Bài toán A. xuanquang1999 và Quá trình chọn lọc đề bài

Ta định nghĩa vị trí i là đẹp nếu như $a_i < a_{i+1}$ (nghĩa là ta có thể thực hiện phép biến đổi tại vị trí i).

Giả sử ta thực hiện phép biến đổi ở vị trí đẹp i. Xét bốn số nguyên $a_{i-1}, a_i, a_{i+1}, a_{i+2}$. Sau khi thực hiện phép biến đổi, ta còn ba số nguyên $a_{i-1}, a_i + a_{i+1}, a_{i+2}$. Ta có các nhận xét sau:

- Nếu $i-1$ là vị trí đẹp trước phép biến đổi ($a_{i-1} < a_i$), thì vị trí này vẫn đẹp sau phép biến đổi (do $a_{i-1} < a_i + a_{i+1}$).
- Nếu $i+1$ là vị trí đẹp trước phép biến đổi ($a_{i+1} < a_{i+2}$), thì vị trí này có thể không còn đẹp sau phép biến đổi (do $a_i + a_{i+1}$ chưa chắc nhỏ hơn a_{i+2}).
- Các vị trí j trước phần tử a_{i-1} và sau phần tử a_{i+2} vẫn sẽ giữ nguyên tính chất đẹp (hoặc không đẹp), do không bị ảnh hưởng gì bởi phép biến đổi trên.

Do đó, cách thực hiện phép biến đổi tốt nhất luôn là lựa chọn vị trí đẹp cuối cùng để thực hiện phép biến đổi (khi đó việc bị mất vị trí đẹp sẽ không xảy ra). Đồng thời, do không có thêm vị trí đẹp xuất hiện sau vị trí thực hiện phép biến đổi, nên ta chỉ cần duyệt i từ $n-1$ ngược về 1, và thực hiện phép biến đổi nếu vị trí i đẹp.

Độ phức tạp: $O(n)$.

Bài toán B. TrungNotChung và Công tác chuẩn bị kì thi

Đáp án là "NO" nếu tồn tại cặp chỉ số i, j ($i \neq j$) thỏa mãn một trong hai điều kiện sau:

- $p_i = p_j$ và $q_i \neq q_j$.
- $p_i \neq p_j$ và $q_i = q_j$.

Ngược lại, với k là số cặp (p, q) phân biệt, luôn tồn tại đáp án với c nhỏ nhất bằng $k+1$. Với các hãng bàn được thí sinh lựa chọn, t_{p_i} có thể nhận giá trị bất kì từ 1 đến k sao cho không có hai hãng nào có chiếc bàn cùng chiều cao. Các hãng màn hình được thí sinh lựa chọn sẽ mua màn hình có chiều cao: $m_{q_i} = k + 1 - t_{p_i}$.

Với các hãng bàn và màn hình không có thí sinh nào lựa chọn, chiều cao của màn hình và bàn mua ở các hãng này có thể nhận giá trị bất kì từ $k+1$ đến 10^9 sao cho không có hai hãng có chiếc bàn hoặc màn hình cùng chiều cao.

Bài toán C. ngfam làm Hoa Tiêu

Từ định nghĩa của đề bài, ta thấy một đỉnh u được gọi là centroid nếu cây gốc u không tồn tại cây con v ($v \neq u$) sao cho cây con gốc v có ít nhất $\frac{n}{2}$ đỉnh.

Subtask 1: $n \leq 100$**.**

Ở subtask này ta có sẽ tận dụng tối đa phép toán `adj` để dựng lại cây gốc. Ta duyệt qua lần lượt các đỉnh từ 2 tới n. Với mỗi đỉnh u được lặp qua, ta sẽ tìm đường đi từ đỉnh hiện tại (gọi là r) tới u bằng cách liên tục truy vấn:

1. `adj u` (trả về `k`)
2. `move k`
3. Gán `r = k`

Qua mỗi bước `move`, ta có thể thêm (u, k) vào danh sách cạnh của đồ thị. Sau khi có đồ thị, ta có thể duyệt qua toàn bộ các đỉnh và kiểm tra điều kiện centroid như mô tả và di chuyển tới đỉnh đó.

Số câu hỏi cần thiết: $O(n^2)$

Subtask 2: $n \leq 1000$**.**

Với giới hạn của subtask này, ta nhận thấy việc dựng lại đồ thị gốc là không khả thi. Vì vậy ta cần một cách tiếp cận để đi tới centroid không thông qua cấu trúc đồ thị.

Sử dụng nhận xét ở phần đầu, ta thấy nếu đỉnh r ta đang đứng không phải centroid, ta luôn có ít nhất $\frac{1}{2}$ cơ hội tìm được một đỉnh ngẫu nhiên mà đỉnh đó nằm trong cây con gốc centroid.

Do đó ta có thể sử dụng thuật toán ngẫu nhiên, lấy mẫu m đỉnh bất kì chưa được chọn và kiểm tra xem cây con chứa các đỉnh đó có kích thước ít nhất $\frac{n}{2}$ hay không. Nếu có thì ta sẽ di chuyển tới đỉnh đó.

Nhận xét: Khoảng cách từ một đỉnh bất kì trên cây tới centroid luôn không quá $\frac{n}{2}$. Dễ thấy nếu khoảng cách này lớn hơn $\frac{n}{2}$, thì nhánh con chứa r trên cây gốc centroid chứa toàn bộ các đỉnh trên đường đi, tương đương ít nhất $\frac{n}{2}$ đỉnh.

Kết hợp cách tiếp cận ngẫu nhiên và nhận xét trên ta có thể lặp lại các bước sau cho tới khi đi tới centroid:

1. Lấy mẫu m đỉnh, với mỗi đỉnh ta hỏi `adj` và `subtree` để kiểm tra nhánh con có chứa centroid hay không
2. Nếu tồn tại nhánh con chứa centroid, dùng phép `move` tới nhánh con đó

Ta có thuật toán sử dụng $O(mn + \frac{n}{2})$ với tỉ lệ sai của mỗi bước là 2^{-m}.

Để đảm bảo tính đúng đắn, ta cần ít nhất $m = 30$ để đạt được tỉ lệ sai đủ nhỏ, tuy nhiên sẽ không đủ để thực hiện trong giới hạn đề bài.

Đến đây ta có thể tối ưu theo nhiều cách khác nhau, một trong số đó là:

1. Sau mỗi bước đi, ta bỏ dần các đỉnh đã đi qua khỏi danh sách lấy mẫu. Mỗi lần danh sách lấy mẫu nhỏ đi 2 lần, ta sẽ giảm m đi 1
2. Thay vì hỏi `subtree` cho cả m lần lấy mẫu, ta sẽ hỏi lấy toàn bộ nhánh từ `adj` và chỉ hỏi `subtree` cho nhánh chứa nhiều mẫu nhất.

Subtask 3: $n \leq 10000$**.**

Cách giải 1: Cải tiến thuật toán random ở bên trên.

Nhận thấy thay vì sau mỗi bước ta thực hiện lấy mẫu để tìm hướng đi tiếp theo, ta có thể sử dụng một đỉnh đích để thực hiện nhiều bước đi một lúc.

1. Tại mỗi thời điểm ta chưa có hướng đi, ta thực hiện lấy mẫu cho tới khi tìm được hướng đi (đỉnh *target* sao cho nhánh con chứa *target* có ít kích thước ít nhất $\frac{n}{2}$.
2. Đi theo hướng tới *target* cho tới khi nhánh con chứa *target* không còn có kích thước ít nhất $\frac{n}{2}$.

Như vậy với mỗi bước `move` ta sẽ tốn 3 phép toán: [`adj`, `subtree`, `move`], trong đó `subtree` để đảm bảo ta không đi vào nhánh sai. Như vậy ta tốn tối đa $\frac{3n}{2}$ cho việc di chuyển.

Như vậy ta có `555` phép toán để lấy mẫu và đảm bảo ta luôn tìm được centroid với xác suất sai rất nhỏ.

Cách giải 2: Lời giải của anh RR.

Nhận xét: Với mỗi gốc u không phải centroid, tồn tại một và chỉ một nhánh con chứa centroid. Từ đó, ta luôn biết chắc chắn nhánh con chứa centroid là cây con trong cây gốc u chứa nhiều đỉnh nhất.

Từ nhận xét trên, ta có thuật toán sau:

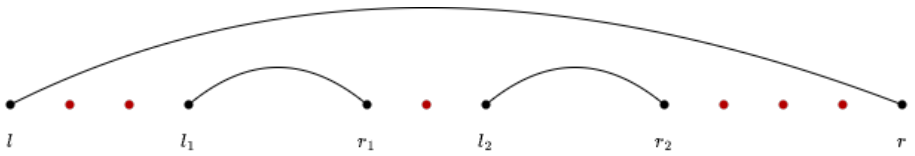
1. Liệt kê kết quả của truy vấn `subtree` cho tất cả các đỉnh trên cây gốc 1
2. Sắp xếp lại danh sách các đỉnh theo kết quả truy vấn giảm dần, gọi dãy đỉnh kết quả là $a_1 = 1, a_2, ..., a_n$.
3. Di chuyển lần lượt từ a_1 tới a_2, a_2 tới a_3, ..., a_{k-1} tới a_k với a_k là đỉnh cuối cùng có kết quả truy vấn lớn hơn hoặc bằng $\frac{n}{2}$. Và a_k cũng chính là centroid ta cần tìm.

Bài toán D. darkkcyan và Quy hoạch vị trí thí sinh

Subtask 1: $m \leq 5000$**.**

Ở subtask này, ta được phép tính lại đáp án trong $O(m)$. Để thuận tiện cho việc tính toán, ta thêm một dây mạng nối giữa 0 và $n+1$.

Ta gọi dây mạng x là con của y nếu y là dây mạng có u_y lớn nhất thoả mãn $u_y < u_x \leq v_x < v_y$.



Xét một dây mạng y có các con là $x_1, x_2, \ldots, x_k$, thì số dây mạng là con của y có thể nối thêm là $cnt_y = \left\lfloor \frac{v_y - u_y - 1 - \sum_{i=1}^{k}(v_{x_i} - u_{x_i} + 1)}{2} \right\rfloor$. Đáp án của bài toán chính là tổng các cnt_y với mọi dây mạng có sẵn.

Ta sắp xếp các dây mạng tăng dần theo vị trí của mút trái. Ta sẽ duy trì một ngăn xếp gồm các dây mạng lồng nhau. Khi thêm một dây mạng x, trước hết ta loại bỏ hết các dây mạng y nằm ở đỉnh ngăn xếp có $v_y < u_x$. Sau bước này, dây mạng y ở đỉnh ngăn xếp (nếu có) sẽ là cha của x. Thêm x vào ngăn xếp và tiếp tục thực hiện quá trình này với các dây mạng còn lại.

Subtask 2: $u_i > u_{i+1}$ **với mọi** $1 \leq i < m$**.**

Ở subtask này, cha của một dây nối sẽ thay đổi nhiều nhất một lần. Ta sẽ dùng `std::set` S để lưu các dây nối là con của $(0, n+1)$. Khi thêm (u_i, v_i), ta chặt nhị phân trên S (hàm `lower_bound`) để tìm các con của dây nối này, sau đó loại bỏ chúng và thêm (u_i, v_i) vào S.

Subtask 3: không có giới hạn gì thêm.

Xét các dây nối sau khi thêm dây nối thứ m, ta dựng một cây T thể hiện quan hệ cha con của các dây nối (việc dựng cây đã được trình bày ở subtask 1). Thay vì thêm dần các dây nối vào tập, ta sẽ làm ngược lại tức là loại bỏ dần các dây nối từ m về 1. Trong quá trình này, ta cần biết cha hiện tại của một dây nối để cập nhật kết quả, dễ dàng thấy rằng cha cần tìm là tổ tiên gần nhất chưa bị loại bỏ ở trên cây T.

Ở đây trình bày một cách làm dùng *DSU* (Disjoint Sets Union). Khởi tạo một DSU gồm m đỉnh tương ứng với các dây nối. Gọi $par[x]$ là dây nối cha của một dây nối x trên cây T. Duyệt x từ m về 1, thực hiện nối cạnh $par[x] - x$ trên DSU, cha hiện tại của x sẽ là đỉnh có độ sâu thấp nhất trong thành phần liên thông chứa x.

Lưu ý rằng ta cần cập nhật mảng *cnt* sau mỗi thao tác ở subtask 2 và 3.

Bài toán E. Lành và Quá trình phân phát áo VNOI cup

Ta thấy, với một bảng xếp hạng gồm đầy đủ m vòng, ban tổ chức sẽ lựa chọn số k như sau:

- Gọi $r[i]$ là thứ hạng tốt nhất của thí sinh thứ i sau m vòng, và $c[j]$ là số lượng thí sinh i thỏa mãn $r[i] = j$.
- Ban tổ chức sẽ chọn số k lớn nhất, thỏa mãn $c[1] + c[2] + \ldots + c[k] \leq s$.

Subtask 1: $n, m \leq 8$**.**

Ở subtask này, ta chỉ cần duyệt qua toàn bộ $n!$ khả năng có thể xảy ra và tính toán thứ hạng tốt nhất của tất cả các thí sinh.

Subtask 2: $n, m \leq 500$**.**

Ta thấy, điều kiện để thí sinh thứ i nhận áo sau m vòng là tồn tại ít nhất $n - s$ thí sinh j khác mà $r[j] > r[i]$.

Gọi $r_2[j]$ là thứ hạng tốt nhất của thí sinh i sau $m - 1$ vòng. Ta thấy, có ba khả năng có thể xảy ra ở vòng thứ m:

1. *Thí sinh i đạt thứ hạng $x < r_2[i]$:*

   Giả sử có a thí sinh j có $r_2[j] > x$ (không gồm thí sinh i) thì phải có ít nhất $n - s$ thí sinh trong đó đạt thứ hạng lớn hơn x ở vòng thứ m.

   Ta có thể xem xét bài toán này như sau: cho n người và n ghế ngồi, trong đó có a người xấu và $n - a - 1$ người tốt. Biết ghế ngồi thứ x đã có người, hãy sắp xếp $n - 1$ người còn lại vào các ghế ngồi, sao cho tồn tại đúng b người xấu ngồi bên phải ghế thứ x.

   Ta có $\binom{n-x}{b}$ cách chọn vị trí cho b người xấu bên phải ghế x, $\binom{x-1}{a-b}$ cách chọn vị trí cho $a - b$ người xấu bên trái ghế x, còn những người tốt có thể ngồi tùy ý. Vì vậy, đáp án cho trường hợp này là $\binom{n-x}{b} \cdot \binom{x-1}{a-b} \cdot a! \cdot (n - a - 1)!$.

   Tính tổng tất cả các trường hợp, ta được đáp án $a! \cdot (n - a - 1)! \cdot \sum\limits_{b=n-s}^{a} \binom{n-x}{b} \cdot \binom{x-1}{a-b}$.

2. *Thí sinh i đạt thứ hạng $x = r_2[i]$:*

   Giả sử có a thí sinh j có $r_2[j] > r_2[i]$ thì phải có ít nhất $n - s$ thí sinh trong đó đạt thứ hạng lớn hơn $r_2[i]$ ở vòng thứ m.

   Lập luận tương tự khả năng 1, ta được đáp án $a! \cdot (n - a - 1)! \cdot \sum\limits_{b=n-s}^{a} \binom{n-x}{b} \cdot \binom{x-1}{a-b}$.

3. *Thí sinh i đạt thứ hạng $x > r_2[i]$:*

   Tương tự trường hợp 2, giả sử có a thí sinh j có $r_2[j] > r_2[i]$ thì phải có ít nhất $n - s$ thí sinh trong đó đạt thứ hạng lớn hơn $r_2[i]$ ở vòng thứ m.

   Lúc này, bài toán của chúng ta sẽ thay đổi đôi chút: cho n người và n ghế ngồi, trong đó có a người xấu và $n - a - 1$ người tốt. Biết ghế ngồi thứ x đã có người, hãy sắp xếp $n - 1$ người còn lại vào các ghế ngồi, sao cho tồn tại đúng b người xấu ngồi bên phải ghế thứ $r_2[i]$.

   Ta có $\binom{n-r_2[i]-1}{b}$ cách chọn vị trí cho b người xấu bên phải ghế $r_2[i]$, $\binom{r_2[i]}{a-b}$ cách chọn vị trí cho $a - b$ người xấu bên trái ghế $r_2[i]$, còn những người tốt có thể ngồi tùy ý. Vì vậy, đáp án cho trường hợp này là $\binom{n-r_2[i]-1}{b} \cdot \binom{r_2[i]}{a-b} \cdot a! \cdot (n - a - 1)!$.

   Tính tổng tất cả các trường hợp, ta được đáp án $a! \cdot (n - a - 1)! \cdot \sum\limits_{b=n-s}^{a} \binom{n-r_2[i]-1}{b} \cdot \binom{r_2[i]}{a-b}$.

Ta duyệt qua tất cả các thí sinh, sau đó duyệt qua các thứ hạng có thể xảy ra ở vòng thứ m của thí sinh i, và duyệt thêm một lần nữa để tính tổ hợp. Vì vậy, độ phức tạp của thuật toán này là $O(n^3)$.

Subtask 3: $n, m \leq 2000$**.**

Chúng ta sẽ xem xét cải tiến từng trường hợp:

1. *Thí sinh i đạt thứ hạng $x < r_2[i]$:*

   Từ lời giải của subtask 2, ta có thể thấy công thức tính không phụ thuộc vào $r_2[i]$, vì vậy ta có thể cải tiến trường hợp này bằng tổng tiền tố, thay vì duyệt qua toàn bộ thứ hạng $x < r_2[i]$.

2. *Thí sinh i đạt thứ hạng $x = r_2[i]$:*

   Đây là một trường hợp riêng lẻ, ta không cần phải tối ưu trường hợp này.

3. *Thí sinh i đạt thứ hạng $x > r_2[i]$:*

   Từ lời giải của subtask 2, ta có thể thấy công thức tính không phụ thuộc vào thứ hạng x mà chỉ phụ thuộc vào $r_2[i]$, vì vậy ta chỉ cần nhân đáp án một trường hợp cho $(n - r_2[i])$ thay vì duyệt toàn bộ thứ hạng $x > r_2[i]$.

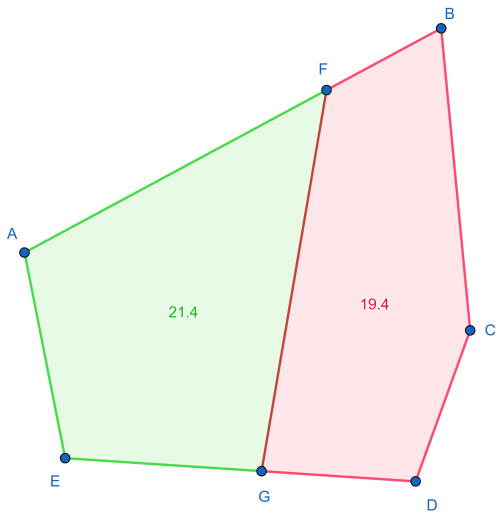
Với những cải tiến trên, ta đạt được độ phức tạp cuối cùng là $O(n^2)$.

Bài toán F1. FireGhost và Lát cắt hoàn hảo 1

Trước hết, nhận xét sau rất quan trọng với mọi subtask của cả F1 và F2: mọi cách cắt thỏa mãn đều chia chu vi miếng bánh thành 2 phần có độ dài bằng nhau. Đây là vì chu vi của mỗi phần bao gồm độ dài của đa giác ở trong phần đó, cộng cho độ dài cạnh cắt; vì độ dài cạnh cắt nằm trong cả chu vi hai phần, ta chỉ cần độ dài của đa giác ở hai phần bằng nhau để có chu vi của hai phần đa giác bằng nhau. Nói cách khác, nếu P, Q là hai điểm cắt trên miếng bánh thì P và Q là *điểm đối cực* của nhau. Dễ thấy với mỗi điểm P, điểm đối cực Q là duy nhất.

Chúng ta sẽ cùng xem một cách làm khá đơn giản của bài F1 trước mà có thể áp dụng cho mọi subtask, nhưng mà cách làm này không thể áp dụng với bài F2. Gọi hàm f nhận một điểm P trên đa giác có ý nghĩa như sau:

- Lấy điểm Q là điểm đối cực của điểm P trên đa giác. Hàm $f(P)$ có giá trị bằng diện tích phần bánh bên trái theo hướng vectơ $\vec{PQ}$ trừ diện tích phần bánh bên phải theo hướng vectơ $\vec{PQ}$ *không lấy dấu trị tuyệt đối.*



Phần bánh bên trái vectơ $\vec{FG}$ có màu đỏ, và phần bánh bên phải vectơ có màu xanh. Vì thế, $f(F) = 19.4 - 21.4 = -2$.

Nhận xét: với điểm P bất kì trên đa giác và điểm Q là điểm đối cực của điểm P, ta có $f(P) = -f(Q)$. Đây là vì phần bánh bên trái vectơ $\vec{PQ}$ là phần bánh bên phải vectơ $\vec{QP}$; tương tự, phần bánh bên phải

vectơ $\vec{PQ}$ là phần bánh bên trái vectơ $\vec{QP}$. Vì thế, theo định nghĩa của hàm f, ta có điều phải chứng minh.

Bởi thế, nếu ta có điểm A nào đó di dọc chiều kim đồng hồ trên biên đa giác từ điểm P với điểm Q, ta nhất thiết phải có một vị trí nào đó mà $f(A) = 0$ (do $f(P)$ và $f(Q)$ trái dấu). Do đó, độ chênh diện tích nhỏ nhất có thể đạt được luôn bằng 0. Ta cũng có luôn một thuật toán chặt nhị phân để tìm điểm A sao cho $f(A)$ bằng 0 như sau.

- Khởi tạo điểm $A := P$ và $B := Q$. Ở đây, giả sử $f(A) > 0$ (và vậy thì $f(B) < 0$).
- Lặp lại thao tác sau đây nhiều lần: chọn điểm C nằm "chính giữa" trên đường đi từ A theo B theo chiều kim đồng hồ. Nếu $f(C) > 0$, gán $A := C$; ngược lại, gán $B := C$.
- Khi hai điểm A và B đủ gần (hoặc sau khi lặp đủ số lần cần thiết), in ra điểm A.

Ta nhận thấy sau mỗi bước của thao tác trên, ta đảm bảo được rằng $f(A) > 0$ và $f(B) < 0$, trong khi khoảng cách từ A tới B theo chiều kim đồng hồ giảm xuống một nửa. Vì thế, sau đủ thao tác, A và B sẽ có khoảng cách đủ gần để có $f(A) \approx 0$.

Về cách cài đặt, cách biểu diễn một điểm P bất kì trên đa giác đơn giản nhất là sử dụng một số thực mô tả khoảng cách từ đỉnh 1 tới điểm P theo chiều kim đồng hồ. Ta chỉ cần cài đặt thêm hai hàm sau: hàm tìm điểm đối cực của một điểm bất kì trên đa giác và hàm f (hoặc tương tự là hàm tìm diện tích phần bên trái và phần bên phải). Hai hàm này có thể được cài đặt trong độ phức tạp $O(n)$, và vì vậy bài toán của chúng ta đã được giải quyết trong độ phức tạp $O(n \log \epsilon^{-1})$ với ϵ là độ chính xác cần có.

(Bạn nào muốn tìm hiểu thêm về bài toán có thể đọc về định lý Borsuk-Ulam và phương pháp tìm nghiệm bisection.)

Bài toán F2. Tahp và Lát cắt hoàn hảo 2

Subtask 1: $n = 3$**.**

Với điều kiện miếng bánh có dạng hình tam giác, ta có thể tìm lát cắt tối ưu bằng cách giải phương trình bậc hai. Cụ thể, giả sử 2 điểm cắt P, Q nằm trên 2 cạnh AB, AC.

Ta có những điều kiện sau:

- $AP + AQ = \frac{AB+BC+AC}{2} = p$
- $AP \le AB$
- $AQ \le AC$
- $|AP \cdot AQ - \frac{AB \cdot AC}{2}| = |AP \cdot AQ - s|$ nhỏ nhất/lớn nhất có thể.

Giả sử $AP \cdot AQ = q$ thì AP, AQ sẽ là hai nghiệm của phương trình bậc hai $x^2 - px + q = 0$ (theo định lý Viète). Vậy ta phải tìm q sao cho $|q - s|$ nhỏ nhất có thể và $x^2 - px + q = 0$ có hai nghiệm $x_1 \le AB$, $x_2 \le AC$.

Giả sử $AB \le AC$, điều kiện trên tương đương với:

- $p - \sqrt{\Delta} \le 2AB$
- $p + \sqrt{\Delta} \le 2AC$

trong đó $\Delta = p^2 - 4q$.

Từ hai điều kiện trên, ta có thể tìm ra giới hạn của q, từ đó tính được giá trị của AP, AQ tương ứng.

Ngoài ra, ta có thể tìm q bằng phương pháp chia tam phân.

Subtask 2: $n \leq 500$**.**

Giả sử 2 điểm cắt P, Q lần lượt nằm trên các cạnh A_iA_{i+1} và A_jA_{j+1}, khi "di chuyển"điểm P trên cạnh A_iA_{i+1} sao cho điểm Q tương ứng vẫn nằm trên cạnh A_jA_{j+1}, ta có thể chứng minh diện tích phần bánh bên trái vector $\vec{PQ}$ có dạng parabol (phần chứng minh cụ thể xin nhường lại cho bạn đọc). Do vậy, để giải bài toán F2, ta chỉ cần tìm điểm P sao cho diện tích tương ứng của phần bánh bên trái lớn nhất/nhỏ nhất. Tương tự, với bài toán F1, ta cần tìm điểm P sao cho diện tích gần với $\frac{1}{2}$ diện tích bánh nhất. Cả 2 phần đều có thể được giải quyết bằng cách dựng phương trình parabol hoặc chia tam phân.

Tóm lại, ta có thể giải hai bài toán F1 và F2 như sau:

- Với mỗi đỉnh A_i trên đa giác, tìm điểm đối cực B_i. Sắp xếp lại tất cả $2n$ điểm này, chia chu vi đa giác thành $2n$ đoạn.
- Trong mỗi đoạn, tìm điểm P sao cho khoảng cách diện tích giữa 2 miếng bánh là tối ưu bằng cách dựng phương trình cho diện tích hoặc chia tam phân.

Phần tìm điểm đối cực và tính diện tích có thể được cài đặt trong độ phức tạp $O(n)$, do đó độ phức tạp của thuật toán này là $O(n^2)$ hoặc $O(n^2 \log \epsilon^{-1})$, trong đó $\epsilon = 10^{-12}$ là sai số tối đa.

Subtask 3: Không có giới hạn gì thêm.

Để giải subtask cuối cùng, ta cần tối ưu thuật toán ở subtask 2.

Trước hết, ta có thể tìm điểm đối cực trong $O(\log n)$ bằng cách chuẩn bị trước tổng cộng dồn của độ dài các cạnh đa giác, kết hợp với kĩ thuật chia nhị phân.

Tiếp theo, ta có thể tính nhanh diện tích của một phần miếng bánh trong $O(\log n)$. Giả sử ta cần tính diện tích của đa giác $PA_iA_{i+1} \cdots A_jQ$. Áp dụng shoelace formula (tạm dịch: công thức buộc dây giày), ta có:

$Area(PA_iA_{i+1} \cdots A_jQ) = \frac{1}{2}(cross(P, A_i) + cross(A_i, A_{i+1}) + \cdots + cross(A_{j-1}, A_j) + cross(A_j, Q) + cross(Q, P))$

trong đó $cross(P, Q) = x_P y_Q - x_Q y_P$.

Đặt $pref_i = cross(A_0, A_1) + cross(A_1, A_2) + \cdots + cross(A_{i-1}, A_i)$, ta có $Area(PA_iA_{i+1} \cdots A_jQ) = \frac{1}{2}(cross(P, A_i) + \mathbf{pref_j} - \mathbf{pref_i} + cross(A_j, Q) + cross(Q, P))$.

Với 2 tối ưu trên, ta có thể cài đặt thuật toán ở subtask 2 với độ phức tạp $O(n)$ hoặc $O(n \log \epsilon^{-1})$, đủ để giải subtask 3.

Bài toán G. MofK và Lắp đặt thiết bị

Ở bài này, ta cần tìm một thứ tự topo của cây $[u_1, u_2, \ldots, u_n]$ sao cho $n \cdot u_1 + (n-1) \cdot u_2 + \cdots + 1 \cdot u_n$ là nhỏ nhất có thể.

Subtask 1: $a_{p_i} \leq a_i$ **với mọi** $2 \leq i \leq n$**.**

Ở subtask này, dễ dàng nhận thấy thứ tự topo tối ưu phải thỏa mãn $a_u \leq a_v$ nếu u đứng trước v trong thứ tự topo. Vì thế, ta có thể sử dụng `std::priority_queue` (hoặc đơn thuần là sắp xếp các chỉ số của dãy a) để tìm đáp án trong độ phức tạp $O(n \log n)$.

Subtask 2: $n \leq 2000$**.**

Ta có khẳng định sau (sẽ chứng minh ở những subtask sau, nhưng không cần thiết cho subtask này): mỗi cây con tồn tại một thứ tự topo tốt nhất của cây con đó, *và thứ tự này không đổi kể cả sau khi thứ tự của cây con này đã được hợp với thứ tự các cây con khác*. Nói cách khác, với mỗi cây con, ta chỉ cần quan tâm tới một thứ tự topo tối ưu duy nhất.

Sử dụng khẳng định trên, thứ tự topo tốt nhất của cây con gốc u bao gồm đỉnh u đầu tiên, sau đó là hợp của các dãy thứ tự topo tốt nhất $t_1, t_2, \ldots, t_k$ của các cây con gốc $v_1, v_2, \ldots, v_k$ là con trực tiếp của u. Nhận xét rằng vì ta có thể xáo trộn các dãy $t_1, t_2, \ldots, t_k$ tùy ý (miễn sao là giữ các phần tử thuộc

cùng dãy theo đúng thứ tự), ta có thể thực hiện quy hoạch động knapsack trên cây để hợp các dãy này như sau:

- Với dãy a và b bất kì, gọi $dp_{i,j}$ là cost nhỏ nhất nếu ta hợp i phần tử đầu của a và j phần tử đầu của b. Như thế thì $dp_{i,j} = \min(dp_{i-1,j} + s \cdot a_i, dp_{i,j-1} + s \cdot b_j)$ với $s = |a| + |b| - (i + j - 1)$.
- Sau khi tính $dp_{|a|,|b|}$, ta có thể quay lại truy vết để đưa ra cách hợp dãy a và b tốt nhất.

Vì độ phức tạp của mỗi bước quy hoạch động này là $O(|a| \cdot |b|)$, bài toán được giải với độ phức tạp $O(n^2)$.

Subtask 3: Tồn tại $3 \le k \le n$ sao cho $p_k = 1$; còn lại, $p_i = i - 1$ với mọi $2 \le i \le n$, $i \ne k$.

Ở subtask này, cây có dạng hai dây nối với nhau tại gốc 1. Bởi thứ tự topo của hai dây này đã được xác định, giả sử hai thứ tự này được biểu diễn bằng hai mảng a và b, ta cần tìm cách hợp hai mảng a và b thành mảng c tối ưu nhất với độ phức tạp hợp lý.

Đầu tiên, nhận thấy rằng với hai phần tử kế tiếp nhau trong c là $c_i = a_x$ và c_{i+1} là b_y, nếu $a_x > b_y$ thì việc hoán đổi vị trí hai phần tử này trong c ($c_i = b_y$, $c_{i+1} = a_x$) sẽ đưa ra kết quả tốt hơn. Ta có thể tổng quát hóa nhận xét này như sau: nếu tồn tại hai dãy con của a và b nằm liên tiếp trong c (tức là dãy c có dạng $[\dots, a_i, a_{i+1}, \dots, a_j, b_k, b_{k+1}, \dots, b_l, \dots]$), nếu trung bình cộng của dãy con từ a lớn hơn trung bình cộng dãy con từ b (tức là $\frac{a_i + a_{i+1} + \dots + a_j}{j-i+1} > \frac{b_k + b_{k+1} + \dots + b_l}{l-k+1}$), việc hoán đổi vị trí hai dãy này sẽ đưa ra dãy c tốt hơn.

Từ đó, ta có nhận xét quan trọng sau: nếu $a_i \ge a_{i+1}$, hai phần tử này sẽ đi kế nhau trong dãy c. Giả sử điều này không đúng, như thế thì dãy c sẽ có dạng $[\dots, a_i, b_j, b_{j+1}, \dots, b_k, a_{i+1}, \dots]$. Xét hai trường hợp sau:

- Trung bình cộng của các phần tử b ở giữa bé hơn a_i. Trong trường hợp này thì dãy $c' = [\dots, b_j, b_{j+1}, \dots, b_k, a_i, a_{i+1}, \dots]$ tốt hơn dãy c.
- Trung bình cộng của các phần tử b ở giữa không bé hơn a_i. Trong trường hợp này thì dãy $c' = [\dots, a_i, a_{i+1}, b_j, b_{j+1}, \dots, b_k, \dots]$ tốt hơn dãy c.

Thế nên ta có thể "dán" hai phần tử a_i và a_{i+1} này thành đoạn con của a; ta liên tục tiền xử lý a như thế này thì cuối cùng ta sẽ thu được một dãy các đoạn con đã được dán lại, và nhận xét quan trọng là trung bình cộng của các phần tử tạo thành các đoạn con này là tăng dần. Tương tự, sau khi tiền xử lý b thì ta cũng thu được một dãy các đoạn con được dán lại có trung bình cộng tăng dần. Việc cuối cùng ta cần làm là ghép các đoạn con được dán này theo thứ tự trung bình cộng tăng dần để thu được dãy c.

Ta đã tìm được cách ghép hai dãy topo a và b trong độ phức tạp $O(|a| + |b|)$. Vì thế, subtask này đã được giải với độ phức tạp $O(n)$.

(Tuy không cần thiết cho thuật toán trên, ta còn có thể chứng minh thêm là thứ tự thực hiện thao tác dán trên không ảnh hưởng tới dãy các đoạn con cuối cùng có thể thu được.)

Subtask 4: không có giới hạn gì thêm.

Do nhận xét trên, mỗi cây con không những chỉ có một thứ tự topo tốt nhất, mà ta có thể biểu diễn thứ tự topo này dưới dạng các đoạn con mà có trung bình cộng tăng dần. Ta có thể sử dụng cấu trúc dữ liệu `std::set` hay `std::priority_queue` để lưu các đoạn này (vì vốn dĩ các cấu trúc dữ liệu này đã sắp xếp giá trị tăng dần). Khi hợp hai cây con lại với nhau, ta có thể hợp cách biểu diễn đoạn lại, áp dụng phương pháp small-to-large để đảm bảo độ phức tạp cuối cùng. Ngoài ra, sau khi hợp các cây con trực tiếp của cây con gốc u, ta cần phải thêm giá trị gốc u vào đầu thứ tự topo; điều này có nghĩa ta cần phải sửa một số đoạn con ở đầu cấu trúc dữ liệu để thu được cách biểu diễn đoạn của toàn bộ cây con gốc u. Vì thế, bài toán đã được giải với độ phức tạp $O(n \log^2 n)$.

(Ngoài ra, việc sử dụng một số cấu trúc dữ liệu đặc thù hỗ trợ merge trong $O(\log n)$ như leftist heap, randomized heap, hay treap có thể đưa độ phức tạp xuống $O(n \log n)$.)

Bài toán H. Kuroni và Những chia sẻ về quá trình ra đề VNOI Cup

Subtask 1: $n \leq 20$, $q \leq 1000$.

Ở subtask này, ta có thể giải bài toán bằng cách quy hoạch động bitmask trước mọi trạng thái của dãy. Gọi dp_{msk} là giá trị lớn nhất có thể đạt được nếu dãy bắt đầu được biểu diễn bằng bitmask msk; ta có thể tính trước mọi đáp án với độ phức tạp là $O(n \cdot 2^n)$. Vì thế, subtask này có thể giải được trong độ phức tạp là $O(n \cdot 2^n + q)$.

Subtask 2: $n, q \leq 1000$.

Nhận xét đầu tiên của chúng ta là nếu ta có thể giải được bài toán với dãy b nhị phân, ta có thể giải được bài toán với dãy b bất kì bằng cách áp dụng phương pháp chặt nhị phân đáp án; ở mỗi lần chặt nhị phân, ta cần truy vấn nếu đáp án của đoạn con trong truy vấn có lớn hơn hoặc bằng x hay không. Ở mỗi phần chặt nhị phân như thế, đoạn con sẽ biến thành dãy nhị phân với mỗi giá trị bằng 0 hoặc 1 tùy thuộc vào việc phần tử ban đầu có lớn hơn hoặc bằng x hay không.

Mục tiêu của chúng ta bây giờ là tìm cách giải mỗi truy vấn cho dãy b nhị phân trong độ phức tạp là $O(n)$. Bởi dãy b nhị phân, với mỗi truy vấn ta chỉ cần kiểm tra xem đoạn con của truy vấn đó có chuyển về 1 được hay không.

Ta có một số nhận xét tham lam sau:

- Nếu vẫn còn tồn tại ba số 0 liên tiếp, ta thực hiện thao tác trên ba số 0 này để biến thành một số 0.
- Còn lại (giả sử mảng vẫn còn giá trị 0) ta muốn thực hiện thao tác trên ba phần tử liên tiếp mà trong đó có ít nhất một giá trị 0 và một giá trị 1. Thao tác này tương đương với việc "chọn hai phần tử 0 và 1 nằm cạnh nhau và xoá hai phần tử này". Vì thế, khi rơi vào trường hợp này, ta tìm cặp 01 đầu tiên và xóa nó đi (ta không tìm cặp 10 đầu tiên vì ở trường hợp 1001..., ta sẽ muốn giá trị 0 đầu tiên nối với ... để có thể tiếp tục tạo ra ba số 0 liên tiếp).

Hai nhận xét trên dẫn tới thuật toán dùng stack sau. Bắt đầu với stack S rỗng, ta duyệt qua các phần tử từ trái sang phải. Với phần tử hiện tại là b_u:

- Nếu stack hiện tại đang rỗng hay kết thúc bằng số 1: đẩy b_u vào đuôi stack.
- Nếu stack kết thúc bằng một số 0: nếu $b_u = 1$, xóa số 0 ở đuôi stack; ngược lại, đẩy $b_u = 0$ vào stack.
- Nếu stack kết thúc bằng hai số 0: bất kể b_u là gì, xóa số 0 ở đuôi stack.

Dễ dàng nhận thấy đáp án là `YES` chỉ ở các trường hợp sau:

- Ở bất kì thời điểm nào, $S = [1, 1]$. Đây là vì ta có thể xử lý các phần tử đằng sau để biến về giá trị bất kì x, rồi thực hiện thao tác cuối cùng với dãy $[1, 1, x]$.
- Sau khi chạy hết dãy, $S = [1]$.

(Lưu ý là nếu stack kết thúc là $S = [1, 0, 0]$ thì đáp án là `NO`.)

Ta có thuật toán với độ phức tạp $O(nq \log(\max b))$ hay $O(nq \log n)$.

Subtask 3: $n, q \leq 100\,000$.

Tương tự như subtask 2, ta sẽ tìm cách giải bài toán với mảng b nhị phân trước; sau đó, bài toán có thể được giải bằng việc áp dụng phương pháp chặt nhị phân song song để tìm đáp án. Mục tiêu của chúng ta là tìm cách giải một đoạn con chỉ toàn giá trị nhị phân trong độ phức tạp $O(\log n)$ sử dụng segment tree.

Tuy thao tác được mô tả trong subtask 2 không có tính chất kết hợp nên ta không thể áp dụng segment tree ngay lập tức, ta có một số nhận xét sau:

- Ở mọi thời điểm, toàn bộ giá trị 1 của S đứng trước toàn bộ giá trị 0.
- Ở mọi thời điểm, S chứa nhiều nhất hai giá trị 0 ở đuôi.

Bởi thế, ta chỉ cần quan tâm tới 9 trạng thái (thật ra là 7) sau của S, được biểu diễn bằng hai số sau:

- Số lượng số 1 ở đầu S (bằng 0, bằng 1, hoặc lớn hơn hoặc bằng 2).
- Số lượng số 0 ở đuôi S (bằng 0, bằng 1, hoặc bằng 2).

Dựa trên nhận xét này, ta có thể lưu 9 (hoặc 7) thông tin sau trong mỗi nút segment tree: nếu stack S bắt đầu với trạng thái i ($1 \leq i \leq 9$), thì sau khi thêm các phần tử trong đoạn theo thứ tự, stack S sẽ kết thúc với trạng thái nào. Sử dụng các thông tin này, ta có thể giải mỗi đoạn con nhị phân trong độ phức tạp $O(\log n)$. Cùng với việc chặt nhị phân song song, bài toán đã được giải với độ phức tạp $O((n+q)\log^2 n)$.