Домашнее задание по АиСД №3

Эмиль Гарипов М3138

2019-10-03

Задача №1

Для начала поймем, сколько обменов делает сортировка выбором. Построим граф следующим образом: для каждого элемента перестановки $a[i]$ проведем направленное ребро в i. Утверждается, что сортировка выбором сделает $n-k$ обменов, где k — количество циклов в таком графе. В таком графе есть только циклы и каждая вершина содержиться в цикле,так как из каждой вершины выходит 1 ребро. Для доказательства утверждения рассмотрим работу сортировки выбором. Каждым обменом сортировка ставит какой-то один элемент на свое место, то есть элемент i ставиться на i-е место, мы удаляем прошлое ребро $a[i] \to i$ и строим новое $i \to i$, тем самым уменьшая длину какого-то цикла на 1. Если длина цикла равна 1, значит вершина из которой исходит ребро стоит на своем месте. Значит если каждый цикл будет иметь длину 1, все элементы будут стоять на своем месте. Таким образом нам осталось лишь понять за сколько обменов мы сделаем каждый цикл длины. Пусть каждый цикл j имеет длину $sz[j]$. Тогда за $sz[j]-1$ обменов он станет длины 1. Просуммируем все такие разности и получим:

$$\sum_{j=0}^{k-1}(sz[j]-1) = \sum_{j=0}^{k-1}(sz[j]) - k$$

Так как сумма длин всех циклов есть n, сортировка сделает $n-k$ обменов.

Таким образом маскимальное число обменов будет тогда, когда в графе всего 1 цикл. Такой перестановкой, в графе которой будет всего 1 цикл, например является перестановка $(n, 1, 2, 3, \ldots, n-1)$. Для подсчета количества таких перестановок рассмотрим их построение. Первым шагом мы ставим в какую-то ячейку y число x, так как $x \neq y$ выбрать x можно $n-1$ способом. Далее, чтобы был всего 1 цикл, мы должны в какую-то свободную ячейку z поставить y, ячейку z можно выбрать (n - 2) способами. Для оставшихся элементов аналогичная ситуация. Таким образом, для постановки i-го элемента существует $i-1$ способов, но отдельно стоить рассмотреть последний непоставленный элемент. Для его постановки существует всего один способ.

Итого таких перестановок $(n-1)!$

Задача №2

Сортировка пузырьком своим i-м шагом проталкивает в конец массива максимум на измененном префиксе $[0; n-i-1]$. Чтобы были совершены все $n-1$ итерации на каждой итерации i должен существовать максимум на префиксе $[0; n-i-1]$, который не стоит на позиции $n-i-1$. Это допустимо только тогда, когда в конце массива стоит 1. Посчитаем количество таких перестановок. В последней ячейке стоит 1, а остальные элементы $n-1$ стоят в произвольном порядке. Значит их количество $(n-1)!$

Задача №4

Разобьем исходный массив на блоки по k элементов и отсортируем каждый блок любой сортировкой, работающей за $\mathcal{O}(n \log n)$. Теперь будем хранить несколько указателей: указатель на минимальный неиспользованный элемент блока,стоящего перед текущим блоком(если такой блок существует), указатель на минимальный неиспользованный текущего блока и указатель на минимальный неиспользованный элемент блока,стоящего после текущего блока(если такой блок существует). На текущее место будем ставить минимум из элементов по указателям.
Так как каждый элемент на позиции i отстоит от своей «правильной» позиции не более чем на k, то элемент мог оказаться либо в своем «правильном» блоке, либо в предыдущем или следующем. Тогда если отсортировать элементы этих блоков и выбирать среди элементов блоков минимальный неиспользованный(все использованные меньше либо равны чем этот минимум, поэтому они стоят на правиьлных позициях), этот минимум будет соответствовать элементу, стоящему на позиции i в отсортированном массиве.

Каждый блок сортируется за $k \log k$, всего блоков $\frac{n}{k}$.
Тогда этот алгоритм работает за $\frac{n}{k} \cdot k \cdot \log k + n = \mathcal{O}(n \log k)$

Задача №6

Решение с рекурсивным алгоритмом «разделяй-и-властвуй»: для каждой из двух половин массива научимся получать массив отсортированных по неубыванию префиксных ($pref[i]$) и суффиксных($suf[i]$) сумм (о том, как их считать описано далее).Пользуясь этими суммами и будем считать количество отрезков с суммой не превыщающей k. Так как рекурсия уже посчитала ответ для каждой из половин отдельно, осталось получить ответ для этих половин вместе.
Пусть в данном вызове рекурсивного алгоритма надо обработать массив длины len, мы имеем массивы $prefl$, $sufl$ — отсортированные суммы для левой половины, и такие же массивы $prefr$ и $sufr$ для правой.
Если сложить какой-нибудь суффикс левой половины и какой-нибудь префикс правой половины, то мы получим отрезок, который лежит в обеих половинах, как раз такие отрезки нам надо обработать.
Правый указатель r будет указывать на максимальный по сумме префикс правой половины, который мы не обработали (изначально $r = len - 1$, так как массив уже отсортирован), левый — на минимальный по сумме суффикс левой половины, который мы не обработали (изначально $l = 0$). Тогда операция подсчета ответа будет выглядить следующим образом:

```
//ansl, ansr - ответы, которые дала рекурсивная функция для правой и левой половин
l = 0, r = len - 1, ans = ansl + ansr;
while (true) {
    if (l == len) {
```

```
     while (r > -1) {
            if (sufl[l] + prevr[r] <= k) {
                ans++;
                r--;
            }
        }
        break;
    }
    if (r == -1) {
        while (l < len) {
            if (sufl[l] + prefr[r] <= k) {
                ans++;
                l++;
            }
         }
    }
    while (r > -1 && sufl[l] + prefr[r] > k) {
        r--;
    }
    if (r == -1) continue;
    x = 0;
    while (l < len && sufl[l] + prefr[r] <= k) {
        x++;
        l++;
    }
    ans += x;
}
```

Для текущих l и r алгоритм считает сумму на отрезке, который образуется каким-то суффикисом и префиксом, и если она больше k, то двигает правую границу, так как только при меньших суммах суффиксов сумма может уменишиться, чтобы стать не больше чем k. А если сумма не превосходит k, то алгоритм пытается сдвинуть левую границу (увелить сумму) и запоминает количество удовлетворяющих отрезков.

Таким образом мы умеем считать ответ для рекурсивного вызова. Осталось лишь показать как считать массив отсортированных по неубыванию префиксных и суффиксных сумм.

Рассмотрим получение только суффиксных сумм, так как префексные суммы получаются аналогичным способом. Для каждой половины будем хранить сумму элементов в этой половине и реализуем алгоритм таким образом, чтобы вызов рекурсивной функции хранил сумму в левой и правой половине. Мы уже имеем отсортированные суффиксные суммы для каждой из половин, осталось только сохранить в отсортированном порядке суффиксные суммы левой половины и суффиксные суммы, образованные левой половиной и правой половины (так как мы сливаем два массива в один). Тогда просто будем сливать отсортированные суффксные суммы левой по-

ловины с отсортированными суффиксными суммами правой половины, к которым прибавим сумму в левой половине. Прибавлением правой половины к суффиксу левой половины мы получаем суффикс нового массива, состоящего из двух половинок. Так как по реализации алгоритма правые суффиксные суммы отсортированы, при прибавлении ко всем суммам константы они так же останутся отсортированными. Просто сольем их операцией merge. То же самое проделаем для префиксов, единственное отличие в том, что мы сливаем суффиксы левой половины и правой с прибавлением суммы всей левой половины.

Подсчет количества отрезков с суммой, не превосходящей k, работает за n, слияние тоже. Имеем реккуренту $T(n) = T(\frac{n}{2}) \cdot 2 + 2n$. Несложно доказать, что $T(n) = \mathcal{O}(n \log n)$