GDG DEVFEST 2021 HA NOI

CODING CHALLENGE TEAM

TWO ARRAY

Lời giải

Ta dùng 2 mảng $g(i), f(i)$ là prefix sum của 2 dãy A và B

- $g(i) = g(i-1) + A_i$
- $f(i) = f(i-1) + B_i$

Gọi p_i là vị trí lớn nhất sao cho $g(p_i) \leq f(i)$ suy ra $f(i) - g(p_i) < n$

Xét $i = 0,1, \dots, n$ với $f(i) - g(p_i) \in [0, n)$, theo nguyên lý Dirichle ta có tồn tại ít nhất một cặp (x, y) $(x < y)$ thỏa mãn

$f(x) - g(p_x) = f(y) - g(p_y)$

$=> g(p_y) - g(p_x) = f(y) - f(x)$

Như vậy trên dãy A ta chỉ cần lấy dãy con liên tiếp từ $p_x + 1 \rightarrow p_y$ và trên dãy B ta chỉ cần lấy dãy con liên tiếp từ $x + 1 \rightarrow y$.

Ta sử dụng một mảng $K[f(i) - g(p_i)]$ là vị trí i đầu tiên có giá trị $f(i) - g(p_i)$ trên dãy B. Ban đầu ta gán bằng -1.

Khi đến một vị trí j nếu giá trị $K[f(j) - g(p_j)] \neq -1$ thì cặp (x, y) ta cần tìm chính là $(K[f(j) - g(p_j)], j)$.

Độ phức tạp: $O(N)$.

Permutation hope

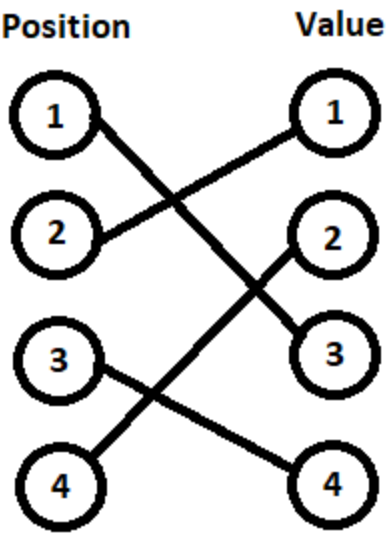
Bài toán này sẽ sử dụng bao hàm loại trừ để xử lý. Gọi P_j là số lượng cách chọn chính xác j vị trí $i(1 \leq i \leq N)$ trong N vị trí thỏa mãn $|a_i - i| = K$.

Kết quả bài toán chính là $\sum_{j=0}^{N}(-1)^j * P_j * (N-j)!$

Để hiểu lí do vì sao kết quả bài toán lại như trên, bạn cần làm quen và hiểu rõ thêm về bao hàm loại trừ.

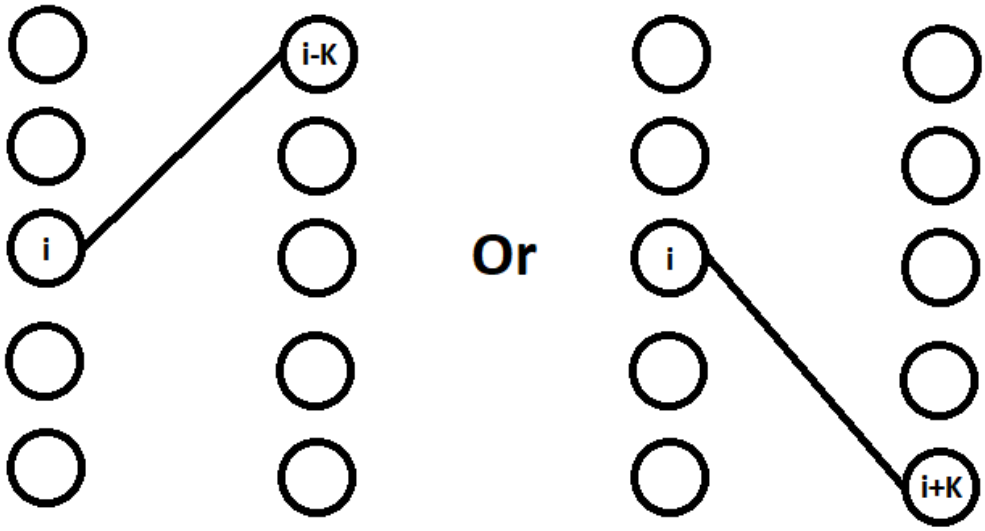
Bây giờ nhiệm vụ của ta là phải xây dựng lên mảng P_j như trên.

Trước hết, ta sẽ coi một hoán vị chính là một cách nối đỉnh như sau:



Đây là một cách nối của hoán vị độ dài 4, là {3,1,4,2}. Mỗi đỉnh chỉ được nối từ đúng chính xác một đỉnh khác.

Số lượng cách chọn j vị trí $i(1 \leq i \leq N)$ trong N vị trí thỏa mãn $|a_i - i| = K$ chính là số lượng cách chọn j đỉnh i ở cột $Position$ thỏa mãn đỉnh i được nối với đỉnh $i - K$ hoặc với đỉnh $i + K$ ở cột $Value$. Vì ta sử dụng tính chất của bao hàm loại trừ, nên tạm thời ta chỉ quan tâm tới j đỉnh được nối thỏa mãn, $N - j$ đỉnh còn lại tạm thời bỏ qua.



Nhận xét các đỉnh ở cột $Position$, chỉ có các đỉnh $mod\ 2 * K$ cùng số dư mới có thể ảnh hưởng tới nhau. Nếu ta chia N đỉnh này vào $2 * K$ nhóm, thì ta hoàn toàn có thể quy hoạch động riêng biệt trên từng nhóm khác nhau. Lí do là bởi ta chỉ xét các đỉnh i thỏa mãn được nối với $i - K$ hoặc $i + K$, nên với mọi hai đỉnh thuộc hai nhóm khác nhau, sẽ không bao giờ được nối tới cùng một đỉnh ở cột $Value$.

Sau khi đã chia thành $2 * K$ nhóm, ta sẽ quy hoạch động riêng biệt trên từng nhóm. Ở bài toán này, quy hoạch động lên sẽ dễ xử lý hơn so với quy hoạch động xuống.

Lưu ý: Công thức Dp dưới đây chỉ đang quy hoạch động mặc dù để chung nhưng thật ra chỉ xử lý riêng biệt trên các nhóm đỉnh có cùng số dư khi $mod\ 2 * K$.

Gọi $Dp[i][j][0/1/2]$ có ý nghĩa như sau:

- Xét tới đỉnh i
- Đã có j đỉnh trước đó được nối thỏa mãn
- $0/1/2$ lần lượt ý nghĩa như sau:
    - 0: Không chọn đỉnh i vào tập các đỉnh thỏa mãn

- 1: Chọn đỉnh i vào tập thỏa mãn, nhưng đỉnh i được nối với đỉnh $i - K$
- 2: Chọn đỉnh i vào tập thỏa mãn, nhưng đỉnh i được nối với đỉnh $i + K$

Trường hợp hai đỉnh cột $Position$ nối cùng một đỉnh bên cột $Value$ chỉ xảy ra khi đỉnh i cột $Position$ được nối với đỉnh $i + K$ cột $Value$ và đỉnh $i + 2 * K$ cột $Position$ được nối với đỉnh $i + 2 * K - K$ cột $Value$. Nên ta phải tránh trường hợp này ra.

Ta có công thức quy hoạch động như sau:

$$Dp[i + 2 * K][j][0] += Dp[i][j][0] + Dp[i][j][1] + Dp[i][j][2]$$

$$Dp[i + 2 * K][j][1] += Dp[i][j][0] + Dp[i][j][1] + Dp[i][j][2]$$

$$Dp[i + 2 * K][j][2] += Dp[i][j][0] + Dp[i][j][2]$$

Gọi $f[i][j]$ là số lượng cách chọn j đỉnh thỏa mãn trong nhóm thứ i(Cho nó gọn gàng dễ nhìn).

$$f[i \ mod \ (2 * K)][j] += Dp[i][j][0] + Dp[i][j][1] + Dp[i][j][2]$$

Sau khi quy hoạch động trên từng nhóm xong, ta sẽ xây dựng mảng P_j như sau:

$$P[j] += P[j - t] * f[i][t]$$

Sau khi có mảng P, dựa theo công thức bao hàm loại trừ ban đầu để tìm kết quả.

<u>Tốc độ thuật toán</u>: $O(N * K)$

MARATHON

Lời giải

Vì thứ tự của dãy C là không quan trọng nên chúng ta có thể sắp xếp dãy C theo thứ tự không giảm.

Gọi dp(i, j, u, v) là số lượng cách xây dựng đường đi thỏa mãn cho với điểm dưới trái là $(C_i, 0)$, điểm dưới phải là $(C_j, 0)$, điểm trên trái là (X_u, Y_u) và điểm trên phải (X_v, Y_v) và sử dụng tất cả các đỉnh trong vùng mình đang có.

Vậy theo định nghĩa kết quả bài toán của mình sẽ là $dp(0, n+1, 0, n+1)$ với $X_0 = 0, Y_0 = \infty, X_{n+1} = \infty, Y_{n+1} = \infty, C_{n+1} = \infty$.

Chúng ta có công thức quy hoạch động như sau:

$$dp(i, j, u, v) = \sum_{p=i+1}^{j-1} dp(i, p, u, k) * dp(p, j, k, v)$$

Với (X_k, Y_k) là đỉnh có Y_k lớn nhất mà vẫn đảm bảo thấp hơn min (Y_u, Y_v) và phải đảm bảo:

- $Vector$ từ đỉnh (X_k, Y_k) đến $(C_i, 0)$ so với $vector$ từ đỉnh $(C_i, 0)$ đến (X_u, Y_u) có hướng rẽ trái để đảm bảo rằng mình có thể tạo được một mặt phẳng và có thể nối các đỉnh khác và đảm bảo không cắt.
- Tương tự như trên, $vector$ từ đỉnh (X_k, Y_k) đến $(C_i, 0)$ so với $vector$ từ đỉnh $(C_i, 0)$ đến (X_u, Y_u) phải có hướng là rẽ phải.

Khởi tạo thì chúng ta chỉ cần gán những $dp(i, j, u, v) = 1$ với $i > j$.

Tốc độ thuật toán: $O(n^5)$.

PROTONX

Gọi $m = \min(N, A)$.

Kết quả của bài: $ans = m * X + (N - m) * Y$.

SUN*

Gọi $cnt[p][s]$ là số lượng cặp (i, j) , $i < j < p$ mà $a[i] + a[j] = s$.

Kết quả của bài toán: $\sum_{p=0}^{n} cnt[p][K - a[p]]$.

REPAIR

Gọi $\boldsymbol{min}$ là số nhỏ nhất trong dãy ban đầu.

Nhận xét: Để làm danh sách rỗng thì ta phải thực hiện nối lại $n - 1$ lần. Trong mỗi lượt, ta luôn có cách xoá sao cho chi phí nối lại sẽ là $\boldsymbol{min}$.

Do đó, kết quả bài toán là: $\boldsymbol{\min * (n - 1)}$.

STRING

Với mỗi thao tác chọn xâu con của s để nối vào z, ta luôn tham lam chọn xâu con dài nhất có thể.

Gọi p là độ dài xâu z hiện tại, c là độ dài của xâu con dài nhất của s có thể chọn trong thao tác tiếp theo. Rõ ràng, xâu con đó phải bằng xâu con $t_{z+1\ldots z+c-1}$.

Ta hoàn toàn có thể tham lam để chọn xâu con dài nhất thoả mãn. Với độ dài của xâu s là 10^5, ta chuẩn bị một mảng tính trước $next[i][c]$ là vị trí của kí tự c gần nhất bên phải tính từ vị trí i của xâu s.

EMATH

Trước hết, ta cần mô hình hóa bài toán theo cách sau: "Cho cây N đỉnh $N-1$ cạnh có trọng số, ban đầu các đỉnh được tô màu trắng. Bạn hãy chọn ít nhất các đỉnh để tô màu đen sau cho với mỗi đỉnh thì luôn tồn tại ít nhất một đỉnh được tô màu đen có khoảng cách không quá K đối với đỉnh đó".

Lưu ý: Ta sẽ gọi một đỉnh là "đã được bao phủ" khi tồn tại một đỉnh đen cách đỉnh đó không quá K. Ngược lại, ta sẽ gọi là "chưa được bao phủ".

Subtask 1: $N \leq 3 \times 10^5; 1 \leq K \leq 20;\ 30 \leq W_i \leq 10^9$

Nhận thấy $W_i > K$ nên khoảng cách giữa hai đỉnh bất kì luôn lớn hơn K, dẫn tới kết quả chính là N.

Độ phức tạp $O(N)$.

Subtask 2: $N \leq 17; 1 \leq K \leq 17;\ W_i = 1$

Duyệt hết tất cả các tập con khác rỗng của tập $\{1,2,\dots,N\}$ để thực hiện tô màu đen cho tập hợp đó. Sau đó với mỗi tập con thì ta có thể thực hiện **loạt BFS/BFS song song** từ tập hợp đó ra tất cả các đỉnh và kiểm tra xem có bao phủ được toàn bộ đồ thị hay không?

Độ phức tạp $O(2^N N)$.

Subtask 3: $N \leq 17; 1 \leq K \leq 10^6;\ 1 \leq W_i \leq 10^4$

Tương tự Subtask 2, ta duyệt hết tất cả các tập con khác rỗng của tập $\{1,2,\dots,N\}$ để thực hiện tô màu đen cho tập hợp đó. Sau đó với mỗi tập con thì ta có thể thực hiện **loạt Dijkstra/ Dijkstra song song** từ tập hợp đó ra tất cả các đỉnh và kiểm tra xem có bao phủ được toàn bộ đồ thị hay không?

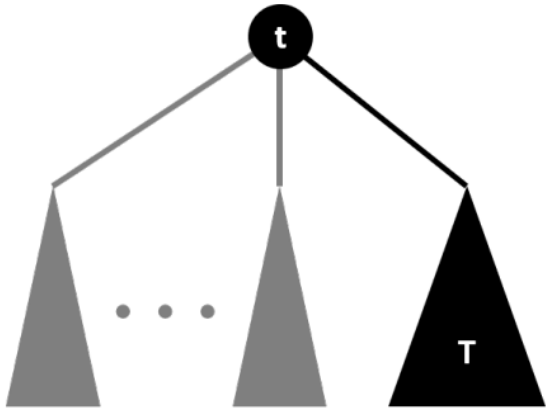
Độ phức tạp $O(2^N N log N)$.

Subtask 4: $N \leq 3 \times 10^5; 1 \leq K \leq 30;\ 20 \leq W_i \leq 10^9$

Giới hạn này sẽ cho ta nhận xét: với hai đỉnh cách nhau không quá K thì sẽ đi qua không quá một cạnh, điều này đương nhiên đúng vì nếu đi qua ít nhất hai cạnh thì khoảng cách sẽ là $2 \times W_i \geq 40 > K$.

Từ đó, ta có thể thực hiện DFS từ đỉnh gốc (chọn đỉnh 1 làm gốc), với mỗi hàm DFS của một cây con gốc T nào đó sẽ trả về cặp giá trị (x, y) (kiểu pair) và cặp giá trị này liên quan trực tiếp đến cha trực tiếp của T tạm gọi là t. Trong đó x là số lượng đỉnh ít nhất cần tô đen trong cây con gốc T và giá trị y gồm các ý nghĩa sau:

- $y = 1$: có x đỉnh màu đen được tô ít nhất trong cây con gốc T để có thể đảm bảo các đỉnh trong cây con gốc T này đã được bao phủ. Ngoài ra đỉnh cha t cũng được bao phủ nhờ một đỉnh đen nào đó trong T.
- $y = 0$: có x đỉnh màu đen được tô ít nhất trong cây con gốc T để có thể đảm bảo các đỉnh trong cây con gốc T này đã được bao phủ. Nhưng đỉnh cha t chưa được bao phủ.
- $y = -1$: có x đỉnh màu đen được tô ít nhất trong cây con gốc T và các đỉnh trong cây con gốc T cũng chưa được bao phủ hoàn toàn, nhưng nếu như tô đen đỉnh cha t thì toàn bộ các đỉnh trong cây con gốc T mới được bao phủ.



Để hiểu hơn, có thể nói cặp (x, y) luôn ưu tiên giá trị x trước, giá trị x nhỏ nhất càng tốt, mặc dù ta có thể thấy rõ $y = 1$ có hợi hơn khi $y = -1$ nhưng với một cặp $(x -$

$1, -1)$ và $(x, 1)$ thì $(x-1, -1)$ sẽ tốt hơn bởi sau này chỉ cần tô thêm một đỉnh đen tại cha t thì có thể đảm bảo được yêu cầu bài toán. Mặt khác, ta sẽ phải đưa ra cặp (x, y) với x nhỏ nhất và có thể chắc chắn được rằng: bây giờ, hoặc sau đó (đối với subtask này là khi đến cha t) ta có thể đảm bảo các đỉnh trong cây con gốc T sẽ được bao phủ.

Chiến thuật tham lam này hoàn toàn đúng, việc xây dựng sẽ dễ dàng khi sử dụng nhận xét đầu tiên của subtask này.

Độ phức tạp $O(N)$.

Subtask 5: $N \leq 3 \times 10^3; 1 \leq K \leq 10^{15};\ 1 \leq W_i \leq 10^9$

Ta có thể thực hiện một chiến thuật tham lam khác hoàn toàn đúng với độ phức tạp $O(N^2)$. Gọi $Depth[u]$ là độ sâu của đỉnh u trong cây con gốc 1 mà ta đang xét. Khi đó ta sẽ lặp đi lặp lại một thuật toán như sau:

- Bước 1: Chọn đỉnh u mà có $Depth[u]$ lớn nhất mà chưa được bao phủ.
- Bước 2: Trên đường đi từ $1 \rightarrow u$ chọn đỉnh P có $Depth[P]$ nhỏ nhất mà đỉnh P bao phủ được đỉnh u. Chọn đỉnh P để tô đen (bất chấp việc đỉnh P đã được bao phủ từ trước), sau đó đánh dấu lại các đỉnh vừa được đỉnh P bao phủ.

Ta cứ thực hiện như vậy cho đến khi nào toàn bộ đồ thị được bao phủ.

Chứng minh: Giả sử tại một bước bạn không chọn đỉnh P để bao phủ đỉnh sâu nhất của đồ thị thì sau bước đó bạn vẫn phải chọn một đỉnh khác để bao phủ đỉnh sâu nhất đó, điều này hoàn toàn không tối ưu. Tiếp theo, do đỉnh u là đỉnh sâu nhất của đồ thị nên việc chọn một đỉnh P trên đường đi từ $1 \rightarrow u$ sẽ bao phủ được ít nhất là toàn bộ các đỉnh trong cây con gốc P, tức là không bị bỏ sót "lá" và đỉnh P cũng là đỉnh cao nhất có thể để sau này khi xây dựng ta càng có lợi.

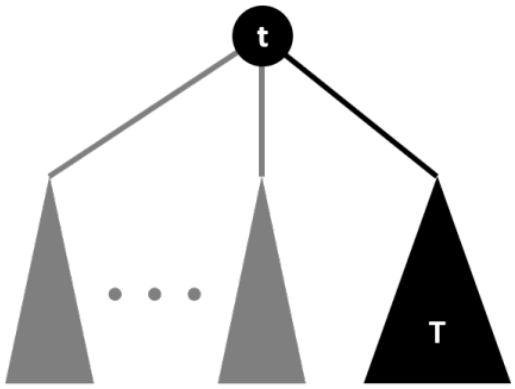
Độ phức tạp: $O(N^2)$.

Subtask 6: $N \leq 3 \times 10^5; 1 \leq K \leq 10^{15};\ 1 \leq W_i \leq 10^9$

Dựa vào subtask 4 và subtask 5, ta sẽ thiết kế một thuật toán DFS dựa trên tư tưởng tham lam và tính đúng đắn của tính chất bài toán.

Tương tự subtask 4, cây con gốc T có cha t khi DFS sẽ trả về hai giá trị (x, y) trong đó x vẫn là số đỉnh đen ít nhất có thể tô và y là thông tin phụ nói chung, thông tin phụ này theo một cách nào đó khi lập trình bạn sẽ truyền tải được hai loại trạng thái sau:

- $overflow(w)$ với $w \geq 0$: tô x đỉnh đen trong T, các đỉnh trong T được bao phủ hoàn toàn. Hơn thế nữa, các đỉnh cách cha t không quá w đều sẽ được bao phủ.
- $underflow(w)$ với $0 \leq w \leq K$: tô x đỉnh đen trong T, các đỉnh trong T chưa được bao phủ hoàn toàn. Tức là đỉnh xa nhất chưa được bao phủ trong T cách đỉnh cha t khoảng cách w, hay nói cách khác, ta kì vọng sau khi đi lên các đỉnh cha có thể tìm được đỉnh tô đen thích hợp cách đỉnh cha t không quá $K - w$ để có thể bao phủ được toàn bộ cây con gốc T.



Như đã nói ở Subtask 4, việc lưu trạng thái phụ y theo kèm với x bạn phải đảm bảo chắc chắn được việc "không sớm thì muộn" các đỉnh tại cây con gốc T sẽ được bao phủ.

Tiếp theo là việc chọn x càng nhỏ chắc chắn sẽ càng có lợi hơn vì thực tế, ta có thể có tô đen ngay đỉnh t để đáp ứng lại ràng buộc của bài toán khi ở trạng thái $underflow$.

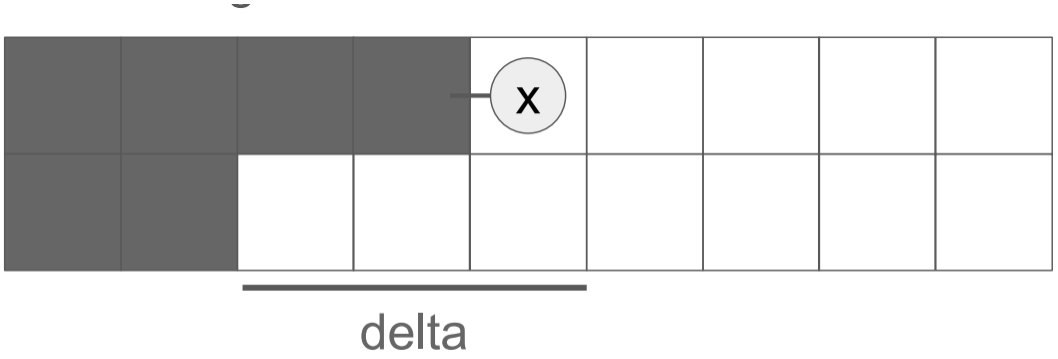
Độ phức tạp $O(N)$.

GRID HARD

Nhận xét: Cây ban đầu gốc là 1 sẽ cần phải là một cây nhị phân. Nếu không, nếu tồn tại một định có bậc lớn hơn 3, thì sẽ không có cách vẽ thoả mãn.

Xây dựng thuật toán quy hoạch động với trạng thái là (x, delta)

Ở trạng thái này, vẽ tất cả các nút trừ các nút là con của cây con gốc x. Khi đó, x sẽ được vẽ ở ô cuối của hàng dài hơn trong 2 hàng và delta là độ chênh lệch khoảng cách giữa 2 hàng.

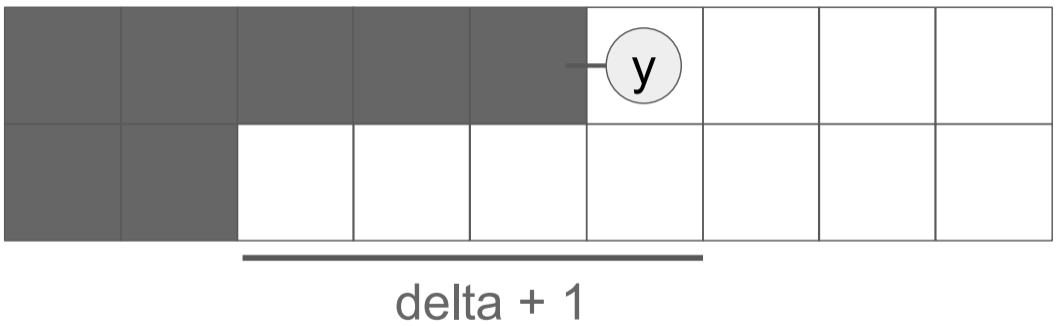


Để chuyển trạng thái tiếp theo, chúng ta xét mọi phép gán có thể của con của x cho những ô lân cận.

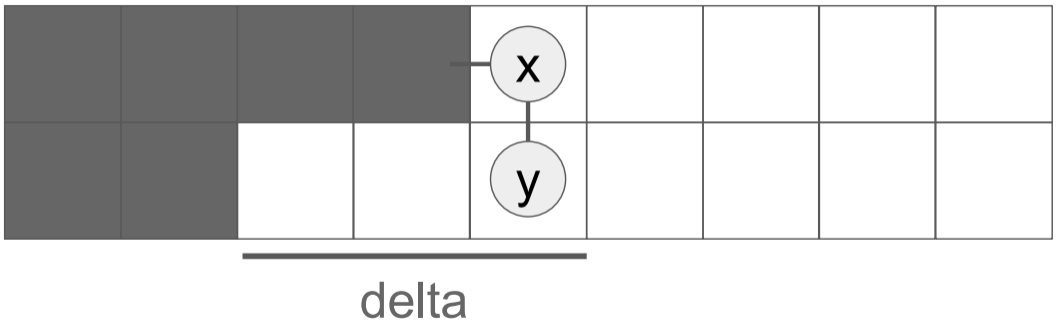
Trường hợp: x không có con. Tìm được một cách vẽ thoả mãn

Trường hợp x có một con duy nhất là y.

+ gán y sang ô bên phải x -> được trạng thái mới là (y, delta+1)

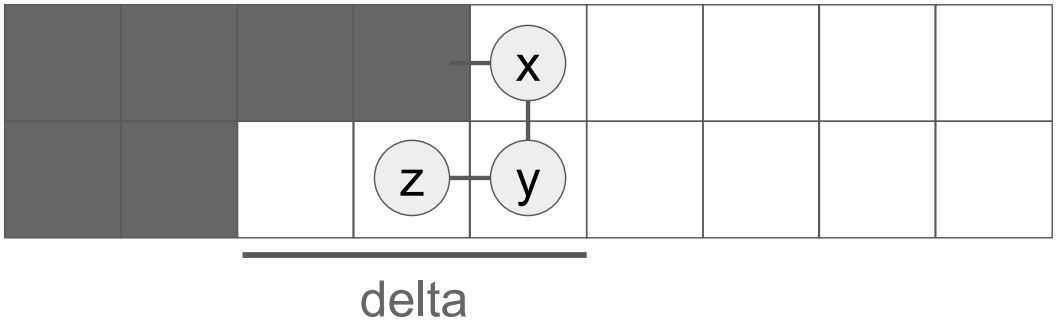


+ gán y sang ô dưới của x

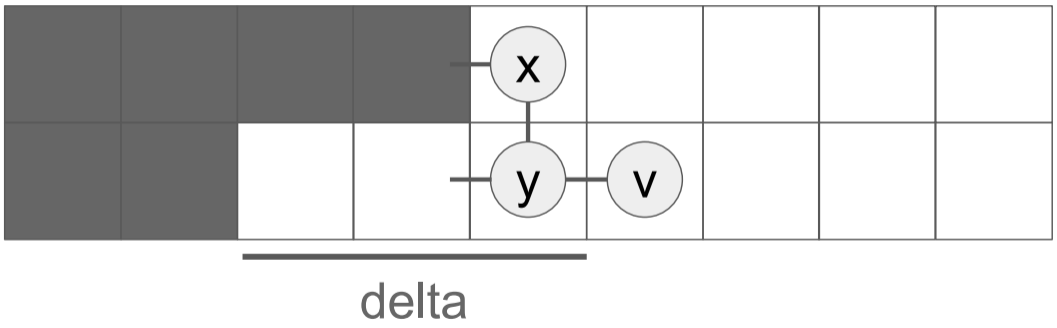


Tiếp tục gán con của y sang các ô bên cạnh

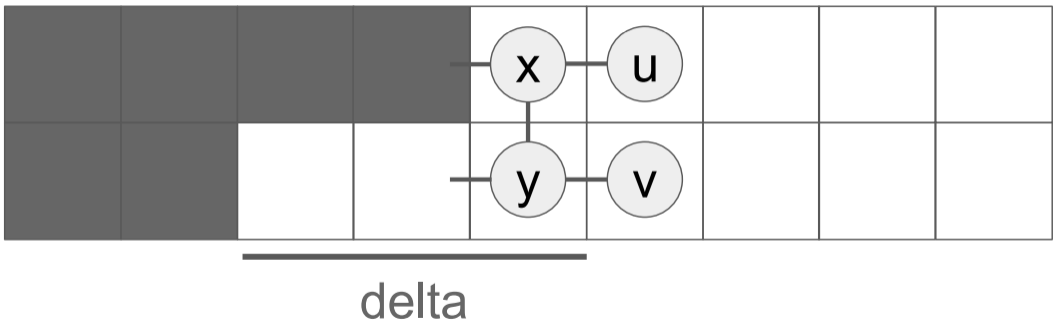
Nếu có một con z được gán sang bên trái, thì cây con của nó sẽ phải tạo thành một đường thẳng độ dài tối đa delta – 1.



Nếu có một con v của y được gán sang bên phải -> sẽ có trạng thái mới là (v, 1)

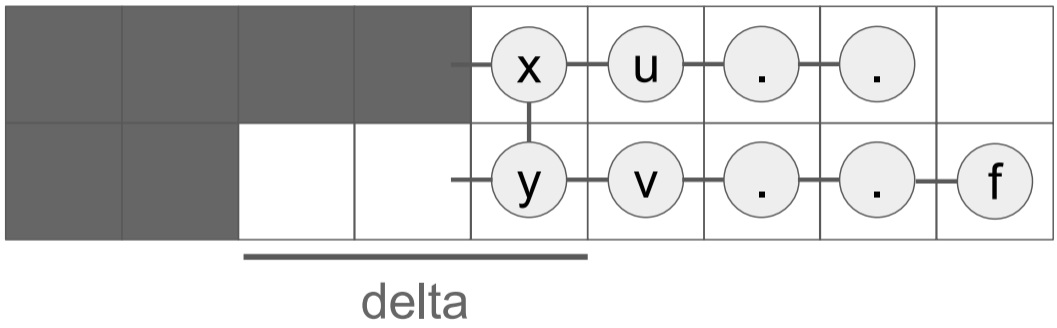


Trường hợp chung, nếu x có 2 con y và u, và y có một con gắn vào bên phải



Thực hiện thêm các node con vào bên phải cho đến khi chúng 1 trong số chúng hết node.

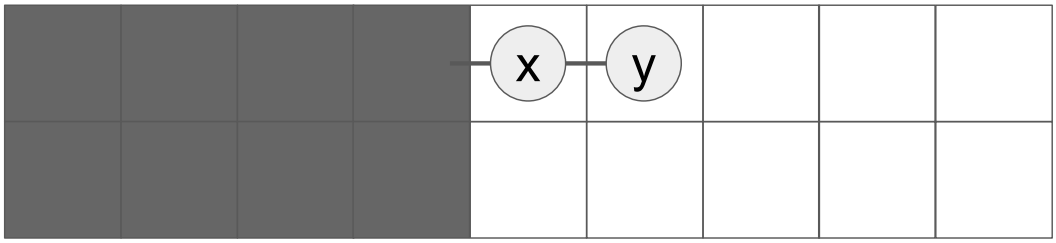
Khi xảy ra, sẽ có được trạng thái mới là (f, 1)



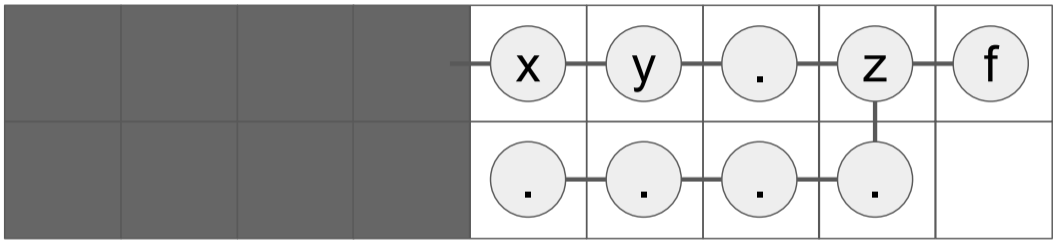
Với cách thực hiện trên sẽ có $O(n^2)$ trạng thái và có thể chuyển trạng thái trong O(1) bằng cách tính trước

Để tăng tốc độ, chúng ta sẽ cố định delta = 1

Trường hợp duy nhất duy nhất mà delta thay đổi là khi x có duy nhất một con và được gán ở bên phải



Chúng ta cần xác định một nút z trong cây con của y mà có sẽ có con được gán vào hàng phía dưới



Chúng ta đã giảm được số trạng thái xuống $O(n)$ và bằng tính trước, các chuyển trạng thái sẽ được hoàn thành trong $O(1)$, độ phức tạp chung là $O(n)$

MATRIX

Bắt đầu bằng phương trình đơn giản hơn

$$F[i,j] = a\,F[i,j-1] + b\,F[i-1,j]$$

Một giá trị ở vị trí $(1,j)$ sẽ đóng góp vào kết quả

$$C(n-j, 2n-j-2) * a^{n-j} * b^{n-1} * x$$

Một giá trị ở vị trí (I, 1) sẽ đóng góp vào kết quả

$$C(n-1, 2n-i-2) * a^{n-1} * b^{n-i} * x$$

Do đó chúng ta có thể tính được sự đóng góp của hàng đầu và cột đầu trong $O(n)$

Quay trờ lại công thức ban đầu,

C đóng góp vào kết quả

$$\sum_{i=2}^{n}\sum_{j=2}^{n}\left(\frac{(2n-i-j)!)}{(n-i)!(n-j)!}\right)a^{n-j}b^{n-i}c$$

$$c\sum_{i=2}^{n}\sum_{j=2}^{n}(2n-i-j)!\left(\frac{a^{n-j}}{(n-j)!}\right)\left(\frac{b^{n-i}}{(n-i)!}\right)$$

$$c\sum_{i=2}^{n}\sum_{j=2}^{n} f(i+j)g(i)h(j)$$

$$c\sum_{k=4}^{2n} f(k)(g*h(k))$$

Tích 2 đa thức có thể sử dùng FFT để tính trong $O(nlogn)$

MODULE

Với mỗi truy vấn thứ i.

Chúng ta sẽ đếm số lượng i ($0 \leq i < n-1$) không thoả mãn $(a_i mod\ m) < (a_{i+1} mod\ m)$

Nếu $(a_i mod\ m) = (a_{i+1} mod\ m)$

Trong trường hợp này, $d_{i\ mod\ k} = 0$. Những i thoả mãn có thể tính được trong $O(K)$ bằng cách đếm bao nhiêu phần tử d có giá trị bằng 0 được thêm vào a

Sau khi trừ đi $n-1$, ta có thể trả lời mỗi truy vấn trong $O(K)$