Тагове	На пълното решение	На подзадачите
	Линеализация на дърво Офлайн обработка на заявки Дърво на Фенуик	Small to Large Treap Merge Sort Tree

Анализ

Подзадача №1

Както винаги, оставих подзадача с тестовите примери, за да има по-добра обратна връзка от системата.

Подзадача №2

Може да намерим реда на върховете в пръчката. Тогава задачата ни се свежда до редица, в която се търси броят инверсии, тоест при редица $a_1, a_2, \ldots, a_N$ се търсят броя двойки (i, j), за които $a_i > a_j$. Най-лесно е тази бройка да се намери чрез квадратно обхождане.

Постигната сложност: $O(N^2)$

Имплементация: `inversions_7p.cpp`

Подзадача №3

Обстановката в подзадачата е стандартната задача за брой инверсии със стандартните ограничения. Тя може да се реши по два начина – чрез Merge sort и чрез дърво на Фенуик. Може да се упражните да я решите тук.

Постигната сложност: $O(N \log_2 N)$

Имплементация: `inversions_32p.cpp`

Подзадача №4

В дърво, задачата за брой инверсии се свежда до N въпроса от вида „Колко числа в поддървото на връх x са $< x$“. Най-лесния начин за намирането на тези бройки е да се направят N обхождания в дълбочина.

Постигната сложност: $O(N^2)$

Имплементация: `inversions_15p.cpp`

Подзадача №5

За подзадачата ще представя три възможни решения.

Решение със Small to Large

Една идея е когато обхождаме дървото и стигнем до връх да поддържаме дърво на Фенуик, в което всички налични върхове в поддървото му са ъпдейтнати с 1. Тогава броят върхове в поддървото, които са $<$ от текущия може да се намерят с 1 заявка към Фенуика. Въпроса е как го поддържаме. Нека правим обхождане в дълбочина, в което индуктивно когато влезнем във връх дървото да е чисто, а когато напуснем връх, всички върхове в поддървото му (включително и него) да са ъпдейтнати с единици във Фенуика. Първоначално ще обходим всички деца, освен това с най-голямото поддърво и когато излезнем от някое от тях ще пуснем още едно обхождане, което ще почисти Фенуика от единиците от поддървото на текущото дете. След това ще пуснем обхождане за най-голямото дете и **няма** да премахваме единиците. Следва обхождане по останалите деца, като ги добавяме към Фенуика. Тогава Фенуика ще съдържа всички върхове от поддървото и ще може да направим заявката, която споменах отгоре. Защо това е бързо? Може да забележите, че когато връх се премахва и добавя повторно към Фенуика, големината на поддървото в което се намира се увеличава поне 2 пъти, тоест всеки връх от дървото ще бъде добавен и премахнат най-много $\log_2 N$ пъти от Фенуика, защото иначе поддървото в което се намира ще има големина $> N$. Този подход е известен като Small to Large и е навсякъде около нас (примерно в DSU).

Постигната сложност: $O(N \log_2^2 N)$

Имплементация: `inversionsSmallToLarge_80p.cpp`

Решение със Treap

Един алтернативен подход е вместо Small to Large да поддържаме трийпове със същата информация за поддърветата на върховете като Фенуика. Когато достигнем до някой връх правим няколко merge-а на трийповете на децата, като получаваме трийпа на текущия връх. Как мърджваме два трийпа, които са с произволни елементи? Ако в задачата се изисква само да се мърджват трийпове, ние може пак да приложим Small to Large подход, като insert-нем всеки връх от по-малкия трийп към по-големия за сумарно $O(N \log_2^2 N)$ сложност, но това би било бавно. Поради вероятно много добре обясними причини, следния подход се държи за $O(\log_2 N)$:

- Нека да трябва да мърджнем два трийпа, съответно X и Y, като нека Б.О.О. приоритета на корена на X е $\geq$ приоритета на корена на Y.
- Тогава сплитваме Y спрямо стойността на корена на X. Нека сме го сплитнали на L и R, като L е с по-малките елементи, а R – с по-големите. Тогава фиксираме корена на X за корена на мърджнатия трийп и извикваме рекурсивно същата процедура за $(X.L, L)$ и $(X.R, R)$, където $X.L$ и $X.R$ са съответно лявото и дясното дете на X.

Остана само да правим заявки за броя числа в трийп $< x$. Това може да стане за $O(\log_2 N)$.

Постигната сложност: $O(N \log_2 N)$ с голяма константа.

Имплементация: `inversionsTreap_80p.cpp`

Решение с Merge Sort Tree

Най-близко до пълното решения от изброените е текущото. Нека направим обхождане в дълбочина на дървото, започващо от корена му, като запишем върховете в редица в реда им на срещане. По този начин ще получим *линеализация* на дървото, тоест редица, в която всяко поддърво се явява подмасив. Линеализацията на дърво е по-известна като ойлеров тур. Така си сведохме задачата до множество заявки „колко числа от l до r са $< x$“. Това може да стане онлайн относително безпрепядствено чрез Merge Sort дърво.

Постигната сложност: $O(N \log_2^2 N)$

Имплементация: `inversionsMST_80p.cpp`

Подзадача №6

В решението с Merge Sort дърво отговаряме на въпросите онлайн, но няма никаква причина за това. Как може да ни помогне офлайн отговаряне? Всеки въпрос от вида „колко числа от l до r са $< x$“ може да го разбием на два префиксни въпроса – преброяваме колко числа от 1 до r са $< x$, както и колко числа от 1 до $l - 1$ са $< x$, като вадим едното от другото. Как може да отговаряме на тези заявки? Познахте – с дърво на Фенуик. Първо ще отговорим на заявките от 1 до 1, после от 1 до 2, после от 1 до 3, …, накрая от 1 до N, като във Фенуика поддържаме числата, които са в текущия префикс. Тогава отговорът на една префиксна заявка за броят числа $< x$ е равен на броят единици във фенуика до $x - 1$-вата позиция, което може лесно да се отговори.

Постигната сложност: $O(N \log_2 N)$, с малка константа

Имплементация: `inversions_100p.cpp`

Автор: Борис Михов

П.П: Може да се чудите защо решението за $O(N \log_2 N)$ сложност с трийп не изкарва 100 точки. По принцип височината на трийп трудно ще бъде точно $\log_2 N$, тя ще бъде $\approx \log_2 N$. Също, една $O(\log_2 N)$ операция на трийп е с доста голяма константа, правят се множество проверки и изчисления, дори и в сегашния му елементарен вид. А и Фенуика е с един от най-бързите логаритми – той прави приблизително $\frac{\log_2 N}{2}$ операции на заявка (интересен факт е, че теоретично едно Сегментно дърво е 8 пъти по-бавно от Фенуик ☺). Тайм лимита не беше твърде щедър, за да се разграничат $O(N \log_2^2 N)$ решенията от авторовото. По принцип решението с трийп на тази задача се пише само за упражнение, но ако някой го е направил и на състезанието, поздравления.