1 Глава 1

1.1 Поиск контрпримеров

1.1.1 #1

Условие. Докажите, что значение $a+b$ может быть меньшим, чем значение $\min(a,b)$.

Допустим:

$$a = 0, b = -1 \tag{1}$$

Тогда:

$$\begin{align} 0 + (-1) = &\quad -1 \tag{2} \\ & -1 \quad < 0 = \min(0,1) \tag{3} \end{align}$$

Что и требовалось доказать.

1.1.2 #2

Условие. Докажите, что значение $a \times b$ может быть меньшим, чем значение $\min(a,b)$.

Допустим:

$$a = \frac{1}{2}, b = \frac{1}{4} \tag{4}$$

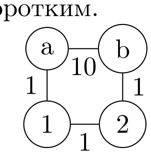
Тогда:

$$\begin{align} \frac{1}{2} \times \frac{1}{4} = \frac{1}{8} \tag{5} \\ \tfrac{1}{8} < \min(\tfrac{1}{2}, \tfrac{1}{4}) = \tfrac{1}{4} \tag{6} \end{align}$$

Что и требовалось доказать.

1.1.3 #3

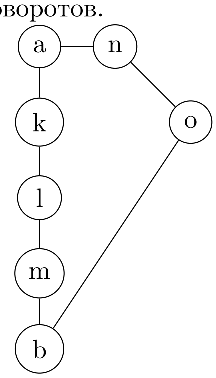
Условие. Начертите сеть дорог с двумя точками a и b, такими, что маршрут между ними, преодолеваемый за кратчайшее время, не является самым коротким.



Цена маршрута «a,1,2,b» равна 3, а цена маршрута «a,b» равна 10. То есть, маршрут, проходящий через больше точек, более длинный, преодолевается за меньшее время. Что и требовалось доказать.

1.1.4 #4

Условие. Начертите сеть дорог с двумя точками a и b, самый короткий маршрут между которыми не является маршрутом с наименьшим числом поворотов.



В данной сети дорог кратчайший маршрут от a до b это $anob$, в котором 3 ребра, но два поворота. Однако, маршрут без поворотов вообще, $aklmb$ не самый короткий — в нём 4 ребра. Что и требовалось доказать.

1.1.5 #5

Условие. Задача о рюкзаке: имея множество целых чисел $S = \{s_1, s_2, \ldots, s_n\}$ и целевое число T, найти такое подмножество множества S, сумма которого в точности равна T. Например, множество $S = \{1, 2, 5, 9, 10\}$, содержит такое подмножество, сумма элементов которого равна T = 22, но не T = 23.

Найти контрпримеры для каждого из следующих алгоритмов решения задачи о рюкзаке, т. е., нужно найти такое множество S и число T, при которых подмножество, выбранное с помощью данного алгоритма, не до конца заполняет рюкзак, хотя правильное решение и существует:

- вкладывать элементы множества S в рюкзак в порядке слева направо, если они подходят (т. е., алгоритм «первый подходящий»);
- вкладывать элементы множества S в рюкзак в порядке от наименьшего до наибольшего (т. е., используя алгоритм «первый лучший»);
- вкладывать элементы множества S в рюкзак в порядке от наибольшего до наименьшего.

Первый подходящий. $T = 10, S = \{9, 2, 2, 2, 2, 2\}$
Первый лучший. $T = 6, S = \{1, 2, 5, 4\}$
От самого большого. $T = 6, S = \{3, 2, 5, 4\}$

1.1.6 #6

Условие. Задача о покрытии множества: имея семейство подмножеств $S_1, \ldots, S_m$ универсального множества $U = \{1, \ldots, n\}$, найдите семейство подмножеств

$T \subset S$ наименьшей мощности, чтобы $\bigcup_{t_i} \in T^{t_i} = U$. Например, для семейства подмножеств $S_1 = \{1,3,5\}$, $S_2 = \{2,4\}$, $S_3 = \{1,4\}$, $S_5 = \{2,5\}$ покрытием множества будет семейство подмножеств S_1 и S_2.

Приведите контрпример для следующего алгоритма: выбираем самое мощное подмножество для покрытия, после чего удаляем все его элементы из универсального множества; повторяем добавление подмножества, содержащего наибольшее количество неохваченных элементов, пока все элементы не будут покрыты.

$$S_1 = \{1,3,5,7,9\} \tag{7}$$
$$S_2 = \{1,2\} \tag{8}$$
$$S_3 = \{3,4\} \tag{9}$$
$$S_4 = \{5,6\} \tag{10}$$
$$S_5 = \{7,8\} \tag{11}$$
$$S_6 = \{9,10\} \tag{12}$$

Наилучшее покрытие это $S_2 \cup S_3 \cup S_4 \cup S_5 \cup S_6$, но жадный алгоритм вначале выберет S_1 как самое мощное подмножество, а потом не останется ничего другого, как добавлять все остальные подмножества, то есть, в итоге алгоритм выберет все исходные подмножества в качестве результата работы, $T = S$.

1.2 Доказательство правильности

1.2.1 #7

Условие. Докажите правильность следующего рекурсивного алгоритма умножения двух натуральных чисел для всех целочисленных констант $c \geqslant 2$:

```
function MULTIPLY(y, z)
    if z = 0 then
        return 0
    else
        return multiply(cy, ⌊z/c⌋) + y * (z mod c)
    end if
end function
```

Вычислим результат работы multiply на каком-нибудь простом примере:

$$
\begin{aligned}
& c = 2, y = 2, z = 3 && (13)\\
& multiply(2, 3) && (14)\\
= \quad & multiply(2 * 2, \lfloor 3/2 \rfloor) + 2 * (3 \mod 2) && (15)\\
= \quad & multiply(4, 1) + 2 * 1 && (16)\\
= \quad & multiply(4, 1) + 2 && (17)\\
= \quad & multiply(4 * 2, \lfloor 1/2 \rfloor) + 4 * (1 \mod 2) + 2 && (18)\\
= \quad & multiply(8, 0) + 4 * 1 + 2 && (19)\\
= \quad & 0 + 4 + 2 && (20)\\
= \quad & 6 && (21)
\end{aligned}
$$

Пусть $z = 0$. Тогда $multiply(y, z) = 0$.

Допустим теперь, что алгоритм правильный, то есть $multiply(y, z) = yz$ при:

$$z \leqslant n \tag{22}$$

$$y \geqslant 1 \tag{23}$$

$$c \geqslant 2 \tag{24}$$

Теперь докажем, что $multiply(y, n + 1) = y(n + 1)$.

$$
\begin{aligned}
& mutiply(y, n + 1) = && (25)\\
& multiply(y * c, \lfloor \frac{n+1}{c} \rfloor) + y * ((n + 1) \mod c) = && (26)\\
& y * c * \lfloor \frac{n+1}{c} \rfloor + y * ((n + 1) \mod c) = && (27)\\
& y * (c * \lfloor \frac{n+1}{c} \rfloor + (n + 1) \mod c) && (28)
\end{aligned}
$$

Теперь докажем, что $c * \lfloor \frac{n+1}{c} \rfloor + (n + 1) \mod c) = n + 1$.

$$n + 1 \equiv z \tag{29}$$

$$c\lfloor \frac{z}{c} \rfloor + z \mod c = z \tag{30}$$

z/c в случае деления произвольных натуральных чисел — операция с остатком. $\lfloor z/c \rfloor$ же избавляется от этого остатка.

В то же время, $z \mod c$ это операция получения *остатка от деления* z на c.

Формула (30) может использоваться как есть, потому что следует из определения операций, которые в ней используются.

Если мы умножим c на операцию, убирающую остаток от деления z на c, то получим как бы «восстановленную» z без остатка после деления.

$$3 \times \lfloor \frac{10}{3} \rfloor = 3 \times 3 = 9 \tag{31}$$

Но если мы потом прибавим буквально остаток от этого же деления, мы получим исходное число!

$$10 \mod 3 = 1 \tag{32}$$

А $9 + 1 = 10$, как и следует из объяснения выше.

Таким образом, из (30) и (28) следует:

$$y \times (c \times \lfloor \frac{n+1}{c} \rfloor + (n+1) \mod c) = y \times (n+1) \tag{33}$$

$$\Rightarrow multiply(y, n+1) = y \times (n+1) \tag{34}$$

Что и требовалось доказать.

1.2.2 #8

Условие. Докажите правильность следующего алгоритма вычисления полинома $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^{+} a_0$:

function HORNER(A, x)
    $p = A_n$
    for $i \leftarrow n-1, 0$ **do**
        $p = p \times x + A_i$
    end for
    return p
end function

Пусть $n = 0$. Тогда цикл не выполнится ни разу и результатом будет A_0.

Пусть $n = 1$. Тогда:

$$p = A_1 \tag{35}$$

$$i = 0: \quad p = A_1 \times x + A_0 \tag{36}$$

Пусть $n = 2$. Тогда:

$$p = A_2 \tag{37}$$

$$i = 1: \quad p = A_2 \times x + A_1 \tag{38}$$

$$i = 0: \quad p = (A_2 \times x + A_1) \times x + A_0 = A_2 \times x^2 + A_1 \times x + A_0 \tag{39}$$

Допустим, что алгоритм правильный при любых n. Посмотрим, что происходит при $n + 1$:

$$p = A_{n+1} \tag{40}$$
$$i = n: \quad p = A_{n+1} \times x + A_n \tag{41}$$
$$i = n-1: \quad p = (A_{n+1} \times x + A_n) \times x + A_{n-1} \tag{42}$$
$$\ldots \tag{43}$$
$$i = n - n = 0: \quad p = A_{n+1}x^{n+1} + A_n x^n + A_{n-1}x^{n-1} + \ldots + A_{n-n} \tag{44}$$

Что и требовалось доказать.

1.2.3 #9

Условие. Докажите правильность следующего алгоритма сортировки:

```
function bubblesort(A: list[1...n])
    var int i,j
    for i from n to 1
        for j from 1 to i - 1
            if (A[j] > A[j+1])
                swap A[j] and A[j + 1]
```

Пусть $n = 2$. Тогда:

$$i = 2, j = 1: \quad A[1] > A[2] ? swap : keep \tag{45}$$

Пусть алгоритм может отсортировать любой массив длиной n. Может ли он отсортировать массив длиной $n+1$?

Передача в этот алгоритм массив длиной $n+1$ эквивалентна замене его на следующий код:

```
function bubblesort(A: list[1...n+1])
    var int i,j
    for j from 1 to n + 1 - 1
        if (A[j] > A[j + 1])
            swap A[j] and A[j + 1]
    for i from n to 1
        for j from 1 to i - 1
            if (A[j] > A[j+1])
                swap A[j] and A[j + 1]
```

По индукции мы допустили, что основная пара вложенных циклов действительно отсортирует все элементы от 1 до n.

Мы же добавили в код ещё один цикл:

```
for j from 1 to n
    if (A[j] > A[j + 1])
        swap A[j] and A[j + 1]
```

Этот цикл проверит два новых элемента: `A[n]` и `A[n+1]`, последние в списке. Элемент `A[n+1]` идёт после отсортированных элементов `A[1] ... A[n]`. Если они не в правильном порядке, он их поменяет местами, в точности как случай когда $n = 2$.

Таким образом, этот алгоритм действительно сортирует массивы любой длины.