Климатик

Решение

Нотация, която използвам в целия анализ:

- Страната на чудесата се разглежда като граф дърво.
- Пътят от връх u до връх v се обозначава с $[u \rightsquigarrow v]$. Когато началният или крайният връх не се включват в пътя, пътят се обозначава съответно със $(u \rightsquigarrow v]$ или $[u \rightsquigarrow v)$ (стандартна нотация за интервали).
- За сбитост общатата промяна в температурата за даден път се нарича *сума* на този път и се обозначава съответно със $sum[u \rightsquigarrow v]$, $sum(u \rightsquigarrow v]$ или $sum[u \rightsquigarrow v)$.

Разгледаните решения съм подредил по подходите, които използват, а не по точките, които очаквам да получат.

Решение за $O(N^2 + Q)$

За фиксиран начален връх a, можем да отговорим на всички заявки за пътища $[a \rightsquigarrow v]$ за $O(N)$: Обхождаме дървото (примерно в дълбочина) и за всеки връх v намираме $sum[a \rightsquigarrow v]$ и броя на върховете $u \in [a \rightsquigarrow v]$ със $sum[a \rightsquigarrow u] > 0$. Прилагайки този алгоритъм за всеки връх a, можем да отговорим за всички възможни заявки за (N^2). Това решение е реализирано в AC-N2.CPP и се очаква да получи 11 точки.

Offline подход

В контекста на задачи със заявки, решение се нарича *offline*, ако разглежда всички заявки наведнъж.

Решение за $v_i = u_i + 1$

Тук дървото се свежда до редица от върхове. Без ограничение на общността, нека $a_i \leq b_i$: за да отговорим на заявките с $b_i \leq a_i$, можем приложим същия алгоритъм върху обърнатата редица.

Нека r е връх номер 1, началото на редицата. Отговорът на заявка i е броят на върховете $v \in [a_i \rightsquigarrow b_i]$ със $sum[a_i \rightsquigarrow v] > 0$. Този брой е равен на броя $v \in [r \rightsquigarrow b_i]$ със $sum[r \rightsquigarrow v] > sum[r \rightsquigarrow a_i)$ минус броя $v \in [r \rightsquigarrow a_i)$ със $sum[r \rightsquigarrow v] > sum[r \rightsquigarrow a_i)$. Използвайки това, свеждаме всяка заявка до две *опростени* заявки с начало r.

За да отговорим на тези заявки, ще използваме структура от данни *nъm*, която:

- Започва с празен път.
- Поддържа добавяне на връх в края на пътя за $O(\log N)$.
- При текущ *nъm* $[s \rightsquigarrow t]$ и зададено x, намира броя на върховете $v \in [s \rightsquigarrow t]$ със $sum[s \rightsquigarrow v] > x$ за $O(\log N)$.

Структурата *nъm* може да се имплементира чрез дърво на Фенуик, в което на позиция i се поддържа броят на градовете $v \in [s \rightsquigarrow t]$ със $sum[s \rightsquigarrow v] = i$. Така, броят v със $sum[s \rightsquigarrow v] > x$ може да се намери чрез заявка са сума, отнемаща $O(\log N)$ време. За да избегнем отрицателни индекси, можем да добавим N към всеки индекс.

Отговаряме на опростените заявки, като позледователно добавяме върховете $1, 2, ..., N$ към първоначално празната структура *nъm*; след като добавим връх i, отговаряме на всички *опростени* заявки за $[r \rightsquigarrow i]$.

Общата сложност на това решение е $O((N + Q) \log N)$. То е реализирано в LINE.CPP и се очаква да получи 30 точки.

Решение за $O((N + Q) \log N)$

Избираме връх r за корен на дървото. Нека l_i е най-близкият общ предшественик на a_i и b_i (lowest common anecstor или LCA). Чрез стандартни алгоритми можем да намерим всички l_i за $O((N + Q) \log N)$ онлайн или за $O((N + Q) \cdot \alpha(Q, N))$ чрез офлайн алгоритъма на Тарджан.

Броят на върховете $v \in [a_i \rightsquigarrow b_i]$ със $sum[a_i \rightsquigarrow v] > 0$ е равен на броя $v \in [a_i \rightsquigarrow l_i]$ със

$sum[a_i \rightsquigarrow v] > 0$ плюс броя $v \in (l_i \rightsquigarrow b_i]$ със $sum(l_i \rightsquigarrow v] > -sum[a_i \rightsquigarrow l_i]$. Използвайки това, както и подходите от решението за редица, свеждаме всяка заявка $[a_i \rightsquigarrow b_i]$ до четири *опростени* заявки, съответно за пътищата $[a_i \rightsquigarrow r]$, $(l_i \rightsquigarrow r]$, $[r \rightsquigarrow b_i]$ и $[r \rightsquigarrow l_i]$.

Обхождаме дървото в дълбочина, поддържайки в *стек* пътя $[r \rightsquigarrow u]$ от корена r до текущия връх u. При влизане във u, отговаряме на всички *оппростени* заявки за $[r \rightsquigarrow u]$ и $[u \rightsquigarrow r]$. За тази цел е необходимо *стек* да подържа още две операции:

- Да премахва последния връх в пътя за $O(\log N)$.
- При текущ път $[s \rightsquigarrow t]$ да отговаря и на заявки за $[t \rightsquigarrow s]$ за $O(\log N)$.

Тези операции също могат да се поддържат чрез дърво на Фенуик: Премахването на последния връх се свежда до изваждане на 1 от стойността на индекс $sum[s \rightsquigarrow t]$. За да намерим броя $v \in [s \rightsquigarrow t]$ със $sum[t \rightsquigarrow v] > x$, използваме, че този брой е равен на броя v със $sum[s \rightsquigarrow v) < sum[s \rightsquigarrow t] - x$, което отново се свежда до заявка за сума в дървото на Фенуик.

Общата сложност на това решение $O((N + Q) \log N)$: $O(N \log N)$ за поддържане на *стек* при обхождането в дълбочина и $O(Q \log N)$ за отговаряне на заявките. Това решение е реализирано в AC.CPP и се очаква да получи пълен брой точки.

Чрез персистентна имплементация на *стек* това решение може да бъде модифицирано да работи и *online*.

Online подход

За разлика от *offline* решенията, *online* решенията отговарят независимо на всяка заявка.

Решение за $u_i = v_i - 1$

Построяваме интервално дърво за редицата от върхове. Чрез него разделяме всяка заявка на $O(\log N)$ интервала, съответстващи на върхове в интервалното дърво. За всеки такъв интервал ще смятаме бързо отговора:

Нека разглеждаме заявка $[a_i \rightsquigarrow b_i]$ $(a_i \leq b_i)$ и интервал $[s \rightsquigarrow t]$ (интервалите са и пътища): искаме да намерим броя $v \in [s \rightsquigarrow t]$ със $sum[a_i \rightsquigarrow v] > 0$. Тъй като това условие е еквивалентно на $sum[s \rightsquigarrow v] > -sum[a_i \rightsquigarrow s)$, този брой може да бъде сметнат за $O(\log N)$, ако за всеки връх в интервалното дърво построим структурата *стек* от offline подхода. Така получаваме решение със сложност $O((N + Q) \log^2 N)$.

Тъй като тук *стекът* не е необходимо да поддържат промени, дървото на Фенуик може да бъде заменено с префиксни суми. Така получените *статични стекове* могат да отговарят на заявки за $O(1)$, което води до решение със сложност $O((N + Q) \log N)$.

За да се справим със заявки с $b_i < a_i$ можем или да използваме наблюденията за обратни заявки от пълното *offline* решение, или да поддържаме по два *статични стека* за всеки връх в интервалното дърво – за $[s \rightsquigarrow t]$ и за $[t \rightsquigarrow s]$.

Това решение е реализирано в LINE-ONLINE.CPP и се очаква да получи 30 точки.

Решение за $O(N + Q\sqrt{N})$

Избираме корен на дървото r и параметър $K \approx \sqrt{N}$. За всеки връх v с дълбочина $d(v) \bmod K = 0$ построяваме *статичния стек* $[v \rightsquigarrow v')$, където v' е K-тият предшественик на v в дървото. Това отнема $O(N)$ време и памет.

Използвайки тези *статични стекове*, всяка заявка $[a_i \rightsquigarrow b_i]$ може да бъде разбита на $O(\frac{|a_i \rightsquigarrow b_i|}{K} + K) = O(\sqrt{N})$ участъка, за всеки от които можем да намерим броя v със $sum[a_i \rightsquigarrow v] > 0$ за $O(1)$. Така отговаряме на всяка заявка за $O(\sqrt{N})$.

Това решение със сложност $O(N + Q\sqrt{N})$ е реализирано в AC-SQRT.CPP. В зависимост от имплементацията, подобни решения се очаква да получат 27–42 точки.

Решение за $O(N \log N + Q \log^2 N)$

Използвайки *heavy-light decomposition*, можем да генерализираме решението за редици до произволни дървета. Това изисква $O(\log^2 N)$ време на заявка, водещо до обща сложност $O(N \log N + Q \log^2 N)$. В подзадачата за редица обаче дървото има само един тежък път и сложността е отново $O((N + Q) \log N)$.

Това решение е реализирано в AC-HLD.CPP. В зависимост от имплементацията, подобни решения се очаква да получат 72–84 точки.

Автор: Александър Кръстев