«Основные алгоритмы»

задачи, указания, решения и критерии проверки

итоговой контрольной 2019

В этом файле приведены условия задач итоговой контрольной, критерии их проверки, а также указания и решения некоторых задач. Приведённые критерии описывают общие случаи и не являются исчерпывающими. Если ваш случай не попадает под критерии, значит задача оценена исходя из общих соображений, согласованных с критериями.

Указания к задаче содержат основные шаги её решения и не являются полным решением. В частности, в них часто опущены обоснования приведённых фактов или доказательство оглашённых утверждений. Подобный подход к решению задачи студентом в большинстве задач приведёт к неполному баллу за её решение.

В случае, если для задачи был предложен правильный алгоритм, то отсутствие доказательства корректности и оценки его сложности существенно снижает оценку за решение задачи как и неоптимальность алгоритма, о чём было сообщено в преамбуле к контрольной.

Преамбула к контрольной

Необоснованные ответы не оцениваются! Если в задаче требуется построение алгоритма, то нужно построить оптимальный алгоритм (за неэффективность снижается оценка), доказать его корректность и оценить время работы.

Задача 1 (4). Определим $f(n)$ как количество выводов «Hello, World!» следующей функцией (на входе n).

```
1  Function HelloWorld(n) :
2      if (n > 0) then
3          HelloWorld(⌊n/4⌋);
4          HelloWorld(⌊n/4⌋);
5          for i = 1 to n do
6              j = 1;
7              while (j < n) do
8                  print("Hello, World!");
9                  j = j * 2;
10             end
11         end
12         HelloWorld(⌊n/4⌋);
13         HelloWorld(⌊n/4⌋);
14     end
15     return;
16 end
```

Оцените асимптотику роста $f(n)$.

Указания. Для f справедлива рекуррентная формула $f(n) = 4f(\lfloor \frac{n}{4} \rfloor) + \Theta(n \log n)$, что является классической рекуррентой из курса.

Решение. Необходимо показать, что $f(n) = \Theta(n \cdot \log^2(n))$. Одним из вариантов доказательства, учитывая что ответ уже известен, является простая индукция (возможно решение через честное расписывание суммы для дерева рекурсии).
Пусть для некоторых начальных n выполнено:
$f(n) \leqslant c_1 \cdot n \cdot \log^2(n)$. Действительно, не связываясь с $n = 0$ из-за проблем с логарифмом, записываем для тех n, что непосредственно ссылаются на $f(0)$:

$$f(1) = 0 \leqslant c_1 \cdot 1 \cdot \log^2(1) = 0;$$

$$f(2) = 2 \leqslant c_1 \cdot 2 \cdot \log^2(2) = 2c_1;$$

$$f(3) = 6 \leqslant c_1 \cdot 3 \cdot \log^2(3)$$

Это и будет базой индукции. Хотим показать, что утверждение выполнено для всех n.
Шаг индукции:
$f(n) = 4f(n/4) + \Theta(n \cdot \log(n)) \leqslant 4c_1 \cdot (n/4) \cdot \log^2(n/4) + d_1 \cdot n \cdot \log(n) = c_1 \cdot n \cdot (\log(n)-2)^2 + d_1 \cdot n \cdot \log(n) = c_1 \cdot n \cdot (\log^2(n) - (4 - d_1/c_1)\log(n) + 4) \leqslant c_1 \cdot n \cdot (\log^2(n))$.
Здесь мы предполагаем, что $4 - d_1/c_1 > 0$ и тогда при $n > exp(4/(4 - d_1/c_1))$ всё выполнено. Очевидно, что такое c_1 можно выбрать относительно d_1. d_1 следует из оценки выполнения программы внутри одного шага, и может быть выбрано равным 2. Очевидно, что внутри одного вызова функции происходит $\leqslant 2n \cdot \log(n)$ выводов. Тогда выбираем $c_1 = 2$, и получаем для

$n > 10$ соотношение $f(n) \leqslant 2 \cdot n \cdot \log^2(n)$

Аналогично докажем оценку $f(n) \geqslant c_2 \cdot n \cdot \log^2(n)$. Перейдём сразу к шагу индукции:
$f(n) = 4f(n/4) + \Theta(n \cdot \log(n)) \geqslant 4c_2 \cdot (n/4) \cdot \log^2(n/4) + d_2 \cdot n \cdot \log(n) = c_2 \cdot n \cdot (\log^2(n) - (4 - d_2/c_2) \cdot \log(n) + 4) \geqslant c_2 \cdot n(\log^2(n)$. Здесь мы предполагаем, что $4 - d_2/c_2 < 0$ и тогда при $n > 1$ всё выполнено. Очевидно, что такое c_2 можно выбрать относительно d_2. d_2 следует из оценки выполнения программы внутри одного шага, и может быть выбрано равным 1. Очевидно, что внутри одного вызова функции происходит $\geqslant n \cdot \log(n)$ выводов. Тогда, выбираем $c_2 = 0,1$, и получаем для $n > 1$ соотношение $f(n) \geqslant 0,1 \cdot n \cdot \log^2(n)$.
Объединяя результаты, получаем, что для любого $n > 10$:
$0,1 \cdot n \cdot \log^2(n) \leqslant f(n) \leqslant 2 \cdot n \cdot \log^2(n)$.

Критерии.

0 выписана неверная рекуррента;

+1 выписана верная рекуррента;

+1 написан верный ответ;

−2 за неверные утверждения в доказательстве

−0,5 за небольшие погрешности в доказательстве

−1 за значительные погрешности в доказательстве

−1,5 O-оценка вместо Θ-оценки.

−2 Если дана только ссылка на обобщённую основную теорему о рекурсии (Master Theorem в английской википедии) или Основную теорему (из курса). Первой не было в курсе по-хорошему, а вторая в данном случае не применима. Если сделаны какие-то разумные шаги для пояснения (указан случай и посчитаны константы), то можно апеллировать у семинариста.

Задача 2 (4). К отсортированному по возрастанию массиву длины n применили несколько раз операцию циклического сдвига (сдвинули его элементы по кругу: $[1, 2, 3, 4] \to [2, 3, 4, 1] \to [3, 4, 1, 2]$). Получившийся массив хранится в оперативной памяти (в этой задаче не нужно считывать вход). Постройте эффективный алгоритм, который находит число совершённых циклических сдвигов.

Решение. Обозначим данный во входе массив как $x[1..n]$.

Рассмотрим отдельно случай, когда сдвиг был нулевым, в этом случае массив останется возрастающим. В этом и только в этом случае первый элемент массива будет меньше последнего. Поэтому далее считаем, что $x[1] > x[n]$.

Найдём бинпоиском пару элементов $(x_i, x_{i+1}) : x_i > x_{i+1}$. Это будут последний и первый элементы в исходном массиве, т.к. для любой другой пары элементов в отсортированном по возрастанию массиве выполняется $x_j < x_{j+1}$. Если последний элемент исходного массива находится на i-м месте, то было произведено $n - i$ циклических сдвигов.

Будем поддерживать инвариант для рассматриваемого подмассива, что его левый конец больше правого, вначале он выполнен $x[1] > x[n]$. Этот инвариант будет гарантировать нам, что внутри текущего подмассива обязана находиться пара $x_i > x_{i+1}$. Пусть мы сейчас находимся в подмассиве $x[l..r]$, $x[l] > x[r]$. Посмотрим на $x[(l + r)/2]$, если он больше $x[r]$, то перейдем к $l = (l + r)/2$. Если $x[(l + r)/2] < x[l]$, то к $r = (l + r)/2$. Оба условия не могут быть не выполнены, т.к. иначе $x[l] \leqslant x[(l + r)/2] \leqslant x[r]$, что противоречит $x[l] > x[r]$. Разность $r - l$ экспоненциально убывает, когда она станет $r - l = 1$ мы автоматически найдем искомую пару индексов $x_l > x_{l+1}$ в силу инварианта. Асимптотика решения $O(\log n)$.

Критерии.

0 Решение за линейное время. В условии было сказано что вход считывать необязательно, это жесткая подсказка что итоговый ответ должен быть сублинеен.

0 Решение за $O(1)$ в предположении что исходный массив это $[1, \ldots, n]$ – на самом деле это не так.

1 Используется бинпоиск, но нет корректного перехода к подмассиву.

2 Другие сублинейные алгоритмы оцениваются из двух баллов.

−0.5 Не разобран случай нулевого сдвига.

−0.5 Не показано как заканчивается работа бинпоиска, какой элемент выбирать из пары.

−0.5 Рассматриваются сдвиги вправо, а не влево.

−0.5 Отсутствие расчёта количества сдвигов после найденной позиции последнего или первого элемента.

Задача 3 (4). Государство решило разобраться, из какого города можно добраться до какого с помощью имеющихся ЖД-дорог (быть может односторонних). Постройте $O(|V||E|)$-алгоритм, который получив на вход граф дорог выводит для каждого города список достижимых из него городов.

Решение.

Для i-й вершины графа заведем пустой список и запустим поиск в ширину на дереве достижимых из нее вершин. Во время обработки очередного ребра (u, v), если конец ребра еще не был помечен «посещенным», добавляем

вершину v в список. В итоге в список попадут все вершины, достижимые из i-й и только они. Сложность: для каждой вершины выполняется $O(|E|)$ операций (по числу обработанных ребер), тогда общая сложность алгоритма $O(|V||E|)$

Критерии.

−1 Асимптотика решения не написана, написана асимптотика $O(E(E+V))$, алгоритм имеет асимптотику $O(E(E+V))$

−1 За недоказанный $O(E)$ алгоритм нахождения списка достижимых вершин из данной в топологически сортированным графе

- Решения с асимптотикой хуже $O(E(E+V))$ (в том числе $O(V^3)$) оцениваются в 0 баллов

Задача 4 (4). Два сотовых оператора устанавливают оборудование в новом городе. Для работы связи необходимо соединить вышки в общую сеть так, чтобы сигнал мог пройти по проводам от любой вышки до любой другой. Помимо вышек каждого оператора, есть общегородские вышки, которые уже образуют общегородскую сеть. Каждый оператор может соединять проводами только свои вышки (быть может вместе с общегородскими). Также каждый оператор хочет быть независимым: работа сети не должна пострадать в случае, если выйдут из строя все вышки другого оператора. Соединение пары вышек (i, j) стоит $c_{i,j} = c_{j,i}$. Постройте эффективный алгоритм, который находит минимальное нужное число соединений минимальной стоимости.

Указания. Приведём два возможных пути решения.

- Сведём задачу к построению минимального остовного дерева каждым оператором. Вершинами графа являются все вышки оператора, а также вершина s, соответствующая общегородской сети. Вес ребра между вершинами-вышками i и j есть $c_{i,j}$, а между вершиной-вышкой i и s — ребро минимального веса среди $c_{i,j}$, где j — вершина общегородской сети.

- Просто используем алгоритм Прима на вершинах оператора и общегородской сети с той поправкой, что поддерживаем структуру дерева, где все общегородские вершины считаются одной вершиной u.

Критерии.

- **0 баллов** — решение не использует общегородскую сеть, строит не дерево;

- **0,5-1 балл** — строится дерево, но не объясняется как и почему;

- **1 балл максимум** — строится минимальное остовное на вершинах оператора, а затем соединяется одним ребром с общегородской сетью;
- **-1 балл** — решение использует алгоритм Краскала, но не объясняется почему он работает: либо не указано как Краскал работает с циклами в общегородской сети, либо не объясняется как Краскал завязан на связности, либо не объясняется как полученное дерево соотносится с целью задачи: при занулении весов ребер в общегородской сети и построении мин. остовного дерева, получается некоторое дерево на вершинах оператора и городской сети, но общегородская сеть может иметь много ребер и почему данное дерево сработает и как именно его использовать — нужно объяснять;
- **-1 балл** — в решении общегородская сеть стягивается в одну вершину, но не говорится какие будут новые веса ребер или как именно и за какое время эти веса находятся;
- **до -1 балла** — другие ошибки.
- **от -50%** от того, что выше вплоть до -4 за отсутствие доказательств и объяснений

Задача 5 (4). Строка w является сцепкой строк u и v, если в промежутки между символами u были вставлены все символы v, при этом при подстановке порядок символов не нарушался: $ACBD$ — это сцепка AB и CD, $ADEBCFG$ — это сцепка ABC и $DEFG$. На вход подаются три строки u, v и w. Нужно проверить является ли w сцепкой строк u и v.

Решение. Заведем двумерный массив F размера $(|u|+1) \times (|v|+1)$ и будем заполнять его следующим образом: $F(i,j) = 1$, если $w(0, i+j) = w_1 \cdots w_{i+j}$ (где w_k - k-й символ строки w) является сцепкой строк $u(0,i)$ и $v(0,j)$ и $F(i,j) = 1$ иначе.

Тогда $F(0,0) = 1$, $F(i,0) = 1 \Leftrightarrow u(0,i) = w(0,i)$ (аналогично для $F(0,i) = 1$) и

$$\begin{cases} F(i-1,j), \text{если } w_{i+j} = u_i \text{ и } w_{i+j} \neq v_j \\ F(i,j-1), \text{если } w_{i+j} = v_j \text{ и } w_{i+j} \neq u_i \\ \max\{F(i-1,j), F(i,j-1)\}, \text{если } w_{i+j} = u_i = v_j. \end{cases}$$

Корректность: Пусть на каком-то шаге приведенный алгоритм корректно заполнил весь подмассив размера $i \times j$, кроме элемента $F(i,j)$, т.е мы знаем, является ли строка $w(0, i+j-1)$ сцепкой строк $u(0,i-1)$ и $v(0,j)$ или $u(0,i)$ и $v(0,j-1)$. Заметим, что, если последний элемент $w(0,i+j)$ совпадает с u_i и не совпадает с v_j, то $w(0,i+j)$ может быть сцепкой $u(0,i)$ и $v(0,j)$ тогда и только тогда, когда $w(0,i+j-1)$ является сцепкой строк $u(0,i-1)$ и $v(0,j)$, т.е. $F(i,j) = F(i-1,j)$. Если последний элемент $w(0,i+j)$ совпадает с v_j и не совпадает с u_i, то $w(0,i+j)$ может быть сцепкой $u(0,i)$ и $v(0,j)$ тогда и только тогда, когда $w(0,i+j-1)$ является сцепкой строк $u(0,i)$ и $v(0,j-1)$,

т.е. $F(i,j) = F(i,j-1)$. Если последний элемент $w(0,i+j)$ совпадает и с u_i, и с v_j, то $w(0,i+j)$ может быть сцепкой $u(0,i)$ и $v(0,j)$ тогда и только тогда, когда $w(0,i+j-1)$ является сцепкой хотя бы одной из пар строк $u(0,i)$ и $v(0,j-1)$, $u(0,i-1)$ и $v(0,j)$ т.е. $F(i,j) = \max\{F(i,j-1), F(i-1,j)\}$. Переход выполнен верно. По окончании работы алгоритма получаем ответ в ячейке $F(|u|,|v|)$. Сложность: заполняем массив размера $(|u|+1) \times (|v|+1)$, для заполнения каждой ячейки выполняется постоянное количество операций. Таким образом сложность алгоритма $O(|u||v|)$

Критерии.

−3,5 Приведен линейный алгоритм, корректно обрабатывающий только строки с различными элементами

−1 Нет доказательства корректности

−1 Нет оценки сложности

Задача 6 (4). Дано полное бинарное дерево с 4^n листьями. Листья покрашены в белый и черный. Ваш преподаватель по алгоритмам предлагает вам сыграть в игру: мы по очереди двигаем фишку из корня дерева вниз к листьям, вы начинаете. Если в конце после n ходов с каждой стороны фишка оказалась на белом поле, вы выиграли и получаете «отлично» по алгоритмам. Если же на черном — что ж, вы не сдали курс. Вам известны метки всех листьев, постройте алгоритм, который подскажет вам, стоит ли играть в эту игру.

Решение. Считая белые листья булевой 1, а чёрные – булевым 0, размечаем дерево снизу вверх по правилу: если в этой вершине ходит преподаватель, вершина есть конъюнкция потомков, если студент – дизъюнкция потомков. Значение корня даёт ответ: если в корне 1 играть можно, если 0, то нет.

Корректность следует из индуктивных соображений – если вершина помечена единицей на вашем ходе, то по свойству дизъюнкции существует потомок, помеченный единицей, в который вы можете перейти своим ходом. Потомок помечен единицей, а значит по свойству конъюнкции оба его потомка помечены единицей, а значит вне зависимости от хода преподавателя фишка останется на поле, помеченном 1. По индукции, фишка всегда стоит на единице – в том числе, и когда вершина является листом, т.е. вы можете выиграть. Если же корень помечен 0, то вне зависимости от вашего хода по свойству дизъюнкции вы перемещаетесь в вершину, помеченную нулём, а преподаватель в свой ход по свойству конъюнкции может найти вершину, помеченную нулём, и переставить фишку туда. По индукции, фишка всегда стоит на нуле – в том числе, и когда вершина является листом, т.е. вы проиграете.

Сложность алгоритма – вычисление булевой функции на всех листьях – в худшем случае $O(4^n)$ (на самом деле даже $\Theta(4^n)$).

Возможны и другие, практически эквивалентные, решения.

Критерии.

- **1 балл максимум** — нетривиальное достаточное условие на листья, не являющееся необходимым;
- **до -1 балла** — неверная асимптотика;
- **-1 балл** — при в целом верном решении перепутан порядок хода соперников;
- **-1 балл** — не бинарное дерево, при верном решении;
- **до -2 баллов** — неясные пояснения (в основном корректности)

Задача 7 (5). В ориентированном взвешенном графе все ребра имеют неотрицательный вес. Описать эффективный алгоритм поиска кратчайшего облегчения пути между заданной парой вершин. Облегчением пути называется сумма всех весов рёбер на этом пути кроме, быть может, одного ребра (его длина нуль на этом пути). Вход задачи: граф G с весами на рёбрах и вершины u и v, между которыми нужно найти кратчайший облегчённый путь $u \to v$.

Указания. Находим кратчайшее расстояние от вершины u до остальных вершин алгоритмом Дейкстры. Находим кратчайшие расстояния от каждой вершины до вершины v алгоритмом Дейкстры, развернув рёбра. Дальше проходим по всем рёбрам (a, b) и ищем минимальную сумму расстояний $d_u[a] + d_v[b]$.

Критерии.

−1 Отсутствие правильного доказательства корректности алгоритма.

/2 Ошибка в верном решении.

−4 Неэффективное решение с запуском дейкстры E раз.

Задача 8 (5). Массив $A = [a_1, a_2, \ldots, a_n]$ является перестановкой чисел от 1 до n. На вход задачи подаётся массив $F = [f_1, f_2, \ldots, f_n]$, в котором f_k — число элементов a_j, стоящих левее числа $k = a_i$ в массиве A и таких, что $a_j > a_i$ $(j < i)$. Постройте эффективный алгоритм, который восстанавливает массив A по массиву F.

Решение. Число 1 в восстанавливаемой перестановке будет стоять на $(f_1 + 1)$-м месте при индексации с единицы, поскольку оно меньше всех остальных чисел. Далее, число $k+1$ будет стоять на $f_{k+1}+1$ еще не занятой позиции, если расстановка всех элементов до $k + 1$ уже известна. Есть два варианта того, как можно находить $f_{k+1} + 1$ еще не занятую позицию.

- За $O(n^2)$: Будет проходить массив слева направо, подсчитывая количество незаполненных элементов массива, и остановимся, когда пройдем ровно $f_{k+1}+1$. Решения с такой асимптоиткой достаточно для получения полного балла.

- За $O(n \log n)$ со структурами данных: Будем поддерживать дерево отрезков для суммы на массиве размера n, при инициаализации состоящем из единиц. Единица в массиве значит, что соответствующее место в восстанавливаемой перестановке еще не занято. В каждом узле дерева будем хранить сумму всех листьев в его поддеревьях. При запросе на нахождение $f_{k+1}+1$ не занятой позиции будем спускаться по дереву, начиная с корня, и поддерживать текущую сумму элементов левее рассматриваемой вершины максимально возможной, но меньшей, чем $f_{k+1}+1$. Найденная таким образом вершина будет соответствовать $f_{k+1}+1$ незанятым местам левее ее. После помещения элемента на его место ($O(\log n)$) обновляем суммы в узлах от измененного с единицы на ноль листа (еще $O(\log n)$). Таким образом, суммарное время работы алгоритма составит $O(\log n)$.

Критерии.

5 Верное решение с асимптотикой не более $O(n^2)$

3 Решение за $n \log n$, в котором не указано, с помощью какой структуры данных реализуется поиск порядковой статистики за $O(\log n)$.

−1.5 Нет асимптотики

−1.5 Нет обоснования корректности

Задача 9 (5). Ориентированный ациклический граф задан матрицей смежности (хранится в оперативной памяти). Докажите, что чтобы найти все стоки (вершины с исходящей степенью ноль) нужно проверить $\Omega(n^2)$ элементов матрицы смежности.

Решение. Возьмем произвольное n. Рассмотрим граф, состоящий из n несвязанных вершин. Каждая из этих вершин является стоком. Допустим существует алгоритм, для которого $\Omega(n^2)$ не является нижней оценкой. Тогда начиная с некоторого n_0 на предложенном графе алгоритм проверит менее чем n^2 ячеек матрицы смежности (т.е. не все). Тогда, если мы возьмем граф, заданный аналогичной таблицей смежности, за исключением ячеек, не проверенных алгоритмом (там мы поставим единицы), то некоторые из вершин перестанут быть стоками, хотя выход алгоритма не изменится. Противоречие. Значит нижней оценкой является как минимум $\Omega(n^2)$.

Критерии.

4 Предложен пример графа в общем случае, проблемы с описанием стратегии оппонента.

3 Предложен пример с логикой оппонента без указаний к обобщению на произвольное n.

2-1 Некоторые некритериальные случаи

0.5 Отсутствие полноценного решения при наличии разумных продвижений.

Задача 10 (7). На вход подаются натуральные числа n, M, x_1, a, b, причем $M << n$. Нужно найти медиану массива $x[1, \ldots, n]$, генерируемого следующим образом: $x[1] = x_1, x[i+1] = (a \cdot x[i] + b) \pmod{M}$. Предложите алгоритм со сложностью $O(M)$ арифметических операций.

Решение. Вначале забудем о том, что последовательность конечная, будем смотреть на ее бесконечное продолжение (не будем обрубать ее после n-го элемента). Каждый элемент последовательности $x[i]$ вычисляется через предидущий. Если какой-то элемент встретится во второй раз, $x[s] = x[e], s < e$, то далее последовательность будет вести себя периодически $x[s+i] = x[e+i]$. Давайте найдем такую пару индексов s, e, что $x[s] = x[e], s < e$ и e минимальный. Тогда все элементы $x[1], \ldots, x[e-1]$ будут различными (поскольку e минимальный). Обозначим $T = e - s$ длину периода, тогда $\forall i \geqslant s, x[i] = x[i+T]$ (для справки, кусок последовательности $x[1], \ldots, x[s-1]$ называют предпериод). Искомую пару s, e можно найти за $O(M)$ времени и памяти. Заведем массив $TimeVisit[0..M-1]$, в который будем записывать минимальный индекс в $x[]$ на котором встретился данный остаток, изначально $TimeVisit[]$ заполнен бесконечностями. Будем вычислять один за другим элементы $x[]$ и записывать информацию о них в $TimeVisit[x[1]] = 1, TimeVisit[x[2]] = 2, \ldots$, пока для очередного $x[i]$ не окажется, что в $TimeVisit[x[i]]$ уже записана не бесконечность. Этот элемент и будет первым, который повторился $e = i$, а $s = TimeVisit[x[i]]$. Поскольку все элементы $x[1], \ldots, x[e-1]$ различные и они все являются остатками по модулю M, то их в сумме не больше M и эта часть алгоритма отработает за $O(M)$.

Теперь мы знаем длину периода и предпериода, вычислим массив $c[0..$ $..M-1]$, количеств раз, сколько каждый остаток встретится в последовательности $x[1..n]$. Вначале в $c[]$ будут записаны одни нули. Мы знаем, что в последовательности встречаются элементы $x[1], \ldots, x[e-1]$ и только они, давайте пройдемся по ним и посчитаем сколько раз кто встретился. Очевидно, что элементы предпериода встретятся не более одного раза, поэтому $\forall i \leqslant min(n, s-1), c[x[i]] = 1$. Посмотрим теперь на элементы периода $x[s], \ldots, x[e-1]$. Каждый из них встречается в последовательности с периодом T и довольно легко получить формулу сколько раз он встретится $\forall i : s \leqslant i \leqslant min(n, e-1), c[x[i]] = \lceil \frac{n-i+1}{T} \rceil$. Таким образом мы вычислим массив $c[]$ также за $O(M)$ времени и памяти.

Заметим, что теперь по сути у нас есть сжатый вариант отсортированной последовательности $x[]$, потому что мы по сути посортировали ее подсчетом. Это позволяет легко найти ее медиану, это такое число x, для которого выполнено, что $\sum_{i=0}^{x-1} < n/2, \sum_{i=0}^{x} \geqslant n/2$. Чтобы это число x найти, достаточно пройтись по массиву $c[]$ вычисляя частичную сумму его элементов начиная с нулевого, и в первый раз когда мы встретим частичную сумму большую или равную $n/2$ мы нашли медиану. Эта часть также будет работать за $O(M)$.

Критерии.

+1 описана структура последовательности через период и предпериод

+1 приведен $O(M)$ алгоритм нахождения периода и предпериода

+4 (не складывается с двумя пунктами выше) приведен и доказан $O(M)$ алгоритм нахождения массива частот (сколько раз каждый остаток встречается в последовательности)

+2 (складывается с предидущими пунктами) Приведен $O(M)$ алгоритм нахождения медианы по массиву частот.

−2 задача решена только для последовательности без предпериода