Студент: Александр Никулин
Дата: 14 сентября 2020 г.

Задача 1. *Дайте ответ для двух случаев* $\mathbb{N} \to \mathbb{N}$ *и* $\mathbb{N} \to \mathbb{R}_{>0}$*:*
(a) Если в определении $\mathcal{O}$ *опустить условие про* N *(т.е. оставить просто* $\forall n$*), будет ли полученное определение эквивалентно исходному?*
(b) Тот же вопрос про o*.*

Решение.

(a) Да, т.к. нам важно поведение функций на бесконечности, а конечный хвост нас не волнует (как и в определении предела последовательности). Если совсем неформально, можно положить $n = 1$ и решить неравенство, тем самым найдя новую константу C. Более формально, $\exists N, C > 0 : \forall n \geq N : f(n) \geq Cg(n)$, тогда положим $\overline{C} = max\{\frac{f(n)}{g(n)} \mid 1 \leq n \leq N\}$:

$$f(n) \leq f(n) = \frac{f(n)g(n)}{g(n)} \leq \overline{C}g(n),\ \forall n$$

(b) Нет, тогда сломается ключевое условие из определения, что для $\forall C > 0 : \exists N : \forall n \geq N : f(n) \leq Cg(n)$. Можно привести пример: положим $f(n) = 20n + 2, g(n) = n^2$. Для $C \leq 22$ такого N не найдется.

В обоих случаях ничего не меняется при $\mathbb{R}_{>0}$ (кажется). Могло бы сломаться, если бы наши фукнции были не из $\mathbb{N}$, тогда в примере выше при $n \to 0$ отношение функций было бы неограниченно и (a) стало бы неверно.

Задача 2. *Считайте здесь, что* $\forall n : f(n) > 1 \wedge g(n) > 1$*. Правда ли, что* $f(n) = o(g(n)) \Rightarrow 2^{f(n)} = o\left(2^{g(n)}\right)$?

Решение. Да, данное следствие справедливо (при $g(n) \to \infty$). Док-во:

$$\lim_{x\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = \lim_{x\to\infty} 2^{f(n)-g(n)} = \lim_{x\to\infty} 2^{g(n)(\frac{f(n)}{g(n)}-1)} = \lim_{x\to\infty} 2^{-g(n)} = \lim_{x\to\infty} 2^{-\infty} = 0$$

Если бы наши функции могли принимать значения из $\mathbb{R}$, то это было бы неверно, например $f(n) = 1,\ g(n) = \frac{1}{n}$.

Задача 3. *Заполните табличку и поясните (особенно строчки 4 и 7). Здесь все буквы, кроме* n*, — константы.*

Решение.

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

Пояснения. Будем пользоваться пределами (если они существуют):
1. $\lim_{x\to\infty} \frac{\log^k n}{n^\varepsilon} = \lim_{x\to\infty} \frac{k!}{\varepsilon^k n^\varepsilon} = 0$ используем лопиталя.
2. $\lim_{x\to\infty} \frac{n^k}{c^n} = \lim_{x\to\infty} c^{k\log_c n - n} = 0$
3. $\exists$ бесконечно много n: $\sin n \geq \frac{1}{2} \Rightarrow n^{\sin n} \geq Cn^{\frac{1}{2}}$, а значит $n^{\sin n} \neq \mathcal{O}(n^{\frac{1}{2}})$. В обратную сторону, $\exists$ бесконечно много n: $\sin n < 1$, а значит $n^{\frac{1}{2}} \neq \mathcal{O}(n^{\sin n})$ (т.к. при любом выборе константы при данных n $\sqrt{n}$ будет иногда больше)
4. $\lim_{x\to\infty} \frac{2^n}{2^{\frac{n}{2}}} = \lim_{x\to\infty} 2^{\frac{2}{n}} = 0$
5. $\lim_{x\to\infty} \frac{n^{\log m}}{m^{\log n}} = \lim_{x\to\infty} \frac{n^{\log m}}{n^{\log m}} = 1$
6. $\lim_{x\to\infty} \frac{\log n!}{n\log n} = \lim_{x\to\infty} \frac{\log n + \log n - 1 + \cdots + \log 1}{n\log n} = \lim_{x\to\infty} \frac{n\log n + \log 1 - \frac{1}{n} + \cdots + \log\frac{1}{n}}{n\log n} = 1$

Задача 4. *Дана последовательность* $a_1, a_2, \cdots, a_n \in \mathbb{N}$ *и* $S \in \mathbb{N}$. *Найти* $l, r (1 \leq l \leq r \leq n)$ *такие, что сумма* $\sum_{i=l}^{r} a_i = S$. *Задачу требуется решить за линейное от* n *время.*

Решение. Решать будем с помощью подхода two pointers. Заведем два указателя left, right, которые будут указывать на границы последовательности, сумму которой мы в данный момент считаем. Если искомая найдется, то именно их значения будут ответом. Заведем так же переменную для суммы.

Можно заметить, что на каждой итерации наша сумма может быть либо $> S$ и тогда нам нужно от нее отнять первый элемент, либо $< S$ и тогда надо добавить новый элемент справа. В третьем случае мы нашли ответ и на этом алгоритм закончил работу. В худшем случае, например когда все элементы до последнего $\leq S$, а последний $\geq S$ нам придется вначале добавлять справа до конца, а потом убавлять слева до конца, совершив тем самым порядка $2n + C$ операций. Поэтому время работы будет $\mathcal{O}(n)$.

Решение на python:

```python
def find_sum(arr: List[int], S: int) -> Tuple[int, int]:
    left, sum = 0, 0

    for right in range(len(arr)):
        sum = sum + arr[right]

        while sum > S and left < right:
            sum = sum - arr[left]
            left = left + 1

        if sum == S:
            return left, right
```

.

P.S. Честно сказать я не понял подсказки :c Что если i элемент больше $\geq S$, как тогда поддерживать неравенство $r_i \leq r_{i+1}$?

Задача 5. *Дана последовательность* $a_1, a_2, \cdots, a_n \in \mathbb{N}$

Решение.

(a) Интуициию мы обсуждали на семинаре. Представим, что у нас есть какой-то большой элемент до которого есть элементы меньше. Тогда данный элемент нигде дальше не будет использовать т.к. мы ищем минимальный. Поэтому нужно завести стэк, который будет хранить только элементы, которые не больше уже встреченных (в убывающем порядке). Для этого мы для каждого нового элементы делаем pop, до тех пор, пока стэк либо не пуст, либо верхний элемент меньше чем i-ый.

Решение на python:

```python
def nearest_left(arr: List[int]) -> List[int]:
    res = [0] * len(arr)

    stack = []
    for i, num in enumerate(arr):
        while stack and num <= nums[stack[-1]]:
            stack.pop()

        if stack:
            res[i] = stack[-1]
        stack.append(i)
    return res
```

.

(b) Решение аналогично (a). Мы можем либо развернуть массив и оставить код таким же, либо повторить его но уже двигаясь справа налево.

(c) Если мы вызовем функции из (a) и (b) для a_i то получим для этого элемента границы, где слева лежит элемент меньше него и где лежит такой же справо. Значит, на промежутке $[l+1, r-1]$ a_i будет минимальным. Так как первые две функции работают за $\mathcal{O}(n)$ и выдают результат сразу для всех элементов, то нам остается только один раз пройтись по массиву, чтобы посчитать произведение и выбрать максимум.

(d) Задача аналогична предыдущей, мы уже умеем найти для a_i границы, на которых он минимален. Остается посчитать сумму на этом промежутке за $\mathcal{O}(1)$. Это можно сделать с помощью префиксных сумм: $sum(i, j) = sum(j) - sum(i-1)$, которые можно подсчитать заранее.

Задача 6. *Вам дан массив из n элементов и список из m запросов $add(x, l, r)$: прибавить x к каждому элементу на отрезке $[l, r]$. За $\mathcal{O}(n+m)$ выведите массив, получающийся из исходного после выполнения заданных запросов.*

Решение.

При наивном подходе, если мы будем для каждого значения из данного массива искать все подходящие значения из запросов, алгоритм будет работать за $\mathcal{O}(nm)$. Чтобы полуичить $\mathcal{O}(n+m)$ нам нужно прежде предобработать запросы. Можно воспользоваться тем, что они даны все сразу и попытаться для каждого элемента массива сразу посчитать итоговое изменение после всех запросов.

Основная идея: предположим, что запросы отсортированы по левой границе. Тогда можно просто идти по ним и поддерживать сумму, прибавляя при открытии границы и отнимая, если граница закрывается. Чтобы нам не пришлось сортировать можно воспользоваться префиксной суммой. Создадим два массива pref_add и pref_sub, в один из которых будем добавлять x на pref_add[l], а -x в pref_sub[r + 1]. Теперь можно заметить формулу:

$$total\_add[i] = pref\_add[i-1] + pref\_add[i] + +pref\_sub[i]$$

Когда граница открывается, мы добавляем в общую сумму, а когда закрывается - отнимаем. И не забываем про запросы, которые еще не закончились. Время работы $T(n) = 2m+n+C$, то есть $\mathcal{O}(m+n)$

Решение на python:

```
1        def array_add(arr: List[int], req: Tuple[int]) -> List[int]:
2            pref_add = pref_sub = [0] * (len(arr) + 1)
3
4            for (x, l, r) in req:
5                pref_add[l] += x
6                pref_sub[r + 1] += -x
7
8            for i in range(1, len(n)):
9                pref_add[i] += pref_add[i - 1] + pref_sub[i]
10
11           return [arr[i] + pref_add[i] for i in range(len(arr))]
```

.

Задача 7. *Определить асимптотику $T(n) = 2 \cdot T(\lfloor \log n \rfloor) + 2^{\log^* n}$, где $\log^* n$ - итерированный логарифм.*

Решение.

$$\begin{aligned} T(n) &= 2T(\lfloor \log n \rfloor) + 2^{\log^* n} \\ &= 2^2 T(\lfloor \log \lfloor \log n) + 2^{\log^* \lfloor \log n + 1} \\ &= 2^3 T(\lfloor \log \lfloor \log \lfloor \log n) + 2^{\log^* \lfloor \log n + 1} \\ &= \cdots \\ &= 2^{\log^* n} T(1) + 2^{\log^* \lfloor \log n + 1} \\ &= 2^{\log^* n} + 2^{\log^* n} \\ &= 2^{\log^* n + 1} = 2^{\log^* n} \end{aligned}$$

Задача 8. *(дополнительно) Дана последовательность $a_1, a_2, \cdots, a_n \in \mathbb{Z}$ Найти $l, r (\leq 1 \leq l \leq r \leq n)$ такие, что сумма $\sum_{i=l}^{r} a_i$ была бы максимальной. Задачу требуется решить за линейное от n время.*

Решение. Более простая задача, где не нужно находить границы, решается с помощью динамического программирования. Пусть dp[i] максимальная встреченная сумма какого-то подмассива до i. Тогда мы либо можем добавить элемент и увеличить эту сумму, продолжив массив, либо i элемент больше, чем сумма + i и тогда начнется новый:

$$dp[i] = max(dp[i-1] + array[i], array[i])$$

Время работы $\mathcal{O}(n)$. Несложно добавить учет границ. Если текущая сумма больше максимальной встреченной, то нам нужно апдейтнуть указатели на best_left, best_right. Мы также двигаем curr_left каждый раз когда начинаем новый массив, а curr_right двигается в цикле.

Решение на python:

```
        def max_subsum(arr: List[int]) -> Tuple[int, int]:
            dp = arr
            best_l, best_r, best_sum = 0, 0, 0

            curr_l = 0
            for curr_r in range(1, len(arr)):
                if dp[curr_r - 1] > 0:
                    dp[curr_r] = dp[curr_r - 1] + dp[curr_r]
                else:
                    curr_l = curr_r

                 if dp[curr_r] > best_sum:
                    best_sum = dp[curr_r]
                    best_l, best_r = curr_l, curr_r

             return best_l, best_r
```

.