

# Лекция

## Элементы теории алгоритмов

### Содержание

1. Интуитивное понятие алгоритма
2. Свойства алгоритмов
3. Формы записи алгоритмов
4. Блок-схемы
5. Запись алгоритмов на псевдокоде
6. Базовые алгоритмические структуры

### 1. Интуитивное понятие алгоритма

Работа вычислительной машины состоит в выполнении алгоритма, поэтому общие возможности применения вычислительных машин определяются тем, что можно и что нельзя представить в виде алгоритма.

Понятие алгоритма, являющееся одним из основных понятий математики, возникло задолго до появления первого компьютера. Слово «алгоритм» происходит от имени узбекского математика Аль-Хорезми, который еще в 9 веке дал правила выполнения четырех арифметических действий в десятичной системе счисления. На протяжении многих веков люди пользовались интуитивным понятием алгоритма, которое можно выразить так:

***Алгоритм** – это строгая и четкая конечная система указаний, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к достижению поставленной цели.*

В частности, система правил является алгоритмом, если ее можно вручить в качестве инструкции разным людям, не знакомым с сутью дела, и они, следуя этой системе правил, будут действовать одинаково. Например, можно предложить такой простейший алгоритм подсчета людей в зрительном зале: пройди по всем рядам и для каждого присутствующего человека прибавляй единицу к общему счетчику, в котором сначала был нуль.

Объекты этого алгоритма – люди в зале и числа. Над людьми выполняется действие «найти следующего», а над числами – действие «прибавить единицу».

Рассмотрим еще один пример простого алгоритма. Древнегреческий математик Евклид предложил алгоритм вычисления наибольшего общего делителя (НОД) двух натуральных чисел  $A$  и  $B$ . Суть этого алгоритма в том, чтобы вычитать из большего числа меньшее, заноса результат на место большего, и действовать так до тех пор, пока числа не станут равны между собой. Эти равные числа и будут наибольшим общим делителем исходных двух чисел. В алгоритме Евклида используется тот факт, что НОД  $A$  и  $B$  является также НОД их разности и любого из чисел  $A$ ,  $B$ .

Сформулируем алгоритм Евклида более четко:

1. Рассмотреть  $A$  как первое число и  $B$  как второе число. Перейти к п. 2.
2. Сравнить первое и второе числа. Если они равны, то перейти к п.5. Если нет, то перейти к п.3.
3. Если первое число меньше второго, то переставить их. Перейти к п.4.
4. Вычесть из первого числа второе и рассмотреть полученную разность как первое число. Перейти к п.2.
5. Рассмотреть первое число как результат. СТОП.

## 2. Свойства алгоритмов

1. **Дискретность** – разделение выполнения решения задачи на отдельные операции, выполняемые исполнителем по определенным командам. Исполнитель выполняет предписания алгоритма последовательно одно за другим, в соответствии с указанным порядком их записи.
2. **Определенность** (или точность) алгоритма. Суть этого свойства состоит в том, что каждая команда должна однозначно определять действия исполнителя. У каждого исполнителя имеется свой перечень команд, которые он может выполнить. Эта совокупность команд называется **системой команд исполнителя**.
3. Для того, чтобы алгоритм был выполнен, нельзя включать в него команды, которые исполнитель выполнить не в состоянии. Алгоритм должен включать только те команды, которые входят в его систему команд. Это свойство алгоритма называется **понятностью**.
4. **Результативность** (или конечность). Свойство означает, что исполнение алгоритма должно закончиться за конечное число шагов.
5. **Массовость** (или вариативность). При разработке алгоритма предпочтительно создавать такие алгоритмы, которые решали бы весь класс задач данного типа. Например, алгоритм нахождения корней кубического уравнения  $ax^3+bx^2+cx+d=0$  должен обеспечивать возможность решения при любых допустимых значениях коэффициентов  $a, b, c, d$ .

Свойство массовости не является необходимым, оно скорее определяет качество алгоритма. Если же последовательности команд не удовлетворяют первым четырем свойствам, то это не алгоритмы.

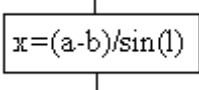
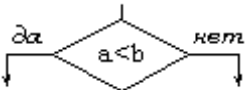
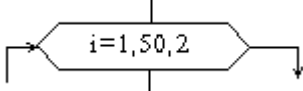
## 3. Формы записи алгоритмов

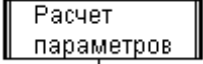
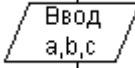


1. Словесная (Записан на естественном языке (см. Алгоритм Евклида)).
2. Графическая (Изображен в виде блок-схемы).
3. **псевдокоды** (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
4. **программная** (тексты на языках программирования).

## 4. Блок-схемы алгоритмов

**Блок-схема алгоритма** – графическое представление алгоритма. Каждый пункт алгоритма отображается на схеме некоторой геометрической фигурой (блоком) и дополняется элементами словесной записи. Правила выполнения схем алгоритмов регламентирует ГОСТ 19.002-80.

### Основные элементы блок-схем

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Вычислительное действие или последовательность действий
Решение		Проверка условий
Цикл		Начало счетного цикла

Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Ввод-вывод		Ввод-вывод в общем виде
Пуск-останов		Начало, конец алгоритма, вход и выход в подпрограмму
Документ		Вывод результатов на печать

В качестве примера построим блок-схему алгоритма Евклида:

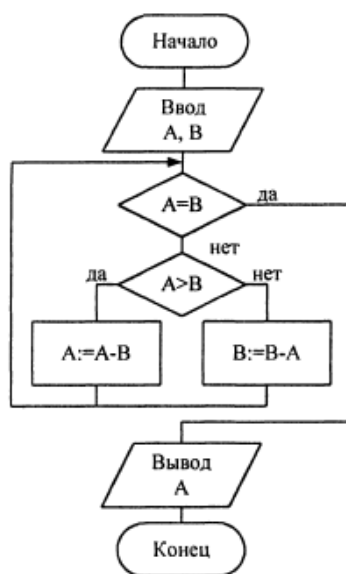


Рис. 1.4. Схема алгоритма Евклида

## 5. Запись алгоритмов на псевдокоде

В качестве примера псевдокода рассмотрим школьный алгоритмический язык.

### Основные служебные слова

<b>алг</b> (алгоритм)	<b>сим</b> (символьный)	<b>дано</b>	<b>для</b>	<b>да</b>
<b>арг</b> (аргумент)	<b>лит</b> (литерный)	<b>надо</b>	<b>от</b>	<b>нет</b>
<b>рез</b> (результат)	<b>лог</b> (логический)	<b>если</b>	<b>до</b>	<b>при</b>
<b>нач</b> (начало)	<b>таб</b> (таблица)	<b>то</b>	<b>знач</b>	<b>выбор</b>
<b>кон</b> (конец)	<b>нц</b> (начало цикла)	<b>иначе</b>	<b>и</b>	<b>ввод</b>
<b>цел</b> (целый)	<b>кц</b> (конец цикла)	<b>все</b>	<b>или</b>	<b>вывод</b>
<b>вещ</b> (вещественный)	<b>длин</b> (длина)	<b>пока</b>	<b>не</b>	<b>утв</b>

Общий вид алгоритма:

```
алг название алгоритма (аргументы и результаты)
  дано условия применимости алгоритма
  надо цель выполнения алгоритма
нач описание промежуточных величин
| последовательность команд (тело алгоритма)
кон
```

Часть алгоритма от слова **алг** до слова **нач** называется **заголовком**, а часть, заключенная между словами **нач** и **кон** — **телом** алгоритма.

В предложении **алг** после названия алгоритма в круглых скобках указываются **характеристики** (**арг**, **рез**) и **тип значения** (**цел**, **вещ**, **сим**, **лит** или **лог**) всех **входных** (аргументы) и **выходных** (результаты) переменных. При описании массивов (таблиц) используется служебное слово **таб**, дополненное **граничными парами** по каждому индексу элементов массива.

Примеры предложений **алг**:

```
алг Объем и площадь цилиндра ( арг вещ R, H, рез вещ V, S )
алг Корни КвУр ( арг вещ a, b, c, рез вещ x1, x2, рез лит t )
алг Исключить элемент ( арг цел N, арг рез вещ таб A[1:N] )
алг Диагональ ( арг цел N, арг цел таб A[1:N, 1:N], рез лит Otvet )
```

Предложения **дано** и **надо** не обязательны. В них рекомендуется записывать утверждения, описывающие состояние среды исполнителя алгоритма, например:

1. **алг** Замена (арг лит Str1, Str2, арг рез лит Text)
2. **дано** | длины подстрок Str1 и Str2 совпадают
3. **надо** | всюду в строке Text подстрока Str1 заменена на Str2
- 4.
5. **алг** Число максимумов (арг цел N, арг вещ таб A[1:N], рез цел K)
6. **дано** |  $N > 0$
7. **надо** | K — число максимальных элементов в таблице A
- 8.
9. **алг** Сопротивление (арг вещ R1, R2, арг цел N, рез вещ R)
10. **дано** |  $N > 5$ ,  $R1 > 0$ ,  $R2 > 0$
11. **надо** | R — сопротивление схемы
- 12.

Здесь в предложениях **дано** и **надо** после знака "|" записаны **комментарии**. Комментарии можно помещать в конце любой строки. Они не обрабатываются транслятором, но существенно облегчают понимание алгоритма.

## Команды школьного АЯ

**Команда присваивания.** Служит для вычисления выражений и присваивания их значений переменным. Общий вид: **A := B**, где знак "=" означает команду **заменить прежнее значение переменной, стоящей в левой части, на вычисленное значение выражения, стоящего в правой части**.

Например, `a := (b+c) * sin(Pi/4); i := i+1.`

### Команды ввода и вывода.

- **ввод** имени переменных

- **вывод** имена переменных, выражения, тексты.

**Команды если и выбор.** Применяют для организации ветвлений.

**Команды для и пока.** Применяют для организации циклов.

### Пример записи алгоритма на школьном АЯ

```

алг Сумма квадратов (арг цел n, рез цел S)
  дано | n > 0
  надо | S = 1*1 + 2*2 + 3*3 + ... + n*n
нач цел i
  ввод n; S:=0
  нц для i от 1 до n
    S:=S+i*i
  кц
  вывод "S = ", S
кон

```

## 6. Базовые алгоритмические структуры

Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных **базовых** (т.е. основных) **элементов**. Естественно, что при таком подходе к алгоритмам изучение основных принципов их конструирования должно начинаться с изучения этих базовых элементов. Для их описания будем использовать язык схем алгоритмов и школьный алгоритмический язык.

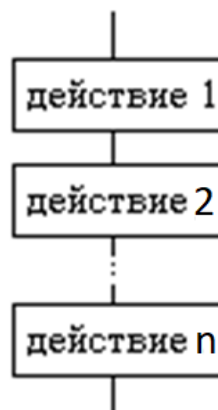
**Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: следование, ветвление, цикл.**

Характерной особенностью базовых структур является наличие в них **одного входа и одного выхода**.

**1. Базовая структура "следование".** Образуется последовательностью действий, следующих одно за другим:

Школьный алгоритмический язык      Язык блок-схем

действие 1  
действие 2  
.....  
действие n



**2. Базовая структура "ветвление".** Обеспечивает в зависимости от результата проверки условия (**да** или **нет**) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к **общему выходу**, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран. Структура **ветвление** существует в четырех основных вариантах:

- если—то;
- если—то—иначе;
- выбор;
- выбор—иначе.

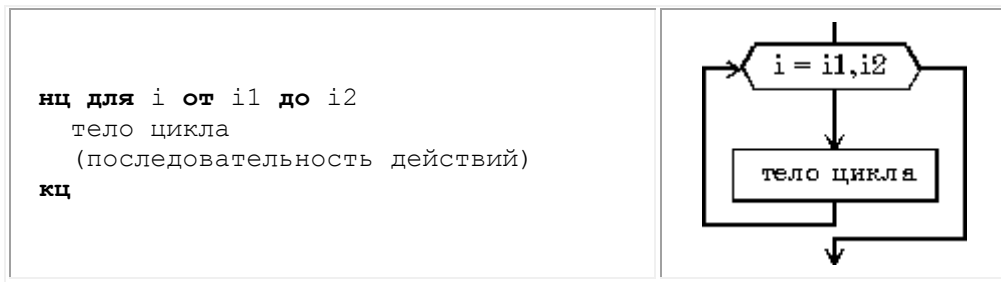
Школьный алгоритмический язык	Язык блок-схем
<b>1. если—то</b>	
<pre> если условие   то действия все           </pre>	
<b>2. если—то—иначе</b>	
<pre> если условие   то действия 1   иначе действия 2 все           </pre>	
<b>3. выбор</b>	
<pre> выбор   при условии 1: действия 1   при условии 2: действия 2   . . . . .   при условии N: действия N все           </pre>	
<b>4. выбор—иначе</b>	
<pre> выбор   при условии 1: действия 1   при условии 2: действия 2   . . . . .   при условии N: действия N   иначе действия N+1 все           </pre>	

## Примеры структуры **ветвление**

Школьный алгоритмический язык	Язык блок-схем
<pre> <b>если</b> x &gt; 0   <b>то</b> y := sin(x) <b>все</b>           </pre>	
<pre> <b>если</b> a &gt; b   <b>то</b> a := 2*a; b := 1   <b>иначе</b> b := 2*b <b>все</b>           </pre>	
<pre> <b>выбор</b>   <b>при</b> n = 1: y := sin(x)   <b>при</b> n = 2: y := cos(x)   <b>при</b> n = 3: y := 0 <b>все</b>           </pre>	

**3. Базовая структура "цикл".** Обеспечивает многократное выполнение некоторой совокупности действий, которая называется **телом цикла**. Основные разновидности циклов представлены в таблице:

Школьный алгоритмический язык	Язык блок-схем
<p style="text-align: center;"><b>Итерационный цикл</b> или цикл типа «<b>пока</b>».</p> <p style="text-align: center;">Предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова пока.</p>	
<pre> <b>нц пока</b> условие   тело цикла   (последовательность действий) <b>кц</b>           </pre>	
<p style="text-align: center;"><b>Счетный цикл</b> или цикл типа «<b>для</b>».</p> <p style="text-align: center;">Предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.</p>	



### Примеры структуры **цикл**

Школьный алгоритмический язык	Язык блок-схем
<pre> <b>нц</b> пока i &lt;= 5   S := S+A[i]   i := i+1 <b>кц</b> </pre>	<pre> graph TD     Start(( )) --&gt; Cond{i &lt;= 5}     Cond -- да --&gt; Body[S:=S+A[i]; i:=i+1]     Body --&gt; Cond     Cond -- нет --&gt; Exit(( ))   </pre>
<pre> <b>нц</b> для i от 1 до 5   X[i] := i*i*i   Y[i] := X[i]/2 <b>кц</b> </pre>	<pre> graph TD     Start(( )) --&gt; Init[i = 1, 5]     Init --&gt; Body[X[i]:=i*i*i; Y[i]:=X[i]/2]     Body --&gt; Init     Init --&gt; Exit(( ))   </pre>

### Какие циклы называют итерационными?

Особенностью итерационного цикла является то, что число повторений операторов тела цикла заранее неизвестно. Для его организации используется цикл типа **пока**. Выход из итерационного цикла осуществляется в случае выполнения заданного условия.

На каждом шаге вычислений происходит **последовательное приближение к искомому результату и проверка условия достижения последнего.**

*Пример. Составить алгоритм вычисления бесконечной суммы*

$$S = x - \frac{x^2}{2} + \frac{x^2}{3} - \frac{x^4}{4} + \dots (-1)^{i-1} \frac{x^i}{i} + \dots$$

*с заданной точностью  $\varepsilon$  (для данной знакопеременной бесконечной суммы требуемая точность будет достигнута, когда очередное слагаемое станет по абсолютной величине меньше  $\varepsilon$ ).*

Вычисление сумм — типичная циклическая задача. Особенностью же нашей конкретной задачи является то, что число слагаемых (а, следовательно, и число повторений тела цикла)



заранее неизвестно. Поэтому выполнение цикла должно завершиться в момент достижения требуемой точности.

При составлении алгоритма нужно учесть, что знаки слагаемых чередуются и степень числа  $x$  в числителях слагаемых возрастает.

Решая эту задачу "в лоб" путем вычисления на каждом  $i$ -ом шаге частичной суммы

$$S := S + ((-1)^{(i-1)}) * (x^{**i}) / i ,$$

мы получим очень неэффективный алгоритм, требующий выполнения большого числа операций. Гораздо лучше организовать вычисления следующим образом: если обозначить числитель какого-либо слагаемого буквой  $p$ , то у следующего слагаемого числитель будет равен  $-p*x$  (знак минус обеспечивает чередование знаков слагаемых), а само слагаемое  $m$  будет равно  $p/i$ , где  $i$  — номер слагаемого.

Сравните эти два подхода по числу операций.

Алгоритм на школьном АЯ	Блок-схема алгоритма
<pre> алг Сумма (арг вещ x, Eps, рез вещ S)   дано   0 &lt; x &lt; 1   надо   S = x - x**2/2 + x**3/3 - ... нач цел i, вещ m, p   ввод x, Eps   S := 0; i := 1   начальные значения   m := 1; p := -1   нц пока abs(m) &gt; Eps     p := -p*x   p - числитель                     очередного слагаемого     m := p/i   m - очередное слагаемое     S := S + m   S - частичная сумма     i := i + 1   i - номер                     очередного слагаемого   кц   вывод S кон </pre>	<pre> graph TD     Start([начало]) --&gt; Input[/x, ε/]     Input --&gt; Init[S:=0; i:=1; m:=1; p:=-1]     Init --&gt; Decision{  m  &gt; ε }     Decision -- нет --&gt; Output[/S/]     Output --&gt; End([конец])     Decision -- да --&gt; Loop[p:=-p*x; m:=p/i; S:=S+m; i:=i+1]     Loop --&gt; Decision </pre>

Алгоритм, в состав которого входит итерационный цикл, называется **итерационным алгоритмом**. Итерационные алгоритмы используются при реализации итерационных численных методов.

В итерационных алгоритмах необходимо обеспечить обязательное достижение условия выхода из цикла (**сходимость итерационного процесса**). В противном случае произойдет "**зацикливание**" алгоритма, т.е. не будет выполняться основное свойство алгоритма — **результативность**.

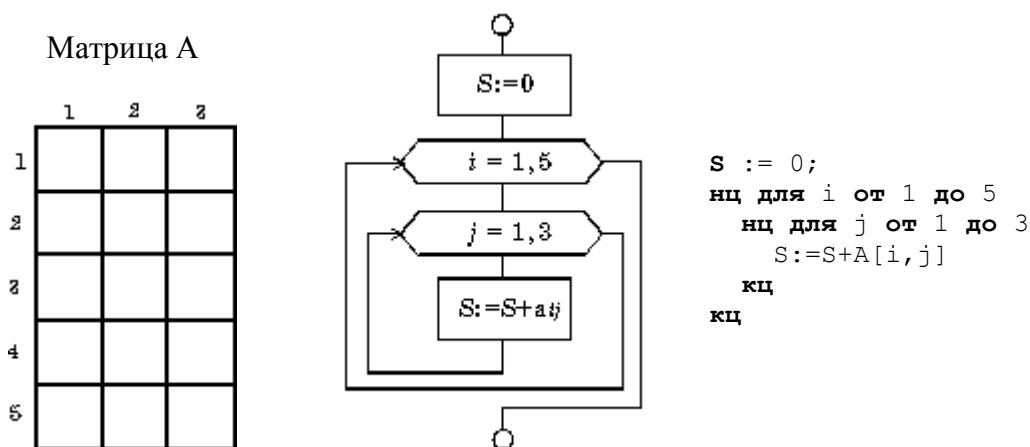
## Что такое вложенные циклы?

Возможны случаи, когда внутри тела цикла необходимо повторять некоторую последовательность операторов, т. е. организовать внутренний цикл. Такая структура получила название **цикла в цикле** или **вложенных циклов**. Глубина вложения циклов (то есть количество вложенных друг в друга циклов) может быть различной.

При использовании такой структуры для экономии машинного времени необходимо выносить из внутреннего цикла во внешний все операторы, которые не зависят от параметра внутреннего цикла.

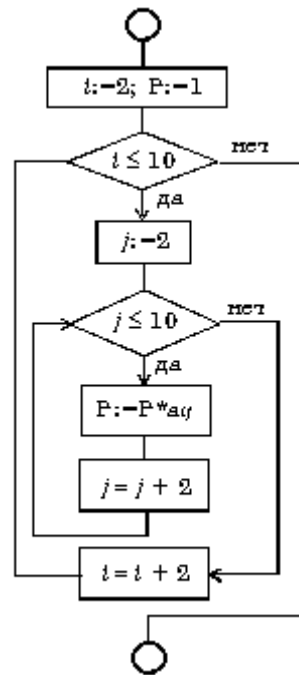
### Пример вложенных циклов «для»

Вычислить сумму элементов заданной матрицы  $A(5,3)$ .



### Пример вложенных циклов «пока»

Вычислить произведение тех элементов заданной матрицы  $A(10,10)$ , которые расположены на пересечении четных строк и четных столбцов.



```

i:=2; P:=1
нц пока i <= 10
  j:=2
  нц пока j <= 10
    P:=P*A[i,j]
    j:=j+2
  кц
  i:=i+2
кц

```