Студент: Маргарита Чудова
Группа: 2
Дата: 27 апреля 2022 г.

---

1. (a) $f(n) = O(n)$, значит $\exists N, C > 0 : f(n) \le C \cdot g(n)$. Допустим, что условие "$\exists\ N$"отбросить нельзя, сразу после какого-то N_1 все ломается. Тогда $\exists N_1 : n = N_1 - 1 \Rightarrow \frac{f(n)}{g(n)} > C$. $C_1 = \frac{f(N_1-1)}{g(N_1-1)} + 1$ - это тоже какая-то константа, причем для нее $\frac{f(n)}{g(n)} < C_1$. Повторим ту же процедуру для $n = N_1 - 2$, получим некоторую C_3, и так далее, пока не получим $n = 0$. Имеем набор $C_1, C_2, C_3 \dots C_N$, из них можем выбрать максимум, и тогда условие "$\exists N$"не будет ни на что влиять.
   Такое рассуждение не зависит от типа функции и подходит и для $\mathbb{N} \to \mathbb{N}$, и для $\mathbb{N} \to \mathbb{R}_{>0}$ (и в том, и в другом случае нам помогает то, что элементов до N конечное количество). В обоих этих случаях условие "$\exists\ N$"можно отбросить.

   (b) Случай $\mathbb{N} \to \mathbb{N}$: возьмем $f(n) = n$, а $g(n)$

$$g(n) = \begin{cases} n^2 - 3 & n \neq 1 \\ 1 & n = 1 \end{cases} \tag{1}$$

   Рассмотрим n = 2, $f(2) = 2$, $g(2) = 1$. По обновленному определению $\forall n,\ C > 0 : f(n) < C \cdot g(n)$, но это условие не выполняется в точке 2 для C = 1 $\Rightarrow$ условие '$\exists\ N$' отбросить нельзя.

   Случай $\mathbb{N} \to \mathbb{R}_{>0}$: пусть $f(n) = \frac{2}{n}$, $g(n) = n$. По обновленному определению $\forall n,\ C > 0 : f(n) < C \cdot g(n)$, но это условие не выполнено в точке 1 для C = 1 $\Rightarrow$ условие '$\exists\ N$' отбросить нельзя.

2. (a) $f(n) = O(n)$, значит $\exists N, C > 0 : f(n) \le C \cdot g(n)$, Так как $\phi(x) = \log(x)$ монотонно возрастающая, из этого неравенства следует

$$\log f(n) \le \log Cg(n) = \log g(n) + \log C$$

   $\log C$ - это какая-то константа. Так как функции $\mathbb{N} \to \mathbb{N}$, то $g(n)$ не может бесконечно убывать, и $\log g(n)$ ограничена снизу (при этом $g(n) > 1$). Тогда введем $a = \frac{\log C}{\min(\log(g(x))}$ - эта величина конечна. Без уменьшения общности можем считать C > 1/2, тогда a > -1. $\log g(n) + \log C < (a+1) \log g(n) \Rightarrow$ утверждение верно

   (b) Рассмотрим $f(n) = 2n$, $g(n) = n$. Проверим по определению: $\exists N, C > 0 : 4^n \le C \cdot 2^n \Rightarrow C > 2^n$ - не получится подобрать такую универсальную константу $\Rightarrow$ утверждение неверно

   (c) Рассмотрим $f(n) = 2^n$, $g(n) = 2^{2n}$. Проверим по определению: $\forall C > 0\ \exists N : n \le C \cdot 2n$ - неправда, неравенство выполнено только для $C > 1/2 \Rightarrow$ утверждение неверно

   (d) $f(n) = o(g(n))$, значит

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \tag{2}$$

   Рассмотрим

$$\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = [g(n) > 1] = \lim_{n\to\infty} 2^{(f(n)/g(n)-1)\cdot g(n)} = 2^{\lim\limits_{n\to\infty} -g(n)} \tag{3}$$

   Так как $g(n) > 1$, условие (2) возможно только при $g(n) \to \infty$. Из этого следует, что (3) = 0, то есть утверждение верно

(e) По определению $\Omega(n)$ $\exists N, C > 0 : \forall n \geq N \ f(n) \geq C \cdot g(n) \Rightarrow \sum\limits_{k=1}^{n} \frac{1}{k} \geq C \cdot \log n$.

Попробуем оценить левую часть снизу - если сможем найти подходящую константу для чего-то меньшего, чем наша сумма, то для суммы такая константа тем более подходит. На роль оценки снизу подходит $\int\limits_1^n \frac{1}{x} dx = \ln n$ (можно нарисовать рисунок с функцией $1/x$ и прямоугольниками, соответствующими каждому слагаемому и сравнить площади под графиком). Получим

$$\ln n > C \cdot \log n \Rightarrow C < \frac{1}{\ln 2}$$

Тогда возьмем для определения Ω $C = \frac{1}{\ln 2} - \varepsilon, \varepsilon = \text{const}$, утверждение верно.

A	B	O	o	Θ	ω	Ω
$\log^k n$	n^2	+	+	-	-	-
n^k	c^n	+	+	-	-	-
$\sqrt{n}$	$n^{sin(n)}$	-	-	-	-	-
2^n	$2^{n/2}$	-	-	-	+	+
$n^{\log m}$	$m^{\log n}$	+	-	+	-	+
$\log n!$	$\log n^n$	+	-	+	-	+

3. (a) Проверим на о-малое:

$$\lim_{n\to\infty} \frac{\log^k(n)}{n^\varepsilon} = \lim_{n\to\infty} \frac{\ln^k(n)}{e^{\varepsilon \ln n} \cdot \ln 2} = [t = \ln(n)] = \lim_{n\to\infty} \frac{t^k}{e^{\varepsilon t} \ln 2} =$$
$$= [\text{применяем k раз правило Лопиталя}] = \lim_{n\to\infty} \frac{k!}{\varepsilon^k e^{\varepsilon t} \ln 2} \to 0 \qquad (4)$$

В определении о-малого ($\forall C > 0 : \exists N : \forall n \geq N : f(n) < C \cdot g(n)$) из $\forall C > 0$ выберем конкретную, тогда отсюда следует определение для О-большого. Из определений следует, что $f(n)$ не может быть $\Omega(n)$ или $\omega(n)$ и $o(n)$ одновременно (будем этим пользоваться и в следующих пунктах). Так как $f(n) = \Theta(n) \Leftrightarrow f(n) = \Omega(n) \cap f(n) = O(n)$, то $f(n) \neq \Theta(n)$ (этим тоже пользуемся далее).

(b) Проверим на о-малое:

$$\lim_{n\to\infty} \frac{n^k}{c^n} = \lim_{n\to\infty} \frac{n^k}{e^{\ln C \cdot n}} = [\text{применяем k раз правило Лопиталя}] = \lim_{n\to\infty} \frac{k!}{\ln^k C e^{\ln C \cdot n}} \to 0 \quad (5)$$

(c) Степень функции B $x = \sin(n)$ является периодической функцией и может меняться от -1 до 1. Будем рассматривать большие n - тогда при $x < 0$ можем сколь угодно сильно приблизить n^x к нулю за счет величины n. Такие встречаются периодически при сколь угодно больших n. Тогда все определения, в которых фигурирует неравенство вида $f(n) < C \cdot g(n)$(то есть о-малое, О-большое) автоматически отпадают, не найдется такого C.

Аналогично для $f(n) > C \cdot g(n)$ (то есть $\omega(n)$, $\Omega(n)$). Если x близко к 1, то n^x близко к n. Увеличивая n, мы сможем сделать n^x сколь угодно близким к $n > n^{0.5}$, значит, неравенство не будет выполняться для любого сколь угодно большого n.

(d) Проверим на $\omega(n)$:

$$\forall C > 0 : \exists N : \forall n \geq N : 2^n > C \cdot 2^{n/2} \Rightarrow C < 2^{n/2}$$

По любому заданному C сможем найти $N = 2\log C - \varepsilon$, где ε - какая-то маленькая константа, следовательно $f(n) = \omega(n)$.

Отсюда следует (аналогично рассуждениям с о-малым и О-большим выше), что $f(n) = \Omega(n)$

(e)

$$\frac{n^{\log(m)}}{m^{\log(n)}} = \frac{2^{\log(n)\log(m)}}{2^{\log(m)\log(n)}} = 1$$

Это одна и та же функция. В определениях возьмем C = 1 $\Rightarrow$ для $\Omega(n), O(n)$ условия выполняются, а в $\omega(n), o(n)$ неравенство неверное.

(f) Воспользуемся формулой Стирлинга:

$$n! \sim \sqrt{2\pi n}\frac{n^n}{e^n} \Rightarrow \log n! \sim (n + 1/2)\log(n) - n$$

Рассмотрим

$$\lim_{n\to\infty} \frac{(n+1/2)\log(n) - n}{\log(n^n)} = \lim_{n\to\infty} \frac{(n+1/2)\log(n) - n}{n\log(n)} \to 1$$

Тогда $\log(n!) \neq o(n)$ (предел тогда должен был бы быть равен 0), но $\log(n!) = O(n)$ (если в определении взять, например, C = 2, то все выполняется с некоторого N). Проверим на $\Omega(n)$. $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$.

$$\lim_{n\to\infty} \frac{n\log(n)}{(n+1/2)\log(n) - n} = \lim_{n\to\infty} \frac{1}{(1 + 1/2n) - 1/\log n} \to 1$$

- аналогично выполнено

5. (a) По сути, в этой задаче требуется для каждого элемента a_i из массива найти d_i - максимальную длину отрезка, на которой он минимален, и взять максимум из $a_i d_i$.

Для того, чтобы найти d_i, нужно найти r_i и l_i. Для l_i делаем это следующим образом: возьмем стек и начинаем по порядку класть туда элементы массива с их номером, при этом предварительно удаляя все верхушки стека, большие или равные элементу, который мы хотим положить. Предположим, мы кладем в стек элемент a_k, и, удалив все, что больше него, получаем элемент a_i верхним в стеке. Тогда a_i - самый левый элемент, меньший a_k, а значит $r = i + 1$.

Почему это работает? Элемент a_l между a_k и a_i не мог удалить a_i до этого шага, так как $a_l \geq a_k,\ a_i < a_k \Rightarrow a_l \geq a_i$. Элементы добавляются/удаляются не более 2 раз, так что время работы этой части алгоритма $O(n)$.

Аналогично ищем r_i, но кладем элементы в стек, начиная с последнего. Вычисление $a_i d_i$ и поиск среди них максимума тоже дадут $O(n)$, так что общее время работы - $O(n)$

(b) Все аналогично предыдущей задаче, но теперь на максимальном отрезке, на котором a_i минимально, нужно искать сумму всех элементов отрезка S_i, а не длину d_i, и выбирать максимум из $a_i S_i$.

Как искать сумму на отрезке так, чтобы это занимало $\mathcal{O}(1)$? Воспользуемся результатами задачи 8 с семинара: сначала посчитаем все кумулятивные суммы, запишем их в массив − это $\mathcal{O}(n)$. После для каждого a_i найдем соответствующие r, l, вычтем одно значение кумулятивной суммы из другого и получим сумму на этом отрезке − тоже $\mathcal{O}(n)$.

6. Пусть на вход дан массив $[a_1..a_n]$.

(a) Преобразуем его в $[\tilde{a_1}..\tilde{a_n}]$ следующим образом:

$$\tilde{a_1} = a_1$$

$$\tilde{a_i} = a_i - a_{i-1},\ i \neq 1$$

(b) Для каждого запроса $add(x, l, r)$ будем проделывать с массивом следующее (так m раз):

$$\tilde{a}_l += x$$

$$\tilde{a}_{r+1} -= x$$

Получим что-то вида $[a_1, .., a_l - a_{l-1} + x, a_{l+1} - a_l.., a_{r+1} - a_r - x, ..]$

(c) После обработки всех запросов заведем переменную temp и запустим цикл:

```
answ = [a[1]] \\для простоты пусть нумерация с 1
temp = a[1]
for i in range(2, n):
   temp +=ta[i]    \\ ta - массив a с тильдой из предыдущих пунктов
   answ.append(temp)
print(answ)
```

Понятно, что предыдущее слагаемое сокращается со следующим так, что остается только a_i + нужные x из запросов. *answ* - это преобразованный после всех запросов массив. (a) займет O(n), (b) - O(1)*m = O(m), (c) - O(n)

7. Рассмотрим дерево решений. Пусть k - глубина дерева, понятно, что спускаемся до тех пор, пока размер подзадачи не станет 1. Посмотрим, как меняются характеристики подзадач:

   1 итерация: задачи размера n, $2^{\log^* n}$ тратится в каждом узле. Количество задач - 1
   2 итерация: задачи размера $\log n$, $2^{\log^* \log n} = 2^{1+\log^* \log\log n}$ тратится в каждом узле. Количество задач - 2
   ...
   k итерация: задачи размера $\log \ldots \log n$, $2^{\log^* \log \ldots \log n}$ тратится в каждом узле (везде k логарифмов). Количество задач - 2^k

   На последнем шаге $\log \ldots \log n$ должен быть равен 0. На шаге $k-1$ в степени двойки будет на один логарифм меньше, можем раскрыть эту степень по рекуррентной формуле и получить формулу через k логарифмов: $1 + \log \ldots \log n = 1$. Продолжим в том же духе до верха дерева, на самом верхнем уровне получим 2^k. Таким образом общее время равно:

   $$T(n) = 2^k \cdot 1 + 2^{k-1} \cdot 2 + \ldots + 1 \cdot 2^k$$

   Поскольку на каждом шаге задача уменьшается в log раз, $k = \log^* n$, $T(n) = 2^{\log^* n} \cdot \log^* n$

   P.S: по непонятной для меня причине мой тех ругается на lfloor и rfloor, поэтому их здесь нет, надеюсь, что суть без них ясна.