Алгоритмы. Домашнее задание №1

Горбунов Егор Алексеевич

17 сентября 2015 г.

Задание 1

(a) Имеем 2 функции $f, g : \mathbb{N} \to \mathbb{R}_{>0}$. Покажем, что

$$\exists_{N,C>0}\forall_{n\geq N}(f(n) \leq Cg(n)) \iff \exists_{C>0}\forall_{n}(f(n) \leq Cg(n))$$

($\Leftarrow$) Достаточно положить тогда, что $N = 1$ (т.к. $n \in \mathbb{N}$). ($\Rightarrow$) В предположении, что мы работаем с функциями f и g такими, что $\forall n \in \mathbb{N}\ (f(n) < \infty,\ g(n) < \infty)$. Тогда имеем, что $\exists_{N,C>0}\forall_{n\geq N} f(n) \leq Cg(n)$. Если $N \neq 1$, то есть номер n', на котором нарушается условие $f(n) \leq Cg(n)$, т.е. $f(n') > Cg(n')$. N — конечно, тогда мы можем перебрать все пары $i,\ j$ и найти такие, что $f(i)$ наибольшее, а $g(j)$ наименьшее. Ясно тогда, что $\forall\ n\ <\ N\ (g(n)\frac{f(i)}{g(j)}\ \geq\ f(n))$, но тогда и подавно в силу положительности $g, f:\ g(n)\frac{f(i)}{g(j)} + Cg(n)\ \geq\ f(n)$, т.е.

$$f(n)\ \leq\ (\frac{f(i)}{g(j)} + C)g(n)\ \forall\ n \in \mathbb{N}$$

Для $n \geq N$ ничего это так же верно в силу того, что мы лишь добавили положительное слагаемое к правой части неравенства $f(n) \leq Cg(n)$. Таким образом мы нашли новую константу $C' = \frac{f(i)}{g(j)} + C$, для которой уже справедливо утверждение без $\exists N$. ■

(b) Вопрос об эквивалентности следующих утверждений:

$$\forall_{C>0}\exists_{N}\forall_{n\geq N}(f(n) \leq Cg(n)) \iff \forall_{C>0}\forall_{n}(f(n) \leq Cg(n))$$

Рассмотрим: $g(n) = log(n),\ f(n) = \frac{1}{n}$. Ясно, что $f(n) = o(g(n))$, но так же, например для $C = 1$ $f(n) > g(n)$ при $n = 1$, а значит n должен быть по крайней мере ≥ 2, чтобы $f(n) \leq g(n)$, таким образом из определения $f = o(g)$ убрать условие о существовании N нельзя. ■

Задание 2

(a) В силу того, что во внутренний цикл вход происходит лишь при условии, что $used[i] == 0$, где i — переменная внешнего цикла, а если исполнение доходит до внутреннего цикла, то как минимум в $used[i]$ будет присвоена 1 и вообще, на каждой итерации внутреннего цикла происходит изменение некоторой ячейки массива $used$ с 0 на 1. Таким образом весь алгоритм сделает n присвоений, а значит суммарно будет сделано n итераций. Ответ: $\mathcal{O}(n)$

(b) Числа длины n в десятичной системе счисления.

(i) Сложение в столбик: будет сделано n операций поразрядного сложения, т.е. сложность $\mathcal{O}(n)$

(ii) Умножение в столбик: n операций умножения для каждого из разрядов второго числа, после чего нужно будет сложить n чисел (столько, сколько у второго числа разрядов). Итого $\mathcal{O}(n^2)$

(iii) Деление в столбик: $\mathcal{O}(n^2)$ Т.к. на каждом шаге мы находим число (от 1 до 10) и производим умножение на него того числа, на которое делим. Это происходит за $\mathcal{O}(n)$. Потом происходит вычитание 2-х чисел за $\mathcal{O}(n)$. И это повторяется $\mathcal{O}(n)$ раз.

(c) Заметим, что строку `int j = pi[i-1];` можно вынести из цикла наружу, т.к. в конце итерации `i` внешнего цикла переменная `j` равняется по значению `pi[i]`, а значит на следующей итерации она и так равняется по значению `pi[i-1]`. В строке `if (s[i]==s[j]) ++j` происходит увеличение `j`, это увеличение может произойти не более чем `n` раз (т.к. итераций внешнего цикла `n`). Уменьшаться же значение переменной `j` может лишь в строке `j = pi[j-1];`. Заметим, что `pi[k]` всегда меньше или равно, чем `k`, а значит, в строке `j = pi[j-1];` происходит гарантированное уменьшение `j` хотя бы на 1 (т.к. `pi[j-1]` $\leq$ `j - 1` < `j`). Но в силу условия цикла `while (j > 0)` и того, что увеличение `j` может произойти не более чем `n`, уменьшение тоже может произойти суммарно не более, чем `n` раз за всю работу алгоритма. А значит, суммарное время работы = $\mathcal{O}(n)$.
PS: данная процедура вычисляет префикс функцию для строки `s`. Известно, что вычисление этой функции, построенное приведённым в условии задачи образом, занимает линейное от длины строки время.

Задание 3

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

Задание 5 Расширенный алгоритм Евклида.

Будем искать $d = gcd(a,b)$, $a \geq b$ и такие x и y, что $d = ax + by$. Известно, что такие x и y существуют. Пускай мы умеем искать такие числа x и y для a и $(a \bmod b)$:

$$gcd(a, a \bmod b) = bx' + (a \bmod b)y' = bx' + (a - \left\lfloor \frac{a}{b} \right\rfloor b)y' = ay' + b(x' - \left\lfloor \frac{a}{b} \right\rfloor y')$$

И т.к. известно, что $gcd(a, a \bmod b) = gcd(a,b)$, то получаем:

$$gcd(a,b) = ay' + b(x' - \left\lfloor \frac{a}{b} \right\rfloor y')$$

Теперь мы можем построить рекурсивную процедуру вычисления разложения $gcd(a,b)$:

Вход: a, b — числа, $a \geq b \geq 0$

Выход: (x, y, d) такие, что $d = gcd(a,b) = ax + by$

```
1: procedure SuperEuclid(a, b)
2:     if b = 0 then
3:         return (1, 0, a)
4:     (x, y, d) ← SuperEuclid(b, a mod b)
5:     return (y, x − ⌊a/b⌋y, d)
```

Задание 6

1. Если $x = y$, то $gcd(x,y) = x = y = \frac{x+y}{2}$

2. Если x и y чётные, то ясно, что их gcd кратен 2, т.к. иначе его можно было бы увеличить умножением на 2. Ясно, что $gcd(x,y) \geq 2gcd(x/2, y/2)$, т.к. если $d|x/2, d|y/2$, то всяко $2d$ делит x и y. Аналогично $gcd(x/2, y/2) \leq gcd(x,y)/2$, ведь если $d|x, d|y$, то всяко $d/2|x/2$ и $d/2|y/2$. А следовательно должно выполняться равенство.

3. Если какой-то один элемент, например x, кратен 2, а y нет. Тогда ясно, что любой их общей делитель не кратен 2, а значит мы спокойно можем делить x на 2 и искать $gcd(x/2, y)$

4. 2 последних условных оператора очевидны. Если d делит x и y, то конечно d делит и их разность, а если d делит разность, то конечно оно делит каждый элемент разности по отдельности.

Таким образом каждый рекурсивный вызов **gcd** верен, а значит алгоритм корректен. Заметим, что в каждом вызове функции хотя бы один из аргументов либо уменьшается вдвое, либо становится чётным (т.к. разность двух нечётных чисел — число чётное), а значит на следующей итерации всяко произойдёт деление вдвое одного из аргументов. Итого получаем, что хотя бы каждые 2 рекурсивных вызова происходит деление одного из аргументов пополам, а значит суммарно алгоритм отрабатывает за $O(log(n))$.