách tìm nghịch đảo modulo tại: http://vnoi.info/wiki/algo/math/modular-inverse.

Ta có thể viết lại công thức trên như sau:

$$S' = S + 2q * \sum_{i=u}^{v} E(a_i) + 1$$

Giải thuật

Từ lý thuyết trên, ta đi đến giải thuật cho bài toán này:

Gọi S là giá trị $E(A)$ cho trạng thái hiện tại. Với mọi i, gọi x_i là giá trị $E(a_i)$ cho trạng thái hiện tại.

Để xử lí truy vấn "1 u v", ta cần thực hiện lần lượt hai thao tác:

- Cộng thêm một lượng $2q * \sum_{i=u}^{v} x_i + 1$ vào S (với q là nghịch đảo modulo $10^9 + 7$ của $v - u + 1$)
- Tăng các giá trị x_i thêm q (với mọi $u \le i \le v$).

Khi đó, với mỗi truy vấn 2, ta chỉ việc in ra giá trị S.

Ta có được thuật toán với độ phức tạp O(Q(N + log M)) (với $M = 10^9 + 7$). Để cải tiến thuật toán, ta nhận xét là mỗi lần thực hiện truy vấn loại 1, ta cần thực hiện hai thao tác:

- Cộng thêm một giá trị nào đó vào các phần tử trong một đoạn con liên tiếp.
- Tính tổng giá trị của các phần tử trong một đoạn con liên tiếp.

Ta có thể sử dụng cây phân đoạn (segment tree) với cơ chế lazy update để thực hiện hai thao tác trên trong O(log N). Các bạn có thể tham khảo về cây phân đoạn tại: http://vnoi.info/wiki/algo/data-structures/segment-tree-extend

Độ phức tạp: O(Q log (N + M)).

Code bằng C++: https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/a.cpp

B

Lời giải: Phạm Văn Hạnh (UET)

Source code:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/b.cpp

Có rất nhiều cách làm cho bài toán này, và đa số các cách giải chính xác đều cho kết quả chính xác mà không yêu cầu thí sinh phải xét rất nhiều trường hợp. Sau đây xin trình bày cách giải của mình.

- Giá trị S (số bước ít nhất để kết quả trận đấu được định đoạt), ta có thể tính như sau:
    - Nếu sau 2K bước, hai bên hoà nhau, S = 2K (Kết quả chỉ được xác định ở lượt cuối cùng, hiển nhiên, vì đảo lộn kết quả bước cuối cùng sẽ làm thay đổi kết quả chung cuộc).
    - Trong trường hợp có một người thắng, ta xét như sau: Giả sử tại một thời điểm, người A đã đi p lượt và có a điểm, người B đã đi q lượt và có b điểm, người A chắc chắn thắng khi và chỉ khi a > b + (K - q). Tương tự, người B chắc chắn thắng khi và chỉ khi b > a + (K - p).
- Trước khi đọc vào truy vấn đầu tiên, ta chuẩn bị trước một cơ sở dữ liệu: Với mỗi giá trị K từ 1 tới 6, ta liệt kê đầy đủ 4^K bộ kết quả có thể có (kết quả của 2K lượt). Với mỗi bộ kết quả, ta tính ba giá trị S, Pa, Pb (S là số bước tối thiểu để xác định duy nhất người chiến thắng (nếu có), Pa là điểm của người A sau S bước, Pb là điểm của người B sau S bước). Ta lập mảng 4 chiều matches, mỗi phần tử là một vector<string>. Với mỗi một bộ kết quả, ta thêm vào vector matches[K][S][Pa][Pb] chuỗi chứa kết quả của S bước đầu tiên.
- Sau khi duyệt toàn bộ 4^K trận đấu, ta dựng được mảng vector matches, ta cần LOẠI BỎ CÁC PHẦN TỬ TRÙNG. Nên nhớ, ở đây ta đã "gom" các trận đấu có chung ba giá trị (S, Pa, Pb) vào cùng một vector, và chỉ quan tâm tới kết quả ở S bước đầu tiên, nên nếu hai trận đấu bất kỳ có cùng (S, Pa, Pb) và diễn biến ở S lượt đầu tiên; thì chúng được coi là như nhau. Ví dụ, hai trận dưới đây được coi là như nhau:
    - Trận 1: A: E E E E E Trận 2: A: E E E E E
    - B: N E E E N B: N E E E E
- Giờ ta xử lý từng truy vấn: Sau khi xác định được giá trị S, Pa và Pb, ta cần tìm giá trị C nhỏ nhất, sao cho **tồn tại duy nhất** một kịch bản nằm trong matches[K][S][Pa][Pb] có chung C bước đầu tiên với input (chính là input). Nói cách khác, nếu ta biểu diễn mỗi kịch bản trong matches[K][S][Pa][Pb] bằng một xâu gồm S ký tự, tồn tại 1 xâu duy nhất có tiền tố độ dài C khớp với input. Do giới hạn của bài toán nhỏ (K < 7, 65 test), bạn có thể duyệt từng giá trị C tăng dần, với mỗi giá trị C, duyệt lại toàn bộ vector matches[K][S][Pa][Pb] để kiểm tra điều kiện trên.

Mình hy vọng lời giải chuẩn của bài toán này không quá phức tạp, nhưng lời giải không phải là điều đáng nói nhất của bài tập này.
Nhìn vào bảng rank, mình thấy có khá nhiều đội nộp bài này. Mình tin bài B này chiếm khá nhiều thời gian và công sức của các đội. Mình xin chúc mừng các đội đã vượt qua bài này, dù sau 5 - 10 lần nộp vẫn kiên trì chiến đấu, và cảm thông cho những đội dù đã cố gắng nhiều nhưng chưa thể vượt qua.
Những con số đã nói lên tất cả, bài B thuộc kiểu bài: Mới nhìn thì thấy dễ, nhìn kĩ sẽ thấy khó, nhìn rất kỹ lại thấy dễ. Đây là kiểu bài truyền thống và khá phổ thông trong các đề thi ACM (các bạn có thể coi lại bài K của đề ACM miền Bắc năm 2017, một bài kiểu tương tự). Bài B có số đội giải được nhiều thứ 4, nhưng tỷ lệ giải được (28/558 ~ 5%) lại thấp thứ hai của đề.

Nhìn qua code các bạn, mình đoán đa số các đội tiếp cận bài toán theo hướng “đoán xem ngta suy luận kiểu gì để kết luận được diễn biến trận đấu”. Và một trường hợp hầu hết các đội đã bỏ sót như sau:
Team A: E N E E N
Team B: E E E E x
(S = 9, Pa = 3, Pb = 4 và ta không cần biết kết quả lượt cuối của B). C = 3. Tại sao lại như vậy?
Ở thời điểm xem xong 3 lượt đầu tiên, thông tin Cee có được:
Team A: E N ? ? ?
Team B: E ? ? ? x
Cee nhận thấy team B đang có 1 điểm và còn 3 lượt nữa. Hiển nhiên ba lượt này có điểm vì Pb = 4. Team A còn 3 lượt, đang có 1 điểm và Pa = 3, do đó trong 3 lượt này, có hai trúng và một trượt.
Nhưng Cee suy nghĩ thêm rằng, trận đấu kết thúc ở lượt thứ 9, là lượt của A. Do trận đấu kết thúc trước 2K lượt (S < 2K), **thông tin ở lượt thứ S phải có lợi cho phe thắng** (nếu thông tin ở lượt thứ S làm thiệt cho phe thắng, đồng nghĩa với việc sau lượt S - 1 phe thắng đang dẫn trước mạnh hơn, phe thắng thừa sức đánh bại sau S - 1 lượt). Vì B thắng nên đội A bắt buộc phải trượt ở lượt thứ 9. Và do đó 3 bước là đủ để Cee khôi phục toàn bộ kết quả.

Tiếp cận bài toán theo hướng phân tích trường hợp mang nhiều rủi ro, và khi ở bài B giới hạn rất nhỏ, hướng tiếp cận đó là sai lầm. Điều đó phần nào khiến nhiều đội nộp rất nhiều lần nhưng vẫn sai.

Fun fact: Mình nghĩ ra bài này khi đang xem tường thuật trận tranh HCĐ Asiad (VN - UAE). Ví dụ trên chính là những gì đã diễn ra trong loạt sút luân lưu của trận đấu này.

C

Lời giải: Vương Hoàng Long (UET - Map)

Bài này lúc mới đọc thì có vẻ vô cùng ảo diệu vì câu trả lời còn chứa số k được máy chọn hoàn toàn random. Nhưng sau khi chơi tay một hồi thì mình nhận ra bài khá dễ =)) . Cách làm của mình là ta sẽ hỏi lần lượt các đoạn [1,1] , [1,2] , [1,3] , …. [1,n]. Mục tiêu của mình là sau khi hỏi đoạn [1,x] thì sẽ tìm được số a[x], vậy cứ hỏi lần lượt ta sẽ có cả mảng.
Đầu tiên ta hỏi đoạn [1,1] => có số a[1]. Khi hỏi đoạn [1,2], gọi rep[1,2] là câu trả lời ta nhận được cho đoạn này, ta sẽ có 2 TH xảy ra:

- Số k = 1 => số a[2] = rep[1,2] - a[1]
- Số k = 2 => số a[2] = rep[1,2] / a[1]

Vậy trong cả 2 TH ta đều tìm được số a[2]. Vậy tổng quát hoá, làm sao để ta tìm được số a[x] khi hỏi đoạn [1,x] và đã biết các số từ a[1] … a[x-1]?

Rất đơn giản:
Gọi K là độ lớn subset được chọn trong câu trả lời, REP là câu trả lời về tổng các subset độ lớn K. Gọi dp[i][j] là tổng các subset độ lớn j của dãy từ 1 -> i.(Lưu ý, ta sẽ set dp[i][0] = 1 để tiện làm trong bài). Vì mình đã biết các số từ 1-> (x-1) nên cũng biết các giá trị dp[i][j] với i chạy từ 1->(x-1)
Ta có: REP = dp[x-1][K] + a[x] * dp[x-1][K-1] , chính là tổng các subset độ lớn K của dãy 1 -> (x-1) + a[x] * tổng các subset độ lớn (K-1) của dãy 1 -> (x-1).
Vậy a[x] = (REP - dp[x-1][K]) / dp[x-1][K-1]
Với phép chia này thì vì ta sử dụng mod 1e9+7 là mod nguyên tố nên thay vì / (dp[x-1][K-1]) thì ta sẽ * (dp[x-1][K-1]^(mod-2).

Vậy cứ làm lần lượt thì ta sẽ có cả mảng. Bài chỉ có vậy thôi :)

D

Lời giải: Lê Minh Phúc (NUS - Illumina)

Source code:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/d.cpp

Đề bài yêu cầu đếm số ngày có dạng DD-MM-YYYY có thể được tạo thành từ 8 chữ số cho trước. Cách đơn giản nhất chính là xét tất cả các hoán vị *abcdefgh* của 8 chữ số đã cho, với *abcd* là năm, *ef* là tháng, *gh* là ngày, và kiểm tra xem nó có đúng là một ngày trong lịch không. Vì 8! = 40,320 khá nhỏ, và chỉ có 50 tests nên chúng ta làm vậy là đủ qua time limit.

Một cách để xét tất cả các hoán vị của 8 chữ số đã cho là:

- Cho 8 số vào trong 1 vector và sort từ nhỏ đến lớn.
- Dùng hàm next_permutation trong C++ để xét các hoán vị từ nhỏ tới lớn. Vì đề bài cũng bắt tìm ngày **nhỏ nhất** sau 01/01/2000, dùng next_permutation cho phép ta lấy luôn hoán vị đầu tiên thoả mãn các điều kiện đã cho làm ngày nhỏ nhất.

Để xét xem 1 hoán vị *abcdefgh* (*abcd* là năm, *ef* là tháng, *gh* là ngày) có thoả mãn các điều kiện đã cho hay không, chú ý các tiêu chuẩn sau:

- $2000 \le abcd$
- $1 \le ef \le 12$, $1 \le gh$
- $gh \le 29$ nếu $ef = 2$, $gh \le 30$ nếu $ef = 4, 6, 9, 11$ và $gh \le 31$ nếu $ef = 1, 3, 5, 7, 8, 10, 12$.
- Đặc biệt, nếu $ef = 2$, ta kiểm tra xem $abcd$ có phải năm nhuận không theo như ghi chú trong đề bài. Nếu có thì chấp nhận $gh \le 29$, nếu không thì chỉ chấp nhận $gh \le 28$

E

Lời giải: Phạm Văn Hạnh (UET)

Source code:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/e.cpp

Đề bài yêu cầu chúng ta đếm số cấp số nhân tăng có tổng bằng S. Ở đây ta chỉ quan tâm đến các dãy có ít nhất 3 phân tử. Việc đếm số dãy có ít hơn 3 phần tử xin dành lại cho bạn đọc.

Ký hiệu dãy là A_0, A_1, ..., A_k. Gọi công bội là $\frac{P}{Q}$ với P và Q nguyên tố cùng nhau. Khi đó Ak = A0 * P^k / Q^k. Do Ak là số nguyên, P và Q nguyên tố cùng nhau nên A0 chia hết cho Q^k. Đặt A0 = x * Q^k, ta có Ai = x * Q^i * P^(k - i). Khi đó S = A0 + A1 + ... + Ak = x * (P^k + P^(k-1)Q + ... + Q^k). Vì k > 1 và S <= 1e6, ta có P, Q <= 1e3.
Do đó, ta sẽ dùng mảng f để lưu lại kết quả với mọi S, và khi đọc input vào ta chỉ dùng mảng f để tính kết quả trong O(1). Để chuẩn bị mảng f, đầu tiên ta duyệt qua mọi bộ ba (P, Q, k), tính giá trị của biểu thức T = P^k + P^(k-1)Q + ... + Q^k, và tăng giá trị của f(T) lên 1. Sau khi đã duyệt qua mọi bộ (P, Q, k) ta sẽ duyệt các giá trị T từ 1e6 giảm về 1, và gán f(bT) += f(T).

F

Lời giải: Nguyễn Ngọc Trung

Dễ dàng nhận thấy rằng ta chỉ cần quan tâm đến những điểm nằm trên bao lồi của những điểm nằm trong đa giác to. Do đó bước đầu tiên là tìm bao lồi của những điểm này. Giả sử các đỉnh của đa giác lồi to là $A_1, A_2, ..., A_n$ và đa giác lồi con tìm được là $B_1, B_2, ..., B_m$ (theo thứ tự ngược chiều kim đồng hồ).
Tiếp theo với mỗi vị trí i ta sẽ tìm tiếp tuyến phía bên phải từ điểm A_i đến đa giác con. Đó là điểm B_j mà tích có hướng $\|A_iB_j \times B_jB_{j-1}\|$ và $\|A_iB_j \times B_jB_{j+1}\|$ là không âm. Ký hiệu vị trí j cần tìm cho vị trí i là $f[i]$, nếu $j < i$ ta quy ước $f[i] = j + n$. Như vậy là dãy $f[1], f[2], \ldots, f[n]$ là không giảm và tập giá trị là từ 1 đến $2n - 1$. Từ đây sử dụng two pointers để tìm mảng $f[]$ trong $O(N)$.
Sau khi tìm được mảng $f[]$, ta sẽ tìm mảng $g[i]$ là vị trí xa nhất từ i có thể đi tới mà đa giác tạo thành từ đỉnh i đến $g[i]$ không chứa bất kỳ điểm nào nằm bên trong hoặc trên biên. Ta phải có điều kiện là tích có hướng $\|A_iA_j \times A_iB_{f[i]}\|$ dương. Ta tiếp tục sử dụng two pointers tại đây và độ phức tạp vẫn là $O(N)$.
Cuối cùng để tìm đa giác có số đỉnh nhỏ nhất. Một cách đơn giản là tham lam, tại mỗi điểm ta cố gắng đi tối đa có thể dựa theo mảng $g[]$ cho đến khi gặp lại hoặc vượt qua điểm ban đầu. Một cách tự nhiên và dễ nghĩ nhất là xây dựng cấu trúc sparse table tương tự như trong *LCA*. Độ phức tạp sẽ là $O(NlogN)$.

Summary:

- Complexity: $O(NlogN + KlogK)$ or $O(N + KlogK)$.
- $O(N + KlogK)$ is left as an exercise for readers.
- Author's code: https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/f.cpp

G

Lời giải: Phạm Văn Hạnh (UET)

Source code:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/g.cpp

Nếu đã để ý được đồ thị trong bài không phải là cây, và thậm chí đồ thị còn có cạnh lặp; thì ta quy về bài toán quen thuộc - Đếm số tam giác trên đồ thị vô hướng. Có khá nhiều cách làm bài toán này, các bạn có thể tham khảo tại đây:
https://www.geeksforgeeks.org/number-of-triangles-in-a-undirected-graph/

Lời giải của mình cho bài này như sau:
Nhận thấy đáp số của bài toán bằng hai lần số bộ ba (u, v, w) sao cho tồn tại cạnh (u, v), tồn tại cạnh (v, w) nhưng không tồn tại cạnh (u, w), ta khởi tạo đáp số của bài toán bằng tổng Du * (Du - 1) với Du là bậc của đỉnh u trong đồ thị. Sau đó, ta duyệt từng cạnh (u, v); với mỗi cạnh, ta đếm số đỉnh w kề với cả u và v, với mỗi đỉnh w, ta trừ đáp số đi 2.

Việc đếm số đỉnh w kề với cả u lẫn v giống như một loại bài toán truy vấn ta hay gặp: Cho n tập hợp S1, S2,..., Sn có tổng lực lượng là K; và Q truy vấn, mỗi truy vấn cần xử lý trên 2 trong số n tập này. Bài toán này có hướng giải quen thuộc: Duyệt trên tập có lực lượng nhỏ hơn trong hai tập được truy vấn, sao cho độ phức tạp xử lý một truy vấn tỷ lệ với lực lượng tập nhỏ mà **không tuyến tính** với lực lượng tập lớn. Khi đó tổng độ phức tạp của bài toán (ước lượng tương đối) là O(K * sqrt(Q)).

Trong bài này, ta có n tập hợp chính là n tập đỉnh kề với mỗi đỉnh của đồ thị. Các truy vấn ở đây là các cạnh (u, v); ở đó ta cần đếm số phần tử chung của hai tập Adj(u) và Adj(v) - hai tập đỉnh kề với u và v.
Với mỗi "truy vấn" (u, v); ta chọn ra một đỉnh (giả sử là u); duyệt tập Adj(u). Với mỗi w thuộc Adj(u), kiểm tra xem w có thuộc Adj(v) hay không. Việc kiểm tra này có thể thực hiện bằng cấu trúc dữ liệu HashMap với độ phức tạp **trung bình** O(1) hoặc Set với độ phức tạp **ổn định** là O(logN).

Tuy nhiên, bạn có cách làm tốt hơn việc kiểm tra cạnh (v, w) được thực hiện với độ phức tạp **ổn định** O(1) (nghĩa là bạn không cần đặt niềm tin vào sự may rủi của HashMap) như sau:
- Với mỗi đỉnh u, bạn lưu lại danh sách các "truy vấn" (u, v) mà ở đó bạn chọn tập Adj(v) để duyệt thay vì Adj(u).
- Với mỗi đỉnh u, ta "xử lý" các "truy vấn" này cùng một lúc: Đầu tiên ta chuẩn bị một mảng đánh dấu các đỉnh trong tập Adj(u). Với mỗi "truy vấn" (u, v) trong danh sách, ta duyệt tập Adj(v), sử dụng mảng đã chuẩn bị để kiểm tra cạnh (w, v).

Có ba điểm khiến cho rất nhiều đội nộp sai ở bài tập này:
1. Các đội nghĩ đồ thị là cây.
2. Các đội nghĩ đồ thị là đơn đồ thị, nên quên mất phải xóa đi những cạnh lặp trước khi tính.
3. Các đội cài thuật toán không tốt dẫn đến chạy quá chậm.

Về điểm thứ ba, giới hạn thời gian 3s là gấp 10 lần lời giải C++ của mình (chạy mất 0.3s). Nếu các bạn lưu hết các cạnh vào map, kiểm tra cạnh với độ phức tạp O(logN), bạn gần như chắc chắn bị TLE. Nếu bạn dùng unordered_map, các bạn cần chống việc rehash của unordered_map trong C++. Sử dụng unordered_map có cơ hội AC lớn hơn dùng map, nhưng nếu code không cẩn thận bạn cũng có thể bị TLE. Cách tốt nhất là bạn tự cài đặt HashMap, và với cạnh (u, v); bạn chọn HashKey là (u * n + v) xor x (n là số đỉnh của đồ thị). x là một số được chọn ngẫu nhiên. Cách làm này khiến HashMap của bạn khó bị tiêu diệt hơn rất nhiều.

Về điểm thứ hai, mình hơi bất ngờ vì mình cho rằng, nếu các bạn không nghĩ đồ thị là cây, thì cũng sẽ không nghĩ đồ thị là đơn đồ thị chứ nhỉ :)

Về điểm thứ nhất, mình nghĩ đề bài không đề cập đến đồ thị liên thông thì không có lý do gì bạn nghĩ nó liên thông cả. Mình có quan sát vòng miền nam, mình có để ý tới bài E:

Mình ngẫm kỹ đoạn *"the transportation system consisting of $n - 1$ two-way streets connecting all the residential areas"* và tự hỏi *"connecting all the residential areas"* có nghĩa là liên thông hay không. Nếu thay đổi thành *"connecting all these areas"* hay *"connecting these areas"* hoặc *"connecting the areas"* thì nghĩa của câu văn thay đổi như thế nào. Mình có cảm giác là một chữ *"all"* chưa đủ nhấn mạnh rằng đồ thị liên thông, và nếu xóa chữ "all" đi mà thay thành "these" hoặc "the" thì lại có vẻ là câu văn không mang ý nghĩa đồ thị liên thông. Mình khá băn khoăn về ngữ nghĩa của câu này.
Nếu dịch ra tiếng Viêt, câu văn này có nghĩa là "Hệ thống giao thông gồm N - 1 con đường hai chiều kết nối tất cả các khu vực lại". Câu này có vẻ mang nghĩa là "N - 1 con đường hai chiều này chỉ dùng để kết nối N khu vực trong thành phố với nhau" với hàm ý là "Các con đường này **không nối** một khu vực trong thành phố ra ngoại thành". Nếu coi hàm ý của câu này là "Việc đi lại trong thành phố là thông suốt" hoặc "Từ một khu vực có thể tới mọi khu vực khác" thì ý nghĩa của nó không rõ nét như hàm ý trên.

Tuy nhiên, khi mình nhìn bảng điểm, trong 60p đầu tiên có tầm 4-5 đội AC bài E. Điều này cho thấy các đội chỉ đọc đề lướt qua và tin khá chắc là đồ thị trong bài là cây.

H

Tác giả/Lời giải: Nguyễn Vương Linh (MIT/Facebook)

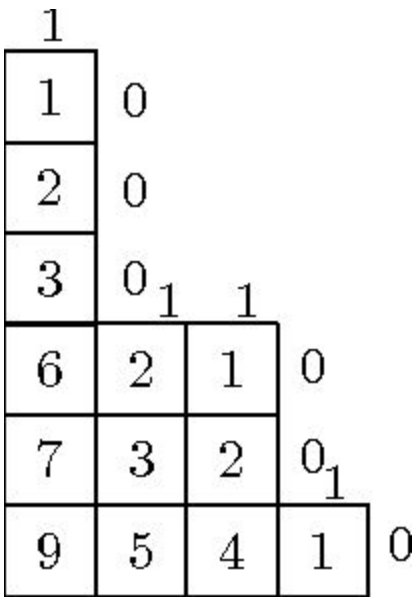
Source code:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/h.cpp

Tóm tắt lời giải:

- Để ý rằng chúng ta chỉ cần giải quyết bài toán cho 1 hình bậc thang. Với 2 hình bậc thang, số cách xếp sẽ là $C(p + q - 2, p - 1) FG$ với p và q là số ô vuông của bậc thang 1 và 2; F là số cách xếp bậc thang thứ nhất và G là số cách xếp bậc thang thứ 2. (Ta nhân với hệ số $C(p + q - 2, p - 1)$ vì 2 ô xanh và đỏ đầu tiêng đã được cố định sẵn).
- Áp dụng công thức độ dài Hook: https://en.wikipedia.org/wiki/Hook_length_formula; số cách xếp bậc thang được tính theo công thức

$$\frac{p!}{c_1!\, c_2! \ldots c_k!}$$

Trong đó $c_1, .., c_k$ là độ dài các *móc câu (hook)* của hình bậc thang, và p là số ô vuông trong hình bậc thang.



Ví dụ minh hoạ cho 1 hình bậc thang và độ dài các móc câu. Mỗi ô vuông có 1 móc câu duy nhất là hình chữ L xuất phát từ ô đấy và kéo dài theo cả 2 chiều đến đường biên của hình bậc thang (kí hiệu bằng số 0/1 trong hình)

- Độ dài các móc câu có thể được tính bằng QHĐ khá cơ bản với độ phức tạp O(MN).
- Cần chú ý xử lí với modulo 100003, tuy nhiên phần xử lí này không quá phức tạp.

Bình luận:

- Đây là một bài khó từ cả việc hiểu đề và tiếp cận vấn đề, nhất là với ví dụ không thực sự trực quan. Khó nhất là tìm ra được công thức Hook và cài đặt hiệu quả; kiến thức này thường không có trong chương trình học bậc phổ thông / ĐH tuy nhiên vẫn có thể tiếp cận được bằng cách cài chương trình duyệt trâu để tìm ra quy luật.
- Chúc mừng đội THREE (với 3 thí sinh là học sinh phổ thông) đã là đội duy nhất vượt qua thử thách này, cho thấy không có gì là không thể :)
- Những kiểu bài này đã từng xuất hiện trong các kì thi trước đây (ví dụ https://vnoi.info/problems/TREELINE/ với công thức mấu chốt là số Catalan https://en.wikipedia.org/wiki/Catalan_number), các đội nên chú ý bổ sung kiến thức toán/tổ hợp ở trong đội; đây có thể là yếu tố quyết định trong các vòng thi cuối cùng.

Fun fact:

- Với hình bậc thang 2 * n, đáp số là số Catalan C_n. Trùng hợp ngẫu nhiên?

I

Lời giải: Vương Hoàng Long (UET - Map)

Bài này là một bài biếu điểm. Ta chỉ cần làm đúng như đề bài yêu cầu. Đầu tiên sort lại tất cả các con Pokenom theo chỉ số Attack, rồi insert vào Set id của K con có Attack cao nhất. Làm tương tự với chỉ số Defend và Health. Sau đó chúng ta chỉ cần in ra Set.size() là xong (do Set đã tự động bỏ hết phần tử trùng lặp)

J

Lời giải: Nguyễn Vũ Hoàng Vương

Trạng thái kết thúc có dạng như thế nào?
Ký hiệu a, b thứ tự là giá trị của ký tự đầu tiên và cuối cùng của xâu S ban đầu. Hai ký tự đầu cuối này không bao giờ được xóa.
Trò chơi kết thúc khi và chỉ khi người chơi không thể xóa ký tự nào được nữa, tức là việc xóa ký tự nào cũng sẽ dẫn tới việc có 2 ký tự kề nhau và bằng nhau.
Xét xâu S khi trò chơi kết thúc.
Ký tự thứ nhất của xâu không bao giờ bị xóa. Ký tự thứ hai nếu xóa thì sẽ dẫn đến việc ký tự thứ nhất và thứ ba bằng nhau => ký tự thứ ba = a. Lập luận tương tự, ta có các ký tự tại vị trí lẻ sẽ có giá trị bằng a.
S có dạng: S = a?a?...?a hoặc S = a?a?...?a?

TH1: S = a?a?...?a
Ký tự cuối của S = a, ký tự cuối của S không bao giờ bị xóa => a = b
Độ dài của xâu S khi trò chơi kết thúc có độ dài lẻ.
Gọi N là độ dài S ban đầu. Nếu N lẻ, người chơi đầu khi đến lượt mình, xâu S luôn có độ dài lẻ; còn người chơi thứ hai khi đến lượt mình xâu S luôn có độ dài chẵn. Suy ra người chơi đầu sẽ gặp trạng thái kết thúc => người chơi đầu tiên thua nếu N lẻ. Tương tự, người chơi thứ hai sẽ thua nếu N chẵn.

TH2: S = a?a?...?a?
Ký tự ? cuối cùng phải = b. Do không có 2 ký tự nào kề nhau và bằng nhau, a khác b.
Độ dài của xâu S khi trò chơi kết thúc có độ dài chẵn.
Lập luận tương tự trên, ta có:

- Nếu N lẻ, người chơi thứ hai thua
- Nếu N chẵn, người chơi thứ nhất thua

Vậy:

- Nếu a = b, TH1 xảy ra
- Nếu a khác b, TH2 xảy ra

Code tham khảo:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/j.cpp

K

Lời giải (Đinh Nguyên Khôi - HCMUS Intimidate)
Ta sẽ đi từ giải thuật "thô thiển" đơn giản nhất mà có thể dễ dàng nghĩ ra, sau đó ta sẽ tối ưu dần lên.

Gọi S(x, y) là TẬP HỢP tất cả các hình chữ nhật phủ lên trên tọa độ (x, y). Ở đây ta định nghĩa hình chữ nhật có tọa độ góc trái dưới là (x1, y1), tọa độ góc phải trên là (x2, y2) "phủ" lên điểm (x, y) khi và chỉ khi: x1 ≤ x ≤ x2 và y1 ≤ y ≤ y2. Ví dụ: Hình chữ nhật thứ nhất góc trái dưới (2, 2) và tọa độ góc phải trên là (8, 8) thì sẽ phủ lên các điểm {(2, 2), (2, 3), ….}.

Nhận xét rằng để (x1, y1) có đường đi đến (x2, y2) thì tập hợp các hình chữ nhật phủ lên điểm (x1, y1) và điểm (x2, y2) phải giống hệt nhau. Như vậy, ta sẽ có một thuật toán "sơ đẳng" nhất ban đầu:

- Với truy vấn loại 1: Với hình chữ nhật thứ i (x1, y1, x2, y2), ta duyệt với mọi (x, y) nằm trong hình chữ nhật này, đưa i vào trong S(x, y).
- Với truy vấn loại 2: Ta duyệt mọi (x, y) nằm trong hình chữ nhật thứ i (x1, y1, x2, y2) và xóa i ra khỏi S(x, y)
- Với truy vấn loại 3, ta so sánh S(x1, y1) và S(x2, y2) xem có bằng nhau hay không. Nếu bằng nhau thì xuất 'Y' và xuất N' trong trường hợp ngược lại.

Vì trong bài này gồm các thao tác "thêm" và "xóa", nên ta cần tổ chức S(x, y) theo kiểu Balanced Binary Search Tree (set) trong c++.

Hiển nhiên ta nhận ra rằng độ phức tạp của hướng giải quyết ở trên đề xuất ra là quá lớn, ta cần một hướng tiếp cận khác dựa trên nền của hướng giải quyết này và tối ưu lên.

Vì sử dụng set để lưu tập hợp thì sẽ rất chậm, vì các thao tác trải qua N phần tử, thì độ phức tạp sau N thao tác là NlogN.
Ta có thể cải tiến công đoạn này làm sao từ việc khi đưa 1 phần tử vào tốn O(log(N)) xuống thành O(1), đồng thời thao tác xóa 1 phần tử cũng tốn O(1), như vậy ta có một cải tiến tiếp theo sử dụng hash như sau: Giả sử như S(x, y) = {1, 3, 4, 6}, ta sẽ hash chúng thành 1 con số nào đó để tiện so sánh. Ở đây, ta sẽ xem những con số {1, 2, 3, ….., 100000} như những CHỮ SỐ trong hệ cơ số 100.000. Tức nếu S(x, y) = {1, 3, 4, 6}, thì con số ta cần lưu là V = (1 * 100.000 + 3 * 100.000^3 + 4 * 100.000^4 + 6 * 100.000^6) mod MOD. Với MOD là một con số nào đó ta có thể chọn. Như vậy, ta có thể cải tiến để S(x, y) không phải là set nữa, mà chỉ là một số nguyên. Tất nhiên kiểu hash này sẽ có xác suất sai, nhưng xác suất này không đáng kể.

Như vậy, với việc thêm một hình chữ nhật thứ i vào trong S(x, y), ta cộng S(x, y) thêm một giá trị là i * 100.000^i, và việc xóa một hình chữ nhật thứ i ra khỏi S(x, y), ta trừ S(x, y) đi một giá trị là i * 100.000^i.

Vấn đề thứ 2 làm giải thuật ban đầu chậm, đó là việc với hình chữ nhật (x1, y1, x2, y2), ta phải duyệt toàn bộ (x, y) để cập nhật, tốn độ phức tạp gần như là O(R * C). (R là số dòng, C là số cột). Để cải tiến công đoạn này, ta cần tổ chức bảng S như là cây fenwick trên bảng 2D, vì cây fenwick trên bảng 2D cho phép ta có thể tăng (giảm) một hình chữ nhật có tọa độ góc trái dưới là (x1, y1) và góc phải trên (x2, y2) trong O(log(R) * log(C)), thao tác truy vấn tìm tổng của một hình chữ nhật trên bảng 2D cũng là O(log(R) * log(C)) (https://www.topcoder.com/community/competitive-programming/tutorials/binary-indexed-trees/)
Như vậy, để cập nhật những giá trị S(x, y) trong hình chữ nhật (x1, y1, x2, y2) thêm một giá trị val, ta sẽ làm như sau:

- Thêm toàn bộ các ô trong hình chữ nhật (1, 1, x2, y2) giá trị val
- Xóa toàn bộ các ô trong hình chữ nhật (1, 1, x1 - 1, y2) giá trị val
- Xóa toàn bộ các ô trong hình chữ nhật (1, 1, x2, y1 - 1) giá trị val
- Thêm toàn bộ các ô trong hình chữ nhật (1, 1, x1 - 1, y1 - 1) giá trị val.

Các thao tác trên, ta sử dụng cây fenwick trên bảng 2D, các thao tác đều là log(R) * log(C). Và với truy vấn so sánh, ta cũng chỉ lấy kết quả các ô (1, 1, x1, y1) và (1, 1, x2, y2) để so sánh là ra kết quả.

Tuy nhiên, với cách giải đã đề xuất ở trên, nếu không cẩn thận, khi ta submit lên sẽ bị Timelimit. Ta cần tối ưu thêm một chút nữa. Để ý rằng đề bài đã nói rằng với các truy vấn loại 1, các x1, y1, x2, y2 đều là số lẻ và với các truy vấn loại 3, các x1, y1, x2, y2 đều là số chẵn. Ta sẽ sử dụng phép đồng dạng, biến hình chữ nhật (x1, y1, x2, y2) về thành một hình chữ nhật với tọa độ (x1', y1', x2', y2'). Trong đó x1' = (x1 + 1) / 2, y1' = (y1 + 1) / 2, x2' = (x2 - 1) / 2 và y2' = (y2 - 1) / 2. Với truy vấn số 3, ta sẽ biến tọa độ điểm đầu và điểm đích thành x1/2, y1/2, x2/2, y2/2 thì bài toán sẽ không thay đổi.

Độ phức tạp: O(Q * (log(R) * log(C))) với Q là số truy vấn, R và C lần lượt là số dòng và số cột của bảng.
Source code:
https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/k.cpp
Bonus: Ta có thể sử dụng hash mà không cần mod cho một số cũng được, ta có thể để cho S(x, y) tràn số trong kiểu long long.

L

Lời giải: Nguyễn Vũ Hoàng Vương

Đề bài: Có N Pokenom và N chiếc bánh đều được đánh số từ 1 tới N. Pokenom i muốn ăn bánh thứ i. Có một số nguyên liệu làm bánh được đánh dấu bởi mọi số nguyên tố <= N. Nếu x có phân tích thừa số nguyên tố x = p1^a1 * p2^a2 * .... thì để làm bánh x cần a1 gram nguyên liệu p1, a2 gram nguyên liệu p2, vân vân. Mua k gram nguyên liệu p mất k^2 * C(p) tiền. Pokenom i sẽ vui và trả Bash V(i) tiền nếu số nguyên liệu Bash mua **có thể** làm được bánh i. Hỏi số tiền lớn nhất mà Bash có thể nhận được là bao nhiêu?

Lời giải:

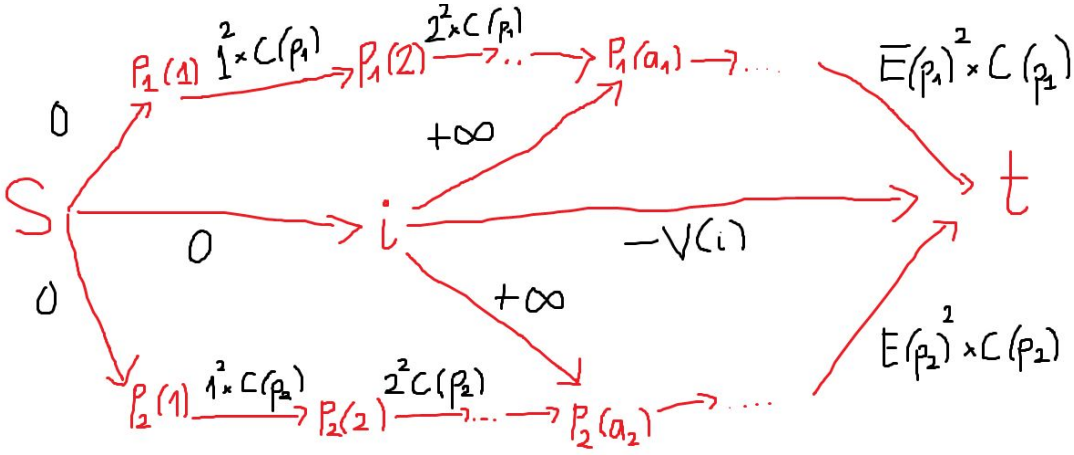
Ta sẽ dựng một đồ thị theo cách như sau.

Các đỉnh của đồ thị là:

- Có 2 đỉnh s và t
- Có N đỉnh tương ứng với N Pokenom, đỉnh i tương ứng với Pokenom i.
- Với mỗi nguyên liệu được đánh dấu bởi số nguyên tố p, ký hiệu E(p) là số nguyên dương lớn nhất mà p^E(p) <= N, đồ thị sẽ có E(p) đỉnh p(1), p(2), … tương ứng với nguyên liệu p. Đỉnh p(k) tương ứng với việc mua k gram nguyên liệu p. Dễ thấy trong phương án tối ưu, ta không bao giờ mua quá E(p) nguyên liệu p.

Các cạnh của đồ thị là:

- Với mỗi đỉnh i ứng với Pokenom i, có cạnh một chiều từ s tới i với trọng số 0, và có cạnh một chiều từ i tới t với trọng số -V(i).
- Với mỗi nguyên liệu p, có cạnh một chiều nối từ s tới p(1) với trọng số 0, có cạnh một chiều nối từ p(k) tới p(k+1) với trọng số bằng chi phí mua k gram nguyên liệu p và bằng k^2 * C(p); có cạnh nối từ p(E(p)) tới t với trọng số E(p)^2 * C(p).
- Với mỗi đỉnh i ứng với Pokenom i, giả sử i có phân tích thừa số nguyên tố i = p1^a1 * p2^a2 * …., có cạnh nối từ i tới p1(a1) với trọng số +oo, có cạnh nối từ i tới p2(a2) với trọng số +oo, vân vân.



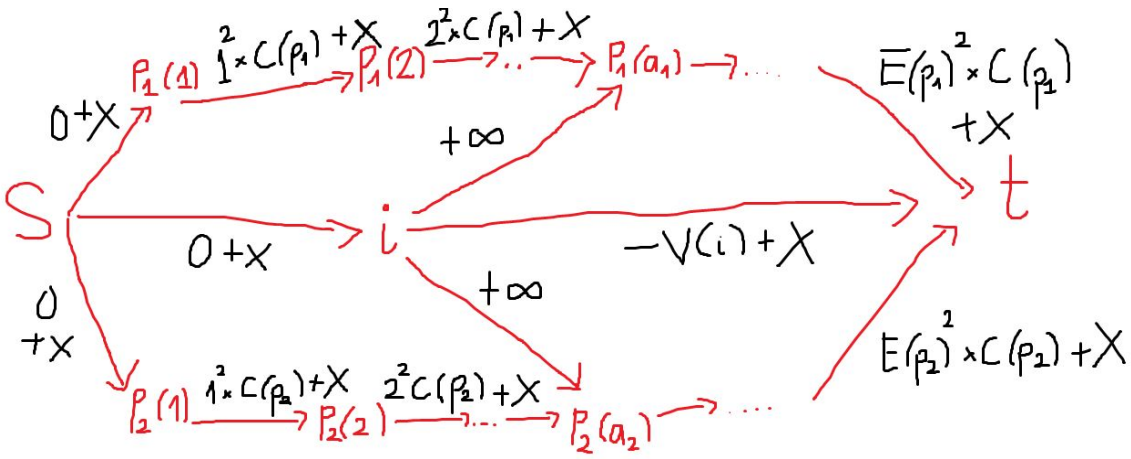
Định nghĩa 1 lát cắt từ s tới t của đồ thị là sau khi loại bỏ 1 số cạnh ra khỏi đồ thị, s không đi được đến t nữa. Ta gọi lát cắt là cực tiểu khi tổng trọng số cách cạnh bị loại bỏ là nhỏ nhất. Ta sẽ dùng từ "cắt" cạnh với ý nghĩa tương đương với việc loại bỏ cạnh đó.

Ta sẽ chứng minh một lát cắt cực tiểu của đồ thị nói trên tương ứng với một lời giải:

- Với mỗi nguyên liệu p, **đúng** 1 trong các cạnh s->p(1), p(1)->p(2), … sẽ bị cắt trong lát cắt cực tiểu, bởi vì nếu trong số các cạnh này không có cạnh nào bị cắt thì s vẫn đi được đến t, và vì ta muốn lắt cắt là **cực tiểu** nên không có lý do gì để cắt 2 cạnh cả. (**Lưu ý:** lập luận này là **sai**! Nó chỉ đúng trong trường hợp trọng số các cạnh là dương, mà cạnh của đồ thị đang xét có trọng số âm. Tuy nhiên ta tạm thời giả sử trọng số các cạnh là dương hết. Cạnh âm sẽ được xử lý ở sau này).
- Với mỗi đỉnh i ứng với Pokenom i, lập luận tương tự như trên, ta có đúng 1 trong 2 cạnh s->i hoặc i->t sẽ bị cắt (**Lưu ý:** tạm thời giả sử trọng số các cạnh là dương).
- Cắt cạnh s->i sẽ tương ứng với việc Pokenom i không vui, Bash nhận được 0 tiền, 0=(-1)*0 = (-1) * trọng số cạnh s->i bị cắt.
- Cắt cạnh i->t tương ứng với việc Pokenom i vui, Bash nhận được V(i) tiền, V(i) = (-1)* (-V(i)) = (-1) * trọng số cạnh i->t bị cắt
- Với nguyên liệu p, cắt cạnh p(k)->p(k+1) tương ứng với việc mua k gram nguyên liệu p, Bash mất k^2 * C(p) tiền, - k^2 * C(p) = (-1) * trọng số cạnh p(k)->p(k+1) bị cắt.
- Giả sử i = p1^a1 * p2^a2 …. Ta sẽ chứng minh nếu Pokenom i vui, thì sẽ phải mua ít nhất a1 gram nguyên liệu p1, ít nhất a2 gram nguyên liệu p2, vân vân. Thật vậy. Pokenom vui tức cạnh i->t bị cắt. Trong lát cắt cực tiểu, cạnh i->p1(a1) với trọng số +oo không bao giờ bị cắt. Giả sử có ít hơn a1 gram nguyên liệu p1, tức là không có cạnh nào trong số các cạnh p1(a1)->p1(a1+1), p1(a1+1)->p1(a1+2), …, p1(E(p1))->t bị cắt cả. Suy ra s vẫn đi được đến t thông qua con đường s->i->p1(a1)->p1(a1+1)->...->p1(E(p1))->t, mâu thuẫn. Vậy phải có ít nhất a1 gram nguyên liệu p1. Chứng minh tương tự với a2 gram p2, vân vân.

Vậy, lát cắt cực tiểu tương ứng với một lời giải của bài toán. Số tiền Bash nhận được trong lời giải ứng với lát cắt cực tiểu, dễ thấy, bằng (-1) * lát cắt cực tiểu. Do lát cắt là cực tiểu nên (-1) * (lát cắt cực tiểu) chính là số tiền lớn nhất mà Bash có thể nhận được.

Xử lý trọng số cạnh âm:
Tìm X là một số nguyên dương đủ lớn mà khi cộng X vào trọng số của tất cả các cạnh, đồ thị không có cạnh với trọng số âm nữa. Có thể chọn X = max(V(1), V(2), …, V(N)) + 1
Đồ thị mới sẽ là:



Khi đó:

- Cắt cạnh s->i sẽ tương ứng với việc Pokenom i không vui, Bash nhận được 0 tiền, 0= -X + X = (-1) * (trọng số cạnh s->i bị cắt) + X.
- Cắt cạnh i->t tương ứng với việc Pokenom i vui, Bash nhận được V(i) tiền, V(i) = -(-V(i)+X) + X = (-1) * (trọng số cạnh i->t bị cắt) + X
- Với nguyên liệu p, cắt cạnh p(k)->p(k+1) tương ứng với việc mua k gram nguyên liệu p, Bash mất k^2 * C(p) tiền, - k^2 * C(p) = (-1) * (trọng số cạnh p(k)->p(k+1) bị cắt) + X.

Suy ra số tiền lớn nhất mà Bash có thể nhận được là:

(-1) * (lát cắt cực tiểu) + X * N + X * m

trong đó m là số lượng nguyên liệu, tức m là số lượng số nguyên tố <= N.

Tìm lát cắt cực tiểu:

Người ta đã chứng minh: luồng cực đại = lát cắt cực tiểu (max flow = min cost).

Do đó để tìm lát cắt cực tiểu, ta tìm luồng cực đại là xong.

Tài liệu tham khảo: http://vnoi.info/wiki/translate/wcipeg/Flows

Code tham khảo:

https://github.com/acmicpc-vietnam/acmicpc-vietnam.github.io/blob/master/2018/national/codes/l.cpp