Тагове	На пълното решение	На частичните решения
	Грийди Сегментно дърво	Backtracking Динамично с битмаска

Анализ

Подзадача №1

Както винаги оставих подзадача с тестовите примери за обратна връзка от системата.

Подзадача №2

Може да разгледаме всяка една възможна игра чрез `backtracking` и да намерим резултата.

Постигната сложност: $O(QN!)$

Имплементация: -

Подзадача №3

Можем да подобрим нашия `backtracking` като му направим мемоизация. Така може да направим динамично с битова маска, която пази за всеки един елемент дали е бил изтрит.

Постигната сложност: $O(Q2^N \times N)$

Имплементация: -

Оптимална игра

Нека разгледаме една игра, в която Сашка и Ибоб трият числа от редицата $a_1, a_2, \ldots, a_N$, като сме я пермутирали по такъв начин, че Сашка първо изтрива a_1, впоследствие Ибоб изтрива a_2 и т.н. За да пресметнем резултата, ние може да преброим за всеки елемент колко допринася към него, т.е. броят срещания в сборовете на Сашка – броят срещания в сборовете на Ибоб. Сборовете на Сашка се формират от $a_i + a_{i+1} + \ldots + a_N$ за нечетно i, а тези на Ибоб – за четно i. Като ги извадим се получава $a_1 + a_3 + a_5 + \ldots$, което ще бъде равно на резултата на играта.

Сега въпросът е коя пермутация на a е тази, по която ще се играе. Всъщност се оказва, че винаги е оптимално за играчите да изтрият най-големият възможен елемент. Нека $a_1 \geq a_2 \geq \ldots \geq a_N$. Тогава, ако на всеки ход Сашка взима най-големия елемент, то тя ще получи резултат $b_1 + b_3 + b_5 + \ldots$, като задължително $b_1 \geq a_1; b_3 \geq a_3; \ldots$. Аналогично, ако Ибоб винаги взима най-големият елемент, то за резултат $b_1 + b_3 + b_5 + \ldots$, $b_1 \leq a_1; b_3 \leq a_3; \ldots$. Така Сашка винаги може да си подсигури, че резултатът е $\geq a_1 + a_3 + a_5 + \ldots$, а Ибоб винаги може да подсигури, че резултатът е $\leq a_1 + a_3 + a_5 + \ldots$. Следователно при оптимална игра от двамата играчи, резултатът винаги ще е $= a_1 + a_3 + a_5 + \ldots$.

Подзадача №4

Наивно може да приложим горната стратегия и винаги да изтриваме най-големия елемент.

Постигната сложност: $O(N^2)$

Имплементация: - .

Подзадача №5

Може да приложим бърза сортировка на числата, след която линейно да намерим $a_1 + a_3 + a_5 + ...$

Постигната сложност: $O(QNlog_2N)$

Имплементация: `game_25p.cpp` .

Подзадача №6 **и** 7

Note към читателя – решението на подзадачата **не** трябва да се измисля по време на състезание. Пълното решение е по-стандартно за такива постановки. Споменал съм техниката в анализа, защото е възможно да бъде полезна на задача с по-трудни заявки.

Сведохме задачата до поддържане на определения сбор за бързо време. Поради потенциално големите стойности на числата в редицата, ние може да приложим компресия. Ще я направим по следния начин – разглеждаме множеството на всички числа, които някога са присъствали в редицата и за всяко намираме кое по големина е. Така всяко число мислено го заместваме с неговата позиция в сортировката, като използваме реалната му стойност единствено при изчисление на отговора.

Така може да си представим числата като последователност от $N + Q$ бита, от които точно N са единици. Съответно присъстващите елементи в редицата са с бит 1, а всички останали – с бит 0. Стойностите в тази последователност са ”сортирани” ненамаляващо.

Нека приложим коренова декомпозиция на тази последователност. Нека за всеки бъкет поддържаме сборовете на елементите на четни и нечетни позиции, както и бройката на елементи на четни и нечетни позиции.

Трябва да поддържаме два ъпдейта – обръщане на 0 към 1 (появяване на число в редицата), както и обръщане на 1 към 0 (премахване на число в редицата). Те са аналогични. Построяваме наново бъкета, в който има промяна (за $O(\sqrt{N+Q})$ време). Изчисляването на отговора се постига чрез следния подход – обхождаме бъкетите отляво-надясно, като поддържаме четността на обходените елементи. Така ще знаем кой сбор търсим за текущия бъкет – на четните или нечетните.

Това е решението на задачата kth от `A НОИ 2 2025`, но в случая решаваме подслучая за $k = 2$. Еквивалентен подход решава задачата за всяко k и изкарва 100 точки на нея. За по-общата задача, решението на 8 и 9 подзадача не е приложимо.

Постигната сложност: $O((N+Q)\sqrt{N+Q})$

Имплементация: - .

Подзадача №8 **и** 9

Наново компресираме и построяваме редицата с 0 и 1. Построяваме и сегментно дърво върху нея, което поддържа абсолютно същите неща, които поддържаме и в кореновата декомпозиция – сборовете на елементите на четни и нечетни позиции спрямо началото на интервала на текущия връх в сегментното, както и бройката на четните и нечетните елементи. Мърджа в сегментното е еквивалентен на мърджа на кореновата декомпозиция – проверявайки на четността на елементите в лявото поддърво намираме търсеният сбор в дясното. За повече детайли – погледнете имплементацията.

Постигната сложност: $O((N+Q)log_2(N+Q))$

Имплементация: `game_75p.cpp` .

Автор: Борис Михов