SOLUTION 13/11

Bài 1. XorSecond

Thay vì đi tính kết quả của mọi đoạn [l,r], ta sẽ đi tính mọi kết quả có thể xảy ra.

Giả sử với một a_i, tồn tại một đoạn [l,r] để nó là MaxSecond(l,r), thì Max(l,r) có thể chọn chỉ có thể là số gần nhất về bên trái với a_i và lớn hơn a_i hoặc số tương tự nhưng gần nhất về bên phải. Như vậy ta dễ dàng tìm được kết quả bài toán bằng ***stack***.

Để chứng minh điều này, giả sử gọi a_j là số gần nhất về bên trái với a_i và lớn hơn a_i, a_k là số gần thứ hai về bên trái với a_i và lớn hơn a_i, sẽ không thể có đoạn [l,r] nào thỏa mãn a_i là MaxSecond(l,r) và a_k là Max(l,r), bởi k < j. Tương tự với vế bên phải.

Ví dụ cho dãy: 1 5 2 4 3 6 7

- Giả sử như có một đoạn [l,r] nào đó chứa a_5 = 3 là MaxSecond(l,r), ta có thể thấy rằng Max(l,r) của đoạn đó chỉ có thể là a_4 = 4 hoặc a_6 = 6.

- Không thể có chuyện tồn tại dãy [l,r] nào đó chứa MaxSecond(l,r) = 3 mà Max(l,r) = a_2 = 5 được, bởi khi đó MaxSecond(l,r) phải có giá trị bằng a_4 = 4.

- Tương tự ta có thể chứng minh rằng không thể có chuyện Max(l,r) = a_7 = 7 được, bởi khi đó MaxSecond(l,r) phải bằng a_6 = 6.

Như vậy, ta sẽ duyệt qua toàn bộ MaxSecond(l,r) có thể và tìm ra mọi Max(l,r) tương ứng với nó, nghĩa là ta sẽ không bỏ sót cặp nghiệm nào cả và có thể tìm ra kết quả tối ưu.

Bài 2: MST

Đầu tiên, ta sẽ gộp các nhóm cạnh có chung trọng số vào, nhóm cạnh có trọng số là W gọi là nhóm W.

Dựa vào thuật toán Kruskal, ta nhận xét rằng, khi ta xét tới nhóm cạnh W và đã DSU tất cả các cạnh trong các nhóm có trọng số ≤ W-1, ***cấu hình*** của DSU sẽ chỉ có 1, nói cách khác, bất kể với từng nhóm cạnh ≤ W-1 ta đưa vào DSU các cạnh

trong đó theo bất kì thứ tự nào, thì tính liên thông của đồ thị trong DSU sau khi xét toàn bộ các nhóm cạnh ≤ W-1 vẫn giữ nguyên.

Ta biết thêm được rằng, một cạnh (u,v,c) chỉ được chọn để nằm trong một MST khi và chỉ khi trước đó u và v chưa liên thông. Từ đây, khi xét tới các cạnh trong nhóm W, giả sử đỉnh u và đỉnh v đã chung một vùng liên thông, chắc chắn nó sẽ không thể thuộc một MST nào cả, vậy nên ta bỏ qua. Ngược lại, với các cạnh có u,v khác vùng liên thông, đây chắc chắn sẽ là các cạnh ***nằm trong ít nhất*** một MST.

Để kiểm tra xem các cạnh (u,v) đó có nằm trong ***mọi MST*** hay không, ta cần xây dựng một đồ thị con chỉ gồm các cạnh này (các cạnh trong nhóm W có u,v khác vùng liên thông). Tuy nhiên, trong đồ thị con, ta sẽ không coi nó là cạnh (u,v) nữa, mà là cạnh (a,b), với a là gốc của vùng liên thông chứa u, tương tự b là gốc của vùng liên thông chứa v. Tưởng tượng như ta đang nối hai vùng liên thông này bằng cạnh (a,b,i), với i ở đây là index cạnh ban đầu, lưu ý cần lưu thêm index của cạnh vì sẽ có rất nhiều cạnh lặp, bởi dù hai cạnh (u,v) và (x,y) ban đầu khác nhau nhưng trong đồ thị mới nó có thể có cùng gốc nên quy về thành cạnh (a,b).

Đến đây, ta sẽ có một đồ thị mới với các đỉnh là các vùng liên thông trong DSU (chú ý rằng đồ thị mới không nhất thiết liên thông với nhau), một cạnh sẽ nằm trong mọi MST khi và chỉ khi bỏ cạnh đó ra, đồ thị mới của ta bị tăng số vùng liên thông lên, bởi nếu không tăng thì tức là không cần dùng cạnh đó thì ta vẫn có thể giữ tính chất liên thông của đồ thị. Hay nói cách khác, các cạnh thỏa mãn chính là ***các cạnh cầu***.

Độ phức tạp: O(n+m).

Bài 3: Giao hàng

- Để giải được bài toán này, ta sẽ áp dụng thuật toán ***tham lam***.

- Nhận thấy rằng, ta có thể chia bài toán ra thành hai pha riêng biệt. Pha đầu tiên ta sẽ chọn **x** nhân viên vào kho **1** và **t-x** người vào kho hai. Sau đó sẽ tối ưu cách giao hàng tới các địa điểm và lấy ***min*** kết quả.

- Vậy ta có thể thực hiện các pha này như thế nào?

- Gọi tập ***a*** là tập hợp gồm các nhân viên đang đi tới kho ***1***, ***b*** là tập các nhân viên đang đi tới kho ***2***, ***c*** là tập các nhân viên đang không được giao việc.

- Đầu tiên, ta sẽ chọn ***t*** nhân viên đi tới kho 1, và từ kho 1 sẽ giao tới ***t*** địa điểm nhận hàng. Gọi giá trị ***cur*** là chi phí cho sự lựa chọn này. Gán biến kết quả ***res = cur***.

- Gọi a[i] = (x,y) là khoảng cách nhân viên i tới kho 1 và kho 2.

- Ta sort các a[i] theo x tăng dần.

- Gọi b[i] = (x,y) là khoảng cách địa điểm i tới kho 1 và kho 2.

- Ta sort các giá trị b[i] theo (y-x) tăng dần.

- Từ giờ, ta sẽ dần dần ***chuyển*** một nhân viên vào kho 2, ***giảm*** một nhân viên tới kho 1 và ***hủy*** một đơn hàng giao từ kho 1 thành giao từ kho 2, và tính lại khoảng cách quãng đường.

- Ta sẽ for i trong khoảng [1,t], mỗi bước ta có hai lựa chọn:

- Chuyển một nhân viên từ tập 1 sang s3 và một nhân viên từ s3 sang s2:

  - Để chuyển tối ưu, ta sẽ lấy nhân viên có giá trị q = a[i].x lớn nhất trong tập a để chuyển sang c, và lấy nhân viên có giá trị p = a[i].y nhỏ nhất trong tập c để chuyển sang b.
  - Độ tăng giá trị của lựa chọn này sẽ là (p-q).

- Chuyển một nhân viên từ tập s1 sang tập s2:

  - Để chuyển tối ưu, ta sẽ lấy nhân viên có giá trị (a[i].y - a[i].x) nhỏ nhất để chuyển.
  - Độ tăng giá trị của lựa chọn này sẽ là (a[i].y - a[i].x).

- Ta sẽ xem xét lựa chọn nào là tối ưu nhất về đương đi trong hai lựa chọn trên.

- Gọi Delta = chi phí tăng lên nhỏ nhất trong hai lựa chọn, vậy lúc này tổng chi phí sẽ là:

- cur += Delta (chi phí chuyển một nhân viên tới kho 2 và giảm một nhân viên tới kho 1).

- cur += b[i].y - b[i].x (chi phí hủy một đơn hàng giao tới địa điểm i từ kho 1 và chuyển thành giao từ kho 2).

- Ta sẽ cập nhất ***res = min (res,cur)*** cho kết quả.

- Để xử lý các thao tác một cách nhanh nhất, ta cần:

- Nhận thấy tập b không cần quan tâm do ta không lấy giá trị ra từ nó, vậy nên ta không cần kiểm soát tập này.
- Tập a ta sẽ lưu bằng hai set như sau:
  - set đầu tiên sort theo thứ tự tăng dần của a[i].x. (phục vụ lựa chọn một)
  - set thứ hai sort theo thứ tự tăng dần của a[i].y-a[i].x. (phục vụ lựa chọn hai)

- Tập c ta chỉ cần lưu bằng một set sort theo a[i].y tăng dần.

- Lúc này những thao tác tìm x lớn nhất, y bé nhất, y-x bé nhất, xóa phần tử khỏi tập và thêm phần tử vào tập chỉ mất O(log) mỗi lần.

- Độ phức tạp; O(n*log).

Bài 4: Vẻ đẹp của dãy

Nhận xét: với bất kì dãy a đẹp, khi sắp xếp dãy a tăng dần, phần tử a_{i+1} phải chia hết cho phần tử a_i.

⇔ Bài toán trở thành tìm độ dài dãy con dài nhất sao cho phần tử a_{i+1} phải chia hết cho phần tử a_i (dãy a sắp xếp tăng dần).

Gọi dp_i là dãy con dài nhất kết thúc tại i, f_x = max dp_j .(với mọi j < i, a_j = x).

dp_i = max f[x] + 1 (với mọi x là ước của $a_{i)}$.

Kết quả chính là n – max dp_i

Gọi div(x) là số lượng ước của a_i. ta có độ phức tạp là O(n.max div(a_i))

Bài 5: Chỉ số đầu tiên

Với subtask đồ thị là cây, ta gán mỗi cạnh trên cây có trọng số w - số thứ tự của cạnh đó trong input.

Xét hai đỉnh a, b bất kì, để a, b liên thông thì kết quả chính là trọng số lớn nhất trên đường đi từ a – b trên cây. Ta có thể tính được max trọng số các cạnh nằm trên đường đi a – b bằng cách sử dụng kĩ thuật binary lifting và LCA (cha chung gần nhất).

Xét các đỉnh liên tiếp từ l đến r, kết quả chính là max trọng số các cạnh nằm trên đường đi từ a đến b với l <= a <= b <= r. Tuy nhiên, ta nhận thấy rằng kết quả đó

cũng chính là max trọng số các cạnh nằm trên đường đi từ a đến b với b = a + 1, l <= a < b <= r.

Từ nhận xét trên, ta gọi f[i] chính là kết quả của đỉnh i và đỉnh i + 1, trả lời mỗi truy vấn chính là max f[i] với l <= i < r. (sử dụng các thuật toán RMQ để lấy max).

Với subtask không phải là cây, ta chỉ cần dựng cây khung nhỏ nhất sau đó cài đặt thuật toán trên.

(Định lý: trên cây khung nhỏ nhất, giá trị lớn nhất của các cạnh trên đường đi hai đỉnh khác nhau sẽ là nhỏ nhất).

Bài 6: Áp đảo

Ta có: $r_i . w_j > r_j . w_j$

$\Leftrightarrow r_i . w_j - r_j . w_i > 0$

$\Leftrightarrow (r_i, w_i) \times (r_j, wj) > 0$ (x ở đây là tích có hướng).

Từ đó ta thấy rằng với bộ i, j, k đôi một khác nhau, nếu đỉnh (r_i, w_i), (r_j, w_j), (r_k, w_k) tạo thành một tam giác chứa điểm (0, 0) $\Leftrightarrow$ trận đấu này sẽ không tìm ra người chiến thắng.

Bài toán trở thành : đếm số tam giác chứa điểm (0, 0).

Việc đếm số tam giác chứa điểm (0, 0) khá là khó. Tuy nhiên ta có thể đếm phần bù của nó, chính là đếm số tam giác không chứa điểm (0, 0).

Sử dụng kĩ thuật sweepline để đếm số tam giác không chứa điểm (0, 0). Lần lượt quét các vecto (r_i, w_i) theo hệ số góc, chú ý đến trường hợp các trường hợp đặc biệt như tam giác có ba đỉnh thẳng hàng, tam giác có 2 đỉnh thẳng hàng với điểm (0, 0).

Độ phức tạp O(nlogn).