**Интуитивное понятие алгоритма и его свойств.**

Алгоритм отностится к основным понятиям математики, а поэтому не имеет определения. Часто это понятие формулируют так:"точное предписание о порядке выполнения действий, из заданного фиксированного множества, для решения всех задач, заданного класса".

Рассмотрим подробнее ключевые слова в этой формулировке:

"точное предписание” означает, что предписание однозначно и одинаково понимается всеми исполнителями алгоритма и при одних и тех же исходных данных любой исполнитель всегда получает один и тот же результат;

“из заданного фиксированного множества” означает, что множество действий, используемых в предписании, оговорено заранее и не может меняться в ходе исполнения алгоритма.

“решения всех задач, заданного класса” означает, что это предписание предназначено для решения класса задач, а не одной отдельной задачи. Позднее мы подробнее рассмотрим смысл выражения “класс задач”.

Эта формулировка требует знания таких понятий, как исходные данные, результат, действие, исполнитель, класс задач. Познакомимся с ними на примере алгоритма Евклида нахождения наибольшего общего делителя (НОД) двух натуральных чисел:

Исходные данные: n, m - натуральные числа

Результат: НОД (n, m) - натуральное число d, такое, что



Алгоритм:

1. Положи a =n, b = m ; (следующий шаг)

2. Сравни a и b ; (следующий шаг)

3. Если a=b, то a - результат; (стоп)

 иначе; (следующий шаг)

4. Если a<b, то поменяй их местами; (следующий шаг)

5. Вычти b из a ; (следующий шаг)

6. Положи a = разность; (перейти к шагу 2)

В этом алгоритме действия обозначены словами: положи, сравни, если\_то\_иначе, вычти, поменяй\_местами, следующий шаг. Все исполнители, реализующие этот алгоритм, должны одинаково понимать смысл этих действий.

Например:

Сравни a и b ;(следующий шаг)

Все исполнители должны “понимать”, что надо сравнить значения, представленные символами а и b, а не сами символы, и перейти к действию 3.

Положи a = разность; (перейти к шагу 2)

Означает, что в качестве текущего значения переменной а надо взять разность между значениями, представленными переменными а и b, вычисленную на предыдущем этапе, и перейти к действию 2.

Последовательность действий, выполняемых исполнителем, образуют процесс. Назовем этот процесс вычислительным процессом. Например, для случая исходных данных n=3, m=2 и алгоритма НОД этот процесс можно описать так:

|  |  |  |  |
| --- | --- | --- | --- |
|  | значение а | значение b | № действия |
| 1 | 3 | 2 | 1 |
| 2 | 3 | 2 | 2 |
| 3 | 3 | 2 | 3 |
| 4 | 3 | 2 | 4 |
| 5 | 3 | 2 | 5 |
| 6 | 1 | 2 | 6 |
| 7 | 1 | 2 | 2 |
| 8 | 1 | 2 | 3 |
| 9 | 2 | 1 | 4 |
| 10 | 2 | 1 | 5 |
| 11 | 1 | 1 | 6 |
| 12 | 1 | 1 | 2 |
| 13 | 1 | 1 | 3 |

Стоп

Нетрудно видеть разницу между алгоритмом и вычислительным процессом, им порождаемым. Так, например, действие, которое встречается в алгоритме один раз, может быть выполнено несколько раз, а следовательно встретиться неоднократно в описании вычислительного процесса. Примерами такого действия может быть действие 3, либо действие 5, либо 6. В нашем алгоритме, как правило, экземпляры одного и того же действия будут различаться данными (операндами), над которыми совершается это действие. Однако, эти данные могут быть представлены одними и теми же именами.

По алгоритму не всегда можно предсказать вычислительный процесс. Например, не всегда можно предвидеть точно, как много действий будет в вычислительном процессе. Примером тому может служить алгоритм вычисления НОД.

Алгоритм всегда определяет однозначно, какое действие должно быть выполнено следующим, равно как и то, какое действие должно быть выполнено первым. Очередное действие всегда определено однозначно. Именно этой цели служат слова “(cледующий шаг).” Они явно указывают какое действие должно быть выполнено следующим. В силу этого свойства алгоритма, которое называется детерминированность, вычислительный процесс всегда для заданных исходных данных определен однозначно. Таким образом, при одних и тех же исходных данных вычислительный процес будет одним и тем же.

Обычно по умолчанию предполагают “естественный” порядок выполнения действий в алгоритме. Это означает, что если нет явного изменения порядка выполнения действий, то они выполняются последовательно одно за другим, а выполнение алгоритма начинают с первого по порядку действия. Поэтому, в дальнейшем слова “(следующий шаг)” мы будем опускать.

Рассмотрим подробнее понятие действия. В нашем примере присутствуют как минимум два вида действий:

действия над данными (например, сравни, вычти, положи);

действия, изменяющие “естественный” порядок вычислений (например, если - то - иначе; перейди к шагу #).

Если присмотреться внимательней, то мы увидим, что действия вида “сравни”, “вычти” и действия вида “положи ” - это разные действия. Разница между ними в том, что действия первого вида указывают что делать с его операндами, но не указывают где “сохранить” полученный результат. Действия второго вида, наоборот “умалчивают” каким образом получено значение, которое надо “сохранить” в качестве нового значения переменной.

Теперь пристальнее рассмотрим правило окончания алгоритма и выбора результата. В алгоритме могут быть использованы десятки переменных, но результат будет храниться лишь в нескольких. Эти “результирующие” переменные должны быть явно указаны в алгоритме, т.к. в противном случае нет никакой возможности их выявить. Момент, когда в этих переменных сформированы нужные значения, определяет правило окончания алгоритма. Правило окончания всегда формулируется как некоторое условие на множестве текущих значений переменных. Достижение этого условия и означает, что в определенных переменных получен результат. В нашем примере таким условием является равенство текущего значения переменной а текущему значению переменной b.

Итак, сформулируем наши наблюдения в общей форме. Исходные данные - это значения, с которых начинается исполнение алгоритма. Множество исходных данных всегда точно определено. Оно может быть определено явно, перечислением его элементов, либо не явно, в виде системы правил, определяющих конструкцию его элементов. (Этот способ мы рассмотрим позднее).

Данные в алгоритме могут быть представлены переменными (в нашем примере именуемые a и b), либо явно, в виде постоянной величины - константы (в нашем примере n и m), которая не меняет своего значения в конце выполнения алгоритма.

Переменная - это имя, с которым связано конкрентное множество значений. В каждый конкретный момент времени исполнения алгоритма каждая переменная принимает одно конкретное значение, из связанного с ней множества.

Определение 1.1

Типом переменной называется множество возможных ее значений.

Пусть у нас есть множество переменных {vi|i=1,k}. (Примеры!)

Определение 1.2

Состоянием на множестве переменных {vi|i=1,k} называется набор значений этих переменных.

Пусть А - алгоритм, {vi|i=1,k} - набор всех переменных, используемых в этом алгоритме. Добавим к этому набору еще одну переменную π, тип которой есть множество индексов действий в алгоритме. Каждый шаг исполнения алгоритма можно однозначно охарактеризовать сосотоянием на множестве переменных ({vi|i=1,k},π).

Определение 1.3

Вычислительным процессом, порожденным алгоритмом, называется последовательность шагов алгоритма, пройденных при исполнении этого алгоритма.

Определение 1.4

Состоянием вычислительного процесса, порожденного алгоритмом А, называется состояние на множестве переменных ({vi|i=1,k},π), где{vi|i=1,k} - набор всех переменных, используемых в алгоритме А.

Нетрудно видеть, что вычислительный процесс можно описать в виде последовательности его состояний.

Определение 1.5

Терминальным состоянием называется состояние вычислительного процесса, на множестве значений которого выполняется определенное условие - правило окончания алгоритма.

Определение 1.6

Результат - это определенная совокупность значений из терминального состояния вычислительного процесса алгоритма.

Определение 1.7

Действие - переход из одного состояния в другое.

Действительно, если действие связано с преобразованием каких-либо данных, то правильность этого утверждения очевидна. Если же действие связано с изменением “естественного” порядка выполнения действий в алгоритме, то состояние вычислительного процесса меняется все равно, т.к. изменяется значение специальной переменной π - индекс действия.

Рассмотрим еще один пример. Пусть нам надо построить алгоритм для решения следующего класса задач:

вычислить значение выражения  , в точке x=b, где

ai∈R, b∈R, где R - множество вещественных чисел.

Множество исходных данных: , где ai∈R, b∈R.

 вектор из n+1 вещественных числа;

b - вещественное число.

Результат b r: , где 

Переменные: i - типа целый; х, r - типа вещественный.

Константы:{ai|i=1, n+1}, п.

Нетрудно видеть, что мы имеем дело с классом задач. Ниже приведен алгоритм для этого класса задач.

Алгоритм:

Положи i равным п, x равным b;

Положи r равным ;

Умножь r на x;

Положи r равным произведению;

Положи i равным i -1;

Положи r равным r+;

Если i = 0, то r - результат

иначе перейди к шагу 3;

Организацию вычислений по этому алгоритму можно пояснить вот таким выражением:



Этот метод вычисления значения полинома в точке называется схемой Горнера. Однако, есть и другой алгоритм для решения этих задач, который мы назовем прямым.

Исходные данные: те же, что и в предыдущем примере.

Результат: тот же.

Переменные: r, s, x - типа вещественный, i - типа целый.

Константы: , п.

Алгоритм:

Положи i равным п, s равным 0, х равным b;

Возведи х в степень i

Умножь  на степень;

Положи s равной сумме s и произведения.

Если i = 0, то s - результат (стоп)

иначе положи i=i -1, перейди к шагу 2.

Организацию вычислений по этому алгоритму описывает выражение 

Читателю предлагается построить вычислительные процессы для этих алгоритмов, например, для полинома  в точке 2, и убедиться, что это разные вычислительные процессы.

Итак мы видим, что для решения одного и того же класса задач может существовать несколько алгоритмов, реализующих разные вычислительные процессы. Эти вычислительные процессы различаются набором действий и их количеством. Количество действий в вычислительном процессе - весьма важная характеристика алгоритма, т.к. оно определяет время и ресурсы исполнителя, необходимые для выполнения алгоритма.

Определение 1.8

Сложностью алгоритма называется количество действий в вычислительном процессе этого алгоритма.

Обратите внимание, именно в вычислительном процессе, а не в самом алгоритме.

Для того, чтобы сравнивать сложность разных алгоритмов, надо чтобы она подсчитывалась для них в терминах одинаковых действий. Например, умножь, сложи, положи. Выразим сложность действия возведения в степень i как (i - 1) операций умножения. Тогда, сложность первого алгоритма (по схеме Горнера) в виде числа операций сложения и умножения будет равна 2п. Для прямого алгоритма она будет равна:



Таким образом, первый алгоритм эффективнее второго.

Вывод: Для решения одного и того же класса задач существуют разные алгоритмы, разной сложности.

Теперь рассмотрим вопрос: всегда ли алгоритм дает точное решение? На примере алгоритмов деления в столбик и вычисления корня квадратного не трудно видеть, что, например, при вычислениях 20:3 и, мы вынуждены довольствоваться лишь приблизительным решением.

Теперь рассмотрим свойство массовости алгоритма, выражающееся в том, что алгоритм предназначен “для решения всех задач заданного класса”, т.е. множества задач. Здесь уместно вспомнить парадокс греческого философа Эвбулида "Что есть куча?" Один камень - это куча? Два - это куча? И т.д. В нашем случае наводящим соображением для решения возникшей проблемы является то, что все эти задачи различаются лишь исходными данными.

Массовость предполагает существование четко определенного класса объектов, которые могут быть исходными данными. Мощность этого класса - свойство класса объектов, а не алгоритма. Массовость означает существование языка данных, т.е. четких правил построения этих объектов, называемых данными, из некоторого, как правило, фиксированного множества базовых объектов, называемого алфавитом. Такие объекты в математике называются конструктивными объектами. Примерами конструктивных объектов могут служить слова в некотором фиксированном алфавите. Таким образом, данные - это слова в алфавите. В рассмотренных примерах исходными данными были числа. В данном случае мы можем рассматривать эти числа, как слова в алфавите {0,1,2,3,4,5,6,7,8,9,}. Позднее мы рассмотрим и другие нечисловые примеры, на которых покажем, что в нечисленных алгоритмах исходные данные - слова в определенном алфавите, т.е. конструктивные объекты. Итак, исходные данные - всегда конструктивные объекты.

До сих пор мы рассматривали лишь алгоритмы над числами, т.е. исходными данными были числа, а результатом - число. Рассмотрим проблему построения выигрышной стратегии для определенного класса игр. Выигрышная стратегия - это набор правил, следуя которым один из игроков обязательно выигрывает. Класс игр, который мы будем рассматривать, определим так:

Двое игроков ходят по очереди;

Обязательно выигрывает только один;

При выборе очередного хода случайность отсутствует;

Каждый играющий знает свои ходы и ходы противника.

Примером такой игры может служить игра “чет-нечет”. Есть шесть предметов и два игрока: А и В. Игрок может за один ход взять от 1 до 2 предметов. Проигрывает тот, кто берет последний предмет. Может ли А, который всегда начинает, “заставить” В взять последний предмет? Оказывается - да! Может! Его выигрышная стратегия очень проста: первым ходом он всегда берет два предмета; если число оставшихся предметов не 0 и В взял l предметов, то А должен взять 3-l предметов. (Читателю предлагается проверить этот факт пока экспериментально).

Первый вопрос, который здесь возникает, как представить игру? До сих пор мы имели дело лишь с числами. Нам же необходимо такое представление игры, которое бы позволило проанализировать все возможные комбинации ходов и найти характерные признаки только тех, которые ведут к победе А.

Воспользуемся для представления игры объектом, который в математике называется двоичное дерево. Двоичное дерево состоит из вершин и дуг, соединяющих вершины. У всех вершин, за исключением одной, есть только одна дуга, “заходящая” в эту вершину, и две дуги, “исходящие” из этой вершины. Единственная вершина, в которую “не заходит” ни одна дуга, называется корнем дерева; вершины, из которых “не исходит” ни одной дуги, называются листьями.

Замечание. Нетрудно видеть, что двоичные деревья - конструктивные объекты, т.к. есть набор базовых объектов: вершины и дуги; и одна операция: соединения двух вершин дугой.

Тогда игру “чет-нечет” в виде дерева можно представить так, как показано на рисунке 1.

A

A

A

A

B

B

B

6

6

6

6

6

6

6

6

6

6

6

6

5

5

5

5

5

5

5

5

3

4

4

-

4

4

6

4

3

3

2

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

+

+

+

+

+

+

+

+

+

+

+

+

-

+

-

+

+

+

2

1

Рис.1. Дерево игры “чет-нечет”.

Здесь каждая вершина представляет состояние в игре, после очередного хода. Каждый ход представлен дугой. Число на дуге показывает, сколько всего предметов было взято в игре с учетом этого хода. Конечные вершины, из которых не исходит ни одной дуги и которые называются листьями, помечаются “+”, если выигрывает А и “-” если - В.

Пометим все вершины в этом дереве “+” или “-” по следующему правилу:

Если ход А заканчивается хотя бы одним “+”, то помечаем вершину, с которой начинается ход А, так же “+”. В противном случае помечаем вершину, с которой начинается ход А ,“-“.

Если ход В заканчивается так, что обе вершины помечены “+”, то вершина, с которой начинается этот ход, так же метится “+”. В противном случае ставим “-”.

Ясно, что если в результате этой разметки корень будет помечен “+”, то есть такая последовательность ходов когда А обязательно выигрывает. Читателю предлагается доказать, что предложенная ранее стратегия игры для А в случае 6 предметов действительно является выигрышной и всегда обеспечивает ему победу. Мы же докажем более общую теорему:

Теорема1.1. В любой игре, из рассматриваемого класса, всегда существует выигрышная стратегия для одного из игроков.

Доказательство:

Доказывать эту теорему будем методом индукции.

Для n=1 ( n - кол-во ходов в игре):

На рисунке 2 показано дерево игры из одного хода где К - максимальное количество предметов, которое можно взять за один ход; символ \* обозначает либо ‘-‘ или ‘+’, т.е. выигрыш А, или выигрыш В соответственно.

•



Рис.2. Дерево игры из одного хода

Тогда ходы, заканчивающиеся ‘+’ и образуют выигрышную стратегию для игрока А. Если среди \* есть хотя бы один ‘+’, то А имеет выигрышную стратегию, если нет, то B выигрывает всегда, т.е. он имеет выигрышную стратегию.

2) По индукции: пусть утверждение теоремы верно для n=S, докажем, что оно верно и для S+1. На рисунке 3 показано дерево игры S+1 ходов, где γ - поддеревья, для игры не более, чем из S ходов .

Поскольку γ: - дерево игры из не более, чем S ходов, то для него утверждение теоремы верно. Поэтому, если среди \* в γесть хотя бы один ‘+’ , то существует выигрышное поддерево (γ) (по предположению индукции) и, следовательно, А выиграл. В противном случае выиграл В. Итак, мы доказали существование решения для рассматриваемой проблемы.

Оказывается, что эта теорема верна и для игр, где возможна ничья. Следовательно, она верна и для шахмат! Но тогда, если есть выигрышная стратегия для шахмат, то почему мы говорим об искусстве шахмат? Зачем нужны чемпионаты мира?

Применительно к играм, подобным шахматам, эта теорема показывает, что рассматриваемая проблема лишь потенциально разрешима. Дело в том, что построение дерева игры, такой как шахматы, и его анализ потребует для современных вычислителей времени больше, чем время жизни одного человека или время от поломки до поломки исполнителя.

Таким образом, мы приходим к понятиям о потенциально разрешимых проблемах и практически разрешимых. У потенциально разрешимой проблемы вычислительный процесс даже у самого эффективного алгоритма может оказаться сколь угодно длинным. Конечным, но сколь угодно длинным. Больше, чем, например, жизнь заказчика, которого интересует решение проблемы. У практически разрешимых, вычислительный процесс всегда реализуем за приемлемое для заказчика время. Иными словами, сложность алгоритма у практически разрешимой проблемы имеет разумную с точки зрения заказчика величину.

 •

 \*1 • \*2 .......... • \*к

γ2

γk

γ1

Рис.3. Дерево для игры из S+1 ходов.

Теперь изучим вопрос: что будет, если на вход алгоритма попадут данные не из множества исходных данных. Такое действительно может произойти, если исполнитель не проверит поступившие данные на принадлежность к исходным данным, либо если в алгоритме не предусмотрено соответствующих проверок, либо если само множество исходных данных четко не определено.

Алгоритм:

Исходное данное умножь на 2;

Прибавь к произведению 1;

Раздели сумму на 3;

Раздели исходное данное на остаток.

Если остаток от последнего деления равен 0, то стоп.

Рассмотрим вычислительный процесс для исходных данных:

|  |  |  |
| --- | --- | --- |
|  | 6 | 7 |
| 1. | 12 | 14 |
| 2. | 13 | 15 |
| 3. | 4 (1 в остатке) | 5 (0 в остатке) |
| 4. | 6 | 7:0? |

В случае исходных данных 7 вычислительный процесс становится не определенным. Рассмотрим еще один пример.

Исходные данные: слова в алфавите {a,b};

Действия: aP → Pb, baP → Paba, где P - любое слово в алфавите {a,b};

Правило остановки: Р= aaW, где W - результат.

Рассмотрим вычислительный процесс этого алгоритма для слов babaa; baaba; abaab.

babaa → baaaba → aabaaba стоп.

baaba → abababa → baabab → abababa → bababab → babababa →

→ ...не останавливается.

abaab → baabb → abbaba → bbabab …нет правила на этот случай.

Случаи 2, 3 демонстрируют примеры исходных данных, к которым алгоритм не применим. Эти примеры показывают важность проблемы строгого определения и проверки исходных данных.

Рассматриваемые примеры - это частные случаи более общей проблемы, которую называют проблемой применимости и формулируют так:

Построить алгоритм, который по алгоритму и исходным данным определяет, применим ли алгоритм к этим данным.

(Как мы увидим позднее, эта проблема имеет решение не всегда.)

Теперь уместно задать вопрос: для всякой ли массовой проблемы существует алгоритм? Как мы уже отмечали, Гедель дал отрицательный ответ на этот вопрос. Существуют проблемы, для решения которых нет алгоритмов. Примером одной из таких проблем является 10-я проблема Гильберта:

построить алгоритм решения диофантовых уравнений.

Пример диофантова уравнения: X2 + Y2 - Z2 = 0 , где

X, Y и Z - целые числа.

Одно из решений: X=3, Y=4, Z=5.

В 1969г. отечественный математик Ю.В. Матеясевич показал, что не существует алгоритма для решения произвольного диофантова уравнения.

Итак, подведем итоги:

Не для всякой массовой проблемы существует алгоритм;

Для одной и той же проблемы могут существовать разные алгоритмы с разной сложностью;

Алгоритм определяет последовательность действий;

Данные, алгоритм и вычислительный процесс - конструктивные объекты;

Исходные данные образуют класс, описываемый некоторым языком;

Алгоритм и исходные данные определяют вычислительный процесс полностью;

При одних и тех же исходных данных алгоритм всегда дает один и тот же результат;

Алгоритм одинаково понимается всеми исполнителями.

Алгоритм - точное предписание, которое задает потенциально осуществимый вычислительный процесс, начинающийся с произвольного исходного данного, из класса исходных данных, к которым данный алгоритм применим, и направленный на получение результата, полностью определенного этими исходными данными.

Основные свойства алгоритма:

Массовость;

Потенциальная осуществимость (конечность, сложность);

Детерминированность;

Однозначность понимания всеми исполнителями.

Вопросы :

Что определяет алгоритм?

Что такое вычислительный процесс?

Всегда ли по алгоритму можно определить точное число действий в вычислительном процессе?

2х2 = 4 - это алгоритм ?

Что такое численный алгоритм?

Всегда ли алгоритм дает точное решение?

Что означает массовость алгоритма?

Что такое конструктивный объект?

Отметить какие из нижеперечисленных объектов являются конструктивными - число 2, множество натуральных чисел, множество текстов на русском языке.

Что означает потенциальная осуществимость алгоритма?

Всякий ли алгоритм должен быть потенциально осуществим?

Что такое сложность алгоритма?

В каких единицах измеряется сложность алгоритма?

Можно ли сказать, что понятие области определения функции есть аналог множества исходных данных для алгоритма?

Как отличить результат от исходных данных и промежуточных результатов?