Неразобранные задачи

Минский ШАД. Осень

17 января 2015 г.

1 Динамическое программирование

1. [1 балл] На прямой своими координатами задано n точек. В этих точках расположеные гвоздики. Два гвоздика, находящихся в позициях x_i и x_j можно соединить ниткой длиной $|x_i - x_j|$ саженей. Необходимо натянуть нитки между гвоздями таким образом, чтоб к каждому гвоздю была присоединена как минимум одна нитка, а суммарная длина нитей была минимальна. Сложность алгоритма должна составлять $\mathcal{O}(n \log n)$.

Решение:

Отсортируем все гвоздики по координате и будем считать, что они пронумерованы в порядке увеличения координаты. Очевидно, что гвоздик стоит соединять только с соседним гвоздём (иначе можно считать что рассматриваемый гвоздь соединён с промежуточным, а промежуточный — с изначальным соседом). Тогда введём величину f_i — ответ на задачу, если бы было задано только первых i гвоздей. Тогда:

$$\boxed{f_i = \min(f_{i-1}, f_{i-2}) + |x_i - x_{i-1}|}$$

Последний гвоздь мы обязаны соединить с предпоследним. Мы выбираем из двух вариантов: первый соответствует случаю, когда мы соединяем гвоздь $i-1$ с гвоздём $i-2$, а второй — нет. Итого $\mathcal{O}(n \log n)$ на сортировку и $\mathcal{O}(n)$ на вычисление ответа.

2 КМП

2. [1 балл] Для каждой позиции строки S вычислить значение a_i — длину максимальной подстроки, которая начинается в i и совпадает с некоторым суффиксом строки S. Решение должно иметь сложность $\mathcal{O}(n)$

Решение:

Развернём строку и посчитаем префикс-функцию. Если мы развернём обратно массив, содержащий значения префикс-функций, то можно заметить, что это и есть ответ на задачу.

3 Разное

3. [3 балла] Дан массив из $n+1$ числа, в котором содержатся целые числа от 1 до n (какие-то числа могут отсутствовать). Необходимо найти любое такое x, что x встречается в массиве как минимум дважды.

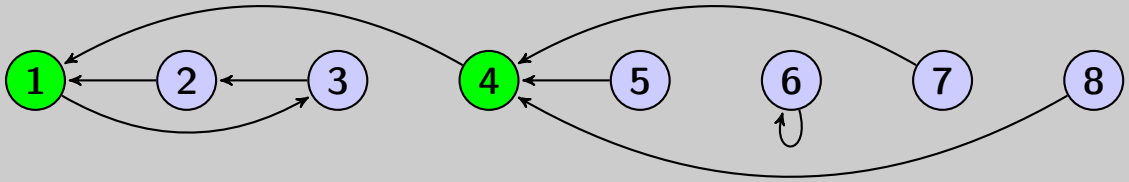
Решение:

Рассмотрим граф из $n+1$ вершины, и дугами $i \to a_i$. В таком орграфе $n+1$ вершина и $n+1$ дуга, а значит он состоит только из циклов, каждая вершина которого — корень какого-нибудь корневого дерева.

Например, исходный массив:

a_i	3	1	2	1	4	3	7	4
i	1	2	3	4	5	6	7	8

Породит следующий граф:



Зелёным отмечены числа, которые подходят под ответ. Эти числа соответствуют вершинам, в которые входят больше одной дуги.

Забудем на время про ориентацию дуг. Посмотрим на компоненту связности, в которой лежит вершина $n+1$. Очевидно, что в этой компоненте есть цикл (в компоненте m вершин и m рёбер). Однако, вершина $n+1$ на цикле лежать не может (в неё не входит ни одна дуга) — значит если мы встанём в неё и будем идти по дугам, то рано или поздно придём в цикл. Заметим, что первая вершина цикла, в которую мы попадём обязательно будет «зелёной». Действительно, в неё входит как минимум одна дуга из цикла и та дуга, по которой мы пришли.

Ну, теперь задача получилась простая. Надо встать в вершину $n+1$ и идти по дугам до цикла, а после найти первую вершину цикла. В данном случае можно сделать это так.

1. Сделаем от вершины $n+1$ ровно $n+1$ шаг. Пусть мы попали в вершину x. Очевидно, это вершина цикла (предпериод не может быть длинней $n+1$).
2. Пойдём от вершины x по дугам и будем считать количество шагов, пока опять не попадём в вершину x. Пусть это количество l. Заметим, что l — длина цикла.
3. Заведём два указателя. Один будет указывать на вершину $n+1$, другой на вершину через l шагов от вершины $n+1$. Будем двигать эти указатели одновременно по шагу, пока они не станут указывать на одну и ту же вершину. Очевидно, что это и будет первая вершина цикла.

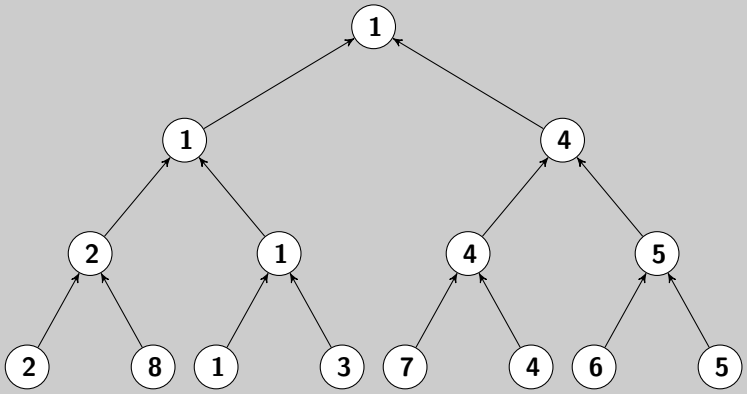
4. [1 ½ балла] Дан массив из $n = 2^k$ различных целых чисел. Необходимо определить 2-ю порядковую статистику за не более, чем $n + k - 2$ сравнения.

Решение:

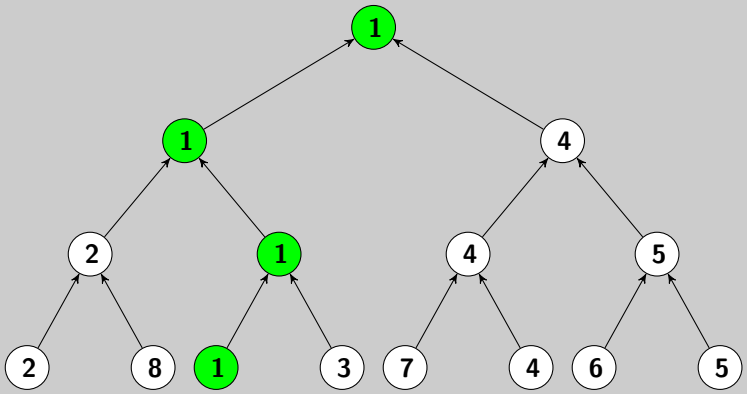
Заметим, что найти минимум в массиве из n элементов можно только с помощью $n-1$ сравнения, причём, очевидно, меньше сделать нельзя.

С другой стороны можно по разному использовать эти сравнения. Давайте будем поступать следующим образом. На первом шаге сравним элементы на первом и втором местах, затем на третьем и четвёртом и так за $n/2$ сравнений оставим ровно $n/2$ кандидатов на минимум. Будем повторять

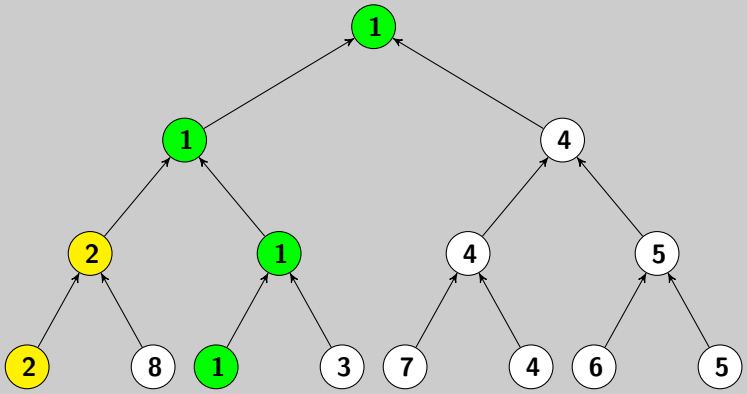
такую операцию, пока не останется ровно один элемент. Такую стратегию легко реализовать в виде дерева. К примеру, рассмотрим массив $(2, 8, 1, 3, 7, 4, 6, 5)$:



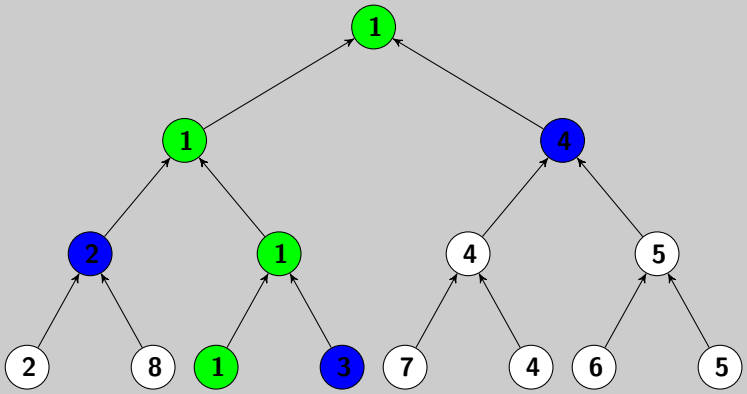
Посмотрим, как минимум проложил себе путь наверх:



Очевидно, он выигрывал при каждом сравнении. Теперь заметим, что среди тех, у кого он выигрывал обязательно есть второй минимум. Действительно, второй минимум таким же образом шёл наверх, выигрывая всех, пока не встретился с минимумом:



Заметим, что всего элементов, которых мы сравнивали с минимумом ровно k — по одному на каждый уровень:



А значит, из них мы можем найти минимум за $k-1$ сравнение. Таким образом нам нужно $n-1+k-1=n+k-2$ сравнения на всё.

5. [1 балл] Пусть мы имеем два положительные неубывающие функции $f(x)$ и $g(x)$, причём $f(n)=\mathcal{O}(g(n))$. Правда, что $2^{f(n)}=\mathcal{O}(2^{g(n)})$? Если это может как выполняться, так и не выполняться, привидете примеры обоих случаев. Иначе докажите утверждение.

Решение:

Иногда это выполнятся, например $f(n)=g(n)$. В частности, это правда, если $f(n)\leqslant g(n)$, при $n\to\infty$.

С другой стороны, если, к примеру, $g(n)=2f(n)$, то $2^{f(n)}$ и $2^{g(n)}$ отличаются уже не в константу раз.

6. [½ балла] Пусть у нас есть k отсортированных последовательностей из n чисел каждая. Предлагается такой алгоритм слияния их в одну: сначала сольём две первых последовательности, затем результат с третьей, и так далее. Какова сложность полученного алгоритма? Считаем, что слияние двух массивов происходит за их суммарную длину. Какова сложность полученного алгоритма.

Решение:

$$\sum_{i=1}^{k-1} n+in = n\sum_{i=1}^{k-1} i+1 = \mathcal{O}(nk^2)$$

7. Дана матрица размером $n\times m$. Каждый элемент матрицы равен либо единице, либо нулю. Нужно преобразовать матрицу таким образом, чтоб элемент $a_{i,j}$ был равен 1 тогда и только тогда, когда в строке i есть хотя бы одна единица или в столбце j есть хотя бы одна единица.
   (a) [1 балл] Решение должно иметь сложность $\mathcal{O}(nm)$
   (b) [1 балл] Решение должно иметь сложность $\mathcal{O}(nm)$ и использовать лишь константу дополнительной памяти (т.е. результат должен оказаться в исходной матрице). Каждый элемент матрицы занимает один бит.

Решение:

Заметим, что задачу можно переформулировать так: если в позиции (i,j) исходной матрицы стоит единица, то надо заполнить строку i и столбец j единицами в результирующей матрице.

Изначально запомним, есть ли в первой строке хотя бы одна единица. Затем для каждой строки, начиная со второй, будем делать следующее:

1. Запомним есть ли в этой строке хотя бы одна единица. Эту информацию не будем запоминать между строками, так что памяти будет $\mathcal{O}(1)$
2. Если в столбце j этой строки стоит единица, поставим единицы в j-й столбец первой строки
3. Если в пункте 1 мы запомнили, что в этой строке была единица, то заполним всю строку единицами

Теперь пройдёмся по первой строке и если встречаем единицу, то заполняем весь встреченный столбец единицами. Если мы изначально запомнили, что первая строка содеражала хотя бы одну единицу, то заполняем единицами всю строку. Полученная матрица — искомая.

8. [½ балла] Дан массив, где **к**аждое число, кроме одного, повторяет**с**я два раза, а одно число — встречается только **о**дин раз. Надо найти это число за 1 п**р**оход по массиву и $\mathcal{O}(1)$ дополнительной памяти.

Решение:

Найдём $\oplus$-сумму всего массива — это и будет искомое число.

9. [1 ½ балла] Дан массив целых чисел, где каждое число, кроме x и y, встречается по два раза, а числа x и y — ровно по одному ($x \neq y$). Надо найти эти числа за $\mathcal{O}(n)$ времени и $\mathcal{O}(1)$ памяти.

Решение:

Найдём $\oplus$-сумму всего массива — это будет $x \oplus y$. Обозначим эту сумму за S. Очевидно, что $S \neq 0$ ($x \neq y$). Найдём любой его единичный бит i (позже покажем, как это сделать за константу времени). Мы знаем, что в этом бите числа x и y различаются. Будем считать, не теряя общности, что x имеет 1 в бите i, а y — 0. Тогда найдём S_0 — $\oplus$-сумму всех чисел, у которых в i-м бите стоит 0, и аналогичную S_1 — для всех чисел, у которых в i-м бите стоит единица. Тогда $x = S_1$, $y = S_0$.

Научимся находить единичный бит у числа за $\mathcal{O}(1)$. Для этого заметим, что $\text{prev}(x) = x\&(x-1)$ — это число x с занулённым младшим единичным битом. Тогда, если мы вычислим $x \oplus \text{prev}(x)$, то мы как раз получим число с одним взведённым битом — самым младшим единичным битом числа x.

Затем, для определения куда отнести число z: в S_0 или S_1 необходимо просто проверять результат $z\&(x \oplus \text{prev}(x))$.

4 Геометрия

10. Дано n точек на плоскости. Необходимо сказать сколько треугольников на этих точках содержат точку $(0, 0)$.
    (a) [½ балла] Решение должно иметь сложность $\mathcal{O}(n^3)$
    (b) [½ балла] Решение должно иметь сложность $\mathcal{O}(n^2 \log n)$
    (c) [1 балл] Решение должно иметь сложность $\mathcal{O}(n \log n)$

Решение:

Посчитаем количество треугольников, которые **не** содержат точку $(0, 0)$, а затем вычтем из общего (C_n^3) количества треугольников найденное количество и получим ответ на задачу.

Все такие треугольники содержатся в одной полуплоскости относительно начала координат. Отсортируем все точки по углу, относительно $(0, 0)$. Начнём с полуплоскости, полученной осью oX. Будем вращать её по часовой стрелке. Когда точка A входит в полуплоскость, посчитаем сколько треугольников будет содержаться в этой новой полуплоскости и иметь A как одну из вершин. Очевидно, что если в полуплоскости ровно m точек, то количество искомых треугольников C_m^2. Число m можно легко поддерживать: при вхождении точки увеличиваем его на 1, при выходе — уменьшаем.

5 Структуры данных

11. Предложить реализацию очереди, используя стурктуру данных стек. Разрешается использовать $\mathcal{O}(1)$ стеков и $\mathcal{O}(1)$ дополнительной памяти. Стек имеет две операции («push» и «pop»), очередь тоже.
    (a) [½ балла] Амортизированная стоимость операций должна быть $\mathcal{O}(1)$
    (b) [2 балла] Стоимость операций должна быть $\mathcal{O}(1)$ в худшем случае

Решение:

Для решения первой части можно использовать известную конструкцию из двух стеков, назовём их A и B. При выполнении операции «push» будем добавлять элемент в стек B. При выполнении «pull» — достаем из стека A. Если же стек A пуст, то просто перекладываем все элементы из стека B в стек A. Легко показать, что тогда выполняется FIFO. Каждый элемент не более одного раза добавляется и достаётся из каждого из двух стеков.

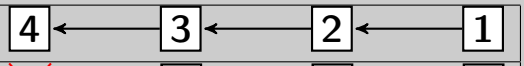
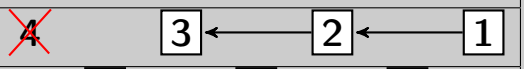
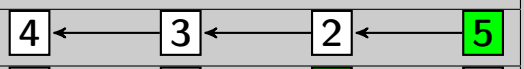
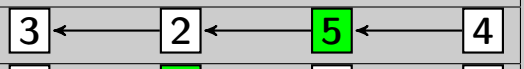
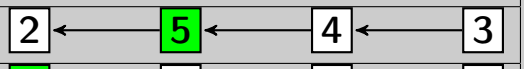
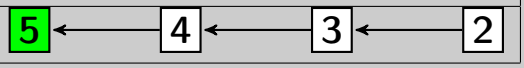
Решение второго пункта довольно большое, описание его можно найти вот тут `http://goo.gl/VjEYxL`

12. Предложить реализацию стека, используя стурктуру данных очередь. Стек имеет две операции («push» и «pop»), очередь тоже. За n будем считать максимальное количество элементов, которые могут находится в стеке в одно время.
    (a) [½ балла] Разрешается использовать $\mathcal{O}(n)$ дополнительной памяти
    (b) [½ балла] Стоимость операции «push» должна быть $\mathcal{O}(n)$, а «pop» — $\mathcal{O}(1)$. Дополнительной памяти — $\mathcal{O}(1)$
    (c) [½ балла] Стоимость операции «pop» должна быть $\mathcal{O}(n)$, а «push» — $\mathcal{O}(1)$
    (d) [2 балла] (Амортизированная) Стоимость обеих операций должна быть $\overline{o}(n)$

Решение:

В первом пункте просто заведём стек в дополнительной памяти.

Будем реализовывать стек на одной очереди. Будем поддерживать в этой очереди порядок LIFO. Если надо забрать элемент, то просто заберём элемент из очереди. Если вставить, то поступим следующим образом. Запомним сколько элементов в очереди до добавления, пусть m. Добавим новый элемент в очередь. Затем сделаем m раз связку «pull»-«push», тем самым выведя новый элемент на первое место, а все остальные оставив за ним. На рисунке элементы пронумерованы в порядке добавления в структуру.

Промежуточное состояние структуры	
«pull»	
После «pull»	
Добавление элемента	
«pull»-«push»	
«pull»-«push»	
«pull»-«push»	

Понятно, что можно при операции «push» просто добавлять элемент в очередь, а при «pop» делать серию «pull»-«push» дабы вывести самый последний элемент на первое место.

Приступим к интересной части. Реализуем операцию «push» за $\overline{o}(1)$ и «pull» за амортизированную оценку $\mathcal{O}(sqrtn)$. Заведём две очереди: A и B. В первой будем хранить $\approx \sqrt{n}$ элементов самых близких к вершине стека (т.е. добавленных позже всего). Хранить их, однако, будем в порядке очереди (иначе сложность, вероятно, бы возрасла опять до $\mathcal{O}(()n)$). При операции «push» просто добавляем элемент в очередь A. Если $|A|^2 > |B|$ перекидываем элемент верхней элемент очереди A в B. Понятно, что таких перекидываний на «push» надо сделать не больше одного.

При операции «pull» серией «pull»-«push» очереди B добиваемся, чтобы последний элемент стека оказался на вершине и возвращаем его. Заметим, что инвариант про $|A|^2 \leqslant |B|$ не нарушился.

Если же очередь A пуста, то необходимо переупорядочить элементы. Для этого переместим (B.pull -A.push) $\lfloor |B| - \sqrt{|B|} \rfloor$ элементов из очереди B в очередь A и просто переименуем очереди. Теперь инвариант выполняется.

В качестве упражнения предлагается доказать, что амортизированная сложность операции «pull» составляет $\sqrt{n}$.