ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

Дисциплины
"ЯЗЫКИ ПРОГРАММИРОВАНИЯ"
"ПРОГРАММИРОВАНИЕ"

Алгоритмы на базе циклических конструкций

Преподаватель:

Доцент Кафедры ВС, к.т.н.

Поляков Артем Юрьевич

Циклические алгоритмы

В основе многих циклических алгоритмов (или их фрагментов) лежит фундаментальная идея о том, что результат вычислений на каждом шаге цикла должен зависеть от результатов предыдущего шага.

```
#include <stdio.h>
int main()
{
  int i, S, N, a;
  scanf("%d",&N);
  i = 1; S = 0;
  while( i <= N ){
    scanf("%d",&a);
    S = S + a;
    i = i + 1;
  }
  printf("Sum a[1...%d] = %d\n", N, S);
  return 0;
}
```

Новое значение *S* выражается через предыдущее. То же самое относится к i. И, в принципе, к a!

В условии наличия циклов программа как бы разделена на дискретные промежутки, на которых значения переменных неизменны, изменение любой из переменных означает изменение состояния всей программы.

Циклические конструкции (2)

С точки зрения математики, переменная в программе, имеющей циклы, может быть представлена в виде *последовательности*. Обобщенным математическим выражением этой идеи являются *рекуррентные соотношения*.

Будем говорить, что последовательность a задана рекуррентным соотношением, если задан начальный ее элемент a_0, а также функциональная зависимость каждого последующего элемента от предыдущего:

$$a_{k+1} = f(a_k)$$

Циклические конструкции (3)

Рекуррентными соотношениями можно описать рассмотренный ранее алгоритм вычисления суммы элементов последовательности:

```
#include <stdio.h>
int main()
{
  int i, S, N, a;
  scanf("%d",&N);
  i = 1; S = 0;
  while( i <= N ){
    scanf("%d",&a);
    S = S + a;
    i = i + 1;
  }
  printf("Sum a[1...%d] = %d\n", N, S);
  return 0;
}
```

Рекуррентными соотношениями можно описать рассмотренный ранее алгоритм вычисления суммы элементов последовательности:

1. Счетчик цикла: $i_0 = 1$, $i_{k+1} = i_k + 1$
2. Сумма:

$$S = a_1 + a_2 + \ldots + a_{N-1} + a_N$$

$$s_0 = 0,\ s_{k+1} = s_{k-1} + a_k,\ S = s_N$$

План лекции

1. Двоичная система счисления и алгоритмы перевода из десятичной системы в двоичную.
2. Элементарная теория чисел. Алгоритмы поиска наибольшего общего делителя и наименьшего общего кратного целых чисел.

Двоичная система счисления и алгоритмы перевода из десятичной системы в двоичную.

Позиционные системы счисления

Под позиционной системой счисления понимается b-ричная система счисления, которая определяется целым числом b, называемым основанием системы счисления.

Целое число без знака x в b-ричной системе счисления представляется в виде конечной линейной комбинации степеней числа b:

$$x_b = \sum_{k=0}^{n-1} a_k b^k,$$

где $\mathbf{0 \le a_k < b}$ – набор весовых коэффициентов (разрядов), k – номер разряда. Если $b = 2$, а $0 \le a_k < 2$ то система счисления называется *двоичной*. Примеры:

$$\mathbf{1010011 = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0}$$

$$\mathbf{a = \{1, 0, 1, 0, 0, 1, 1\}}$$

Перевод между системами счисления

Одно и то же число может быть записано в различных системах счисления (СС). На рис. показаны фотографии автомобилей. Опишем их количество с помощью нескольких систем счисления:

22_5 15_7 C_{16} 110_3

12_{10} 20_6 30_4

13_9 10_{12}

1100_2 14_8 11_{11}

Существует два способа перевода между системами счисления:

- При первом все операции производятся в целевой СС.
- Во втором – в исходной СС.

Перевод целых чисел между позиционными системами (способ I)

Перевод из системы счисления с основанием b в СС с основанием c:

$$x_b \rightarrow x_c$$

$$x_b = a_N a_{N-1} \ldots a_1 a_0$$

$$x_c = (a_N)_c \cdot (b_c)^N + (a_{N-1})_c \cdot (b_c)^{N-1} + \ldots + (a_1)_c \cdot (b_c)^1 + (a_0)_c \cdot (b_c)^0$$

Примеры:

$$10110_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 =$$

$$= 32 + 8 + 4 + 2 = 46_{10}$$

$$2AF_{16} = 2 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0 = 512 + 160 + 15 = 687_{10}$$

$$48_{10} = 4_{16} \cdot A^1 + 8_{16} \cdot A^0$$

$$48_{10} = 100_2 \cdot 1010^1 + 1000_2 \cdot 1010^0$$

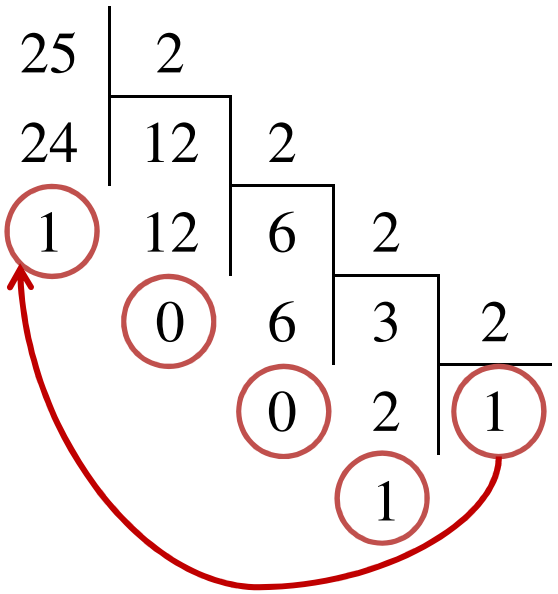
Вычисление в недесятичной системе счисления затруднительны для человека

Перевод целых чисел между позиционными системами (способ II)

Для перевода числа x_a (записанного в системе счисления a) в систему с основанием b необходимо разделить x_a на b. Деление продолжать до тех пор, пока частное не станет меньше b.

Остатки от деления, записанные в обратном порядке, начиная с частного, будут искомым числом.

$$25_{10} = 11001_2$$



Соотношения между числами в разных системах счисления

Вне зависимости от используемой системы счисления числа явление, описываемое этим числом, остается **одним и тем же**. Например, количество машин не изменяется от выбора системы счисления:

$$1100_2 = 110_3 = 30_4 = 22_5 = 20_6 = 15_7 = 14_8 = 13_9 = 12_{10} = 11_{11} = 10_{12}$$

Кроме того, что само число остается одним и тем же, результаты арифметических действий над одинаковыми числами в разных системах счисления также должны давать одинаковые результаты. В противном случае описываемые явления перестанут согласовываться с реальностью. Например, количество машин на фотографии, **уменьшенное в три раза**, будет составлять:

$$100_2 = 11_3 = 10_4 = 4_5 = 4_6 = 4_7 = 4_8 = 4_9 = 4_{10} = 4_{11} = 4_{12}$$

Способ II. Математическая основа

Рассмотрим некоторое число x, записанное, например, в десятичной и b-ричной системах счисления:

$$x_{10} = x_b = a_N a_{N-1} \ldots a_1 a_{0(b)}$$

Арифметические операции над числом, записанным в разных системах счисления, дают одинаковый результат.

Если это так, то справедливо следующее утверждение:

$$x_{10} \textbf{ mod } b = x_b \textbf{ mod } b,$$

где y **mod** z – остаток от деления y на z.

Особенностью этой операции является то, ее результатом является значение младшего разряда b-ричного представления числа x:

$$x_{10} \textbf{ mod } b = x_b \textbf{ mod } b = \boldsymbol{a_0}$$

Например: 12_{10} % 3 = **0** ($12_{10} = 11\mathbf{0}_3$); 12_{10} % 9 = **3** ($12_{10} = 1\mathbf{3}_9$);

Способ II. Математическая основа (2)

Результат деления нацело числа x на b в разных СС также будет одинаков:

$$x_{10} \textbf{ div } b = x_b \textbf{ div } b,$$

где y **div** z – целая часть от деления y на z.

Особенностью этого результата является то, что мы исключаем младший разряд в b-ричном представлении числа x:

$$x_{10} \textbf{ div } b = x_b \textbf{ div } b = a_N a_{N-1} \ldots a_1 \cancel{a_0}_{(b)} = a_N a_{N-1} \ldots a_{1(b)}$$

Например:

$12_{10} / 3 = \mathbf{4 = 11_3 = 10_3 + 1_3 = 3_{10} + 1}$, $12_{10} = \mathbf{11}0_3$;

$12_{10} / 9 = \mathbf{1,}$ $12_{10} = \mathbf{1}3_9$;

$12_{10} / 2 = \mathbf{6 = 110_2 = 2^2 + 2^1 = 4 + 2 = 6,}$ $12_{10} = \mathbf{110}0_2$;

Продолжая действовать аналогичным образом можно получить значение второго разряда b-ричного представления и т.д.

Рекуррентное соотношение (способ II)

$$
\begin{aligned}
& i_0 = 0, \\
& w_0 = x, \\
& \mathrm{s}_0 = 1 \\
& v_0 = 0 \\
& \left\{ \begin{array}{l}
i_{k+1} = i_k + 1 \\
d = w_k \ \mathbf{mod}\ b \\
v_{k+1} = v_k + (d)_b \cdot s_k \\
s_{k+1} = s_k \cdot b \\
w_{k+1} = w_k \ \mathbf{div}\ b
\end{array} \right.
\end{aligned}
$$

Рекуррентное соотношение (способ II) [Пример 1]

Пусть $x = 126_{10}$, $b = 3$

$i_0 = 0,$
$w_0 = x,$
$s_0 = 1$
$v_0 = 0$

$$\begin{cases} i_{k+1} = i_k + 1 \\ d = w_k \ \mathbf{mod}\ b \\ v_{k+1} = v_k + (d)_b \cdot s_k \\ s_{k+1} = s_k \cdot b \\ w_{k+1} = w_k \ \mathbf{div}\ b \end{cases}$$

k	i_k	w_k	$d_{(10)}$	$d_{(3)}$	$s_{k(10)}$	$s_{k(3)}$	v_k
0	0	126	0	0	1	1	0
1	1	42	0	0	3	10	0
2	2	14	2	2	9	100	200
3	3	4	1	1	27	1000	1200
4	**4**	**1**	**1**	**1**	**81**	**10000**	**11200**
5	5	0	0	0	243	10000	011200
6	6	0	0	0	729	100000	0011200

…

Рекуррентное соотношение (способ II)
[Пример 2]

Пусть $x = 1771_{10}$, $b = 16$

$i_0 = 0,$
$w_0 = x,$
$s_0 = 1$
$v_0 = 0$

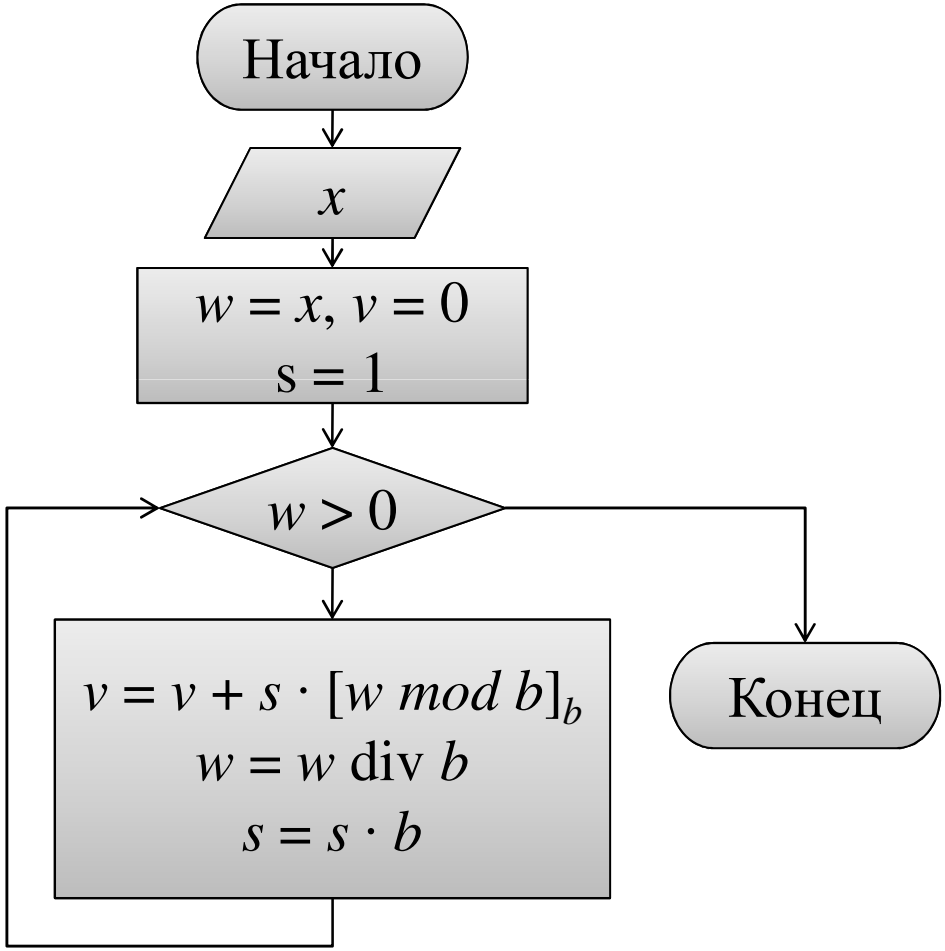
$$\begin{cases} i_{k+1} = i_k + 1 \\ d = w_k \ \mathbf{mod}\ b \\ v_{k+1} = v_k + (d)_b \cdot s_k \\ s_{k+1} = s_k \cdot b \\ w_{k+1} = w_k \ \mathbf{div}\ b \end{cases}$$

k	i_k	w_k	$d_{(10)}$	$d_{(16)}$	$s_{k(10)}$	$s_{k(16)}$	v_k
0	0	1771	11	B	1	1	B
1	1	110	14	E	16	10	EB
2	**2**	**6**	**6**	**6**	**256**	**100**	**6EB**
3	3	0	0	0	4096	1000	06EB
4	4	0	0	0	65536	10000	006EB

. . .

Способ II. Алгоритм.

$$
\begin{cases}
i_0 = 0, \\
w_0 = x, \\
s_0 = 1 \\
v_0 = 0 \\
i_{k+1} = i_k + 1 \\
d = w_k \ \mathbf{mod}\ b \\
v_{k+1} = v_k + (d)_b \cdot s_k \\
s_{k+1} = s_k \cdot b \\
w_{k+1} = w_k \ \mathbf{div}\ b
\end{cases}
$$



Перевод вещественных чисел между системами счисления

Ранее были рассмотрены способы, позволяющие выполнять целочисленные преобразования между системами счисления. Однако большой диапазон реальных явлений не может быть достоверно описан при помощи целых чисел.

В позиционных системах счисления вещественное число, состоящее из N целых и K дробных разрядов записывается следующим образом:

$$\boldsymbol{x_b = a_N a_{N-1} \ldots a_1 a_0 , a_{-1} a_{-2} a_{-3} \ldots a_{-(K-1)} a_{-K}} =$$

$$a_N \cdot b^N + a_{N-1} \cdot b^{N-1} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0 + a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + \ldots + a_{-K} \cdot b^{-K}$$

Любое вещественное число $x = a_N\, a_{N-1} \ldots a_1 a_0 , a_{-1} a_{-2} a_{-3} \ldots a_{-K}$ может быть представлено в виде суммы $x = x' + x''$, где

x' = [x] – целая часть x, $x' = a_N a_{N-1} \ldots a_1 a_0 , \mathbf{0}$

x'' = x – [x] – дробная часть x, $0 \le x'' < 1$: $x'' = \mathbf{0}, a_{-1} a_{-2} a_{-3} \ldots a_{-K}$

Перевод вещественных чисел между системами счисления (2)

Любое вещественное число $x = a_N\, a_{N-1} \ldots a_1 a_0, a_{-1} a_{-2} a_{-3} \ldots a_{-K}$ может быть представлено в виде суммы $x = x' + x''$, где

x' = [x] – целая часть x, $x' = a_N\, a_{N-1} \ldots a_1 a_0$**,0**

x'' = x – [x] – дробная часть x, $0 \le x'' < 1$: x'' = **0,**$a_{-1} a_{-2} a_{-3} \ldots a_{-K}$

Алгоритм перевода для целых чисел x' был рассмотрен ранее, поэтому сосредоточимся на алгоритме перевода дробной части.

Используем то же свойство, что и при построении алгоритма для целых: арифметические операции над числами в любой системе счисления дают одинаковый результат:

$$x''_{10} \cdot b = x''_b \cdot b = a_{-1}, a_{-2} a_{-3} \ldots a_{-K(b)}$$

Таким образом **умножение дробной части на основание целевой системы счисления в результате выводит наиболее значимый дробный разряд в целую часть числа**.

Перевод вещественных чисел между системами счисления (3)

$$\boldsymbol{x''_{10} \cdot b = x''_{b} \cdot b = a_{-1},a_{-2}a_{-3}\ldots a_{-K(b)}}$$

Таким образом умножение дробной части на основание целевой системы счисления в результате выводит наиболее значимый дробный разряд в целую часть числа.

Далее с помощью операции взятия целой части можно определить значение первого разряда, а потом исключить его из рассмотрения:

$$a_{-1} = [x''_{10} \cdot b] = [a_{-1},a_{-2}a_{-3}\ldots a_{-K(b)}] = a_{-1}$$

$$x'' = x''_{10} \cdot b - [x''_{10} \cdot b] = 0,a_{-2}a_{-3}\ldots a_{-K(b)}$$

Продолжая аналогичным образом можно выделить остальные дробные разряды b-ричного представления.

Рекуррентное соотношение (дробная часть)

$$
\begin{cases}
i_0 = 0,\ w_0 = x - [x], \\
\mathrm{s}_0 = 1 / b,\ v_0 = 0 \\
i_{k+1} = i_k + 1,\ d = [w_k \cdot b] \\
v_{k+1} = v_k + (d)_b \cdot s_k,\ s_{k+1} = s_k / b \\
w_{k+1} = w_k \cdot b - [w_k \cdot b]
\end{cases}
$$

Алгоритмы элементарной теории чисел

Наибольший общий делитель

Наибольший общий делитель (НОД, GCD от англ. Greatest Common Divisor) для двух целых чисел m и n называется наибольший из их общих делителей.

Принятые обозначения НОД чисел m и n:

- НОД(m, n)
- (m, n) ← В рамках лекции
- gcd(m, n)

Пример:

$$(60, 84) = 12$$
$$(17\,640, 26180) = 140.$$

НОД существует и однозначно определён,
если хотя бы одно из чисел m или n не ноль.

$(n, m) = (m, n)$ - симметричность

Поиск НОД разложением на множители

Одним из способов нахождения (n,m) и $[n,m]$ является разложение чисел n и m на простые множители.
Пусть

$$n = p_1^{d_1} \cdot p_2^{d_2} \cdot p_3^{d_3} \cdot \ldots \cdot p_k^{d_k}$$

$$m = p_1^{e_1} \cdot p_2^{e_2} \cdot p_3^{e_3} \cdot \ldots \cdot p_k^{e_k}$$

Тогда

$$(n,m) = p_1^{\min(d_1,e_1)} \cdot p_2^{\min(d_2,e_2)} \cdot p_3^{\min(d_3,e_3)} \cdot \ldots \cdot p_k^{\min(d_k,e_k)}$$

$$[n,m] = p_1^{\max(d_1,e_1)} \cdot p_2^{\max(d_2,e_2)} \cdot p_3^{\max(d_3,e_3)} \cdot \ldots \cdot p_k^{\max(d_k,e_k)}$$

Поиск НОД разложением на множители (2)

Примеры:

$$60 = 2 \cdot 2 \cdot 3 \cdot 5 = 2^2 \cdot 3 \cdot 5 = 2^2 \cdot 3^1 \cdot 5^1 \cdot 7^0$$

$$84 = 2 \cdot 2 \cdot 3 \cdot 7 = 2^2 \cdot 3 \cdot 7 = 2^2 \cdot 3^1 \cdot 5^0 \cdot 7^1$$

$$(60, 84) = 2^2 \cdot 3^1 \cdot 5^0 \cdot 7^0 = 12$$

$$[60, 84] = 2^2 \cdot 3^1 \cdot 5^1 \cdot 7^1 = 420$$

$$17\,640 = 2^3 \cdot 3^2 \cdot 5 \cdot 7^2 = 2^3 \cdot 3^2 \cdot 5 \cdot 7^2 \cdot 11^0 \cdot 17^0$$

$$26\,180 = 2^2 \cdot 5 \cdot 7 \cdot 11 \cdot 17 = 2^3 \cdot 3^0 \cdot 5 \cdot 7^1 \cdot 11^1 \cdot 17^1$$

$$(17\,640, 26180) = 2^2 \cdot 3^0 \cdot 5 \cdot 7^1 \cdot 11^0 \cdot 17^0 = 140$$

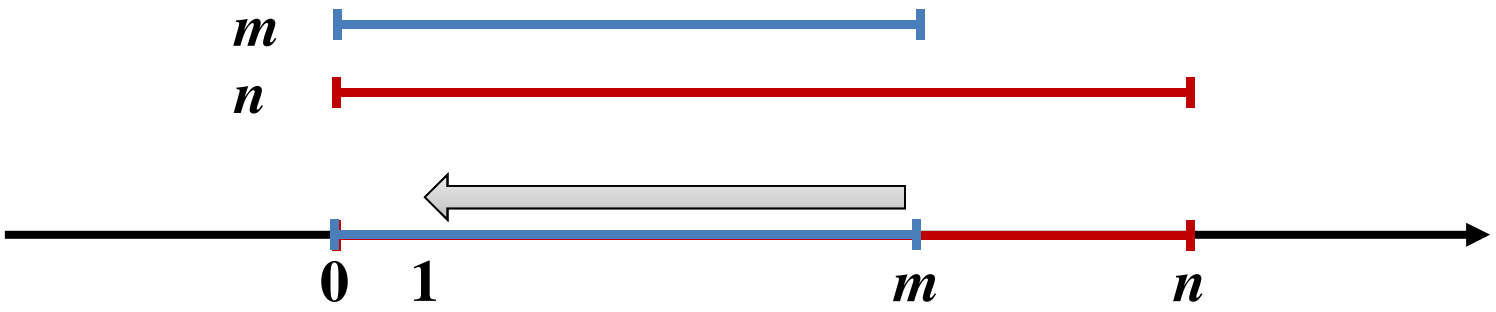
$$[17\,640, 26180] = 2^3 \cdot 3^2 \cdot 5 \cdot 7^2 \cdot 11^1 \cdot 17^1 = 3\,298\,680$$

Линейный метод поиска НОД

Пусть даны целые числа *n* и *m*. Необходимо найти их НОД (n, m).
Решение:

По определению (*n*, *m*) – это наибольшее целое число, на которое нацело делится и *n* и *m*. Поэтому наиболее очевидным способом вычисления (*n*, *m*) является перебор ВСЕХ чисел *x*, которые могут быть делителями и *n* и *m*.

Очевидно, что некоторое целое число *n* не может поделиться нацело на число *x*: *x* > *n*. Поэтому естественным будет искать делители в диапазоне [2, **min**(*n*,*m*)]:



Линейный метод поиска НОД (рекуррентное соотношение)

$i_0 = 0,$
$w_0 = m,$
$f = 0$

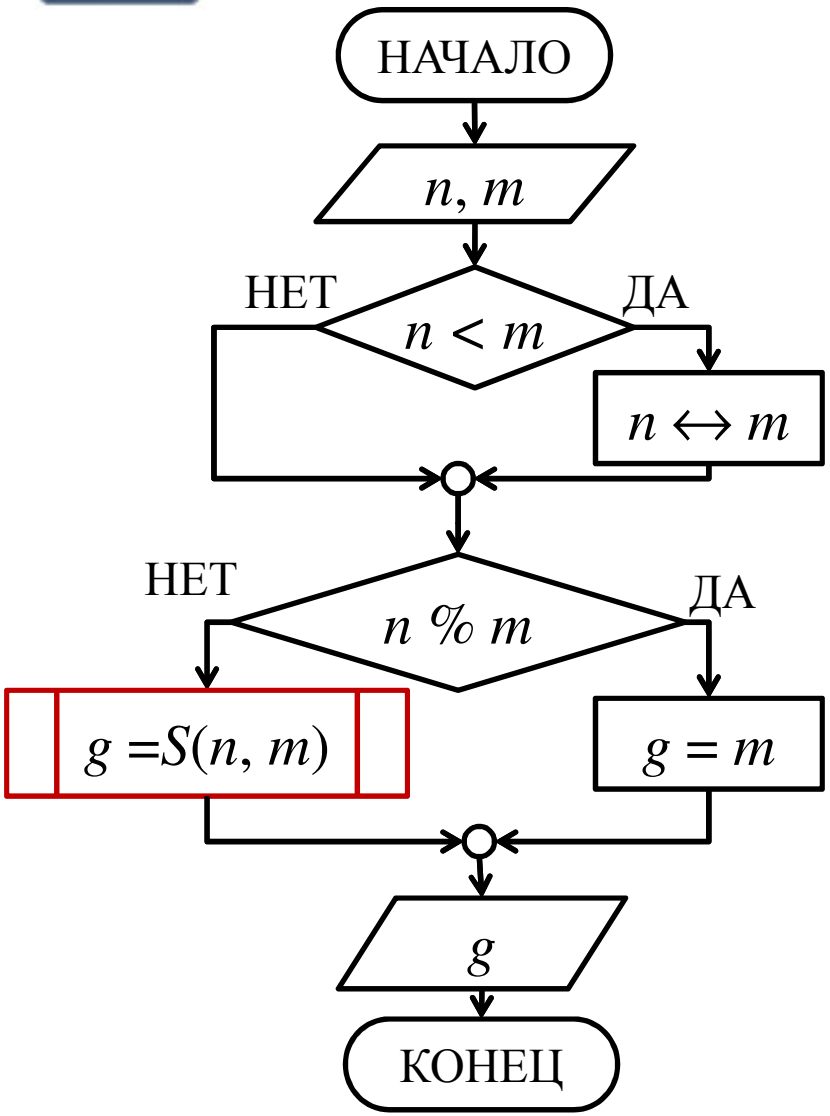
$$\begin{cases} i_{k+1} = i_k + 1 \\ f' = n \ \mathbf{mod}\ w_k = 0 \\ f'' = m \ \mathbf{mod}\ w_k = 0 \\ f = f \ \text{ИЛИ} \ f' \ \text{И} \ f'' \\ w_{k+1} = w_k - 1 \end{cases}$$

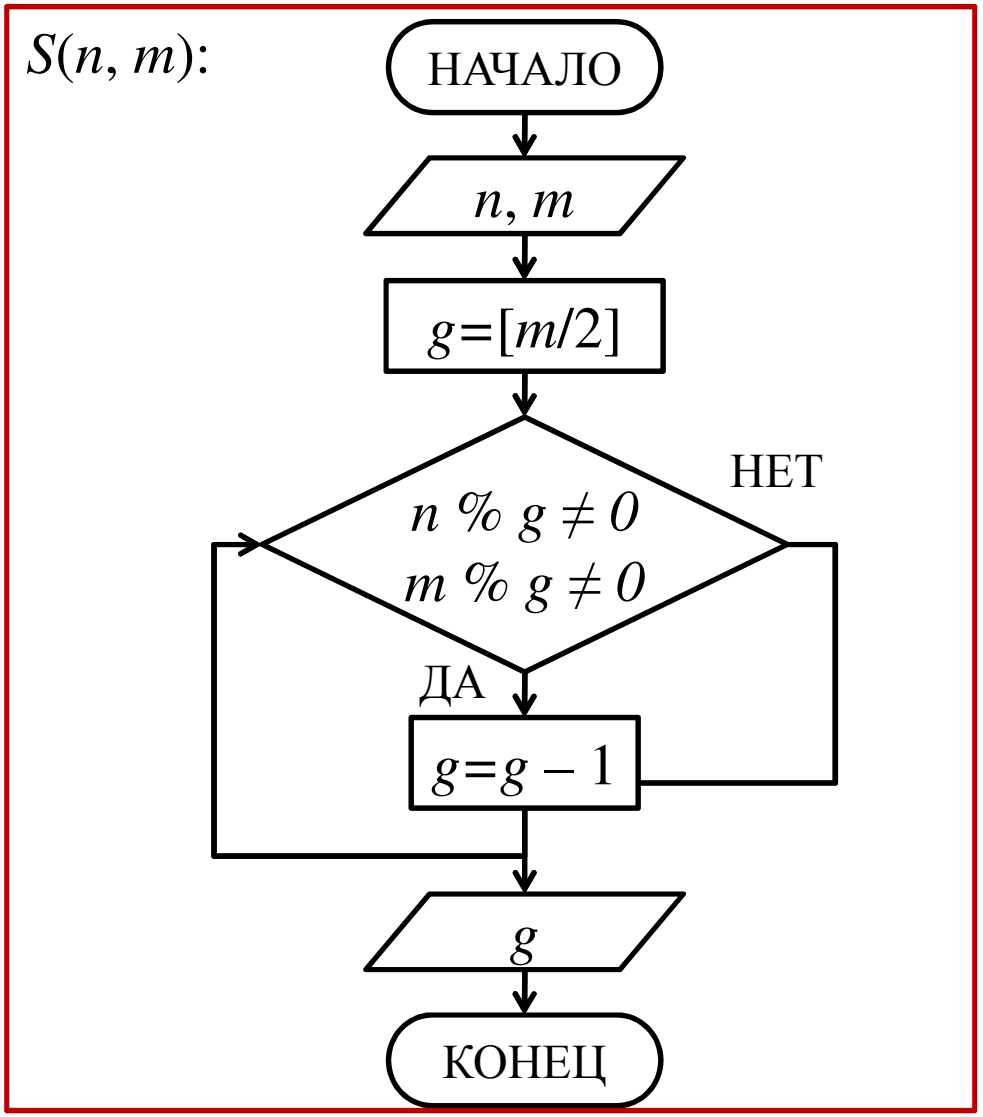
Пусть $n = 9,\ m = 6$

k	i_k	w_k	n **mod** w_k	m **mod** w_k	f
0	0	6	3	0	0
1	1	5	4	1	0
2	**2**	4	1	2	0
3	3	**3**	**0**	**0**	**1**
4	4	2	1	0	1
5	5	1	1	1	1
6	6	0	∞	∞	∞
7	7	-1			

…

Линейный метод поиска НОД (алгоритм)





Анализ линейного алгоритма

Простейшие случаи:

$n = 1$ или $m = 1$ => $(n, m) = 1$

$n = m$ => $(n, m) = m$

$m<n$, $n \bmod m = 0$ => $(n, m) = m$

$m = a \cdot b$

a	b	
1	m	
2	$[m / 2]$	if m **mod** 2 = 0
3	$[m / 3]$	if m **mod** 3 = 0
…		
$[m / 2]$	2	if m **mod** 2 = 0
m	1	

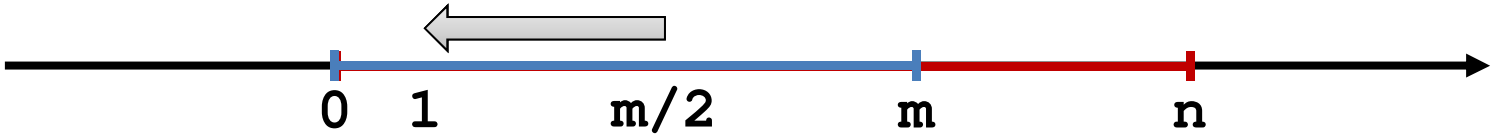
В противном случае:

Пусть $m < n$ и $m \neq (m, n)$.

Тогда $n = m \cdot d + q$, при этом $d \geq 1$, иначе $m \geq n$.

Как было показано ранее $(m, n) \in [1, m - 1]$ $(m, n) < m$ и (так как m делится на НОД): $m = (m, n) \cdot d'$, $d' = 1, 2, \ldots, l, \ldots$

Если $m \neq (m, n)$, то первым потенциальным разложением m на делители является $m = [m/2] \cdot 2$ => $(n, m) \in [1, [m/2]]$



Псевдокод

Блок-схемы являются наглядным, но громоздким способом описания алгоритмов. Альтернативным подходом к описанию алгоритмов является *псевдокод* – язык описания алгоритмов, использующий ключевые слова языков программирования, но опускающий подробности и специфический синтаксис. В рамках данного курса будем придерживаться следующих соглашений:

1. Обозначение операции присваивания: $t \leftarrow n$
2. Для обозначения тел циклов и ветвлений используются **только** отступы.
3. Ветвление **if** <условие> **then** <тело ветви с новой строки>:

```
if m > n then
        t ← n, n ← m, m ← t
```

4. Цикл while: **while** <условие> **do** <тело ветви с новой строки>

```
while (m mod g) ≠ 0 OR (n mod g) ≠ 0 do
        g ← g – 1
```

Псевдокод (2)

Блок-схемы являются наглядным, но громоздким способом описания алгоритмов. Альтернативным подходом к описанию алгоритмов является *псевдокод* – язык описания алгоритмов, использующий ключевые слова языков программирования, но опускающий подробности и специфический синтаксис. В рамках данного курса будем придерживаться следующих соглашений:

5. Логические операции: НЕ – **not**, ИЛИ – **or**, И – **and**.
6. Целочисленная арифметика: **div** – деление нацело, **mod** – остаток

Алгоритм линейного поиска НОД (v2)

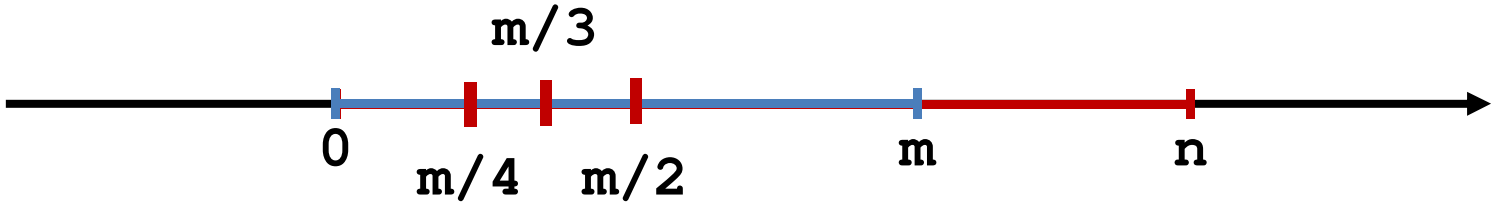
input n, m
if $m > n$ **then**
    $t \leftarrow n$, $n \leftarrow m$, $m \leftarrow t$
if (n **mod** $m = 0$) **then**
    $g \leftarrow m$
else
    $g \leftarrow [m/2]$
    while (m **mod** g) $\neq 0$ **or** (n **mod** g) $\neq 0$ **do**
        $g \leftarrow g - 1$
output g

Дальнейшее усовершенствование линейного алгоритма

Количество операций можно существенно сократить за счет перебора **только возможных** делителей и **исключения** чисел из диапазонов:

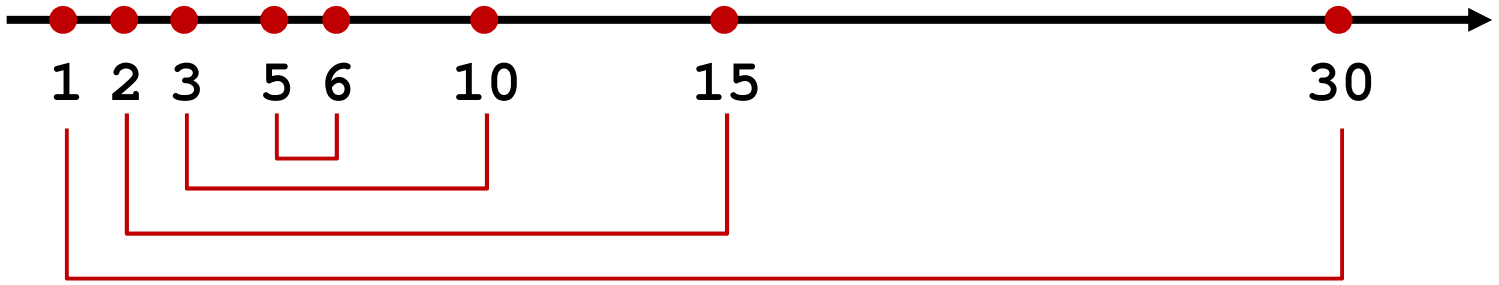
1. [m/2] + 1, … , *m* – 1
2. [m/3] + 1, … , [m/2] – 1
3. [m/4] + 1, … , [m/3] – 1
4. [m/5] + 1, … , [m/4] – 1
5. [m/6] + 1, … , [m/5] – 1

….

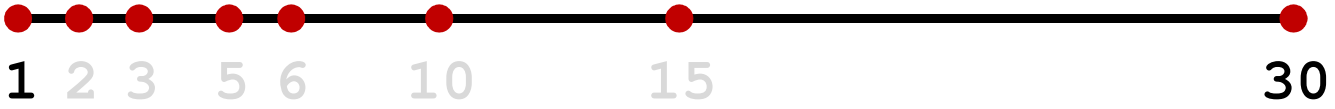


Дальнейшее усовершенствование линейного алгоритма (2)

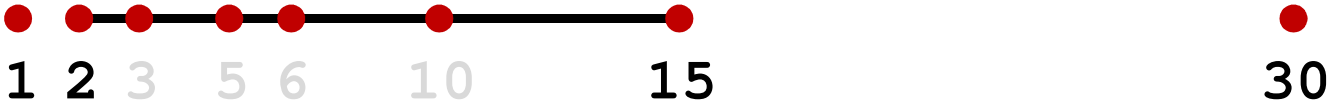
Например, число 30 = 2·3·5:



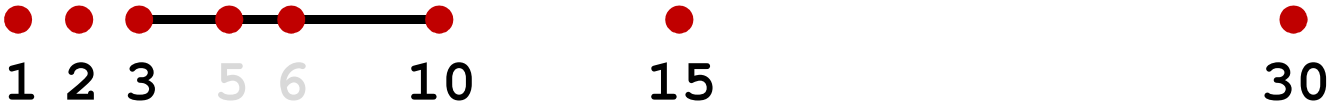
1) Инициализация



2) $k = 2$

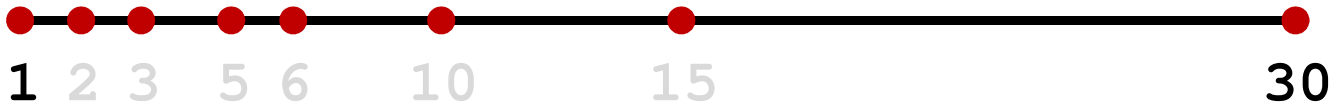


3) $k = 3$

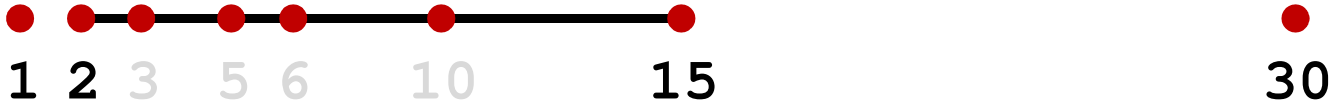


Дальнейшее усовершенствование линейного алгоритма (3)

1) Инициализация



2) $k = 2$, 30 **mod** 2 = **0**, 30 **div** 2 = **15**

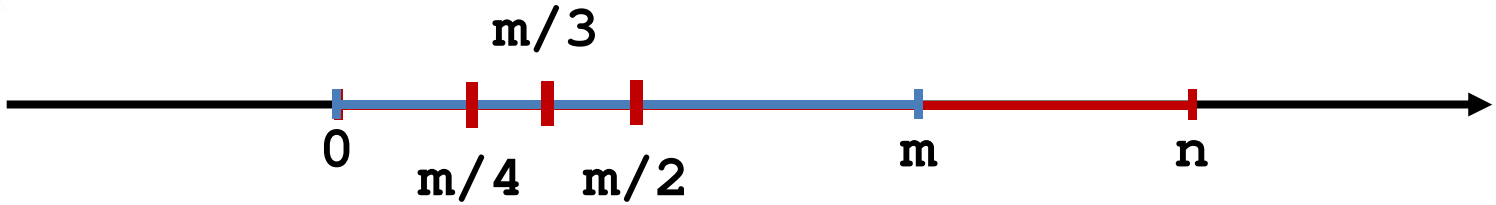


3) $k = 3$, 30 **mod** 3 = **0**, 30 **div** 3 = **10**

4) $k = 4 - 30$ **mod** $4 \neq 0$

5) $k = 5$, 30 **mod** 5 = **0**, 30 **div** 6 = **10**

Алгоритм линейного поиска НОД (v3)



Простейшие случаи:

$n = 1$ или $m = 1$ => $(n, m) = 1$

$n = m$ => $(n, m) = n$

$m<n$, n **mod** $m = 0$ => $(n, m) = m$

В противном случае:

Пусть $m < n$, (иначе поменять их местами)

1. Если (m **mod** 2) = 0 И (n **mod** (m **div** 2)) = 0, то (n, m) = m/2, иначе (n,m) $\in$ [1, [m/3]]
2. Если (m **mod** 3) = 0 И (n **mod** (m **div** 3)) = 0, то (n, m) = m/3, иначе (n,m)$\in$ [1, [m/4]]
3. Если (m **mod** 3) = 0 И (n **mod** (m **div** 3)) = 0, то (n, m) = m/3, иначе (n,m)$\in$ [1, [m/4]]

…

Алгоритм линейного поиска НОД (v3)

ввод n, m
if $m > n$ **then**
    $t \leftarrow n$, $n \leftarrow m$, $m \leftarrow t$
$i \leftarrow 1$, $e \leftarrow m$, $f \leftarrow 0$
while ($i < e$ и $f \neq 1$) **do**
    $t_1 \leftarrow m$ div i
    $t_2 \leftarrow m$ mod i
    if ($t_2 = 0$ и (n mod $t_1 = 0$) **then**
        $g \leftarrow t_1$, $f \leftarrow 0$
    else if ($t_2 = 0$ и (n mod $i = 0$)) **then**
        $g \leftarrow i$
    $i \leftarrow i + 1$, e $\leftarrow t_1$
вывод g

Алгоритм Евклида поиска НОД (на базе вычитания – v1)

Для данных n и m строится последовательность чисел:

$$r_0, r_1, r_2, \ldots, r_k$$

элементы r_i которой определяются следующим образом:

1) если $n \geq m$, то $r_0 = n - m$, $n = m$, $m = r_1$
   иначе $r_0 = m - n$, $m = r_0$
2) $r_1 = n - m$,
   если $r_1 < m$, то $n = m$, $m = r_1$
   иначе $n = r_1$
3) $r_2 = n - m$,
   если $r_2 < m$, то $n = m$, $m = r_2$
   иначе $n = r_2$

…

если $r_k = 0$, то $(n, m) = r_{k-1}$

Алгоритм Евклида (v1). Рекуррентные соотношения.

$$
\begin{cases}
i_0 = 0, \\
n_0 = \mathbf{max}(n, m) \\
m_0 = \mathbf{min}(n, m) \\
r_0 = m_0 \\
i_{k+1} = i_k + 1 \\
r_{k+1} = n_k - m_k \\
n_{k+1} = \mathbf{max}(r_{k+1}, m_k) \\
m_{k+1} = \mathbf{min}(r_{k+1}, m_k)
\end{cases}
$$

Пусть $n = 24,\ m = 18$

k	i_k	n_k	m_k	r_k
1	0	24	18	18
2	1	18	6	6
3	2	12	6	12
4	3	6	6	6
5	**4**	**6**	**0**	**0**
6	5	6	0	0

…

Алгоритм Евклида (v1)

input n, m
if $m > n$ **then**
    $t \leftarrow n$, $n \leftarrow m$, $m \leftarrow t$
while $(n - m) \neq 0$ **do**
    $t \leftarrow n - m$
    $n \leftarrow \max(t, m)$
    $m \leftarrow \max(t, m)$
output n

Алгоритм Евклида поиска НОД (на базе деления – v2)

Для данных n и m строится последовательность чисел:

$$n > m > r_1 > r_2 > \ldots > r_n,$$

элементы r_i которой определяются следующим образом:

1) $n = m \cdot d_0 + r_1$;
2) $m = r_1 \cdot d_1 + r_2$;
3) $r_1 = r_2 \cdot d_2 + r_3$;
…
$n - 1)$ $r_{n-1} = r_n \cdot d_n + 0$;

тогда $(n, m) = r_n$

1) $r_1 = n \ \% \ m$;
2) $r_2 = m \ \% \ r_1$;
3) $r_3 = r_1 \ \% \ r_2$;
…
$n–2)$ $r_n = r_{n-2} \ \% \ r_{n-1}$.

Алгоритм Евклида (v2). Рекуррентные соотношения.

$$
\begin{cases}
i_0 = 0, \\
n_0 = \mathbf{max}(n, m) \\
m_0 = \mathbf{min}(n, m) \\
r_0 = n_0 \\
i_{k+1} = i_k + 1 \\
r_{k+1} = n_k \ \% \ m_k \\
n_{k+1} = m_k \\
m_{k+1} = r_{k+1}
\end{cases}
$$

Пусть $n = 24,\ m = 18$

k	i_k	n_k	m_k	r_k
1	0	24	18	24
2	1	18	6	6
3	**2**	**6**	**0**	**0**
4	3	0	∞	∞

…

Алгоритм Евклида (v2)

input n, m
if $m > n$ **then**
    $t \leftarrow n$, $n \leftarrow m$, $m \leftarrow t$
while (n **mod** m) $\neq 0$ **do**
    $t \leftarrow n$ **mod** m
    $n \leftarrow m$
    $m \leftarrow t$
output m

Математическая основа алгоритма Евклида

Утверждение:

$$(n, m) = (m, n \textbf{ mod } m), \text{ при } n > m$$

Доказательство:

$$n = m \cdot d + q,\ d = n \textbf{ div } m,\ q = n \textbf{ mod } m$$

$$(n \textbf{ mod } x) = (m \textbf{ mod } x) \cdot d + (q \textbf{ mod } x) \textbf{ (1)}$$

Необходимость:

$x = (m, n)$, тогда n **mod** $(m, n) = 0$.

т.к. m, d и $q > 0$, то из **(1)** => m **mod** $x = 0$ и q **mod** $x = 0$

Достаточность:

$x = (m, q)$, тогда m **mod** $x = 0$ и q **mod** $x = 0$,

то из **(1)** => n **mod** $x = 0$.