Подзадача 1: $N \leq 2.5 \times 10^1$

За тази подзадача е предвидено решение със сложност $O(2^N)$ с пълно изчерпване, за всяка точка пробваме двата варианта: да е в A или да е в B. За всяко разпределение смятаме търсената сума и връщаме минимума от откритите възможности. Както и във всички по-нататъшни решения, трябва да внимаваме и A, и B да са непразни.

Подзадача 2: $N \leq 8.5 \times 10^2$

За тази подзадача е предвидено решение със сложност $O(N^3)$. Нужно е да направим наблюдението, че винаги има оптимално решение, в което множеството A се състои от непрекъсната редица точки (когато те са сортирани по X, както е във входа), а Y се състои от всички точки преди тази редица и всички точки след нея, т.е. от даден префикс и даден суфикс от точки. Има $O(N^2)$ възможни начина да фиксираме двата края на A и за всеки от тях линейно смятаме стойността на търсената сума.

Подзадача 3: $N \leq 1.9 \times 10^4$

За тази подзадача е предвидено решение със сложност $O(N^2)$. Трябва да видим, че можем да смятаме широчината на A за дадени лява и дясна граница в $O(1)$, като акумилираме минимума и максимума в A докато местим дясната граница (а ги инициализираме, когато преместим лявата граница). В същото време можем да акумилираме минимума и максимума в префиксната част на B докато местим лявата граница, а за суфиксната му част можем да прекомпютнем всички суфиксни минимуми и максимуми на Y-ци от всяка позиция нататък. Така за всички леви и десни граница смятаме стойността на търсената сума в $O(1)$.

Подзадача 4: $N \leq 9.0 \times 10^4$

За тази подзадача са предвидени всякакви решения със сложност $O(N\sqrt{N})$. Такива са разгледани на идейно ниво, но тъй като никое реално не е по-лесно от решението със сложност $O(N \log N)$, няма имплементирано такова авторово решение.

Подзадача 5: $N \leq 6.1 \times 10^5$

За тази подзадача е предвидено решение със сложност $O(N \log N)$. Изисква се да забележим, че за всяка комбинация от лява и дясна граница сме в един от четири случая: 1) и минимума, и максимума на B са в суфикса; 2) минимума е в префикса, а максимума в суфикса; 3) минимума е в суфикса, а максимума в префикса; 4) и минимума, и максимума са в префикса. Сега нека разглеждаме фиксирана дясна граница R. Лесно можем с двоично търсене да открием, за коя стойност на лявата граница се сменя къде е минимума и за коя ѝ стойност се сменя къде е максимума. Нека това са P_1 и P_2 и нека приемем, че $P_1 \leq P_2$. Т.е. за леви граници L в $[0; P_1)$ сме в случай 1), за L в $[P_1; P_2)$ сме в случай 2), а за L в $[P_2; R)$ сме в случай 4). (Ако $P_2 < P_1$ средният случай става случай 3).) Сега за всеки от четирите възможни случая, можем да разделим стойността на търсената сума на два компонента: членове зависещи само от R и членове зависещи само L. За фиксирано R първият компонент е константен, а вторият е променлив. Т.е. за всеки от трите интервала, търсим това L в него, което минимизира вторият компонент на съответния случай. Това всъщност е RMQ (Range Minimum Query) и за тази задача най-удачно е да го правим със сегментно дърво. (Всъщност лесно се вижда, че за случай 1) не ни трябва сегментно дърво, защото отговорът винаги е най-десният елемент в интервала.)

Подзадача 6: $N \leq 2.0 \times 10^6$

За тази подзадача е предвидено решение със сложност $O(N)$. Основната идея за решението е същата както това за предната подзадача. Нека първо видим как да намираме P_1 и P_2 в $O(1)$ за дадено R. Всъщност можем да видим, че суфиксният минимум намалява като R намалява, а префиксният максимум намалява като L се увеличава, т.е. можем да използваме two pointers: започваме с R най-вдясно и го местим наляво, като на всяка стъпка местим P_1 наляво докато префиксният минимум не достигне суфиксния такъв. Еквивалентно и с P_2. Естествено $P_1, P_2 \leq R$, т.е. трябва да ги намаляваме с по едно на всяка стъпка ако са достигнали R.

Сега да видим как да отговаряме на RMQ-тата в $O(1)$. Естествено, на теория можем да използваме подход през LCA с Cartesian Trees, но това е доста бавно като константа и доста неприятно за писане. Вместо това предлагаме два по-приятни подхода. Първият се базира на това, че заявките са „горе-долу монотонни“ надясно, т.е. в общи линии и левите им краища, и десните им краища се движат само надясно. Ако това винаги беше вярно, можеше да използваме трик за монотнно RMQ с deque. Тъй като обаче и P_1 и P_2 започват да се движат наляво, когато достигнат R, първо ще си прекомпютнем стойностите на P_1 и P_2 за всяка стойност на R и ще видим кога първо се срещат. Нека тези точки са Q_1 и Q_2 и нека $T_1 = \min(Q_1, Q_2)$ и $T_2 = \max(Q_1, Q_2)$. Сега можем да забележим, че RMQ-тата за случаи 2) и 3) са монотонни отляво на T_1, а тези за случай 4) – отляво на T_2. От друга страна, левите краища на нито едно трите вида RMQ-та никога не надвишават T_1 и T_2 съответно. Това ни позволява да разделим всяко куери на две подкуерита: едно отляво на съответното T и едно отдясно на него. Лявото подкуери си е точно монотонно RMQ, на което отговаряме с deque, а дясното е просто минимум от съответното T до съответното P, т.е. префиксен минимум от T-то, всички от които можем да си прекомпютнем. Така отговаряме на всяко от шестте подкуерита в $O(1)$, комбинираме ги по двойки в отговори на трите желани куерита и от там процедираме както в решението на предната подзадача.

Друг подход със сложност $O\big(\alpha(N)\big) \approx O(1)$ се възползва от това, че на RMQ-тата не е нужно да бъде отговаряно онлайн, т.е. можем първо да си генерираме всички куерита по случаи (като за всяко запазваме какво трябва да направим с резултата му) и после да им отговорим в друг, по-удобен ред. Един много лесен алгоритъм за RMQ, който се възползва от това, е такъв базиран на Union-Find/Disjoint-Set Union. Детайлите му може да намерите онлайн или в авторовото решение.

Автор: Емил Инджев
Начална идея: Радостин Чонев