Homework

Panov Ivan, M3139

11-09-2019

Задача 1

Пусть требуется отсортировать массив $a[\ ]$. Тогда построим граф на данном массиве по следующему правилу: проведем ребро из ячейки i в ячейку $a[i]$. Данный граф является интерпритацией задачи сортировки, для каждого элемента записано место, в которое его нужно поставить. Заметим, что этот граф состоит только из простых циклов. Также заметим, что за один обмен какой-то цикл уменьшает свою длину на 1. Следовательно сортировка выбором делает мексимальное количество обменов, когда граф состоит из одного простого цикла, проходящего по всем вершинам.
Существует $(n-1)!$ перестановок для которых выполняется максимальное количество обменов. Докажем этот факт: первое число можно поставить только на $n-1$ место, так как если его поместить в ячейку с индексом равным значению, то образуется цикл длины 1 и станет невозможным построить цикл проходящий по n вершинам. Следующую вершину можно поставить только в $n-2$ ячейки из-за аналогичных рассуждений. Для следующих значений проведем те же рассуждения и получим ответ, перемножив количество возможных расстановок для каждого элемента, получим $(n-1)*(n-2)*...*2*1*1 = (n-1)!$.

Задача 2

Посмотрим на последнюю итерацию сортировки пузырьком, совершится обмен первых двух элементов 1 и 2, которые встанут на свои места. Заметим, что каждую итерацию внешнего цикла 1 сдвигается на однин влево, если 1 и так не стоит в самой левой позиции. Следовательно чтобы совершилось $n-1$ итераций сортировки пузырком, нужно чтобы 1 изначально стояла на крайней справа позиции.
Всего существует $(n-1)!$ перестановок на которых совершается $n-1$ итераций сортировки пузырком, так как позиция 1 зафиксирована, а остальные $n-1$ элементов могут стоять в любом порядке.

Задача 3

Будем строить биекцию следующим образом: пусть у нас есть перестановка на которой сортировка выбором делаем максимальное количество обменов, вспомним, что если интерпритировать эту перестановку в виде графа по особым правилам, то получится простой цикл из n вершин. Випишем вершины цикла в порядке следования в графе так, чтобы последней вершиной оказалась 1. Заметим, что на такой перестановке сортировка пузырьком делает $n-1$ итерацию.
Теперь получим обратное. Пусть у нас есть перестановка, на которой сортировка пузырьком делает $n-1$ итерацию. Построим перестановку $b[\ ]$ по перестановке $a[\ ]$: $\forall i : b[a[i]] = a[(i+1)\%n]$. Заметим, что из $b[\ ]$ можно получить простой цикл, и если применить к $b[\ ]$ алгоритм описанный для

получния перестановки, на которой сортировка пузырьком делает $n-1$ итерацию, то получим $a[\ ]$ (по построению $b[\ ]$). Следует, что можно составить биекцию между этими множествами перестановок.

Задача 4

Разобьем массив на блоки по k элементов начиная с начала (последний блок может быть меньше). Всего таких блоков $\frac{n}{k}$, отсортируем каждый из них сортировкой слиянием за $O(k\log(k))$ операций. Так как всего блоков n, суммарно сортировка всех блоков совершится за $O(n\log(k))$ операций. Теперь научимся сливать блоки в один массив. Заметим, что если мы хотим поставить элемент на i-ую позицию, то нужный нам элемент может храниться в одном из трех подряд идущих блоков, так как по условию задачи i-ый элемент отстоёт от своей позиции максимум на k, он может находится в отрезке от $i-k$ до $i+k$, и этот отрезок гарантированно покрывают три отрезка длины k. Тогда давайте сливать блоки по следующему алгоритму, будем хранить указатели в трех подряд идущих блоках на первый необработанный элемент. На каждой итерации будем выбирать минимум из необработанных элементов, класть его в массив ответа и увеличивать указатель, если элементы в блоке закончились, то просто заменяем блок на следующий за текущими. Соединеие всех блоков совершается за $O(n)$. Суммарное время работы алгоритма равно $O(n\log(k))$.

Задача 5

Отсортируем массив p подсчетом за $O(m)$ операций. Заведем массив ответов $ans[\ ]$. Теперь будем решать задачу рекурсивно. Пусть есть функция $find\_kth(intl, intr, intval_l, intval_r)$, где l и r границы отрезка p для которого мы решаем задачу, val_l и val_r индексы элементов между которыми лежит ответ на запросы для данных границ. Изначально запустим функцию от таких аргументов (везде левая граница включительна, а правая нет): $find\_kth(0, n, 0, n)$. Теперь сделаем *partition* a по $p[m]$(найдем $p[m]$-ий элемент в a), где $m = \frac{l+r}{2}$. Таким образом мы найдем ответ для $p[m]$. Теперь сделаем два вызова функции: $find\_kth(l, m, val_l, p[m])$ и $find\_kth(m, r, p[m], val_r)$. Докажем, что данные вызовы коректны. Когда мы сделали *partition* по $p[m]$, все меньшие элементы оказались слева от него, а большие справа. Так как элементы в p отсортированы, следовательно запросы являются правильными. Теперь оценим время работы данного алгоритма. Максимальная глубина рекурсии равна $O(\log(m))$, так как каждый раз мы делим отрезок пополам в массиве p. На каждом уровне рекурсии мы делаем *partition* на массиве $a[\ ]$, но *partition* делается по непересекающимся отрезкам, что делается суммарно за $O(n)$, следоветельно общая сложость алгоритма $O(n\log(m))$.

Задача 6

Будем рашать данную задачу методом "разделяй-и-властвуй". Пусть мы сделали какое-то количество итераций и сейчас хотим решить данную задачу для отрезка с l по r. Пусть мы посчитали ответ для $[l, m)$ и $[m, r)$, где $m = \frac{l+r}{2}$. И для каждога отрезка у нас посчитаны префиксные и суффиксные суммы в отсортированном порядке $(prefL[\ ], prefR[\ ], suffL[\ ], suffR[\ ])$. Так как ответы на $[l, m)$ и $[m, r)$ посчитанны, тогда нам интересны только те суммы, которые начинаются в $[l, m)$ и заканчиваются в $[m, r)$. Давайте перебирать суффиксные суммы первого отрезка и подбирать префиксные суммы второго так, чтобы их общая сумма была не больше k. Эти суммы можно перебирать двумя указателями, поставим первый указатель i на начало массива суффиксных сумм отрезка $[l, m)$ - $suffL[\ ]$, а второй указатель j на конец массива префиксных сумм $[m, r)$ - $prefR[\ ]$. Теперь будем уменьшать второй указатель пока сумма $suffL[i] + prefR[j] > k$. Когда j перестанет уменьшаться добавим $j + 1$ к ответу, так как для всех $t < j$ сумма $suffL[i] + prefR[t] < k$. Таким образом мы посчитали все суммы, которые возможны, если мы взяли $suffL[i]$, сдвинем i на один вправо и повторим алгоритм подбора $prefR[\ ]$. Понятно, что ответы, которые начинаются в $[l, m)$ и заканчиваются в $[m, r)$, мы посчитаем за $r - l$ операций, так как i мы только увеличиваем, а j уменьшаем из-за того, что $suffL[i]$ возрастает, следовельно нужно уменьшать $prefR[j]$. Теперь нужно как-то объеденить $suffL[\ ]$ и $suffR[\ ]$, $prefL[\ ]$ и $prefR[\ ]$, чтобы получить суммы для $[l, r)$. Прибавим ко всем элементам $suffL[\ ]$ значение $suffR[r - 1]$, а ко всем элементам $prefR[\ ]$ прибавим $prefL[m - 1]$ (мы делаем прибавления, так как нам нужны суммы на целом отрезке). Теперь можно просто объединить эти суммы слиянием за линейное время, и получим корректные значения. Оценим время работы алгоритма: если решать эту задачу рекурсивно, то есть сначала посчитать для меньших отрезков, а потом перейти к большим. Изначально у нас n отрезков длины 1, потом мы их попарно объединим и получим $\frac{n}{2}$ отрезков. Каждую итерацию количество отрезков уменьшается в 2 раза, следует итераций будет не больше $\log(n)$, на каждой итерации совершается $O(n)$ действий исходя из выше описанного алгоритма. Получается, общая сложность алгоритма $O(n \log(n))$.