Tổng quan

Bài	**Độ khó nghĩ / cài**	**Tag**	**unsigned's Result**
A. Invariant Sets	☆☆☆/☆☆	Đồ thị	+4 / 185
B. Scheduling Events	☆/☆	DP	+2 / 11
C. XOR	☆☆/☆	Tham lam, bitwise	+ / 23
D. Nasa	☆☆☆☆/☆☆☆	Tham lam, DP	
E. Determinant of the matrix	☆☆☆/☆	Constructive	+1 / 60
F. Two-person game	☆☆☆☆/☆☆	Đồ thị, DP, Game	+1 / 124
G. Overloaded Ants	☆☆☆/☆☆☆☆☆	BST	
H. Schedule the Football Tournament	☆☆☆/☆☆☆	Constructive, Đồ thị, Cặp ghép, Notebook	+2 / 227
I. Graph Representation	☆☆/☆	Đồ thị	+ / 30
J. Replacement Roads	☆☆☆/☆☆☆	Đồ thị	-4 QQ
K. Pairing	☆☆/☆☆	Duyệt	+2 / 72

A. Invariant Sets

Tóm tắt đề bài

Đây là một bài có đề khá khó hiểu, đã làm nhiều team phải “vắt óc” đọc đề và bỏ cuộc. Thực chất, đề rất đơn giản.
Cho 2 đa thức **P** và **Q** trên hệ số modulo M, tìm số tập con của {0, 1, 2, … M - 1} mà “invariant” với cả đa thức P lẫn Q.
Một tập S được gọi là invariant đối với đa thức P nếu thay mọi phần tử x trong S thành P(x) thì tạo ra tập S’ giống hệt tập S.

Solution

Từ định nghĩa invariant, ta có thể dễ dàng nhận thấy tính chất invariant tương đương với: với mọi phần tử x thuộc S, tồn tại a và b thuộc S thỏa mãn P(a) = x và P(x) = b.

Từ đây, nếu ta coi các số từ 0 đến M - 1 là các đỉnh, và nối các cạnh từ x đến P(x), thì tập con mà ta chọn phải là một tập hợp các chu trình (hiển nhiên là các chu trình đơn, vì mỗi đỉnh chỉ có 1 bậc ra).

Như vậy, bài toán trở thành: cho 2 đồ thị M đỉnh M cạnh, đếm số tập con mà là tập các chu trình trong cả 2 đồ thị.

Việc đầu tiên cần làm là loại bỏ các đỉnh không thuộc chu trình nào. Việc này khá đơn giản: ta liên tục loại bỏ các đỉnh không có bậc vào bằng BFS. Lưu ý một đỉnh chỉ cần không thuộc chu trình trong 1 trong 2 đồ thị là ta phải xóa trên cả 2 đồ thị. Vì mỗi đỉnh chỉ có thể bị xóa 1 lần, nên độ phức tạp của việc này là O(M).

Sau khi thực hiện xóa bỏ các đỉnh không được chọn, ta sẽ còn lại các chu trình trên cả 2 đồ thị. Không khó để nhận thấy nếu chọn 1 đỉnh, ta sẽ phải chọn cả chu trình chứa đỉnh đó, và cả đỉnh tương đương của đồ thị bên kia. Như vậy, nếu ta nối cạnh giữa 2 đỉnh `x` của 2 đồ thị, chúng sẽ tạo ra các thành phần liên thông, mà khi chọn bạn phải chọn tất cả đỉnh trong TPLT đó. Bạn sẽ phải đếm số thành phần liên thông này. Việc này có thể thực hiện bằng nhiều cách (duyệt đồ thị, dsu,...) với độ phức tạp O(M).

Khi đã có các thành phần liên thông, dễ dàng nhận thấy việc chọn 1 TPLT không ảnh hưởng đến lựa chọn các TPLT khác. Như vậy đáp số sẽ là 2^{số TPLT}.

Độ phức tạp của cả bài là O(M).

B. Scheduling events

Tóm tắt đề bài

Đề bài của bài này khá trực quan.
Cho N ngày và K sự kiện. Các sự kiện phải diễn ra lần lượt, sự kiện thứ i sẽ phải được tổ chức trong Li ngày liên tiếp. Tổ chức sự kiện trong ngày thứ i sẽ tốn Ai. Tìm cách tổ chức tiết kiệm nhất sao cho 2 sự kiện không được trùng ngày.

Solution

Đây là một bài dễ của đề. Thuật toán chỉ đơn giản là quy hoạch động.
Gọi f[i][j] là chi phí nhỏ nhất để tổ chức j event đầu tiên trong khoảng thời gian từ ngày 1 đến ngày i. Ban đầu, f[0][0] = 0.
Xét trạng thái f[i][j], ta có 2 trường hợp:

- Ta không sử dụng ngày i. Trường hợp này f[i][j] = f[i - 1][j].
- Ta sử dụng ngày i - Lj + 1 … i. Điều kiện xảy ra là i >= Lj. Khi đó f[i][j] = f[i - Lj][j - 1] + A[i - Lj + 1] + A[i - Lj + 2] + … + A[i].

Đáp số của f[i][j] sẽ là trường hợp có lợi hơn (min) của 2 trường hợp trên. Để tính trường hợp 2 trong O(1), ta có thể sử dụng mảng tổng dồn cho mảng A[].
Đáp số của bài toán là f[N][K].

Độ phức tạp của thuật toán là O(NK).

C. XOR

Tóm tắt đề bài

Cho 2 số A và N. Tìm B nhỏ nhất thỏa mãn (A xor B) chia hết cho N.

Solution

Có 1 cái sol rất là bựa là in min 2 cái gì đó mình không nhớ lắm, nhưng cũng khó chứng minh nên mình sẽ không viết ở đây. Thay vào đó mình sẽ trình bày sol O(log maxB) của team mình.

Lần lượt chốt các vị trí của từng bit của B, từ lớn đến bé, thử 0 trước để tìm số bé nhất. Giả sử ta đã chốt i vị trí đầu tiên, ta chỉ cần kiểm tra xem 63 - i vị trí còn lại có cách điền nào thỏa mãn điều kiện chia hết không?

Việc kiểm tra rất đơn giản. Vì ta còn 63 - i bit có thể gán thoải mái giá trị, nên trong phép A xor B các giá trị đó có thể là 0 đến 2^(63 - i) - 1. Gọi B' là số mà ta đã xây i bit, các số (A xor B) ta có thể tạo được nằm trong khoảng [(A xor B'), (A xor B') + 2^(63-i)). Ta cần kiểm tra khoảng này có chứa số nào chia hết cho N không:

- Nếu 2^(63-i) >= N, hiển nhiên tồn tại 1 số chia hết cho N.
- Nếu (A xor B') % N == 0, hiển nhiên.
- Nếu (A xor B') % N > ((A xor B') + 2^(63-i) - 1) % N, khoảng đi qua 0, tồn tại số chia hết cho N.
- Các trường hợp còn lại không tồn tại.

Việc kiểm tra hoàn toàn có thể xử lí trong O(1). Như vậy độ phức tạp sẽ là O(log maxB).

D. Nasa

Tóm tắt đề bài

Bạn cần cắt một đoạn N phần thành các đoạn con đánh số từ 1 đến X (X không cho trước) từ trái sang phải thỏa mãn:

- Mỗi đoạn có độ dài trong khoảng [P, Q].
- Có M yêu cầu có dạng: phần thứ Ai phải nằm trong đoạn thứ Bi. Bạn cần thỏa mãn tất cả các yêu cầu này.
- Trong tất cả các cách chia thỏa mãn, tìm cách chia sao cho dãy đánh số của các phần có thứ tự từ điển nhỏ nhất.

Solution

Gọi R[i] = vị trí lớn nhất trên dãy mà sao ta có thể điền các số từ 1 đến i. Cụ thể hơn R[i] = p tức là có thể điền các số từ 1 đến i vào vị trí từ 1 đến p trên dãy.
Tương tự ta có L[i] là vị trí bé nhất.
Nhận xét : với mỗi i, nếu L[i] <= p <= R[i] thì có thể điền các số từ 1 đến i vào vị trí từ 1 đến p.
Có thể chứng minh bằng quy nạp hoặc tham lam.
Từ đó, có thể quy hoạch động để tính mảng L và R.

- Nếu i chưa được xuất hiện trong dãy, thì dễ thấy L[i] = L[i - 1] + p và R[i] = R[i - 1] + q.
- Nếu i đã được xuất hiện trong dãy, giả sử u là vị trí đầu tiên i xuất hiện, và v là vị trí cuối cùng. Hiển nhiên từ u đến v phải được fill toàn số có giá trị i. Để tạo ra L[i], ta sẽ fill từ L[i - 1] + 1 đến u toàn giá trị là i. Thế nhưng nếu từ L[i - 1] + 1 đến v lớn hơn q thì ta chỉ fill từ v - p + 1 đến v toàn i thôi. Sau khi fill xong nếu như từ L[i - 1] + 1 đến v vẫn còn chưa đủ p thằng i, thì ta sẽ fill tiếp cho đủ p thằng để được L[i].
- Tương tự tính mảng R.

Việc tính mảng L có thể làm như code sau:

```
u = appearance[i][0];
v = appearance[i].back(); /// vị trí đầu tiên và cuối cùng của i
if (L[i - 1] < u)
    u = L[i - 1] + 1; // fill từ L[i - 1] + 1 đến u toàn giá trị i.
len = v - u + 1;
if (len < p) len = p; /// nếu còn thiếu thì fill cho đủ p
v = max(v, u + len - 1);
L[i] = v;
```

Vì đề bài giả sử luôn luôn tồn tại kết quả, nên mảng L và R luôn luôn tính được.
Khi đã có 2 mảng L và R, công việc còn lại chỉ là truy vết.
Tìm số M lớn nhất sao cho L[M] <= n <= R[M].

Đầu tiên fill p ô cuối cùng bằng M. Nếu như từ R[M - 1] < n - p + 1 thì ta sẽ fill tiếp để cho đủ R[M - 1] + 1. Tương tự làm với các giá trị M - 1, … 1

E. Determinant of the matrix

Tóm tắt đề bài

Cho 2 số N và K. Tìm ma trận N x N thỏa mãn:

- Các số của ma trận nằm trong khoảng [1, K - 1]
- Định thức của ma trận là K.

Solution

Để có thể giải được bài này, bạn cần có một số hiểu biết về định nghĩa cũng như tính chất của định thức trong ma trận.
Bài có nhiều hướng giải, mình sẽ trình bày hướng giải của team mình.

Trước tiên, nếu K = 2, ta in NO vì định thức ma trận toàn 1 luôn là 0. Còn lại, ta luôn có cách dựng thỏa mãn trong O(N^2).
Cách làm của team mình dựa trên 3 tính chất sau của định thức:

- Nếu ta lấy một hàng cộng hoặc trừ vào một hàng khác trong ma trận, ta được ma trận mới có định thức không đổi.
- Nếu ta đổi chỗ 2 hàng, ta được ma trận mới có định thức là đối của định thức cũ.
- Nếu ta có ma trận mà phần trên hoặc dưới đường chéo chính là 0, thì định thức là tích của các số trên đường chéo chính.

Xét K chẵn

Ý tưởng khi K chẵn là tạo ra một ma trận đường chéo trong đó 2 phần tử đầu của đường chéo chính là 2 và K/2, các phần tử còn lại nằm trên đường chéo chính là 1, các phần tử phía trên đường chéo chính là 1 và dưới là 0. Sau đó, ta biến đổi ma trận theo tính chất 1 để được ma trận thỏa mãn.

Giả sử N = 5, K = 6, ta có ma trận:

```
2 1 1 1 1
0 3 1 1 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
```

Ta thực hiện cộng hàng 1 vào tất cả các hàng còn lại, ra ma trận như sau:

```
2 1 1 1 1
2 4 2 2 2
2 1 2 2 2
```

```
2 1 1 2 2
2 1 1 1 2
```

Phần tử lớn nhất là (K / 2 + 1) < K vì K >= 4. Ma trận của ta thỏa mãn tính chất.

Xét K lẻ

Ý tưởng khi K lẻ là tìm cách nào đấy để tạo ra số K trong đường chéo. Trong giờ bọn mình nghĩ ra rằng tạo ra K thì khó, nhưng -K thì dễ: -K = 1 - 2((K + 1) / 2). Khi đó nếu đường chéo có 1 số -K và N - 1 số 1, định thức sẽ là -K. Để đảo dấu lại, ta chỉ cần đảo chỗ 2 hàng là được!

Ví dụ, xét N = 5, K = 7. Để tạo ra ma trận định thức -K, ta tạo ra ma trận như sau:

```
1 4 1 1 1 // 1 (K + 1)/2 1 1 .. 1
2 1 2 2 2 // 2 1 2 2 2 .. 2
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
```

Trong đó 4 = (K + 1) / 2. Ma trận này có thể biến đổi thành

```
1  4  1  1  1
0 -7  0  0  0
0  0  1  1  1
0  0  0  1  1
0  0  0  0  1
```

Bằng cách trừ 2 lần hàng 1 vào hàng 2. Dễ dàng nhận thấy ma trận này có định thức -7. Để có ma trận định thức 7, ta đổi chỗ 2 hàng đầu trong ma trận ban đầu:

```
2 1 2 2 2
1 4 1 1 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
```

Sau đó, thực hiện như K chẵn để biến đổi ma trận cho thỏa mãn điều kiện số. Ta cộng tất cả các hàng >2 một lần hàng 2.

```
2 1 2 2 2
1 4 1 1 1
1 4 2 2 2
1 4 1 2 2
1 4 1 1 2
```

Dễ dàng nhận thấy số lớn nhất là (K + 1) / 2 < K vì K >= 3. Ma trận của ta thỏa mãn.

F. Two-person game

Tóm tắt đề bài

Cho một đồ thị N đỉnh M cạnh. 2 người chơi lần lượt thêm các cạnh chưa tồn tại. Ai làm đồ thị liên thông thì chiến thắng. Tìm kết quả khi 2 người chơi tối ưu.

Solution

Trong mỗi lượt, người chơi có thể đi 1 trong 2 nước:

- Nối 2 đỉnh trong một thành phần liên thông, nếu nó chưa có cạnh nối.
- Nối 2 đỉnh trong 2 thành phần liên thông, nối chúng lại thành một.

Từ đây ta có thể có thuật toán mô phỏng trạng thái: với từng TPLT lưu số lượng cạnh và đỉnh (để tính lại số cạnh khi ghép) của TPLT đó với độ phức tạp rất lớn.

Tuy nhiên ta có thể nhận thấy:

- Với mỗi TPLT, thực chất ta chỉ cần quan tâm đến tính chẵn lẻ của số cạnh còn lại. Thật vậy, nếu số cạnh còn lại là chẵn và lớn hơn 0, nếu A thêm 1 cạnh, B luôn có cách đưa số cạnh trở về trạng thái chẵn. Như vậy trạng thái chẵn > 0 và chẵn = 0 là như nhau. Tương tự, TH lẻ luôn có thể đi sang TH chẵn.
- Khi đó, ta không cần lưu số lượng đỉnh của từng TPLT nữa, mà chỉ cần biết tính chẵn lẻ của nó.

2 nhận xét này giúp ta đưa trạng thái về chỉ còn N^4:

- Số TPLT chẵn đỉnh lẻ cạnh
- Số TPLT chẵn đỉnh chẵn cạnh
- Số TPLT lẻ đỉnh lẻ cạnh
- Số TPLT lẻ đỉnh chẵn cạnh

Từ đây ta chỉ cần thực hiện QHD kiểu đệ quy có nhớ và trả lời từng truy vấn (lưu ý số trạng thái đi đến được là không lớn, và đáp số từng trạng thái không phụ thuộc vào đồ thị input). Độ phức tạp là O(N^4 / niềm_tin).

Ta có thể nhận xét thêm để bỏ bớt 1 chiều chẵn lẻ, hạ ĐPT xuống O(N^2), xin nhường việc này cho các bạn đọc.

G. Overloaded Ants

Tóm tắt đề bài

Cho dãy số W[N] và A[N], các A[i] ban đầu bằng 0. Thực hiện M truy vấn.

- Xoay đoạn [l..r] của A[] và W[] đi x đơn vị về bên trái. Phép xoay 1 đơn vị về bên trái xoay đoạn [l..r] thành [l + 1..r, l].
- Cộng vào đoạn [l..r] một lượng (x, y): A[i] += x * (i - l) + y (x, y >= 0). Bạn phải tìm số phần tử A[i] <= W[i] trước truy vấn và > W[i] sau truy vấn.

Solution

Chưa đảm bảo đây là thuật chuẩn (vì mình chưa AC), nhưng ý tưởng là:

- Chặt nhị phân song song tất cả các phần từ 1..N, tính xem đến truy vấn nào thì chúng bị sai.
- Ta cần giải quyết bài toán, giải các truy vấn 1 và tính W[i] - A[i] ở một thời điểm nào đó. Để xử lí việc này ta có thể sử dụng cây cân bằng có khả năng cắt ghép như Splay Tree / Treap / … để làm trong O(N log N).

Độ phức tạp sẽ là O(N log^2 N).

H. Schedule the football tournament

Tóm tắt đề bài

Cho 2N đội bóng và K cặp đội bóng “yêu thích”. Tìm cách xây dựng lịch thi đấu trong 2N - 1 ngày thỏa mãn:

- Mỗi đội phải được đá với tất cả 2N - 1 đội kia.
- Trong ngày cuối cùng có nhiều trận đấu có cặp đội bóng “yêu thích” nhất.

Solution

Bài toán này sẽ gồm 2 phần:

- Dựng ra một cấu hình lịch thi đấu.
- Hoán vị lại các đội bóng để có nhiều cặp yêu thích trong ngày cuối nhất.

Dựng cấu hình

Đây là một bài toán cổ điển. Có rất nhiều lời giải, ở đây mình chỉ chỉ ra cách mà team mình sử dụng, đpt O(N^2).
Đầu tiên, viết các số từ 1 đến 2N vào bảng 2 x N.

1	2	3	4
5	6	7	8

Đây sẽ là cấu hình của ngày đầu tiên, mỗi cột là một trận đấu. Sau mỗi ngày, ta “xoay” tập các số từ 2 đến 2N, theo ví dụ:

1	3	4	8
2	5	6	7

Ngày 2

1	4	8	7
3	2	5	6

Ngày 3, và vân vân. Ta có thể chứng minh với 2N - 1 phép xoay ta sẽ có lịch thi đấu thỏa mãn cho 2N - 1 ngày.

Tìm cấu hình ngày cuối

Nếu ta coi mỗi đội là một đỉnh, và một cặp yêu thích là một cạnh, thì bài toán sẽ trở thành một bài tìm cặp ghép cực đại trên đồ thị tổng quát. Đây cũng là một bài toán cổ điển, có thể giải bằng thuật toán Edmonds/Blossom trong O(K*sqrt(N)). Khuyến khích các team ACM có thuật toán này trong notebook vì đây là bài toán khá hay xuất hiện trong các kì thi.

Khi đã tìm được cặp ghép cực đại, ta đặt các cạnh được chọn vào các cột đầu tiên, rồi điền bừa các đội còn lại vào các cột còn lại. Khi đó ta được một hoán vị các đội, ta chỉ cần in ra lịch thi đấu đã tìm được ở phía trên theo hoán vị này là được.

Độ phức tạp là O(N^2 + Ksqrt(N)).

I. Graph representation

Tóm tắt đề bài

Cho 2 ma trận A[N x 1] và B[1 x N] gồm các số {0, 1}. Xét đồ thị có mảng cạnh là A x B. Tìm số thành phần liên thông.

Solution

Từ phép nhân ma trận, ta có thể suy ra các tính chất sau:

- Nếu A[i] = 1, đỉnh i có cạnh đến mọi đỉnh j thỏa mãn B[j] = 1.
- Nếu B[i] = 1, đỉnh i có cạnh đến mọi đỉnh j thỏa mãn A[j] = 1.
- Nếu A[i] = B[i] = 0, đỉnh i không có cạnh.

Hiển nhiên mỗi đỉnh i loại 3 là một thành phần liên thông. Ta chỉ cần xét các đỉnh còn lại.

Ta có các trường hợp sau:

- Nếu tồn tại i thỏa mãn A[i] = B[i] = 1, đỉnh i có cạnh nối đến mọi đỉnh đang xét. Đáp số sẽ là [số đỉnh loại 3] + 1.
- Nếu tồn tại i, j thỏa mãn A[i] = B[j] = 1, đỉnh i có cạnh nối đến mọi đỉnh B[x] = 1, j có cạnh nối đến A[y] = 1, như vậy các đỉnh đang xét liên thông. Đáp số như trường hợp trên.
- Nếu 2 TH trên đều không thỏa mãn, A = {0} hoặc B = {0}, khi đó các đỉnh đang xét đều có bậc 0. Đáp số là N.

Độ phức tạp là O(N).

J. Replacement roads

Tóm tắt đề bài

Cho đồ thị có trọng số không âm N đỉnh M cạnh (các cạnh đánh số từ 1 đến M). Ta định nghĩa:

- f(x, y) là đường đi ngắn nhất từ đỉnh x đến đỉnh y.
- F(x, y, i) là đường đi ngắn nhất từ đỉnh x đến đỉnh y **không đi qua cạnh i**.

Tìm giá trị lớn nhất của F(x, y, i) / f(x, y) với mọi x, y, i.

N <= 200, M <= 2000

Solution

[Đây là solution intended(?) của tác giả. Team mình không giải được bài này trong giờ QQ]

Ý tưởng: với mỗi cạnh i tìm F(x, y, i) / f(x, y) lớn nhất.

Nhận xét: khi xóa cạnh (u, v), đường đi có tỉ lệ thay đổi lớn nhất sẽ là u -> x hoặc v -> x với x nào đó.
Thật vậy: Xét đường đi từ a đến b trong đó f(a, b) đi qua cạnh i = (u, v) (theo chiều u->v). Khi đó f(a, b) = f(a, u) + (u, v) + f(v, b) = f(a, u) + f(u, b).
Ta có F(a, b, i) >= F(a, u, i) + F(u, b, i) = f(a, u) + F(u, b, i) (do f(a, u) không đi qua i).
Như vậy:
f(a, b) / F(a, b, i)
<= (f(a, u) + f(u, b)) / (f(a, u) + F(u, b, i)) mà ta lại có phân số >= 1, f(a, u) >= 0
<= f(u, b) / F(u, b, i)

Như vậy, đầu tiên ta tính f(x, y) với mọi x, y (floyd O(n^3)), sau đó với mỗi cạnh (u, v) ta tính F(u, x, i) và F(v, x, i) (dijkstra O(M log M)).

Độ phức tạp của thuật toán là O(N^3 + M^2logM).

K. Pairing

Tóm tắt đề bài

Cho 2 công ty, với mỗi công ty ta biết số người muốn ghép cặp với người cùng công ty và số người muốn ghép cặp với công ty còn lại. Tìm cách ghép cặp mà nhiều người thỏa mãn nhất.

Solution

Ta định nghĩa A, B, C, D như đề bài. Lần lượt for các số lượng:

- a: nhóm A->C
- b: nhóm A->D
- c: nhóm B->C
- d: nhóm B->D

Giờ chỉ còn lại việc ghép nhóm A với A/B, C với C/D. Để tối ưu, ta ghép tất cả nhóm A với nhau, nếu còn thừa thì dùng nhóm B. Tương tự với C và D.

Ta có thể tính được việc nhóm này trong O(1), nên độ phức tạp bài này là O(n^4).