**АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА**

**ЕТАЖЕРКА**

Да обърнем внимание, че за решаването на задачата, освен ограничението по време, съществена роля има и броя на заявките към робота за разместване на тетрадки, който не трябва да надхвърля 200 000.

Големият проблем в тази задача беше написването на бърз грейдър, който да отговаря на заявките към робота. Той трябва да може бързо да пресмята броя на инверсиите на пермутации на 100 000 елемента при 200 000 заявки към робота (разбира се използвайки това, че при всяка следваща заявка са разменени само два елемента на предходната пермутация). Голямата заслуга тук е на Енчо Мишинев, който се справи блестящо с тази задача.

**Решение, основано на сортиране**

Нека *p = (p1, p2,…..,pn)* е първоначалното подреждане (което, разбира се, Вие не знаете). С помощта на две викания на функция *bookswap(i,j)* може да се установи дали *pi*  e по-малко от *pj*. Като имаме такава функция за сравнение, можем с произволен метод на сортиране (нормално е да се избере бързо сортиране, а не квадратична сортировка) да се получи търсената начална пермутация, сортирайки базовата пермутация *(1, 2, 3,…..,n)* и вземайки обратната на получената пермутация, т.е. *ans[p[i]]=i*. Ако използваме бързо сортиране, то времевата сложност ще бъде *O(nlogn)* и броят на заявките към робота за разместване на тетрадки ще бъде *O(nlogn)*. Такова решение доста бързо надхвърля ограничението от 200 000 заявки към робота и работи за 30% от тестовете. То е реализирано във файл **bookshelf\_NlogN.cpp** и носи 30 точки. Разбира се, ако някой направи квадратична сортировка или някакво друго квадратично решение, то ще „хване“ само 10% от тестовете, т.е. 10 точки.

**Решения, основани на таблица на инверсиите**

Таблица на инверсиите на дадена пермутация *p = (p1, p2,…..,pn)* се нарича масив *T*, такъв че *T[i]* е равно на броя на елементите в пермутацията, които са разположени надясно от индекс *i* и чиято стойност е по-малка от *pi*. Например, ако пермутацията е (3, 4, 1, 2), то таблицата на инверсиите е (2, 2, 0, 0).

Известно е, че една пермутация може да бъде еднозначно възстановена от таблицата на инверсиите ѝ. Най-бързите алгоритми за това представляват линейно минаване през таблицата на инверсиите с използване на интервално дърво. Няма да се спираме конкретно на такива алгоритми, тъй като те са добре известни. Времевата сложност на такова възстановяване е *O(nlogn).*

След като сме се сетили за такъв подход, остава да се намери бърз начин (O(n)) за построяване на таблицата на инверсии чрез заявки към робота. Освен това заявките трябва да не надхвърлят 200 000.

**Решение с 4n заявки към робота**

* С *2n* заявки намираме мястото на максималния елемент в търсената пермутация и го поставяме на първо място.
* След това строим таблицата на инверсиите по следния начин: за всяко *i* викаме *bookswap(1,i)* два пъти. Нека броят инверсии след първото извикване е *a*, а броят инверсии след второто извикване е *b.* Очевидно *a ≤ b*.
* Броят на инверсиите за елемента с индекс *i* е *invi* = *(b-a+1)/2*

Това решение строи таблицата на инверсиите за линейно време, използвайки 4n заявки към робота. То е реализирано във файл **bookshelf\_4N.cpp** и ще мине на примерите, в които *n* ≤ 50 000 т.е. в 60% от тестовете.

**Решение с 2n заявки към робота**

Решението, което позволява да се вместим в ограничението от 200 000 заявки към робота се базира на идеята да разбием търсената пермутация на две пермутации – първата е на елементите, които са по-малки от първия елемент, а втората – на тези, които са по-големи от него.

За всяка от тези пермутации строим таблицата на инверсиите. Възстановяваме всяка от тези пермутации и след това от тях строим цялата търсена пермутация.

Нека разглеждаме елемента с индекс *i* от търсената пермутация. Викаме *bookswap(1,i)* два пъти. Нека броят инверсии след първото извикване е *a*, а броят инверсии след второто извикване е *b*.

Ако *a*>*b*, този елемент ще отиде в първата пермутация (ние не го знаем все още кой е). Броят на неговите инверсии в първата пермутация ще бъде *(a-b-1)/2*.

Ако *a<b*, този елемент ще отиде във втората пермутация. Броят на неговите инверсии във втората пермутация ще бъде *count2*-*(b-a-1)/2*, където *count2* е броят на елементите във втората пермутация преди добавянето на този елемент.

След като сме построили таблиците на инверсиите на двете пермутации, то ги възстановяваме и след това лесно възстановяваме цялата без повече заявки към робота.

Това решение използва *2n* заявки към робота и е реализирано във файл **bookshelf.cpp.**

То получава 100 точки.

*Автор: Руско Шиков*