Bài 1: Dãy số

Sub 1: Độ phức tạp O(Q.n^3)

Với mỗi truy vấn, ta xét mọi dãy con liên tiếp và kiểm tra trực tiếp điều kiện bài cho

***Sub 2: Độ phức tạp* O(n+ Q.log2(n))**

- Trước hết, chúng ta sẽ tạo mảng L[i] là vị trí xa i nhất về bên trái sao cho a[L[i]] > a[i]. Như vậy tất cả các phần tử thuộc đoạn [L[i]+1 …i] đều không lớn hơn a[i].
- Tương tự, tạo mảng R[i] là vị trí xa i nhất về bên sao cho a[R[i]] > a[i] , như vậy tất cả các phần tử thuộc đoạn [i…R[i]–1] đều không lớn hơn a[i]
  Suy ra các phần tử thuộc đoạn [L[i]+1…R[i]–1] (có tất cả R[i] – L[i] – 1 phần tử) đều không lớn hơn a[i].
- Ta sẽ lưu tất cả các giá trị (R[i] – L[i] – 1) vào mảng b[i] tương ứng.
- Sau đó, chúng ta sẽ sort mảng A tăng dần, mảng B sẽ tráo đổi các phần tử phụ thuộc vào sự sắp xếp của mảng A.
- Với mỗi truy vấn, ta chặt nhị phân trên dãy A để tìm vị trí p thỏa mãn a[p] ≤ K và p lớn nhất. Sau đó giá trị MAX(b[1]→ b[p]) chính là kết quả.

Chú ý:

- Các mảng L và R có thể dùng Deque hoặc QHĐ trực tiếp để tính trong O(n).
- Để tính MAX(b[1] →b[i]) bất kì ta ta có thể chuẩn bị trước một mảng maxb[i] lấy ra MAX(b[1] → b[i]) trong O(1).

Bài 2: Khu đất

<u>Trường hợp k = 1:</u>

Cách 1: ĐPT O(n^2 * log n)

For góc trái trên (hoặc góc phải dưới đều được), chặt nhị phân cạnh hình vuông.

Rõ ràng là hình to tồn tại thì đương nhiên hình bé cũng tồn tại. Để kiểm tra có tồn tại hình vuông không thì đơn giản ta dùng mảng cộng dồn, tính tổng là ok.

Cách 2:

Có thuật toán đơn giản hơn, không cần chặt nhị phân, tìm kiếm ở đây là tuyến tính

Vẫn For điểm trái trên

Khởi tạo best = 0;

```
For (i: 1..m)
    For (j: 1..n)
        For (r: 1..min(m,n))
```

```
If (psum(i, j, r) <= S)
    If (r > best) best = r;
```

Cài đặt như trên thì **ĐPT: O(n^3)**

- **Cải tiến:**

```
For (i: 1..m)
    For (j: 1..n)
        For (r: best + 1..min(m,n))
            If (psum(i, j, r) <= S
                {  If (r > best) best = r;}
             Else break;
```

ĐPT: $O(n^2 + n)$

<u>Trường hợp k = 2:</u>

Bất kì 2 hình vuông không nằm đè lên nhau thì luôn tồn tại một lát cắt để tách nó làm hai nửa.

Xét 2 TH: cắt ngang hoặc cắt dọc

Bây giờ ta xét cắt dọc. Còn TH cắt ngang giải quyết tương tự/ hoặc xoay bảng.

Khi cắt dọc cần thử xem điểm cắt là điểm nào?

Nếu có được mảng F với ý nghĩa:

F(c, t) = r: sử dụng xét các cột từ 1 đến c, có số tiền là t thì sẽ có được hình vuông cạnh lớn nhất là bao nhiêu?

Số tiền t quá lớn → QHĐ đảo nhãn

$F(c, r_1) = t_1$: Từ cột 1 đến cột c, muốn lấy được hình vuông cạnh r_1 thì số tiền ít nhất cần dùng là bao nhiêu?

$G(c+1, r_2) = t_2$: từ cột c+1 đến n, muốn lấy ra hình vuông cạnh r_2 thì số tiền ít nhất cần dùng là bao nhiêu?

Sau khi có được mảng F và G thì ta xử lí như sau:

For c | For r_1 | For r_2

$F(c, r_1) + G(c+1, r_2) <= t$ thì có thể chọn phương án $r1* r1 + r_2* r_2$ → cập nhật kết quả

Tính F, G?

Mảng h(c, r): hình vuông có cạnh r và bên phải tiếp xúc với c thì số tiền ít nhất là bao nhiêu

For c | For r | for i → $O(n^3)$

Trượt trên c

F(c, r) = min (F(c-1, r), h(c,r))

Bài 3: Đường đi

Sub1: Áp dụng **Dijsktra** (cải tiến) để tìm đường đi ngắn nhất từ 1 đỉnh thuộc nA tới 1 đỉnh thuộc nB rồi kiểm tra trực tiếp xem có bằng W.

Thuật full:

Ta thấy đếm số lượng đường đi có trọng số nhỏ nhất bằng đúng W khó hơn rất nhiều so với cách đếm số lượng đường đi có trọng số nhỏ nhất <= W

→ Tư duy tính phần bao.

Tính P1 là số lượng đường đi có trọng số <=W-1;

Tính P2 là số lượng đường đi có trọng số <= W.

Từ đó dễ dàng tính được số lượng đường đi có trọng số đúng bằng W sẽ là P2-P1.

Nhận xét ta chỉ cần xét những cạnh có trọng số <=W. Khi nhập dữ liệu ta loại bỏ những cạnh có trọng số >W, các cạnh còn lại sẽ phân vào 2 tập:

X: tập các cạnh có trọng số <W

Y: tập các cạnh có trọng số =W

Tính P1 trên tập cạnh X; tính P2 trên tập cạnh X U Y

Tính P1 như sau:

Với mỗi thành phần liên thông:

DFS(): tính được a, b là số lượng đỉnh thuộc tập n_A, n_B trong thành phần liên thông đó

P1 = P1 + a * b

Tương tự tính P2.

❖ **Ta có thể giải quyết bài toán theo cách khác, đó là dùng DSU**

Nhận thấy là đường đi nhỏ nhất (trọng số cạnh lớn nhất trên đường đi là nhỏ nhất) thì đường đi này phải nằm trên cây khung nhỏ nhất

Do vậy ta sắp xếp các cạnh tăng dần theo trọng số

Không phải dùng DSU thông thường (mỗi đỉnh u lưu trữ parent[u]) mà mỗi đỉnh lưu trữ thêm số lượng đỉnh thuộc nA là cntA[u], số đỉnh thuộc nB là cntB[u] nằm trong thành phần liên thông chứa u

Xét lần lượt các cạnh, mỗi khi xét cạnh (u, v) nếu thực hiện join được thì

- Nếu trọng số cạnh (u, v) == W thì ta đếm số cặp đỉnh có đường đi qua cạnh (u, v)
  Thỏa mãn yêu cầu đề bài là cntA[u] * cntB[v] + cntB[u] * cntA[v]
- rồi ta mới thực hiện join 2 cạnh này (lưu ý cập nhật cả cntA[], cntB[])