Занятие 2. Основные понятия об алгоритмах

СОДЕРЖАНИЕ

1. ПОНЯТИЕ АЛГОРИТМА

Понятие алгоритма является одним из основных понятий современной математики. Еще на самых ранних ступенях развития математики (Древний Египет, Вавилон, Греция) в ней стали возникать различные вычислительные процессы чисто механического характера. С их помощью искомые величины ряда задач вычислялись последовательно из исходных величин по определенным правилам и инструкциям. Со временем все такие процессы в математике получили название алгоритмов (алгорифмов).

Термин *алгоритм* происходит от имени средневекового узбекского математика Аль-Хорезми, который еще в IX в. (825) дал правила выполнения четырех арифметических действий в десятичной системе счисления. Процесс выполнения арифметических действий был назван *алгоритмом.*

С 1747 г. вместо слова *алгоризм* стали употреблять *алгорисмус,* смысл которого состоял в комбинировании четырех операций арифметического исчисления — сложения, вычитания, умножения, деления.

К 1950 г. *алгорисмус* стал *алгорифмом.* Смысл алгорифма чаще всего связывался с алгорифмами Евклида — процессами нахождения наибольшего общего делителя двух натуральных чисел, наибольшей общей меры двух отрезков и т.п.

Под алгоритмом понимали конечную последовательность точно сформулированных правил, которые позволяют решать те или иные классы задач. Такое определение алгоритма не является строго математическим, так как в нем не содержится точной характеристики того, что следует понимать под классом задач и под правилами их решения.

2. ХАРАКТЕРИСТИКИ ИСПОЛНИТЕЛЯ

Всякий алгоритм составляется в расчете на конкретного исполнителя с учетом его возможностей. Для того чтобы алгоритм мог быть выполнен, нельзя включать в него команды, которые исполнитель не в состоянии

выполнить. Нельзя повару поручать работу токаря, какая бы подробная инструкция ему не давалась. У каждого исполнителя имеется свой перечень команд, которые он может исполнить. Совокупность команд, которые могут быть выполнены исполнителем, называется *системой команд исполнителя.* Каждая команда алгоритма должна определять однозначно действие исполнителя. Такое свойство алгоритмов называется *определенностью (или точностью) алгоритма.*

Алгоритм, составленный для конкретного исполнителя, должен включать только те команды, которые входят в его систему команд. Это свойство алгоритма называется *понятностью.* Алгоритм не должен быть рассчитан на принятие каких-либо самостоятельных решений исполнителем, не предусмотренных составленным алгоритмом.

3. ТРЕБОВАНИЯ К АЛГОРИТМАМ

Алгоритмом, таким образом, называется система четких однозначных указаний исполнителю, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к получению требуемого результата.

Свойства алгоритмов. Каждое указание алгоритма предписывает исполнителю выполнить одно конкретное законченное действие. Исполнитель не может перейти к выполнению следующей операции, не закончив полностью выполнения предыдущей. Предписания алгоритма надо выполнять последовательно одно за другим, в соответствии с указанным порядком их записи. Выполнение всех предписаний гарантирует правильное решение задачи.

Поочередное выполнение команд алгоритма за конечное число шагов приводит к решению задачи, к достижению цели. Разделение выполнения решения задачи на отдельные операции (выполняемые исполнителем по

определенным командам) — важное свойство алгоритмов, называемое *дискретностью.*

Анализ примеров различных алгоритмов показывает, что запись алгоритма распадается на отдельные указания исполнителю выполнить некоторое законченное действие. Каждое такое указание называется *командой.* Команды алгоритма выполняются одна за другой. После каждого шага исполнения алгоритма точно известно, какая команда должна выполняться следующей. Алгоритм представляет собой последовательность команд (также инструкций, директив), определяющих действия исполнителя (субъекта или управляемого объекта).

Таким образом, выполняя алгоритм, исполнитель может не вникать в смысл того, что он делает, и вместе с тем получать нужный результат. В этом случае говорят, что исполнитель действует формально, т.е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции.

Это очень важная особенность алгоритмов. Наличие алгоритма формализовало процесс, исключило рассуждения. Если обратиться к другим примерам алгоритмов, то можно увидеть, что и они позволяют исполнителю действовать формально. Таким образом, создание алгоритма дает возможность решать задачу формально, механически исполняя команды алгоритма в указанной последовательности.

Построение алгоритма для решения задачи из какой-либо области требует от человека глубоких знаний в этой области, бывает связано с тщательным анализом поставленной задачи, сложными, иногда очень громоздкими рассуждениями. На поиски алгоритма решения некоторых задач ученые затрачивают многие годы. Но когда алгоритм создан, решение задачи по готовому алгоритму уже не требует каких-либо рассуждений и сводится только к строгому выполнению команд алгоритма.

Алгоритм, составленный для конкретного исполнителя, должен включать только те команды, которые входят в его систему команд. Это

свойство алгоритма называется *понятностью.* Алгоритм не должен быть рассчитан на принятие каких-либо самостоятельных решений исполнителем, не предусмотренных составленным алгоритмом.

Еще одно важное требование, предъявляемое к алгоритмам, — *результативность (или конечность)* алгоритма. Оно означает, что исполнение алгоритма должно закончиться за конечное число шагов.

Разработка алгоритмов — процесс творческий, требующий умственных усилий и затрат времени. Поэтому предпочтительно разрабатывать алгоритмы, обеспечивающие решения всего класса задач данного типа. Например, если составляется алгоритм решения кубического уравнения $ax^3 + bx^2 + cx + d = 0$, то он должен быть *вариативен,* т.е. обеспечивать возможность решения для любых допустимых исходных значений коэффициентов *a, Ь, c, d.* Про такой алгоритм говорят, что он удовлетворяет требованию *массовости.* Свойство массовости не является необходимым свойством алгоритма. Оно скорее определяет качество алгоритма; в то же время свойства дискретности, точности, понятности и конечности являются необходимыми (иначе это не алгоритм).

4. СПОСОБЫ ЗАПИСИ АЛГОРИТМОВ

Алгоритмы можно записывать по-разному. Форма записи, состав и количество операций алгоритма зависят от того, кто будет исполнителем этого алгоритма. Если задача решается с помощью ЭВМ, алгоритм решения задачи должен быть записан в понятной для машины форме, т. е. в виде программы. Всякий алгоритм может быть:

- записан на естественном языке (словесное представление алгоритмов);
- изображен в виде блок-схемы;
- записан на языке программирования.

5. СЛОВЕСНОЕ ПРЕДСТАВЛЕНИЕ АЛГОРИТМОВ

Первоначально для записи алгоритмов пользовались средствами обычного языка (словесное представление алгоритмов).

Уточним понятие словесного представления алгоритма на примере нахождения факториала числа *n* — произведения *n* натуральных чисел от 1 до *n*.

Пусть c =1*2*3*4*...*n.

Процесс вычисления факториала может быть записан в виде следующей системы последовательных указаний (пунктов):

1. Полагаем c равным единице и переходим к следующему пункту.
2. Полагаем i равным единице и переходим к следующему пункту.
3. Полагаем c = i*c и переходим к следующему пункту.
4. Проверяем, равно ли i числу n. Если i = n, то вычисления прекращаем.

   Если i < n, то увеличиваем i на единицу и переходим к пункту 3.

Рассмотрим еще один пример алгоритма — нахождение наименьшего числа M в последовательности из n чисел a_1, a_2, ..., a_n (n> 0). Прежде чем записать словесный алгоритм данного примера, детально рассмотрим сам процесс поиска наименьшего числа. Будем считать, что процесс поиска осуществляется следующим образом.

Первоначально в качестве числа M принимается a_1 т. е. полагаем M = a_1, после чего M сравниваем с последующими числами последовательности, начиная с a_2.

Если M <= a_2, то M сравнивается с a_3, если M <= a_3, то M сравнивается с a_4, и так до тех пор, пока найдется число a_i < M.

Тогда полагаем M = a_i, и продолжаем сравнение с M последующих чисел из последовательности, начиная с a_{n+1} и так до тех пор, пока не будут просмотрены все n чисел.

В результате просмотра всех чисел M будет иметь значение, равное наименьшему числу из последовательности (*i* - текущий номер числа). Этот

процесс может быть записан в виде следующей системы последовательных указаний:

1. Полагаем i = 1 и переходим к следующему пункту.
2. Полагаем M = ai, и переходим к следующему пункту.
3. Сравниваем i с n; если i < n, переходим к 4 пункту, если i = n, процесс поиска останавливаем.
4. Увеличиваем i на 1 и переходим к следующему пункту.
5. Сравниваем ai с M. Если M < ai, то переходим к пункту 3, иначе (M > ai) переходим к пункту 2.

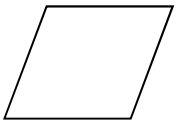
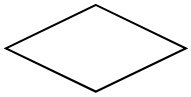
В первом алгоритме в качестве элементарных операций используются простейшие арифметические операции умножения, которые могли бы быть разложены на еще более элементарные операции. Мы такого разбиения не делаем в силу простоты и привычности арифметических правил.

6. ГРАФИЧЕСКИЙ СПОСОБ ОПИСАНИЯ АЛГОРИТМОВ

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 (единая система программной документации).

Блоки на схемах соединяются линиями потоков информации. Основное направление потока информации идет сверху вниз и слева направо (стрелки могут не указываться), снизу вверх и справа налево — стрелка обязательна. Количество входящих линий для блока не ограничено. Выходящая линия должна быть одна (исключение составляет логический блок).

	Начало и конец схемы алгоритма (вход в программу или выход из программы)
	Описание данных (переменных), описание процесса, вычисления

	Ввод данных или вывод данных
	Ветвление: выбор направления выполнения алгоритма в зависимости от результата выполнения некоторого условия
	Используется совместно с блоком Ветвление. Позволяет строить циклические алгоритмы «Пока», «До», «Для»
	Вызов вспомогательного процесса (процедуры, подпрограммы, функции)

7. ЗАПИСЬ АЛГОРИТМА НА ПСЕВДОКОДЕ

Один из методов представления алгоритмов является представление в псевдокодах. Данное представление – это частичный возврат к сценарию. Также как и в представлении в виде сценария запись алгоритма в псевдокодах разбивается на предложения, при этом каждое предложение описывает некоторый шаг алгоритма.

Для записи предложений используются:

- русский язык,
- формальные языки предметных областей, в которых решается исходная задача;
- ключевые слова псевдокодов.

Для реализации псевдокодов, в них резервируются следующие ключевые слова:

АЛГОРИТМ,

НАЧАЛО_алгоритма,

КОНЕЦ_алгоритма,

ПОДАЛГОРИТМ,

НАЧАЛО_вспомогательного алгоритма,

КОНЕЦ_вспомогательного алгоритма,

НАЧАЛО_описания переменных,

КОНЕЦ_описания переменных,

НАЧАЛО_если <условие>,

ТО,

ИНАЧЕ,

КОНЕЦ_если,

НАЧАЛО_цикла с предусловием <условие входа в цикл>,

КОНЕЦ_цикла с предусловием,

НАЧАЛО_цикла с постусловием,

КОНЕЦ_цикла с постусловием <условие выхода из цикла>,

НАЧАЛО_цикла с параметром <параметр, его диапазон и шаг>,

КОНЕЦ_цикла с параметром <параметр цикла>.

Основное внимание при представлении алгоритма в псевдокодах уделяется структуре алгоритма. Особенность псевдокодов заключается в том, что каждое предложение начинается со звездочки или нескольких звездочек. В псевдокодах звездочка используется как индикатор начала строки. В псевдокодах вместо звездочек, в принципе, можно использовать любые другие символы (например: пробел, – как это делается в школьном алгоритмическом языке, но это менее наглядно). Число звездочек определяет уровень вложенности (о вложенности структур алгоритма читай далее) данного предложения в алгоритме, то есть одна звездочка – первый уровень вложенности, две звездочки – второй уровень вложенности и т.д. Исключением является только нулевой уровень, в котором звездочки отсутствуют. Звездочки используются по той простой причине, что данный символ, используемый в языках программирования как символ умножения, никогда не стоит в начале предложения, но в данном случае позволяют структурировать алгоритм, то есть подчеркнуть, выделить структуры алгоритма, уровень их вложенности.

Звездочки в псевдокодах позволяют решить (попутно) еще одну задачу. При кодировании алгоритма в конкретном языке программирования

звездочки кодируются пробелами, тем самым структура алгоритма будет определять структуру блока операторов.

Правила звездочек:

1)число звездочек в первом и последнем предложениях должно быть одинаково;

2)количество звездочек от одного предложения к другому не изменяется, если только в них не встречаются ключевые слова НАЧАЛО…, КОНЕЦ….

3)число звездочек в предложениях, следующих после предложения, словом НАЧАЛО … увеличивается на одну;

4)число звездочек в предложениях, имеющих слово КОНЕЦ…, уменьшается на одну по сравнению с предыдущим.

Пример записи алгоритма на псевдокоде:

```
НАЧ
ЦЕЛ N, min;
МАССИВ arr[N];
ЦЕЛ i;
min = arr[0];
НЦ (i=0;i < N;i++)
 ЕСЛИ (arr[i]<min)
 ТО min=arr[i];
КЦ
ВЫВОД min;
КОН
```