Понятие алгоритма

Алгоритм и его свойства

В математике для решения типовых задач мы используем определенные правила. Обычно любые инструкции и правила представляют собой последовательность действий, которую необходимо выполнить в определенном порядке. Для решения задачи надо знать:

— что дано,

— что следует получить и

— какие действия, и

— в каком порядке следует для этого выполнить.

Алгоритм - это точное предписание исполнителю совершить указанную последовательность действий для получения решения задачи за конечное число шагов.

Приведенное определение не является определением в математическом смысле слова, а, скорее, ***описание интуитивного понятия алгоритма***, раскрывающее его сущность. Но для общего понимания сущности алгоритма такого толкования оказывается, как правило, достаточно.

Название "алгоритм" произошло от латинской формы имени величайшего среднеазиатского математика Мухаммеда ибн Муса ал-Хорезми (Alhorithmi), жившего в 783 -850 гг**.** В своей книге "Об индийском счете" *он изложил правила записи натуральных чисел с помощью арабских цифр и правила действий над ними "столбиком",* знакомые теперь каждом школьнику. *В XII веке эта книга была переведена на латынь и получила широкое распространение в Европе.*

Понятие алгоритма является не только одним из главных понятий математики, но одним из главных понятий современной пауки. Более того, с

наступлением эры информатики алгоритмы становятся одним из важнейших факторов цивилизации.

Свойства алгоритмов

К основным свойствам алгоритмов относятся следующие свойства:

1. Понятность для исполнителя, — исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.

2. Дискретность (прерывность, раздельность) - алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов (этапов).

3. Определенность — каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола.

Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

4. Релевантность (или конечность) состоит в том, что за конечное число шагов алгоритм либо должен приводит к решению задачи, либо после конечного числа шагов останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения, либо неограниченно продолжаться в течение времени, отведенного для исполнения алгоритма, с выдачей промежуточных результатов.

5. Массовость означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

Правила построения алгоритма

Чтобы алгоритм выполнил свое предназначение, его необходимо строить по определенным правилам. В этом смысле нужно говорить не о свойствах алгоритма, а о правилах построения алгоритма, или о требованиях, предъявляемых к алгоритму.

Первое правило — при построении алгоритма, прежде всего, необходимо задать множество объектов, с которыми будет работать алгоритм. Формализованное (закодированное) представление этих объектов носит название данных. ***Алгоритм приступает к работе с некоторым набором данных, которые называются входными, и в результате своей работы выдает данные, которые называются входными.*** Таким образом, алгоритм преобразует входные данные в выходные.

Это правило позволяет сразу отделить алгоритмы от "методов" и “способов”. Пока мы не имеем формализованных входных данных, мы не можем построить алгоритм.

Второе правило — для работы алгоритма требуется память. В памяти размещаются входные данные, с которыми алгоритм начинает работать, промежуточные данные и выходные данные, которые являются результатом работы алгоритма. Память является дискретной, т.е. состоящей из отдельных ячеек. Поименованная ячейка памяти носит название переменной. В теории алгоритмов размеры памяти не ограничиваются, т. е. считается, что мы можем предоставить алгоритму любой необходимый для работы объем памяти.

Третье правило — дискретность. Алгоритм строится из отдельных шагов (действий, операций, команд). Множество шагов, из которых составлен алгоритм, конечно.

Четвертое правило — детерменированность. После каждого шага необходимо указывать, какой шаг выполняется следующим, либо давать команду остановки.

Пятое правило — сходимость (результативность). Алгоритм должен завершать работу после конечного числа шагов. При этом необходимо указать, что считать результатом работы алгоритма.

Итак, алгоритм — неопределяемое понятие теории алгоритмов. Алгоритм каждому определенному набору входных данных ставит в соответствие некоторый набор выходных данных, т. е. вычисляет (реализует) функцию. При рассмотрении конкретных вопросов в теории алгоритмов всегда имеется в виду какая-то конкретная модель алгоритма.

Формы записи алгоритма

Алгоритм, как последовательность шагов или инструкций, может быть представлен в различных формах.

На практике наиболее распространены следующие ***формы представления алгоритмов:***

1. словесная *(запись на естественном языке);*
2. графическая *(изображения из графических символов);*
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).*

Словесная форма записи алгоритмов

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Например. Записать алгоритм нахождения наибольшего общего делителя (ИОД) двух натуральных чисел (алгоритм Эвклида).

Алгоритм может быть следующим:

1) задать два числа;

2) если числа равны, то взять любое т них в качестве ответа и

остановиться, в противном случае продолжить выполнение

алгоритма;

3. определить большее из чисел;
4. заменишь большее из чисел разностью большего и меньшего из чисел;
5. повторить алгоритм с шага 2.

Описанный алгоритм, применим к любым натуральным числам и должен приводить к решению поставленной задачи.

Самостоятельно: определить с помощью этого алгоритма наибольший общий делитель чисел 125 и 75.

Словесный способ не получил широкого распространения из-за следующих ***недостатков***:

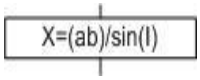
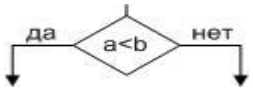
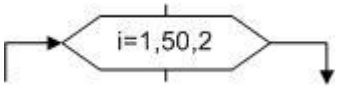
1. Строго не формализуем;
2. Страдает многословностью записей;
3. Допускает неоднозначность толкования отдельных предписаний.

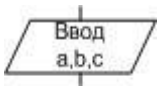
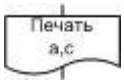
Графическая форма записи алгоритмов

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным. ***При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.***

Такое графическое представление называется схемой алгоритма или блок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий. В таблице 8.1 приведены наиболее часто употребляемые блочные символы.

Таблица 7.1

Название символа	Обозначения и пример заполнения	Пояснения
Процесс		Вычислительное действие или последовательность действий
Решение		Проверка условий
Модификация		Начало цикла

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

Пояснения к таблице 7.1.

1. Блок ***"процесс"*** применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.
2. Блок ***"решение"*** используется для обозначения переходов управления по условию. В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет.
3. Блок ***"модификация"*** используется для организации циклических конструкций. Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.
4. Блок ***"предопределенный процесс"*** используется для указания обращений к вспомогательным алгоритмам, существующим автономно

в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

Типы базовых алгоритмических структур

В общем случае блок-схема алгоритма имеет сложную структуру и, следовательно, может быть выражена композицией элементарных блок-схем, принято называть базовыми.

Логическая структура любого алгоритма может быть представлена комбинацией трех базовых

алгоритмических структур:

1. алгоритмы линейной структуры, которые иногда называют следованием (последовательностью)

2. алгоритмы разветвляющейся структуры, называемые ветвлением

3. алгоритмы циклической структуры, называемые циклами.

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

Линейная базовая структура (“последовательность”)

Линейная базовая структура – это алгоритм, в котором блоки выполняются последовательно друг за другом, в порядке, заданном схемой. Такой порядок выполнения называется естественным. Образуется последовательностью действий, следующих одно за другим:

Таблица 7.2.

Процесс	**Блок-схема**
действие 1 действие 2	

…….. действие n	

Пример:Вычислить высоты треугольника со сторонами a, b , c, используя формулы:

$$h_a = \frac{2}{a}\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)};\quad h_b = \frac{2}{b}\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)};$$

$$h_c = \frac{2}{c}\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)};\qquad \text{где } p=(a+b+c)/2.$$

Для решения любой нетривиальной задачи существует несколько алгоритмов, приводящих к получению результата. Из возможных алгоритмов следует выбирать наилучший по некоторому критерию. Чаще всего в качестве критерия выбирается либо оценка точности решения задачи, либо затраты времени на ее решение, либо некоторый интегральный критерий, включающий оценки точности и затраты времени.

При решении данной задачи для исключения повторений следует вычислять высоты не по приведенным выше формулам непосредственно, а используя промежуточную переменную

$$t = 2\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)},$$

тогда $h_a = t/a$, $h_b = t/b$, $h_c = t/c$.

При этом схема алгоритма решения имеет вид, представленный на рис. 8.1.

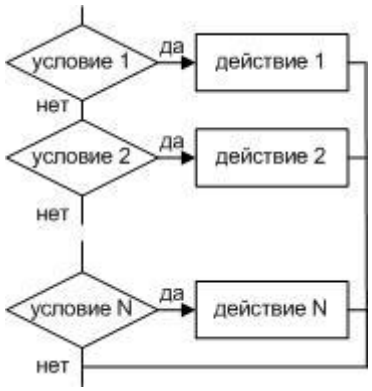
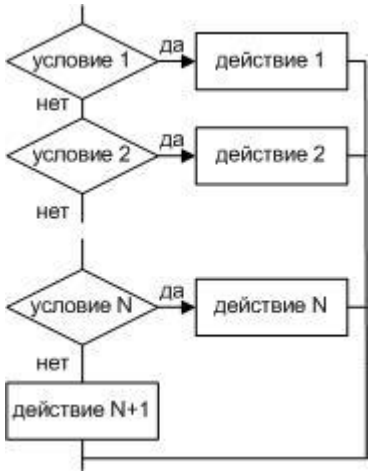
Базовая структура “ветвление”

*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:***

1. если ─ то;
2. если ─ то ─ иначе;
3. выбор;
4. выбор ─ иначе.

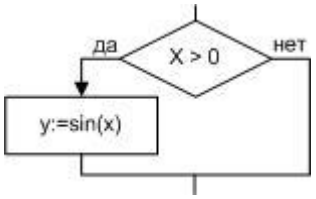
Таблица 7.3

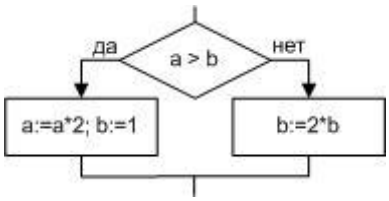
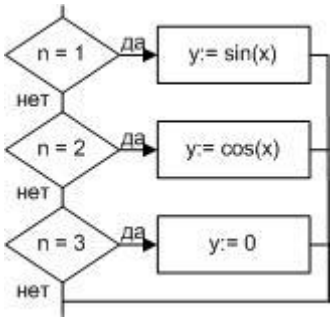
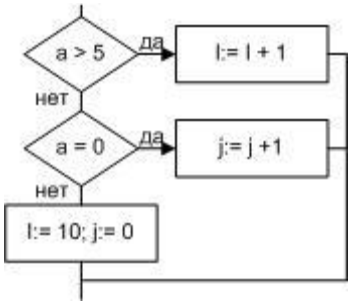
Выполняемые действия	**Блок-схема**
1. **если─то**	
если условие **то** действие **все**	
2. **если─то─иначе**	
если условие **то** действие **иначе** действие 2 **все**	
3. **выбор**	

выбор **при** условие 1: действие 1 **при** условие 2: действие 2 …….. **при** условие N: действие N **все**	
4. **выбор─иначе**	
выбор **при** условие 1: действие 1 **при** условие 2: действие 2 …….. **при** условие N: действие N **иначе** действия N + 1 **все**	

Примеры структуры «ветвление»

Таблица 7.4

Выполняемые действия	**Блок-схема**
если x > 0 **то** y:= sin(x) **все**	

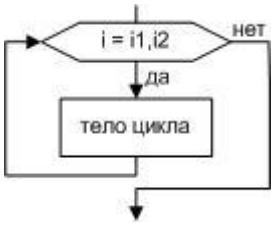
если a>b **то** a:= 2*a; b:= 1 **иначе** b:= 2*b **все**	
выбор **при** n = 1: y:= sin(x) **при** n = 2: y:= cos(x) **при** n = 3: y:= 0 **все**	
выбор **при** a > 5; I:= I + 1 **при** a = 0; j:= j + 1 **иначе** I:= 10; j:= 0 **все**	

Базовая структура “цикл”

Обеспечивает многократное выполнение некоторой совокупности действий, которая называется телом цикла. Основные разновидности циклов представлены в таблице 8.5.

Таблица 7.5

Выполняемые действия	**Блок-схема**
Цикл типа **пока**.	

<table>
<tr><td colspan="2">Предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова пока.</td></tr>
<tr><td>пока условие тело
тело цикла
(последовательность действий)</td><td> 
</td></tr>
<tr><td colspan="2">Цикл типа для.
Предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.</td></tr>
<tr><td>для i от i1 до i2
тело цикла
(последовательность действий)</td><td> 
</td></tr>
</table>