Домашняя работа №2

Бредихин Александр

23 февраля 2020 г.

Задача 1

Дано: $g(n) = 1 + c + c^2 + \ldots + c^n$, где $c > 0$

- **a)** Пусть $c < 1$, доказать, что $g(n) = \Theta(1)$:
  $\square$ По определению Θ оценки:

  $g(n) = \Theta(f(n)) \Leftrightarrow \exists c_1, c_2 > 0, \exists N \in \mathbb{N} : \forall n > N \longrightarrow c_1 f(n) \leqslant g(n) \leqslant c_2 f(n)$

  В нашем случае $f(n) = 1$. Снизу функцию $g(n)$ можно ограничить первым слагаемым 1 (так как все слагаемые положительные, то понятно, что $g(n) = 1 + c + c^2 + \ldots + c^n \geqslant 1$). Сверху можно ограничить бесконечной суммой (сумма первых n положительных членов меньше или равна бесконечной суммы), которая является суммой бесконечно убывающей геометрической прогрессией, так как $c < 1$. Получается:

  $$g(n) = \sum_{k=0}^{n} c^k \leq \sum_{k=0}^{\infty} c^k = \frac{1}{1-c} = c_2$$

  Следовательно мы нашли $c_1 = 1, c_2 = \frac{1}{1-c}, N = 1$ такие, что $c_1 \leqslant g(n) \leqslant c_2$, значит $g(n) = \Theta(1)$ по определению.

  $\blacksquare$

- **b)** Пусть $c = 1$, доказать, что $g(n) = \Theta(n)$:
  $\square$ Так как $c = 1$, то

  $$g(n) = 1 + c + c^2 + \ldots + c^n = 1 + 1 + \ldots + 1 = n$$

  Значит, $g(n) = \Theta(n)$, по определению Θ оценки взяв $c_1 = 1, c_2 = 1, N = 1$

$\blacksquare$

- **c)** Пусть $c > 1$, доказать, что $g(n) = \Theta(c^n)$:
  $\square$ Снизу $g(n)$ можно ограничить последним слагаемым c^n (так как все слагаемые положительные, то понятно, что $g(n) = 1 + c + c^2 + \ldots + c^n \geqslant c^n$). Сверху полуим такую оценку (с помощью формулы суммы геометрической прогрессии) :

$$\sum_{k=0}^{n-1} c^k = \frac{c^n - 1}{c - 1} \leq \frac{c^n}{c - 1} = c_2 \cdot c^n$$

$$\sum_{k=0}^{n} c^k \leq c^n + \frac{c^n}{c - 1} = \frac{c}{c - 1} \cdot c^n$$

  Значит, $g(n) = \Theta(c^n)$, по определению Θ оценки взяв $c_1 = 1, c_2 = \frac{c}{c-1}, N = 1$

$\blacksquare$

Задача 2

a) для среднего арифметического последовательности чисел индуктивное расширение:

$$g = \begin{pmatrix} Sum \\ n \end{pmatrix},$$

где Sum – сумма всех элементов последовательности на текущем шаге: первоначально равна 0, затем на каждом шаге прибавляем текущие число последовательности.
n - количество элементов в последовательности на текущем шаге, т.е. изначально 0, на каждом шаге прибавляется 1.
Тогда функция $f(g)$ возвращает ответ задачи (индуктивную функцию):

$$f\left(\begin{bmatrix} Sum \\ n \end{bmatrix}\right) = \frac{Sum}{n}$$

b) для числа элементов последовательности целых чисел, равных её максимальному элементу индуктивное расширение:

$$g = \begin{pmatrix} Cur\_max \\ n \end{pmatrix},$$

где Cur_max - максимальный элемент на текущем шаге,
n - количество элементов на текущем шаге, равные Cur_max

Изначально Cur_max и n равны 0, затем на каждом шаге происходит сравнение:

- если текущий элемент равен Cur_max, то $n+=1$;
- если текущий элемент больше Cur_max, то присваиваем Cur_max значение текущего элемента а, $n=1$;
- если текущий элемент меньше Cur_max, то ничего не делаем и переходим на следующий шаг

Тогда $f(g)$ возвращает ответ задачи (индуктивную функцию):

$$f\left(\begin{bmatrix} Cur\_max \\ n \end{bmatrix}\right) = n$$

c) для максимального число идущих подряд одинаковых элементов индуктивное расширение:

$$g = \begin{pmatrix} Cur\_len \\ Max\_len \\ Cur\_el \end{pmatrix},$$

где Cur_el - текущий элемент последовательности (изначально равняется первому элементу последовательности),
Cur_len - длина текущий последовательности одинаковых элементов,
Max_len - длина максимальной последовательности подряд идущих элементов
Изначально $Cur\_len = Max\_len = 0$, на каждом шаге происходит сравнение:

- если текущий элемент равен Cur_el, то $Cur\_len += 1$;
- если текущий элемент не равен Cur_el, тогда:
  - присваиваем Cur_el значение текущего элемента,
  - если $Cur\_len > Max\_len$, то присваиваем Max_len значение Cur_len,
  - $Cur\_len = 1$

Тогда $f(g)$ возвращает ответ задачи (индуктивную функцию):

$$f\left(\begin{bmatrix} Cur\_len \\ Max\_len \\ Cur\_el \end{bmatrix}\right) = Max\_len$$

Задача 3

Алгоритм:

Создадим три указателя: первоначально все они указывают на первые элементы в каждом из трёх массивов и переменную $count = 0$, в которой будет находиться результат: количество различных элементов в объединение массивов.
Сравниваем элементы, на которые указывают указатели, находим наименьший из элементов, прибавляем переменной $count$ один и сдвигаем указатель(элемент в котором наименьшее число) на следующий элемент массива, если массив заканчивается, то указатель оставляем на месте, но в последующих сравнениях не учитываем этот элемент(аналогично двигаем 2 или 1 других указателя, пока все не дойдут до конца).
Если в сравнении получилось, что в двух массивах лежит одинаковые минимальные на текущий момент элементы, то переменную $count$ увеличиваем на единицу и сдвигаем сразу два указателя на следующие элементы в своих массивах (аналогично, если все 3 элемента равны, прибавляем к переменной 1 и сдвигаем все 3 на следующий элемент).
Алгоритм заканчивается, когда все указатели дошли до конца, результат будет находиться в переменной $count$.

Корректность:

Пусть мы будем не только считать количество неповторяющихся элементов в объединении, но и находить их, для этого создадим 4ый массив, куда будем класть элемент, который минимальный в сравнении на текущем шаге(если таких элементов одновременно 2 или 3, то все равно кладём 1).
Заметим, что в этот массив на каждом шаге мы добавляем наименьший из оставшихся элементов, докажем это по индукции по k– номеру шага:
База индукции: $k = 1$. Все 3 массива отсортированы по возрастанию, значит первые элементы каждого массива - минимальные в нём, из них выбираем минимальный, значит кладём элемент минимальный из оставшихся.
Шаг индукции: пусть на $k-1$ шаге утверждение выполнено, тогда указатели указывают на наименьшие элементы в своих массивах, аналогично

рассуждению в Б.И. кладём наименьший из оставшихся элементов
Из доказанного утверждения следует, что мы не могли «пропустить никакой элемент» (так как всегда кладём наименьший из оставшихся) и все они различны, так как не кладём 2 одинаковых не на каком из шагов и массив строго возрастающий, значит алгоритм работает корректно.

Сложность:
Алгоритм заканчивается, когда каждый указатель дойдёт до конца массива, на каждое перемещение одного указателя в худшем случае приходится сравнение($\mathcal{O}(1)$), следовательно кол-во сравнений в худшем случае равно $n_1 + n_2 + n_3$, где n_i – количество элементов в iом массиве, следовательно сложность алгоритма, $\mathcal{O}(n)$(линейная) (где n максимальная длина из массивов

Задача 4

Задача: на вход подаётся последовательность чисел $a_1, b_1, a_2, b_2, \dots, a_n, b_n$, нужно построить онлайн-алгоритм, который вычисляет сумму $\sum\limits_{i \neq j} a_i \times b_j$

Заметим, что (аналогично идеи из семинара):

$$\sum_{i \neq j} a_i \times b_j = \sum_{1 \leq i,j \leq n} a_i \times b_j - \sum_{i=j} a_i \times b_j$$

$$\sum_{i \neq j} a_i \times b_j = \sum_i a_i \sum_j b_j - \sum_{i=j} a_i \times b_j \tag{1}$$

Построим индуктивное расширение:

$$g = \begin{pmatrix} S_a \\ S_b \\ flag \\ el \\ S_{ab} \end{pmatrix},$$

где S_a -сумма всех чисел a_i, то есть $S_a = \sum\limits_i a_i$, аналогично S_b: $S_b = \sum\limits_j b_j$,
$flag$ - переменная, которая показывает какой элемент был считан последним: $flag = 0$ когда считываем элемент «a», $flag = 1$ - когда считываем элемент «b»
el - последний считанный элемент «a»
S_{ab} - сумма вида: $S_{ab} = \sum\limits_{i=j} a_i \times b_j$

Алгоритм:

Изначально все описанные выше переменные равны 0, начинаем считывать последовательность и выполняем следующие:

- Если $flag = 0$ переводим флаг в значение 1, прибавляем текущий элемент к S_a, записываем в el значение текущего элемента
- Если $flag = 1$ переводим флаг в значение 0, прибавляем текущий элемент к S_b, значение текущего элемента умножаем на el и прибавляем полученное произведение к S_{ab}

Функция $f(g)$ возвращает ответ задачи(индуктивную функцию):

$$f\left(\begin{bmatrix} S_a \\ S_b \\ flag \\ el \\ S_{ab} \end{bmatrix}\right) = S_a \times S_b - S_{ab}$$

Корректность:
Из уравнения (1) следует корректность алгоритма: нужно найти 3 указанные суммы, что и делает описанный алгоритм.

Сложность:
Для вычисления ответа проходим по последовательности длины $2n$ (так как n чисел «a» и n чисел «b») один раз. На каждом шаге делаем арифметические операции сложность которых $\mathcal{O}(1)$, следовательно сложность алгоритма - $\mathcal{O}(n)$(линейная).

Задача 5

Алгоритм:
Создадим массив buf, в котором будем хранить размер максимальной возрастающей подпоследовательность для iго элемента данного в задаче массива a. Изначально массив buf заполнен единицами (так как сам элемент – минимальная возрастающая подпоследовательность для семя самого). Пройдёмся по массиву a двумя вложенными циклами:

$$for(int\, i = 0; i < n; i++)$$

$$for(int\, j = 0; j < i; j++)$$

Принцип динамического программирования:
Внутри этого вложенного цикла, находим максимальное из чисел $buf[j]$,

таких что элемент $a[j]$ удволетворяет условию: $a[i] > a[j]$ (max) (то есть для скольких предыдущих элементов jый элемент может являться продолжением подпоследовательности, заканчивающийся iым элементом и кол-во элементов будет наибольшем на данном этапе). Изменяем массив $buf[j] += max$ (так как изначально элементы массива уже равны 1, поэтому без +1)
Ответом в задаче будет максимальное число в массиве buf после прохождения вложенного цикла (частный случай: если $n = 1$, выводим 1)

Корректность:
Так как мы смотрим наилучшее из продолжений только при условии, что $a[i] > a[j]$, то рассматриваем только строго возрастающие подпоследовательности. На каждом шаге заполнения массива buf выбираем наибольшую из возможных подпоследовательностей, значит в итоге должна получиться максимальная длина возрастающей последовательности для каждого из чисел массива a

Сложность:
В алгоритме есть вложенный цикл, проходящий по массиву размером n, значит n^2 операций на каждой из которых делаем арифмемтические операции, сложность которых $\mathcal{O}(1)$, следовательно сложность алгоритма $\mathcal{O}(n^2)$

Задача 6

Алгоритм:
Создадим 2 переменные:
$candidat$ - тут будет лежать элемент, кандитат на нужный нам
$flag$ - счётчик

Проходимся по массиву циклом и делаем следующие:

- Если переменная $flag$ равна 0, то в переменную $candidat$ кладём текущий элемент, а $flag$ увеличиваем на 1
- Иначе если текущий элемент равен тому, который в $candidat$, увеличиваем $flag$ на 1
- во всех остальных случаях уменьшаем $flag$ на 1

После этого прохождения по массиву в переменной *condidat* будет лежать элемент, который встречается более чем $n/2$ раз (majority), если такой в массиве существует (что гарантирует условие задачи)

Корректность:
Докажем по индукции, по длине входа – n:
База индукции: $n = 1$, очевидно, алгоритм выведет этот единственный элемент в списке, он и является majority, верно
Предположение индукции: предположим, что наш алгоритм корректно работает для списков размером до длины n. Теперь возьмем список длиной $n + 1$.
Шаг индукции. Рассмотрим 2 случая:

1. В первом случае представим, что первый элемент не является majority. Затем в какой-то момент в этом списке счётчик $flag$ упадет до 0 (так как есть majority элемент, который не равен первоначальному, каждый раз, когда мы его будем встречать счётчик уменьшается на 1, получается, больше $n/2$ минусов, следовательно счётчик обязательно упадёт до 0), и будет выбран новый кандидат. На этом этапе наш алгоритм неотличим от того, что мы начали с этого индекса в массиве. Более того, majority элемент в остальной части нашего списка совпадает с majority элементом в исходном списке, потому что если наш majority элемент появился k раз первоначально в списке из n элементов, то в нашем подсписке из первых i элементов наш majority элемент находится там $\leqslant \frac{i}{2}$ раз. Так как $k > \frac{n}{2}$, то в оставшемся списке длинной $n - i$: $k - \frac{i}{2} \geqslant \frac{n-i}{2}$ - верно, значит. Можно пременить наше предположение индукции, как длина «новой задачи» меньше либо равна n.

2. Второй случай: первый элемент-это наш majority элемент. Если весь список обрабатывается без сброса, то, очевидно, был возвращен majority элемент. Если он был сброшен, то мы можем использовать точно такие же рассуждения, как и в пункте 1, чтобы показать, что большинство в подсписке, подлежащем обработке, совпадает с исходным большинством, и поэтому мы все равно будем возвращать правильный элемент.

Сложность:
Получаем алгоритм сложность которого $\mathcal{O}(n)$, так как проходим по входному списку один раз и на каждом шаге делаем арифмитические операции, сложность которых $\mathcal{O}(1)$
Сложность по памяти — $\mathcal{O}(1)$ (создаём только 2 новые переменные и не

храним входные данные)