Lời giải VOI 2021

Tổng quan đề thi

Dạng các bài vv, đánh giá độ khó

Bài 1: Dãy con tăng

Bài 2: Knapsack trên cây

Bài 3: Sum-of-subset + Bao hàm loại trừ

Bài 4: Trie + chia căn

Bài 5: Quy hoạch động

Bài 6: Đồ thị + Casework + cấu trúc dữ liệu cơ bản (nhưng lại không cơ bản)

Bài 1 – NOEL

Đề bài

Cho dãy a[1], a[2], ..., a[n] là một **hoán vị**.

Tìm dãy con b của a gồm 2M số dài nhất sao cho abs(b[i] - b[i + M]) <= D.

Giới hạn:

1 <= N <= 1000

1 <= D <= 5

Quy hoạch động cơ bản?

F[i][j] là dãy con dài nhất sao cho 2 b[k] và b[k + M] vừa chọn là a[i] và a[j].

Với mỗi i chỉ có 2 * D số j thỏa mãn.

F[i][j] = max(F[u][v] + 1) sao cho u < i, v < j

Độ phức tạp O(N ^ 4)

Để giảm xuống O(N ^ 2) ta có thể thêm một bước tính prefix max đơn giản cho mảng F:

G[i][j] = max(F[i][j], G[i - 1][j], G[i][j - 1])

Đến đây ta có F[i][j] = G[i - 1][j - 1] + 1

The answer is NO

Xét dãy a =[1. 2. 3], D = 1

Ta có

f(1, 2) = 1

f(2, 3) = f(1, 2) + 1 = 2.

Kết quả này hết sức vô lý, do 2 * M > N.

Lý do thuật toán này là ngộ nhận: với định nghĩa và công thức f(i, j) đã có, thuật toán đã vô hình chung coi như mình đang giải bài toán trên hai dãy A phân biệt chứ không phải một dãy. Do đó một chỉ số có thể được sử dụng hai lần. (a[2] = 2)

Then how?

Để tránh việc một số được sử dụng hai lần (một lần làm b[k] và một lần làm b[k + M), thì cách đơn giản nhất là ta đặt một “ranh giới" giữa b[M] và b[M + 1].

↠ **Solution**: Duyệt qua tất cả các ranh giới

Subtask 1:

N <= 10.

Yêu cầu bài toán**:** Chọn ra **một dãy con 2M phần tử** thỏa mãn...

Thuật toán sinh nhị phân: O(2 ^ N * N), cần đảm bảo số lượng số được chọn là chẵn

Subtask 2:

N <= 100

Sau khi chọn ranh giới X, ta hoàn toàn có thể sử dụng công thức F[i][j] như đã định nghĩa trong thuật toán quy hoạch động đã mô tả.

Bằng cách đảm bảo i <= X và j > X với mọi cặp (i, j) trong công thức, ta có một thuật toán chuẩn với độ phức tạp **O(N^3)**

The Final Boss

N <= 1000

Đến đây, ta cần phân tích xem vì sao lời giải subtask 2 lại chạy trong O(N ^ 3).

Đối với thuật toán quy hoạch động (làm N lần cho N ranh giới):

- Chi phí chuyển: O(N * D)
- **Chi phí xây mảng G: O(N ^ 2)**

O(N * D) là một độ phức tạp nhỏ chấp nhận được, nhưng O(N ^ 2) sẽ không hiệu quả cho subtask cuối

The Final Boss

N <= 1000

Nếu nhìn bài toán theo một hướng khác, sau khi đặt ranh giới X, mỗi chỉ số i phần bên trái có 2 * D chỉ số j có thể ghép với nó.

Cần chọn một vài chỉ số i và **ghép nó với các chỉ số j** sao cho từ trái sang phải các chỉ số **j phải tăng dần.**

↠ **Bài toán LIS**

Lời giải: Thay vì duy trì mảng G trong O(N ^ 2), ta sẽ giảm G xuống 1 chiều duy nhất là G[j], và dựng fenwick tree, interval tree của mảng G để truy vấn và cập nhật trong O(log)

The Final Boss

N <= 1000

Độ phức tạp: O(N ^ 2 * D * log)

Funfact: Ngay khi đề ngày 1 vừa được upload thì đã có >= 5 admin VNOI tham gia giải đề. Và có 2 admin tên T, Q rơi vào thuật toán ngộ nhận và **một admin họ Lê** không có ý tưởng gì cho bài toán.

Bài 2 - COMNET

Đề bài

Có một cây gồm N đỉnh, cần chọn ra K đỉnh sao cho số cạnh ít nhất cần dùng để làm K đỉnh này liên thông nằm trong đoạn từ L tới R.

- Subtask1 (20%): N <= 100, K = 2
- Subtask2 (30%): N <= 100, K = 3
- Subtask3 (20%): N <= 100, K = 4
- Subtask4 (20%): N <= 1000, K = 3
- Subtask5 (10%): N <= 1000, K = 4

Subtask 1 (20%): N <= 100, K = 2

Với K = 2, số cạnh ít nhất cần để nối 2 đỉnh chính là khoảng cách của 2 đỉnh đó.

Ta có thể duyệt 2 đỉnh sau đó dùng thuật toán BFS/DFS để tính khoảng cách của 2 đỉnh.

Độ phức tạp sẽ là O(N^3), ta mất O(N^2) cho việc duyệt 2 đỉnh, và O(n) cho việc BFS/DFS.

Ta cũng có thể làm subtask này trong O(N^2).

Subtask 2 (30%): N <= 100, K = 3

Thay vì duyệt 2 đỉnh như subtask 1, ta có thể duyệt cả 3 đỉnh với độ phức tạp O(N^3), tính số cạnh cần dùng trong O(N) bằng DFS. Tổng độ phức tạp sẽ là O(N^4), sẽ khoảng 10^8 phép tính, vẫn đủ an toàn để chạy trong 1 giây.

Cách tính số cạnh cần dùng bằng DFS, tạm gọi 3 đỉnh ta đang duyệt là a, b, c:

- Ta xem gốc của cây là a
- Một đỉnh u được gọi là quan trọng nếu như trong cây con gốc u có chứa đỉnh b hoặc c.
- Số cạnh cần dùng chính bằng số đỉnh quan trọng trừ đi 1.

Subtask 2 (30%): N <= 100, K = 3

bool DFS(u):

    nếu u = b hoặc u = c thì u là đỉnh quan trọng;

    duyệt v kề với đỉnh u:

        Nếu DFS(v) đúng thì ta suy ra u là đỉnh quan trọng;

    return u có là đỉnh quan trọng hay không;

gọi DFS(a);

Subtask 2 (30%): N <= 100, K = 3

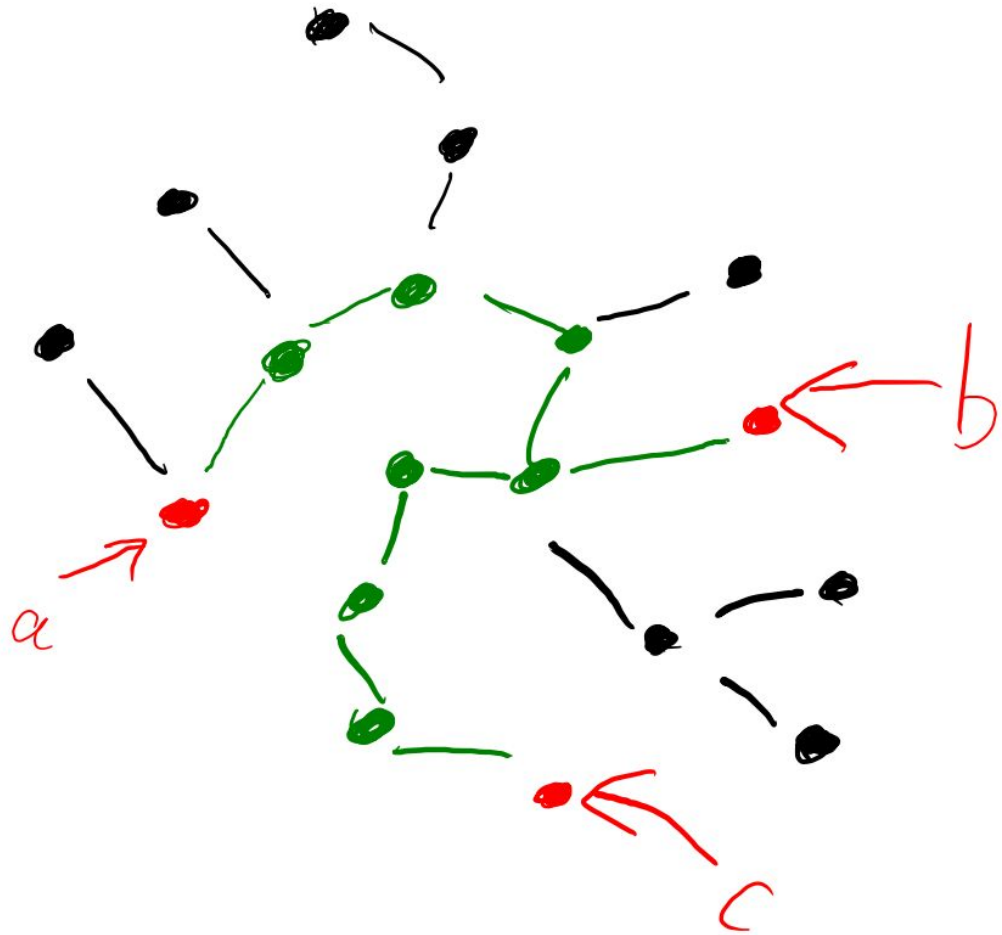
Có cách tính số đỉnh quan trọng trong O(1):

//số đỉnh quan trọng = (dist(a, b) + dist(b, c) + dist(a, c)) / 2

Khởi tạo trước mảng dist(i, j) trong O(N^2) bằng N lần BFS/DFS;

Subtask 3 (20%): N <= 100, K = 4

Nếu làm giống subtask 2, duyệt qua 4 đỉnh sau đó mới tính DFS để tính thì độ phức tạp sẽ lên tới O(N^5)!!!



Chú thính:
Xanh lá: đỉnh quan trọng
Đỏ: đỉnh a, b, c (lưu ý a b c cũng là đỉnh quan trọng, mình chỉ tô khác cho dễ nhìn thôi)
Đen: các đỉnh khác

Việc chọn thêm 1 đỉnh thứ 4, sẽ làm tăng số cạnh lên đúng bằng số cạnh đen
Số cạnh đen chính là số cạnh trên đường đi ngắn nhất từ đỉnh đó tới các đỉnh xanh/đỏ

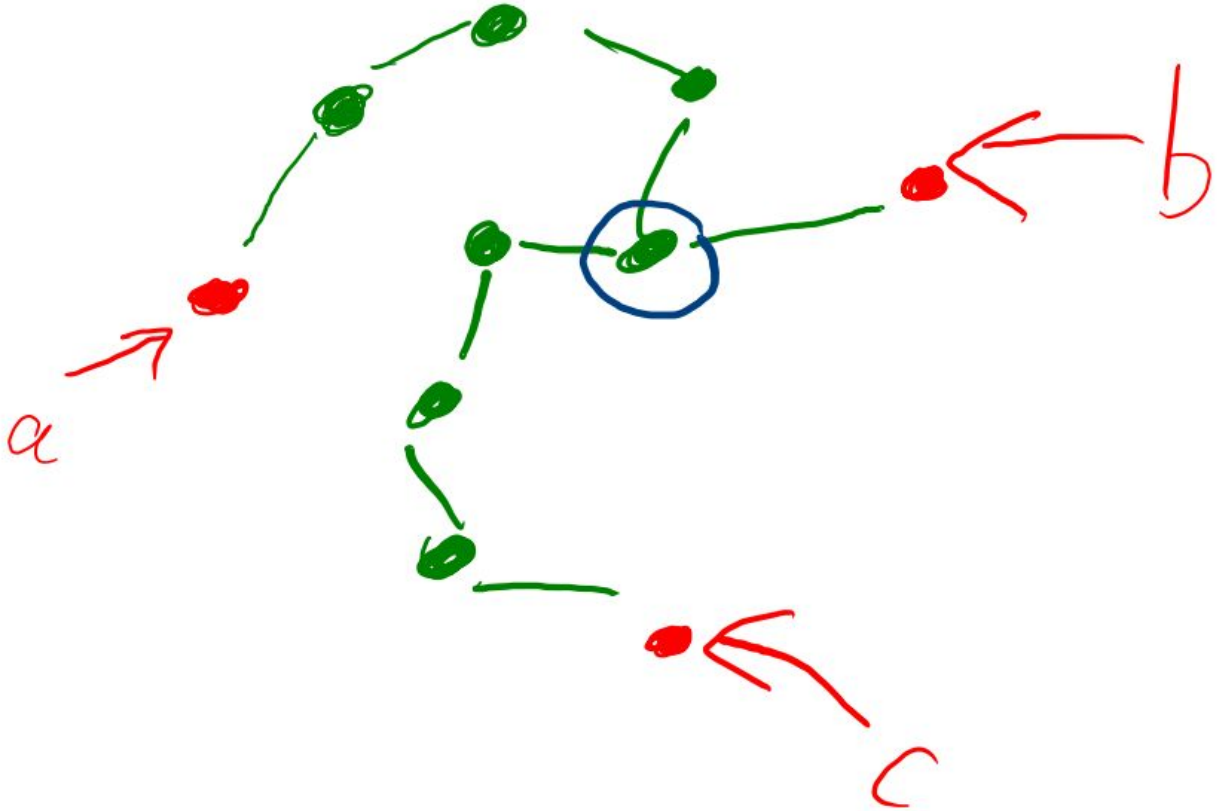
Subtask 3 (20%): N <= 100, K = 4

Tới đây ta có thể làm như sau:

- Duyệt qua 3 đỉnh a, b, c.
- Tìm ra các đỉnh quan trọng như subtask 2.
- Tính G(u) = khoảng cách từ đỉnh u tới một trong các đỉnh quan trọng. Ta có thể tính mảng này bằng cách đẩy hết các đỉnh quan trọng vào queue xong BFS (BFS nhiều nguồn). Độ phức tạp phần này là O(n).
- Sau đó ta duyệt qua đỉnh thứ 4 để tính kết quả.

Độ phức tạp: O(N^4).

Subtask 4 (20%): N <= 1000, K = 3



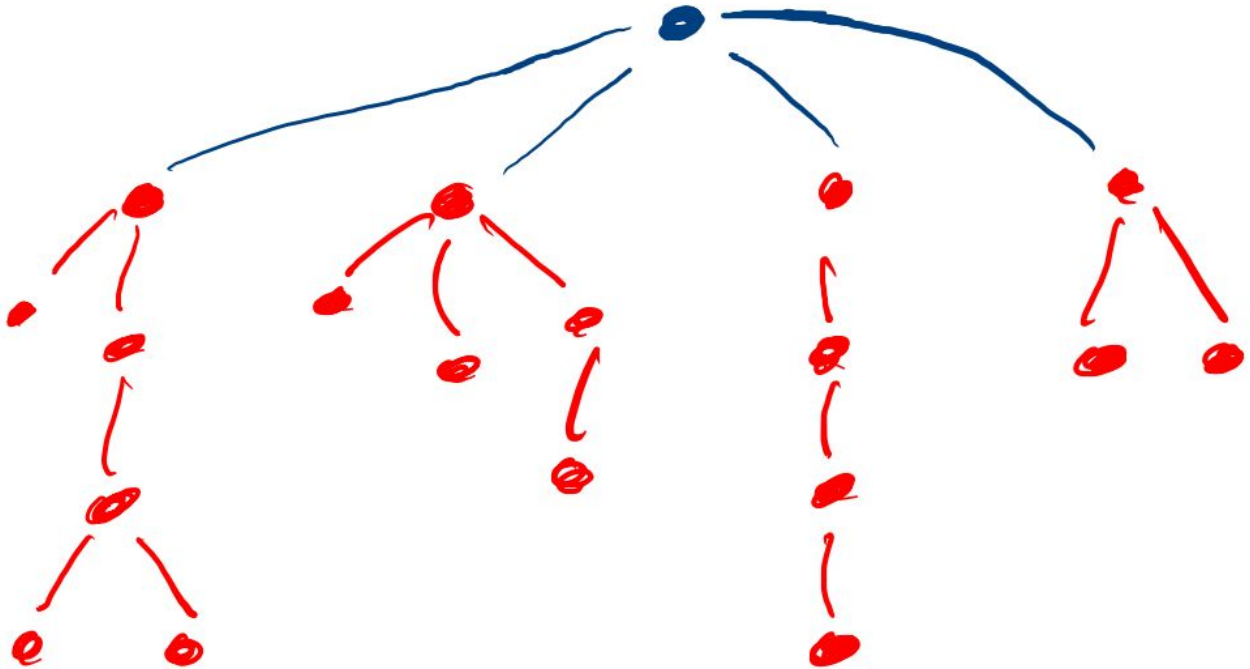
Với 3 đỉnh a, b, c, sẽ có 1 và duy nhất 1 đỉnh thỏa mãn nếu cắt đỉnh đó đi khỏi cây thì a, b, c sẽ nằm ở 3 thành phần liên thông khác nhau.

Tạm gọi đỉnh đó là đỉnh cắt.

Lưu ý đỉnh cắt cũng có thể nằm trong 3 đỉnh a, b, c

Subtask 4 (20%): N <= 1000, K = 3

Từ nhận xét trên ta có thể nghĩ tới việc duyệt qua điểm cắt, sau đó đếm số cách chọn 3 đỉnh a, b, c nằm ở 3 cây con khác nhau sao cho tổng khoảng cách tới đỉnh cắt nằm trong đoạn L, R.



Subtask 4 (20%): N <= 1000, K = 3

Tới đây bài toán quy về như sau:

Ta có M vector V1 V2 V3 … VM các số (với M chính là số con của đỉnh cắt)

Vector V[i] lưu các khoảng cách (số cạnh) từ các đỉnh trong cây con i đi tới đỉnh cắt.

Ta cần chọn ra 3 số trong 3 vector Va Vb Vc khác nhau sao cho tổng của 3 số này nằm trong đoạn L R.

Với bài toán này ta có thể giải bằng Quy Hoạch Động cái túi.

Gọi F[i][sum][k] là số cách chọn ra k số từ các vector V1 V2 … Vi, với tổng các số là sum.

Subtask 4 (20%): N <= 1000, K = 3

Gọi F[i][sum][k] là số cách chọn ra k số từ các vector V1 V2 … Vi, với tổng các số là sum.

Cách tính F[i][sum][k]:

Ta duyệt qua từng số trong vector i, tạm gọi số đó có giá trị là x:

- Nếu ta chọn x: F[i][sum][k] += F[i - 1][sum - x][k - 1];
- Nếu ta không chọn x: F[i][sum][k] += F[i - 1][sum][k];

Khởi tạo F[0][0][0] = 1;

Và F[0][0][1] = 1; (Vì ta có thể chọn điểm cắt cũng là 1 số).

Kết quả chính là tổng của F[M][sum][K] với sum duyệt từ L tới R.

Subtask 4 (20%): N <= 1000, K = 3

Chung quy lại thuật toán của ta sẽ qua các bước như sau:

- Duyệt qua các đỉnh cắt
- Với mỗi đỉnh cắt thì tính lại hàm quy hoạch động F[i][sum][k]. Trên thực tế, khi cài đặt ta có thể chỉ dùng mảng 2 chiều để tiết kiệm bộ nhớ.

Để đánh giá độ phức tạp của thuật toán này, nếu chỉ nhìn vào các vòng for mà đánh giá thì độ phức tạp của nó sẽ là O(N^3 * K) hoặc thậm chí là O(N^4 * K).

Tuy nhiên, nếu cài đặt tối ưu, duyệt các trạng thái quy hoạch động một cách hợp lý thì độ phức tạp của toàn bộ thuật toán này chỉ là **O(N^2 * K). (.O.)**

Phần chứng minh độ phức tạp mình sẽ dẫn chứng ở subtask 5.

Subtask 5 (10%): N <= 1000, K = 4

Với 4 đỉnh thì ta không còn đỉnh cắt nữa, nhưng từ subtask 4 nó cho ta 1 hướng đi mới đó là dùng Quy hoạch động để đếm số cách chọn 3 đỉnh, tiếp tục theo hướng nghĩ này thì ta sẽ dùng thuật toán Quy hoạch động cái túi trên cây (Knapsack on tree) để làm. Kiến thức này đã được cho ra trong bài 6 năm ngoái.

Hàm qhđ có thể làm như sau: F[u][sum][k] chính là số cách chọn k đỉnh trong cây con gốc u, sao cho số lượng cạnh để k đỉnh đó **liên thông với đỉnh u** là sum.

Để tính hàm F cho từng đỉnh u thì ta có thể làm giống như ở subtask 4, nhưng lưu ý rằng mỗi subtree bây giờ ta có thể chọn nhiều hơn 1 đỉnh.

Subtask 5 (10%): N <= 1000, K = 4

Gọi G[u][i][sum][k] là số cách chọn k đỉnh trong cây con gốc u khi xét các cây con V1 … Vi, số lượng cạnh cần dùng là sum.

Với mỗi cây con v thứ i của u:

G[u][i][sum][k] += G[u][i - 1][**sum - (cntEdge_v + 1)**][k - k_v] * F[v][cntEdge_v][k_v];

Lưu ý ở đây nếu toàn bộ cây con gốc v dùng **cntEdge_v** cạnh thì để nối nó lên đỉnh u ta sẽ mất thêm 1 cạnh nữa (để nối từ v tới u).

Cần cẩn thận trường hợp **k_v = 0** (ta không chọn đỉnh nào từ cây con gốc v).

Trong thực tế, ta không cài đặt hàm G mà ta sẽ tính trực tiếp trên mảng F để tiếp kiệm thời gian và bộ nhớ.

Subtask 5 (10%): N <= 1000, K = 4

Thuật toán trên có độ phức tạp O(N^2 * K^2), dù nghe rất vô lý.

Các bạn có thể tham khảo phần chứng minh ở comment này:
https://codeforces.com/blog/entry/52742?#comment-367974

Bài 3 - OR

Đề bài

Cho một dãy a[1], a[2], ..., a[N].

Đếm số cách chọn một dãy con b của a có K phần tử sao cho:

v = b[1] or b[2] or ... or b[k]

L <= v <= R

v mod 3 = 0

Các subtask cuối của bài tập này cần khá nhiều kiến thức.

Subtask 1

N <= 20

Tại subtask này ta hoàn toàn có thể duyệt nhị phân tìm các dãy con có K phần tử và kiểm tra điều kiện của tổng OR.

Độ phức tạp: O(2 ^ N * N)

Subtask 2

N <= 200, a[i] <= 200

Để ý rằng sub này N và a[i] khá nhỏ, đủ nhỏ để chúng ta có thể đưa vào trạng thái của một hàm quy hoạch động.

Gọi f[i][j][v] là số cách chọn ra j số từ i số đầu tiên trong dãy và đạt tổng or là v

Chuyển trạng thái:

- không chọn a[i + 1]: f[i + 1][j][v] += f[i][j][v]
- chọn a[i + 1]: f[i + 1][j + 1][v or a[i + 1]] += f[i][j][v]

Subtask 3

N <= 1000 000, a[i] <= 2^k, L = R

Do L = R nên v chỉ có duy nhất một giá trị cuối.

A[i] đều là lũy thừa của 2.

Do đó chỉ cần chọn k số sao cho bit 1 nào của v cũng tồn tại trong các số đã chọn.

Subtask 3: Vậy làm sao để đảm bảo nó or = V

Nếu ta tập hợp được số bit là con của V = count thì số cách chọn ra k số từ các số này là C(count, k) (tổ hợp).

Nhưng tất nhiên là tổ hợp không thể đảm bảo tổng or các số đã chọn = V.

Tới đây, rất đơn giản là ta chỉ cần trừ đi những trường hợp mà tổng or ra khác V.

Phạm trù kiến thức mới: **bao hàm bù trừ**

Bao hàm loại trừ - VNOI

Subtask 3: Bao hàm bù trừ

Gọi count(j) là số lượng số a[i] là một bit con của j.

Gọi f(j) = C(count(j), k).

answer(v) = tổng sign * f(j)

Với:

j là một **submask** của v (tính cả v)

sign = 1 nếu **v và j cùng tính chẵn lẻ về số bit**, ngược lại là -1

Subtask 4

n <= 1e6, a[i] <= 1e6, L = R

tới đây việc tính count(j) không còn có thể thực hiện trong log bằng cách for bit nữa.

Một kỹ thuật tiếp theo trong bài này là: Quy hoạch động sum of subset (DP SOS).

Với kỹ thuật này, ta có thể tính được mảng count trong tổng cộng O(N log N), qua đó đưa subtask 4 về giống với subtask 3

Subtask 5

n <= 1e6, a[i] <= 1e6, **L** <= **R**

tới đây bài toán không còn chỉ với một số v nữa mà là với rất nhiều số v khác nhau.

Do đó tại bước bao hàm loại trừ cuối cùng, ta không thể for chay j để tính answer(v) với mọi v được.

Giờ bài tập đã yêu cầu chúng ta cần biết về bao hàm bù trừ và dp SOS, vậy kiến thức tiếp theo để có thể giải được subtask 5 là **Bao hàm loại trừ cải tiến bằng dp SOS**

Subtask 5

Đặt f(j) = C(count(j), k) nếu j có lẻ bit và -C(count(j), k) nếu j có chẵn bit

thực hiện tính dp sum of subset cho f ↠ dp

ta có answer(v) = dp(v) nếu v có lẻ bit và ngược lại bằng -dp(v) nếu v có chẵn bit

Đó bài chỉ có mỗi vậy thôi, đề dễ ấy mà

Bài 4 - BONUS

Đề bài

Cho n thẻ xếp trên một hàng, thẻ thứ i có nhãn c_i (là một xâu kí tự) và giá trị p_i.

Thực hiện các bước sau:

- Chọn một thẻ bất kì.
- Giả sử thẻ vừa được chọn là i, có thể tiếp tục chọn thẻ j (j > i) thỏa mãn: c_j là *tiền tố* của c_i <u>hoặc</u> (p_i, p_j) là một cặp số may mắn. Lặp lại như vậy cho đến khi không tìm được j thỏa mãn.
- Tổng điểm nhận được là tổng các p_i trên các thẻ được chọn.

Tìm cách chọn các thẻ sao cho tổng điểm nhận được là lớn nhất.

Giới hạn

- $1 \le n \le 300\,000$
- $0 \le m \le 300\,000$
- $\Sigma|c_i| \le 1\,000\,000$
- $1 \le p_i \le 100\,000$

- Subtask 1 (40%): $n \le 100$, $m \le 100$, $|c_i| \le 100$
- Subtask 2 (40%): $m = 0$
- Subtask 3 (20%): Không có điều kiện gì thêm

Subtask 1

- $n \leq 100$, $m \leq 100$, $|c_i| \leq 100$.
- Ý tưởng quy hoạch động quen thuộc: dp(i) = điểm thưởng lớn nhất có thể nhận được nếu thẻ cuối cùng được chọn là i.
- $dp(i) = p_i + \max(0, \max\{dp(j) \mid j < i$ và có thể chọn i sau khi chọn j$\})$.
- Duyệt các j từ 1 đến i-1, sau đó kiểm tra c_i có phải tiền tố của c_j hoặc (p_i, p_j) có phải cặp số may mắn hay không.
- Độ phức tạp: $O(n^2 * \max\{|c_i|\})$.

Subtask 2

- m = 0.
- Không cần quan tâm đến điều kiện của p_i; tuy nhiên, không thể duyệt qua tất cả các j từ 1 đến i-1 hay kiểm tra tiền tố trong $O(|c_i|)$.
- Sử dụng cây tiền tố (trie): mỗi đỉnh trên trie thể hiện một xâu w, lưu lại tất cả các thẻ i có w là tiền tố của c_i.
- Với mỗi i,
  - duyệt qua tất cả các thẻ được lưu trong đỉnh thể hiện xâu c_i trên trie để tính dp(i).
  - thêm thẻ i vào các đỉnh thể hiện các tiền tố của c_i.
- Thuật toán này KHÔNG đủ nhanh: xét đầu vào có 150 000 xâu “ab” và 150 000 xâu “a”.

Subtask 2 (tiếp)

- Nhận xét: trong những thẻ được lưu ở một đỉnh trên trie, ta chỉ quan tâm đến thẻ i có giá trị dp(i) lớn nhất.
- Với mỗi đỉnh u thể hiện xâu w, thay vì lưu lại **tất cả** các thẻ i có w là tiền tố của c_i, ta lưu lại B(u) = giá trị dp(i) lớn nhất của các thẻ đó.
- Với mỗi i,
  - tính dp(i) = B(u) + p_i, trong đó u là đỉnh thể hiện xâu c_i.
  - với mỗi đỉnh v thể hiện các tiền tố của c_i, cập nhật B(v) = max(B(v), dp(i)).
- Độ phức tạp: $O(n + \Sigma|c_i|)$.

Subtask 3

- Nhận thấy rằng 2 điều kiện để chọn thẻ i sau thẻ j là độc lập với nhau.
- Sửa lại thuật toán ở subtask 2 như sau: với mỗi i,
  - tính dp(i) = B(u) + p_i, trong đó u là đỉnh thể hiện xâu c_i.
  - **cập nhật lại dp(i) = max(dp(i), dp(j) + p_i) với j < i và (p_i, p_j) là cặp số may mắn.**
  - với mỗi đỉnh v thể hiện các tiền tố của c_i, cập nhật B(v) = max(B(v), dp(i)).
- Cần tìm cách làm bước 2 trong thời gian nhanh hơn O(n).

Subtask 3 (tiếp)

- Giống như subtask 2, nhận xét rằng trong những thẻ có cùng giá trị x, ta chỉ quan tâm đến thẻ i có giá trị dp(i) lớn nhất.
- Với mỗi giá trị x, lưu lại C(x) = giá trị dp(i) lớn nhất sao cho (p_i, x) là cặp số may mắn.
- Với mỗi i,
  - tính dp(i) = **max(B(u), C(p_i)) + p_i**, trong đó u là đỉnh thể hiện xâu c_i.
  - với mỗi đỉnh v thể hiện các tiền tố của c_i, cập nhật B(v) = max(B(v), dp(i)).
  - **với mỗi x sao cho (p_i, x) là cặp số may mắn, cập nhật C(x) = max(C(x), dp(i)).**
- Thuật toán này KHÔNG đủ nhanh: xét đầu vào có $p_i = i + 1$ với $1 \le i < 100\,000$, $p_i = 1$ với $100\,000 \le i \le 300\,000$ và (1, x) là cặp số may mắn với mọi x thỏa mãn $1 < x \le 100\,000$.

Subtask 3 (tiếp)

- Nhận xét: chọn một số nguyên dương T bất kì (chẳng hạn, T = 600). Khi đó, có tối đa 2m/T số xuất hiện trong ít nhất T cặp số may mắn.
- Gọi những giá trị như vậy là giá trị *quan trọng*, những giá trị còn lại là *không quan trọng*.
- Với mỗi i, ta chỉ cập nhật các C(x) khi p_i là giá trị *không quan trọng*.
- Với mỗi giá trị *quan trọng* x, lưu thêm F(x) = dp(i) lớn nhất sao cho p_i = x.

Subtask 3 (tiếp)

- Thuật toán cuối cùng: với mỗi i,
  - tính dp(i) = **max(B(u), C(p_i)) + p_i**, trong đó u là đỉnh thể hiện xâu c_i.
  - **với mỗi giá trị quan trọng x sao cho (p_i, x) là cặp số may mắn, cập nhật dp(i) = max(dp(i), F(x) + p_i).**
  - với mỗi đỉnh v thể hiện các tiền tố của c_i, cập nhật B(v) = max(B(v), dp(i)).
  - **nếu p_i là giá trị không quan trọng**, với mỗi x sao cho (p_i, x) là cặp số may mắn, cập nhật C(x) = max(C(x), dp(i)).
  - **nếu p_i là giá trị quan trọng**, cập nhật F(p_i) = max(F(p_i), dp(i)).
- Độ phức tạp: $O(n(T + m/T) + \Sigma|c_i|) = O(n\sqrt{m} + \Sigma|c_i|)$ (chọn $T = O(\sqrt{m})$).

Bài 5 - RMAT

Đề bài

Cho ma trận A gồm m hàng và n cột.

Có hai phép rút gọn ma trận A:

- Chọn 2 hàng i và i+1 có tổng bằng nhau, gán A(i, j) = A(i, j) + A(i+1, j) và xóa hàng i+1.
- Chọn 2 cột j và j+1 có tổng bằng nhau, gán A(i, j) = A(i, j) + A(i, j+1) và xóa cột j+1.

Tìm cách rút gọn ma trận sao cho có ít phần tử nhất.

Giới hạn

- $1 \le m$
- $1 \le n$
- $|A(i, j)| \le 1\,000\,000$

- Subtask 1 (30%): $m = 1, n \le 10$
- Subtask 2 (20%): $m = 1, n \le 100$
- Subtask 3 (30%): $mn \le 500$
- Subtask 4 (10%): $mn \le 5\,000$
- Subtask 5 (10%): $mn \le 1\,000\,000$

Subtask 1

- $m = 1, n \leq 10$.
- Sử dụng kĩ thuật duyệt quay lui.
- Thử tất cả các cách ghép 2 cột liên tiếp (thực chất là 2 số liên tiếp).
- Độ phức tạp: $O(n!)$.

Subtask 2

- $m = 1$, $n \leq 100$.
- Quy hoạch động dp(i) = số phần tử nhỏ nhất có thể khi rút gọn ma trận gồm các số từ A(1, 1) đến A(1, i).
- dp(0) = 0.
- dp(i) = 1 + min{dp(j) | $0 \leq j < i$ và đoạn A(1, j+1..i) có thể rút gọn còn duy nhất 1 phần tử}.
- Ta cần thêm một hàm QHĐ phụ: check(i, j) = true/false: đoạn A(1, i..j) có thể rút gọn còn duy nhất một phần tử hay không.

Subtask 2 (tiếp)

- check(i, i) = true với mọi $1 \le i \le n$.
- check(i, j) = true nếu như tồn tại $i \le k < j$ sao cho check(i, k) = true, check(k+1, j) = true và sum(i, k) = sum(k+1, j), trong đó sum(i, j) là tổng các phần tử từ A(1, i) đến A(1, j). Nếu không tồn tại k thỏa mãn, check(i, j) = false.
- Tính check(i, j) bằng đệ quy có nhớ hoặc duyệt các cặp (i, j) theo thứ tự j-i tăng dần.
- Sau khi tính được check(i, j), ta có thể tính được dp(i) như trên.
- Độ phức tạp: $O(n^3)$.

Subtask 3

- $mn \leq 500$.
- Nhận xét quan trọng: phép rút gọn hàng **không** làm thay đổi tổng giá trị các cột (và ngược lại)!

1	2	3	4	5	**15**
5	4	3	2	1	**15**
4	-1	-1	-1	2	**3**
10	**5**	**5**	**5**	**8**	

6	6	6	6	6	**30**
4	-1	-1	-1	2	**3**
10	**5**	**5**	**5**	**8**	

Subtask 3 (tiếp)

- Tách bài toán thành 2 bài toán con độc lập (trên dãy số gồm tổng mỗi hàng và dãy số gồm tổng mỗi cột), sau đó sử dụng thuật toán ở subtask 2 để giải.
- Đáp số là tích kết quả của 2 bài toán con.
- Độ phức tạp: $O(m^3 + n^3)$.

Subtask 4

- $mn \leq 5\,000$.
- Liệu có thể bỏ qua bước duyệt k khi tính check(i, j) được không?
- Nhận xét: không thể ghép số âm với số dương, hoặc số 0 với số khác 0.
- Chia dãy số thành các đoạn liên tiếp chỉ bao gồm số dương/số âm/số 0, giải riêng từng đoạn rồi cộng đáp án lại.

Subtask 4 (tiếp)

- Với đoạn gồm toàn số 0: đáp án luôn là 1.
- Với đoạn gồm toàn số dương: tồn tại không quá 1 vị trí k sao cho sum(i, k) = sum(k+1, j) (do sum(i, k) là hàm tăng chặt khi k tăng dần),
  - Sử dụng kĩ thuật tìm kiếm nhị phân để tìm vị trí này.
- Làm tương tự với đoạn gồm toàn số âm (lưu ý điều kiện của bước tìm kiếm nhị phân).
- Độ phức tạp: $O(m^2 * \log(m) + n^2 * \log(n))$.
- Có thể cải tiến bằng cách tính trước vị trí k với mỗi đoạn (i, j) bằng kĩ thuật 2 con trỏ (độ phức tạp $O(m^2 + n^2)$).

Subtask 5

- $mn \leq 1\,000\,000$.
- Nhận xét: số y (y lẻ) chỉ có thể ghép được với các số có dạng $2^x y$.
- Ta chia dãy thành các đoạn sao cho các số đều có dạng $2^x y$ với y là một hằng số lẻ nào đó.
- −3 −12 8 1 1 2 0 0 5 −5 −5 −10 −15
- Với mỗi dãy này, ta có thể phát biểu lại bài toán như sau: cho một dãy số nguyên **không âm**, có thể ghép 2 số x liên tiếp nhau để nhận được số **x+1**, tìm cách ghép để thu được dãy mới gồm ít phần tử nhất.

Subtask 5 (tiếp)

- Quy hoạch động check(i, j, x) = true/false: có thể rút gọn đoạn [i..j] thành số x hay không.
- Nhận xét: với mỗi cặp (i, x) chỉ tồn tại không quá 1 giá trị j thỏa mãn check(i, j, x) = true.
- Có thể quy hoạch động dp(i, x) = j sao cho đoạn [i..j] có thể rút gọn thành số x. Nếu không tồn tại j thỏa mãn, dp(i, x) = -1.
- dp(i, A[i]) = i.
- dp(i, x) = dp(dp(i, x-1) + 1, x-1) nếu dp(i, x-1) ≠ -1.
- Giá trị tối đa của x không vượt quá 38 (đạt được khi dãy gồm 2^{19} số 2^{19}).
- Độ phức tạp: O((n + m) * 38).

Bài 6 - TEAMUP

Đề bài

Cho đồ thị N đỉnh N cạnh (bậc các đỉnh >= 1).

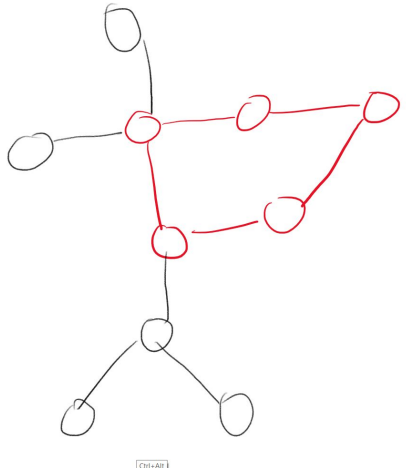
Mỗi cạnh có một trọng số Wi.

Cho M truy vấn **độc lập**, mỗi truy vấn thử thêm 1 cạnh vào đồ thị. Với mỗi truy vấn in ra tổng chi phí nhỏ nhất để xóa các cạnh sao cho phần còn lại là một đồ thị 2 phía.

Kiến thức

Điều kiện để một đồ thị là một đồ thị hai phía đó là không tồn tại chu trình lẻ.

Do đồ thị N cạnh, bậc của mỗi đỉnh >= 1 , nên dồ thị có hình dạng là một rừng các đồ thị mặt trời.



Subtask 1

M = 0, P là một hoán vị

Do P là một hoán vị nên đồ thị được tạo ra bởi N đỉnh sẽ là môt **tập các hình tròn** thay vì một tập các đồ thị mặt trời như trường hợp tổng quát.

Vậy nếu một vòng tròn có lẻ đỉnh, nó sẽ tạo ra một chu trình đơn và lẻ. Cách để xóa cạnh chi phí nhỏ nhất để không tồn tại chu trình lẻ nữa đơn giản là tìm cạnh nhỏ nhất trong chu trình đó và xóa đi,

Đpt: O(N)

Subtask 2

M = 0

Trong subtask này thì đồ thị được cho là một tập các đồ thị mặt trời. Tuy nhiên thì cách giải vẫn là liệt kê các hình tròn ra và tìm cạnh nhỏ nhất cho mỗi hình tròn.

Thay vì chỉ cần for như subtask 1 thì trong subtask này sẽ cần phải dfs

Subtask 3

P là một hoán vị

Đến đây M > o nhưng đồ thị được cho lại là một tập các hình tròn.

Đây là một bài casework. Việc thêm một cạnh và một tập các hình tròn thì có các khả năng sau:

- Hai hình tròn nối với nhau
- Tự một hình tròn nối với chính nó

Subtask 3: Hai hình tròn nối với nhau

Trong trường hợp này, cạnh được thêm vào không tạo ra thêm một chu trình lẻ nào. Do đó đáp án của bài toán được giữ nguyên như không thêm cạnh (Subtask 1).

Subtask 3: Tự hình tròn nối chính nó

Trong trường hợp này lại có hai trường hợp con sau:

- Một hình tròn cắt ra thành 2 hình tròn chẵn
- Một hình tròn cắt ra thành hai hình tròn lẻ

Trong trường hợp đầu tiên: ko có chu trình lẻ được tạo ra, đáp án được giữ nguyên

Trường hợp thứ hai, ta có hai lựa chọn:

- Xóa cạnh vừa thêm vào
- Xóa từ hai nửa hình tròn, mỗi nửa một cạnh để mất hai chu trình lẻ

Trong hai lựa chọn, ta chọn cách cho cost nhỏ hơn.

Subtask 4

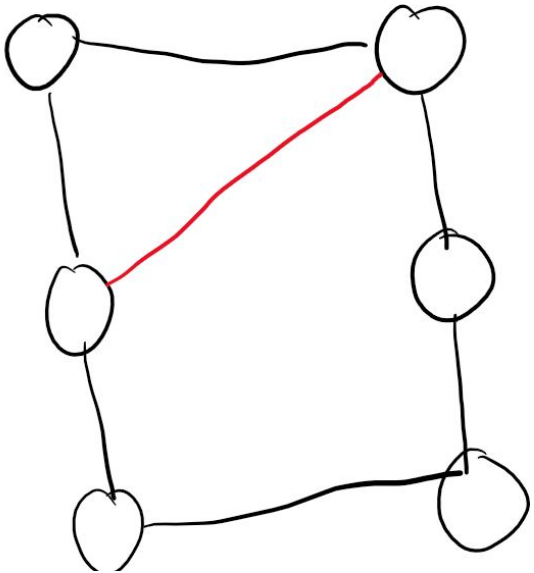
Tới subtask này ta lại quay về với đồ thị mặt trời.

Tới subtask này thì mình ko biết nói gì nữa rồi =)). Mình sẽ liệt kê các case ra và các bạn có thể ngồi khóc cùng mình.

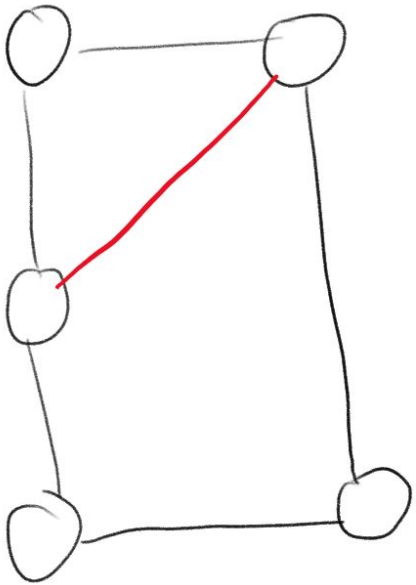
Việc còn lại sau casework không có gì khác ngoài tìm cạnh min trên một phần của hình tròn, trên một nhánh của đồ thị mặt trời, hoặc cả hai, hoặc thậm chí còn tệ hơn thế :(((

Subtask 4

1. Một chu trình chẵn bị cắt làm đôi

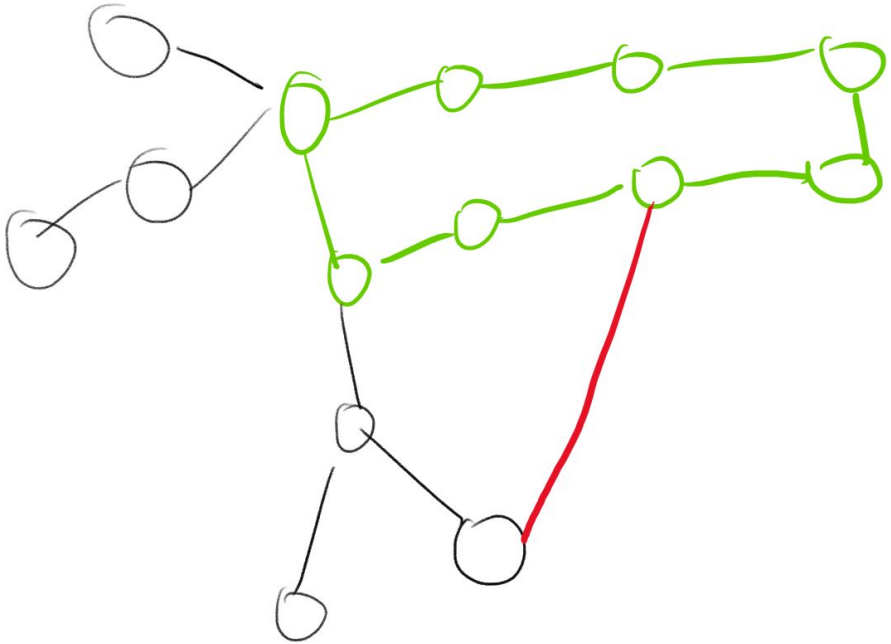


2. Một chu trình lẻ bị cắt thành một chu trình lẻ và một chu trình chẵn



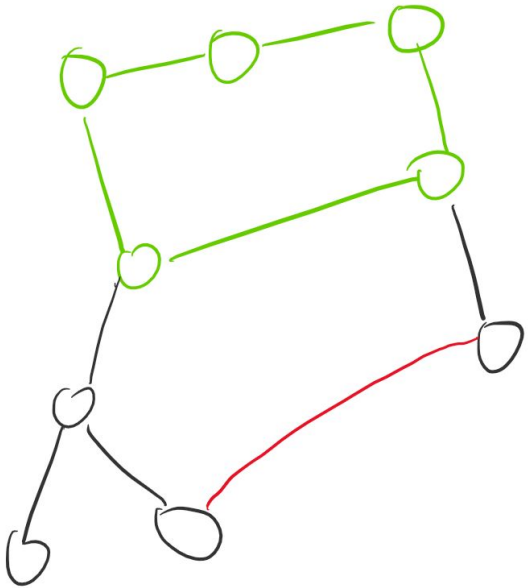
Subtask 4

3. Từ một đỉnh ngoài rìa nối vào vòng tròn tạo ra hai chu trình lẻ.
Trong case này ta có thể xóa một cạnh xanh, một cạnh đỏ hoặc một cạnh đen trên đường tới đỉnh lá.



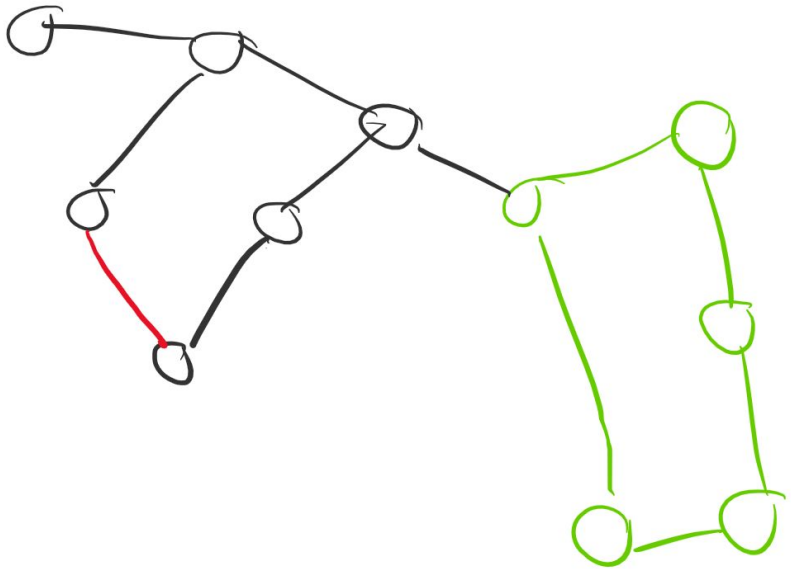
Subtask 4

4. Hai nút là từ hai nhánh phân biệt nối tới nhau.



Subtask 4

5. Hai nút là từ cùng một nhánh nối đến nhau. Tức là đến đây là bài Lubenica trên spoj.



Subtask 4

Ngoài ra thì việc hình tròn màu xanh là chẵn hay lẻ sẽ tiếp tục chia 3 case 3 4 5 ra thành 2 case nhỏ nữa.