Levko và dãy – Levko and array

Nhận thấy rằng $c(a)$ càng lớn thì càng dễ làm, nên ta có thể tìm kiếm nhị phân kết quả, rồi kiểm tra xem một kết quả có thỏa mãn được hay không.

Xét trường hợp kiểm tra một kết quả x, nghĩa là hai số cạnh nhau không được chênh lệch quá x.

Nhận thấy, nếu $k = n$ thì kết quả là 0; Nếu $k < n$ thì ta phải có một số số không bị thay đổi.

Giả sử một số i không bị thay đổi, thì các số ở các vị trí $j < i$ giá trị bao nhiêu không quan trọng, ta chỉ cần biết là nó có thỏa mãn hay không.

Nếu có nhiều cách thỏa mãn thì dĩ nhiên là ta cần dùng cách mà phải sửa ít số nhất.

Vậy thì có ý tưởng quy hoạch động $f[i]$ là nếu ta không sửa vị trí i, ta cần phải bỏ ra ít nhất bao nhiêu lần sửa để các vị trí $j < i$ thỏa mãn.

$f[i] = i - 1$ nếu ta sửa tất cả các vị trí $j < i$.

Nếu không thì tồn tại vị trí j là vị trí lớn nhất trước i mà không được sửa.

Vì thế, $f[i] = \min\big(f[j] + (i - j - 1)\big)$ nếu ta có cách để sửa các vị trí từ $j + 1$ đến $i - 1$ sao cho thỏa mãn điều kiện.

Điều kiện để giữ nguyên j và i, $j < i$ là $|a_i - a_j| \le (i - j) \times x$. Điều này dễ thấy được vì mỗi lần ta được chênh lệch quá x, và ta được đổi $i - j$ lần.

Tính được $f[i]$ thì ta có thể tính thêm một mảng $g[i]$ ngược lại là làm từ phải về. Nếu tồn tại $f[i] + g[i] \le k$ thì kết quả là làm được.

Tuy vậy, việc tính $g[i]$ là không cần thiết. Nhận thấy nếu ta chọn i là vị trí cuối cùng mà sửa thì tất cả các vị trí cuối đều phải sửa, vì vậy ta chỉ cần kiểm tra xem có tồn tại i sao cho $f[i] + n - i \le k$ không là đủ.

https://codeforces.com/contest/360/submission/33282980

Các bài có kiểu thao tác tối ưu thường sẽ là quy hoạch động. Có nhiều trường hợp ta có thể lưu lại trạng thái quy hoạch động trực tiếp, như trong bài này có thể quy hoạch động $f[i][j]$ là đến i và giá trị a_i sửa thành j. Cũng có nhiều trường hợp không thể lưu được, có thể phụ thuộc vào giới

hạn hay bài toán không cho phép. Để quy hoạch động được thì ta cần phải dựa vào những giá trị không thay đổi.

Tô màu nghịch đảo - Inverse Coloring

Nhận thấy là điều kiện mà mỗi cặp hàng kề nhau thì hoặc là giống nhau hoặc là khác nhau ở mọi vì trí sẽ dẫn đến hàng $i > 1$ có hai cách chọn, hoặc là giống hệt hàng i hoặc là khác hoàn toàn hàng i.

Cũng nhận thấy luôn rằng điều kiện về hàng mà đúng thì điều kiện về cột cũng tự đúng luôn. Cụ thể, với hai cột i và $i + 1$, dù ta có đảo ngược hàng hay không thì hai ô tương ứng trên cùng hàng vấn sẽ luôn giống như ở hàng 1.

Vậy thì ta sẽ đi xây dựng hàng 1 trước rồi đi xây dựng cả bảng dựa trên điều kiện đó.

Vậy thì diện tích lớn nhất trong bảng là diện tích nào?

Nhận thấy là chiều rộng của hình chữ nhật này luôn là một đoạn liên tiếp bằng nhau trên cột 1, vì thế ta chỉ cần quan tâm đến đoạn dài nhất mà có màu giống nhau trên cột 1.

Nếu biết đoạn lớn nhất thì ta biết được là chiều cao lớn nhất là bao nhiêu. Tương tự như vậy thì chiều cao chính là đoạn dài nhất giống màu nhau trên cột 1.

Tổng quan lại thì ý tưởng có thể làm là với mỗi độ dài i, tính $f[i]$ là số cách xây dựng hàng 1 mà có độ dài lớn nhất là như vậy.

- Cái này có thể quy hoạch động $O(n^3)$: $g[i][j][k]$ là số cách sao cho đang chọn màu cho đến i, độ dài hiện tại là j và độ dài cao nhất là k. Có thể thực hiện duyệt k trước rồi tính $g[i][j]$, đảm bảo sao cho $j \leq k$ ở mọi thời điểm. Giá trị cần tính $f[i] = \sum g[n][j][i]$ với mọi j.
- Cũng có cách làm $O(n^2)$: duyệt độ dài k. Quy hoạch động $g[i]$ là số cách tô màu từ đầu đến i mà không có đoạn nào có độ lớn dài hơn k. Ta có thể tình $g = sum(g[j], i - j \leq k)$ (ta sẽ tô các ô từ $j + 1$ đến i cùng màu, màu này sẽ khác màu tô ô j. Có thể tính nhanh bằng cách tính tổng dồn mảng g. Giá trị cần tính $f[i] = g[i] - g[i - 1]$, lấy số cách dùng tối đa i số trừ đi số cách dùng tối đa $i - 1$ số sẽ ra số cách dùng đúng i số.

Sau đó với mỗi độ dài i, ta lại đi quy hoạch động để được kết quả cho toàn bảng.

Có thể thực hiện quy hoạch động giống cách thứ nhất để tính, mất $O(n^3)$.

https://codeforces.com/contest/1027/submission/43054518

Cũng có cách tối ưu bằng tổng dồn tương tự để được $O(n^2)$.

Một cách tốt hơn là nhận ra rằng cột và hàng có vai trò như nhau. Nói cách khác, các số ở cột 1 và hàng 1 quyết định cả bảng. Diện tích tối đa là đoạn dài nhất cùng màu trên cột 1 nhân với đoạn dài nhất cùng màu trên hàng 1. Vì thế ta có thể dùng lại giá trị $f[i]$ đã tính. Kết quả là:

$$\frac{sum(f[i] \times f[j])}{2} \ (1 \le i, j \le n; i \times j < k)$$

Có chia 2 vì cột 1 và hàng 1 chung nhau ô $(1,1)$. Để không phải xử lí chia 2, các bạn có thể cho là ô $(1,1)$ luôn có một màu rồi nhân 2 sau này.

https://codeforces.com/contest/1027/submission/97566277

Bài này quan trọng là nhận ra được cái dãy chỉ phụ thuộc vào dòng và cột đầu tiên. Việc quy hoạch động $O(n^3)$ khá là thẳng thắn, cải tiến thành $O(n^2)$ cũng không có gì đặc biệt.

Lại phủ bảng - Another Filling the Grid

Điều kiện $min = 1$ tương ứng với việc trên mỗi hàng và mỗi cột có ít nhất một số 1.

Giả sử ta đi điền số vào bảng theo thứ tự từ trên xuống dưới từ trái sang phải. Vậy thì với mỗi hàng, ta phải điền ít nhất 1 số 1. Sau khi đi hết hàng thì ta cũng phải điền hết các cột.

Thế thì có ý tưởng là quy hoạch động đến một hàng, cột nào có số 1 hay chưa. Làm thế này mất $O(n \times 2^n)$ trạng thái.

Thì nhận ra là các cột thực tế như nhau, nên ta có thể quy hoạch động chỉ cần lưu lại số cột chưa được điền 1.

Cụ thể, sẽ quy hoạch động $f[i][j]$ là điền xong hàng thứ i mà còn j cột chưa có số 1 được điền thì có bao nhiêu cách.

Kết quả là $f[n][0]$.

Trạng thái ban đầu $f[0][n] = 1$.

Ta xét trạng thái $f[i][j]$, liệu trạng thái này có thể phát triển thành các trạng thái nào khi ta thực hiện điền hàng i?

Có hai trường hợp cần quan tâm:

- Trường hợp mà không điền 1 vào một cột chưa được điền:
  - Cả j cột chưa được điền sẽ phải điền từ $2 \to k$, vậy bọn này có $(k-1)^j$ cách để điền.
  - Trong $n - j$ cột đã được điền, ta phải điền ít nhất một số 1 cho hàng i. Thì số lượng cách điền thỏa mãn là tổng số lượng cách điền trừ đi số lượng cách điền mà không dùng số 1, cụ thể là $k^{n-j} - (k-1)^{n-j}$.
  - Vậy trong trường hợp này, ta chuyển trạng thái $f[i+1][j] += f[i][j] \times (k-1)^j \times \left[k^{n-j} - (k-1)^{n-j}\right]$
- Trường hợp mà ta điền vào ít nhất một cột chưa được điền.
  - Giả sử ta điền sao cho còn lại $l < j$ cột chưa được điền.
  - Vậy thì có $jC(j-l) = jCl$ cách chọn các cột này. Các cột này có 1 cách điền duy nhất là điền số 1 như đã chọn.

- $n-j$ cột đã điền 1 rồi thì ta điền cái gì cũng được, nên có k^{n-j} cách điền bọn này.
- l cái cột không được điền mới 1 sẽ có $(k-1)^l$ cách để điền.
- Vậy ta có thể chuyển trạng thái trong trường hợp này là: $f[i+1][l] += f[i][j] \times (jCl) \times (k-1)^l \times k^{n-j}$

Cần tính trước các giá trị $(k-1)^x$, k^x và xCy để có thể sử dụng lại trong $O(1)$. Vì có $O(n^2)$ trạng thái, mỗi trạng thái ta duyệt lại $O(n)$ để cập nhật, độ phức tạp là $O(n^3)$.
https://codeforces.com/contest/1228/submission/61493792

Cách quy hoạch động này là cách đơn giản và khá cổ điển, tuy nhiên bài này vẫn có các cách giải tốt hơn. Ý tưởng của cách đó là có $2n$ điều kiện cần phải thỏa mãn của bảng (n dòng $min = 1$ và n cột $min = 1$), vậy thì nếu tính được là có bao nhiêu cách mà thỏa mãn tối đa $0 \leq x \leq 2n$ điều kiện, ta có thể tính được bằng bao hàm loại trừ. Điều kiện sẽ có dạng là có bao nhiêu cách mà thỏa mãn tối đa i cột và tối đa j dòng. Điều kiện này tương ứng với số cách chọn ra i cột, j dòng, các số trên cột này được phép bằng 1, các số không trên cột này không được bằng 1, … Nói chung là một cách giải rất hay. Cách này sẽ có độ phức tạp là $O(n^2)$ nếu chuẩn bị trước. Độ phức tạp có thể đạt được $O\big(nlog(n)\big)$ nếu ta tiếp tục biến đổi.