Тагове	На пълното решение	На подзадачите
	Сегментно Дърво Двоично търсене	STL Set Linked List

Анализ

Подзадача №1

В тази подзадача е тестовият пример. Тя е за обратна връзка от системата.

Подзадача №2

Ще намерим пермутацията постепенно, като намираме стойностите на елементите отляво-надясно. Нека сме намерили първите m елемента от пермутацията и при сортиране техните стойности са $b_1 < b_2 < \cdots < b_m$. Ако разгледаме стойностите на първите $m+1$ елемента сортирани $c_1 < c_2 < \cdots < c_{m+1}$, то ние ще трябва да намерим на коя позиция сме „вмъкнали" a_{m+1} в c. По този начин ще може с един въпрос да намерим стойността на a_{m+1}. Всъщност, ако a_{m+1} е на x-та позиция в c, то за всяко $x \leq j \leq m$, $c_j \neq b_j$, а за всяко $1 \leq i \leq x-1, c_i = b_i$. Така може с вложен цикъл да търсим на коя позиция е a_{m+1}. Елементите може да поддържат сортирани в STL-ски set или linked list, което не би било проблем, защото и двете имат линейна сложност за обхождане на всички елементи в тях.

Постигната сложност: $O(N^2)$.

Имплементация: `FirstLinkedList_26p.cpp` и `FirstSet_26p.cpp`

Подзадача №3

Трябва да се забележи, че във всеки суфикс на пермутацията числата в него са с поредни стойности. Така всеки префикс се състои от числа със стойности $\{1,2,\ldots,x,y,y+1,\ldots,N-1,N\}$ за някакви $1 \leq x < y \leq N$. Отново намираме числата отляво-надясно, като поддържаме стойносттите на x и y. Ако знаем стойносттите на x и y за някоя позиция i, то елемента на $i+1$-ва позиция би бил равен на $x+1$ или $y-1$. Заради това се пита въпрос за стойността на $x+1$-вия най-голям елемент измежду първите $i+1$ числа.

Постигната сложност: $O(N)$.

Имплементация: `Second_17p.cpp`

Подзадача №4

Нека се върнем на идеята от първата позиция. Ние целим да намерим кой по големина е $m+1$-вия елемент. Вместо с вложен цикъл е по разумно да се направи с двоично търсене. Нека можем да намерим кой е k-тия елемент по големина измежду първите x числа от редицата. Нека

търсим кой е елементът на $m+1$-ва позиция. Тогава може да се приложи следното двоично търсене:

- Нека знаем, че a_{m+1} в сортировката се намира на позиция $> l$ и $\leq r$. Първоначално $l = 0$ и $r = m+1$. Нека $mid = (l+r)/2$.
- Намираме кой е mid-тия елемент по глемина измежду първите m елемента на редицата.
- Намираме кой е mid-тия елемент по глемина измежду първите $m+1$ елемента на редицата чрез въпрос.
- Ако двата намерени елемента са равни, то a_{m+1} е на поне $mid+1$-ва позиция в сортировката на първите $m+1$ елемента от пермутацията, следователно $l \coloneqq mid$.
- Ако двата намерени елемента са различни, то a_{m+1} е на най-много mid-та позиция в сортировката на първите $m+1$ елемента от пермутацията, следователно $r \coloneqq mid$.
- Когато двоичното търсене приключи, ще знаем, че a_{m+1} е на r-та позиция в сортировката на първите $m+1$ елемента от пермутацията.

Сега остава въпросът – как да намираме кой елемент е k-ти по големина.

Намиране на k-то по големина число

Това може да стане с още едно двоично търсене. В дърво на Фенуик си поддържаме 1-ци за числата, които са измежду първите m от редицата и 0 за останалите. Нека k-тото число по големина е равно на x. Тогава броят на елементите, означени с 1-ца, по-малки или равни на $x-1$ ще бъде точно $k-1$, а за тези по-малки или равни на x, ще са точно k. Заради това лесно може да се направи двоично търсене. Така сложността е $O(N\ log_2^3\ N)$, защото има по един логаритъм за двете двоични търсения и един за дървото на Фенуик.

Постигната сложност: $O(N\ log_2^3\ N)$.

Имплементация: `Third_76p.cpp`

Подзадача №5

За да се изкарат 100 точки може да се оптимизира намирането на k-тия по големина елемент. Вместо с двоично търсене и дърво на Фенуик, ние ще използваме сегментно дърво и ще се „спускаме“ в него. В сегментното дърво ще пазим същите стойности които пазихме и във дървото на Фенуик. означава Нека знаем, че k-тия елемент е в поддървото на x-тия връх в сегментното дърво. Нека броят на единиците в поддървото на лявото дете на x е $leftCnt$. Тогава, за да намерим k-тия по големина, трябва да разгледаме два случая:

- Ако $leftCnt \geq k$ единици, то k-тия елемент ще се намира в лявото поддърво, заради това ще търсим k-тия по-големина елемент в поддървото на лявото дете на x.

- Ако $leftCnt < k$ единици, то k-тия елемент ще се намира в дясното поддърво, заради това ще търсим $k - leftCnt$-тия по-големина елемент в поддървото на дясното дете на x.

Така за да намерим k-тия по големина елемент ще ни трябват $O(log_2 N)$ стъпки.

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

Имплементация: `binSearchSegTree_100p.cpp`

П.П: Тази оптимизация може да се реализира и с дърво на фенуик, но това не се очакваше на контролното състезание.

Допълнително решение

Вместо да правим двоично търсене и спускане в сегментно дърво, ще можем да направим само едно спускане в сегментното. Това би станало като за всяко поддърво освен сума си поддържаме и максимален елемент, който е отбелязан с единица. Когато се спускаме ще си поддържаме $smaller$, равно на броя на елементите, означени с единица, които са с по-малък индекс от най-левият елемент в поддървото в сегментното дърво. Тогава трябва да се разгледат няколко случая:

- Няма елементи означени с единица в текущото поддърво, следователно $a_{m+1} = question(m + 1, smaller + 1)$.
- В лявото поддърво няма елементи означени с единица, следователно търсим отговорът в дясното поддърво.
- Нека $number = question(m + 1, smaller + \text{броя\_на\_единици\_в\_лявото\_поддърво})$.
- Тогава ако $\text{максималният\_елемент\_в\_лявото\_поддърво} = number$, то отговорът се съдържа в дясното поддърво.
- В противен случай отговорът се съдържа в лявото поддърво.

Постигната сложност: $O(Nlog_2 N)$.

Идея за решението: Илиян Йорданов

Имплементации: `author.cpp` и `present_iliyan.cpp`

П.П: Можех лесно да разгранича допълнителното решение от предвиденото решение за 100 точки, но реших да съм щедър... Щедър към мозъчните си клетки.

Автор: Борис Михов