Алгоритмы и Структуры Данных ДЗ-3

Гарипов Роман M3138

06.10.2019

Задача №1

(a) - Как устроены такие перестановки

Рассмотрим какую-то перестановку (p_i) из n элементов. Построим на ней граф. Добавим в наш граф ребра по следующему правилу :

$$\text{Для каждого } i \text{ добавим ребро } (i \to p[i])$$

Замечание : Получим граф состоящий из нескольких циклов.
Док-во : В действительности, каждый элемент должен встать на свое место, с которого должен уйти другой элемент. Если какая-то компонента в этом графе не является циклом, то получается какой-то элемент должен встать на место, на котором уже стоит какой-то элемент.
Теперь посмотрим, что делает сортировка выбором в терминах нашего графа. На каждой итерации она ставит один элемент из какого-то цикла на нужное место, уменьшая размер цикла на 1.
Таким образом, очень просто оценить количество обменов которое выполнит сортировка обменом, зная размер каждого цикла. Обозначим это количество за $sw(p_i)$, за k обозначим кол-во циклов, sz_i - размер i-го цикла.

$$sw(p_i) = \sum_{i=1}^{k}(sz_i - 1) = \sum_{i=1}^{k}(sz_i) - k = n - k$$

Таким образом, чтобы сортировка выбором сделала максимальное число обменов, необходимо, чтобы k было наименьшим, то есть $k = 1$. Следовательно, все элементы должны находиться в одном большом цикле.

(b) - Количество таких перестановок

Зафиксируем какое-то *начало* цикла, к примеру v_1. Ребро из него можно направить в $(n-1)$ вершину, так как можно выбрать любую, кроме самого *начала*. Пусть это будет какая-то вершина v_2.
Для v_2, можно выбрать $(n-2)$ вершины, все кроме ее же самой и v_1, ведь нам необходимо построить цикл включающий в себя все вершины.
Продолжим делать так, пока не останется одна вершина v_n. Все вершины v_1, v_2, , $v_3 \dots v_{n-1}$ не могут быть выбраны, отсается ровно один вариант направить ребро в v_1, чтобы получить цикл.
Из этих рассуждений видно, что количество таких перестановок ровно $(n-1)!$.

Задача №2

(a) - Как устроены такие перестановки

Посмотрим на сортировку пузырьком. По факту, на i-ой итерации внешнего цикла, мы проталкиваем слева направо к своему месту максимальный

элемент, среди тех, кто не стоит на своем месте. То есть после i итераций внешнего цикла, у нас будет упорядочен суффикс размером i.
Пусть последний элемент равен x. На первой итерации мы сдвинем его на $n-1$ место, поставив n на свое место. На второй итерации сдвинем его на $n-2$ место, поставив $n-1$ на свое место. И так далее.
Для того чтобы получилась $n-1$ итерация, необходимо чтобы этот x был равен 1. Ведь иначе, нашлось бы такое число j(номер итерации), что элемент с индексом $n-j+1$ стоит на своем месте. Тогда сортировка пузырьком сделала бы меньше чем $n-1$ итерация, ведь один элемент сам встал на свое место.
Поэтому для того, чтобы сортировка пузырьком выполнила ровно $n-1$ итерацию внешнего цикла, нужно чтобы на конце массива стояло число 1.

(b) - Количество таких перестановок

Кол-во перестановок, для которых сортировка пузырьком выполнит $n-1$ итерацию внешнего цикла, это $(n-1)!$. Зафиксируем последний элемент равный 1. На 1-ое место мы сможем поставить – $(n-1)$ элемент, на 2-ое – $(n-2)$ и т д. Получаем $(n-1)!$.

Задача №4

Для удобства будем считать, что количество элементов n во входном почти-отсортированном массиве a кратно k. Тогда сделаем следующее : поделим его на блоки размером k, всего их будет $\frac{n}{k}$, посортиртируем каждый из них используя Merge Sort. Merge Sort работает за $\mathcal{O}(m\log_2(m))$, поэтому суммарно получится $\frac{n}{k}\cdot\mathcal{O}(k\log_2(k)) = \mathcal{O}(n\log_2(k))$. Теперь надо как-то объединить все эти отсортированные блоки в целиком отсортированный массив. Пусть сейчас мы хотим поставить элемент с номером i. Он может находиться как в блоке с номером $(\frac{n}{k})$, так и в блоках с номерами $(\frac{n}{k}-1)$ и $(\frac{n}{k}+1)$, ведь по условию каждый элемент отстает от своей правильной позиции не более чем на k. Поэтому будем делать так : для того чтобы поставить i-ый элемент на нужную позицию, будем рассматривать 3 блока, предыдущий($prev$), текущий(cur) и следующий($next$), для каждого из них будем хранить указатель на первый неиспользованный в этом блоке элемент. Возьмем минимум из них, сдвинем указатель на свободный элемент в соответсвующем блоке на один вправо, если этот блок целиком использован, больше его не рассматриваем, далее сдвинем блоки, т е $prev = cur$, $cur = next$, $next = x$, где x - следующий ещё не использованный блок. Делаем почти так же как слияние в Merge Sort, только для 3 массивов. Осталось только сказать, что для первых k элементов не будет блока $prev$, просто отдельно обработаем первые k элементов и всё. Этот алгоритм выполнит сортировку блоков их слияние за $\mathcal{O}(n\log_2(k)+n) = \mathcal{O}(n\log_2(k))$.

Задача №6

Заведем функцию $solve(l, r)$ которая будет возвращать $pref[]$ отсортированные префикс-суммы и $suf[]$ суффикс-суммы для отрезка l и r, а так же в счетчик ans будет записывать колво отрезков сумма которых не превыщает k на $[l; r]$. Для того чтобы посчитать ответ на $[l; r]$ надо рекурсивно запуститься от $[l, m]$ и $[m+1, r]$. Нужно посчитать кол-во отрезков которые начинаются где-то в $[l, m]$ и заканчиваются в $[m+1, r]$, причем их сумма не должна превыщать k.
Заметим, что интересующие нас отрезки представляются в виде суффикса отрезка $[l, m]$ и префикса отрезка $[m+1; r]$. Все отрезки которые начинаются и заканчиваются либо в левом отерзке либо в правом отрезке уже посчитаны.
Будем считать колво интересующих нас отрезков двумя указателями. Поставитим указатель i на самый маленький суффикс(на начало отсортированного массива суффиксных сумм для левой части), а указатель j на самый большой префикс(на начало массива отсортированных префикс-сумм для правой части).
Будем двигать i пока выполняется: $suf[i] + pref[j] \leq k$ и считать на сколько мы его сдивинули в какой-то переменной $delta$. В тот момент, когда это перестанет выполнятся, мы сможем точно сказать сколько для текущего $pref[j]$ существует суффиксов, что они вместе образуют отрезок сумма на котором не превосходит k, их ровно $delta$. Прибавим i к ответу и уменьшим j. Будем проделывать то же самое, пока не упремся в границы для i и j.
Мы посчитали колво отрезоков для $[l; r]$. Теперь нужно вернуть отсортированные суффикс суммы и отсортированные префикс суммы. Ну для этого посчитаем руками сумму в левом отрезке и в правом.
Чтобы посчитать массив отсортированных префикс сумм, надо массив с префикс-суммами для левого отрезка померджить с новым массивом, элементы которого равны $sum(l, r) + pref[i]$ для всех $i \in [m+1; r]$. Чтобы посчитать массив с суффикс-суммами надо проделать аналогичные операции, только с суммой на левом отрезке $[m+1, r]$ и массивом суффикс-сумм для правого отрезка.
Таким образом, получаем решение методом разделяй и властвуй за $\mathcal{O}(n \log(n))$.