АЛГОРИТМЫ И ИСЧИСЛЕНИЯ

ПЛАН ЛЕКЦИИ

1. Алфавитный оператор
2. Кодирующий оператор
3. Рекурсия в вычислениях
4. Арифметические функции
5. Конструктивные приёмы
6. Частично-рекурсивные функции
7. А.Н. Колмогоров о числах

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
1. Алфавитный оператор (1)

Абстрактный алфавит – любая конечная совокупность (≡множество) объектов, называемых буквами (≡ литерами, символами).

Примеры: $B = \{0, 1\}$, $A = \{a, b, c, ␣\}$

Слово (≡строка) в (над) алфавите (ом) W – любая последовательность символов.

Примеры: 1101, 000, 1000, 10111, … *abc*, *ccbba*, *cba␣aac*, …

Существует ***пустое слово***, не содержащее ни одной литеры.

Число символов в слове называется его ***длиной***.

Алфавитный оператор (≡ *алфавитное отображение*) – всякое соответствие, сопоставляющее словам какого-либо алфавита слова в том же либо другом алфавите.

Эти алфавиты называются, соответственно, *входными* и *выходными* алфавитами.

Существуют области *определения* и *области* значения оператора.

Алфавитный оператор, *не сопоставляющий* некому входному слову φ ни какого, даже пустого слова, называется ***не определённым на слове*** φ.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
1. Алфавитный оператор (2)

$$N \to^{\upsilon} N^{\nabla},$$

N – область определения, N^{∇} – область значения, υ – отображения,

$n \subset N$, $n^{\nabla} \subset N^{\nabla}$ – входное и выходное слова.

Пример: Формальная грамматика, описывающая язык программирования, по сути является алфавитным оператором, принадлежащим к классу нормальных алгорифмов Маркова.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
2. Кодирующий оператор (1)

Кодирующий оператор (≡кодирующее отображение) – соответствие, сопоставляющее любому символу входного алфавита некоторую конечную последовательность символов выходного алфавита.

Пример 1. {♣, ♦, ♥, ♠] – коды карточных мастей.

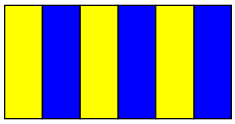
Пример 2. $A = \{a, b, c, d\}$, $B = \{0, 1\}$,

$\upsilon = \{a \to 001, b \to 010, c \to 011, d \to 100\}$.

$n = acdc$, $n^{\nabla} = 001\ 011\ 100\ 011$.

Слово n^{∇} в алфавите B является кодом исходного слова n.

Пример 3: Флаг семафора



АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
2. Кодирующий оператор (2)

Процесс, обратный кодированию, называется *декодированием*

$$N^{\nabla} \to^{\upsilon-1} N .$$

Если слово *n*, закодированное в n^{∇}, декодированием восстанавливается в *n*, то имеет место *обратимость*, заключающаяся в *однозначном соответствии кодирующего и декодирующего отображений*.

Понятие алфавитного оператора общее место процессов преобразования информации и управления, например:

$$y(t) = \int_0^{\infty} h(\tau) \cdot x(t-\tau) d\tau .$$

Алфавитные операторы, задаваемые с помощью *конечной системы правил*, являются алгоритмами. Алгоритм при этом трактуется как *алфавитный оператор* совокупно *с правилами*, определяющими его действие.

Алфавитные операторы ***равны***, если

- они имеют одну и ту же область определения;
- сопоставляют любому наперед заданному входному слову из области определения одинаковые выходные слова.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
2. Кодирующий оператор (3)

Алгоритмы ***равны***, когда

- равны алфавитные операторы, соответствующие им;
- совпадает система правил, задающая их действие на выходные слова.

Алгоритмы ***эквивалентны***, когда

- алфавитные операторы совпадают;
- система правил, задающая их действие на выходные слова, не совпадает.

Алгоритмы и соответствующие им алфавитные операторы, которые любому входному слову соотносят только одно выходное слово, называются ***детерминированными***.

Алгоритмическая система – всякий общий способ задания алгоритма. Приоритетными будут системы, характеризующиеся свойством универсальности, которые способны задать алгоритм, эквивалентный любому, наперед заданному.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
3. Рекурсия в вычислениях (1)

"*… daβ dieser Wurm an Würmen fitt,*
die wiederum an Würmen litten"
Joachim Ringelnatz (1863 – 1934)
"*… а этот глист страдал глистами,*
что мучились глистами сами"
Перевод Л. Макаровой

Рекурсия [прогр.] – способ организации вычислений при котором функция вызывает сама себя с другими аргументами

Примеры.

$i = i + 1$. (1)

$n! = (n - 1) \times n$; $0! = 1$. (2)

Рекурсия [Бауэр Фридрих, Гооз Герхард] – приём, сводящий общую задачу к более простой задаче того же класса.

Индукция – есть разновидность рекурсии в методологии поведения доказательств

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
3. Рекурсия в вычислениях (2)

Рекурсия – способ задания функции, при котором значения определяемой функции для произвольных значений аргументов выражается известным образом через значения определяемой функции для меньших значений аргументов.

Если удаётся показать, что функция, решающая некоторую задачу, *не может быть рекурсивной*, то *задача не разрешима*.

Применение рекурсивных функций в теории алгоритмов основано на предложении перенумеровать слова в произвольном алфавите натуральными числами в порядке возрастания.

В качестве способа такой нумерации, слова располагаются в порядке возрастания их длин, а слова одинаковой длины упорядочиваются в алфавитном порядке.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
4. Арифметические функции

Если пронумеровать входные и выходные данные произвольного алфавитного оператора, последний трансформируется в функцию

$$y = f(x), \qquad (1)$$

где аргумент x и функция y – целые неотрицательные значения.

Такие функции называются ***арифметическими***.

ЭЛЕМЕНТАРНЫЕ АРИФМЕТИЧЕСКИЕ ФУНКЦИИ

- ***Тождественно равные нулю***:

$$0^n(x_1, x_2, \ldots, x_n) = 0.$$

- Тождественные функции, ***повторяющие значения своих аргументов***:

$$I^n{}_i(x_1, x_2, \ldots, x_n) = x_i, \ (1 \le i \le n, n = 1, 2, \ldots).$$

- Функции ***непосредственного следования***:

$$S^1(x) = x + 1.$$

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
5. Конструктивные приёмы (1)

1. Операция суперпозиции функций

Сущность операции: подстановка одних арифметических функций вместо аргументов других.

Пусть заданы функции $f_i(x_1, x_2, \ldots, x_m)$, $i = 1, n$ и $\varphi(y_1, y_2, \ldots, y_n)$.
Суперпозиция функций f_i, $i = 1, n$ и φ даёт новую функцию:

$$g(x_1, x_2, \ldots, x_m) = \varphi[f_1(x_1, x_2, \ldots, x_m), f_2(x_1, x_2, \ldots, x_m), \ldots, f_n(x_1, x_2, \ldots, x_m)].$$

Операция суперпозиции обозначается как S^{n+1}, где n – число функций.

$f(x) = 0$, $g(x) = x + 1$.
$h(x) = g[f(x)] = 0 + 1$, а
$h(x) = g[g(x)] = (x + 1) + 1 = x + 2 \ldots$

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
5. Конструктивные приёмы (2)

2. Операция примитивной рекурсии

Используется для конструирования $n + 1$ – местной арифметической функции по двум заданным: n-местной и $(n + 2)$-местной.

Заданы:

$g(x_1, x_2, \dots, x_n)$ – n-местная

$h(y_1, y_2, \dots, y_{n+2})$ – $(n + 2)$-местная функции.

Если для любых натуральных $x_1, x_2, \dots, x_n$ и z имеем

$f(x_1, x_2, \dots, x_n, 0) = g(x_1, x_2, \dots, x_n)$ и

$f(x_1, x_2, \dots, x_n, z + 1) = h[x_1, x_2, \dots, x_n, z, f(x_1, x_2, \dots, x_n, z)]$

то возникает $(n + 1)$-местная частично-рекурсивная функция f из g и h *примитивной рекурсией*.

Важно! Функция для меньшего числа аргументов можно рассматривать с любым б**О**льшим числом аргументов.

$f(0) = a$

$f(x + 1) = h[x, f(x)]$

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
5. Конструктивные приёмы (3)

Для каждой пары g и h существует единственная функция f. Если способ нахождения g и h известны, то значение f можно вычислить для любого значения m.

Пусть $(d_1, d_2, \dots, d_n) \in D_A$ – набор данных.

$b_0 = g(d_1, d_2, \dots, d_n)$
$b_1 = h(d_1, d_2, \dots, d_n, 0, b_0)$
$b_2 = h(d_1, d_2, \dots, d_n, 1, b_1)$
…
$b_{m+1} = h(d_1, d_2, \dots, d_n, m, b_m)$ то есть $f(d_1, d_2, \dots, d_n, m + 1)$.

Примитивно-рекурсивными называются функции, которые могут быть построены из элементарных арифметических операций с помощью операций суперпозиции и примитивной рекурсии, примененных произвольное число раз и в произвольной последовательности

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
5. Конструктивные приёмы (4)

3. **Операция наименьшего корня** или операция **минимизации**

Для любого заданного набора данных (d_1, d_2, …, d_n) в качестве значений переменных $x_1, x_2, \dots, x_n$ функции f применяется наименьший целый неотрицательный корень $y = z$ уравнения $g(x_1, x_2, \dots, x_n, y) = 0$.

Если целых неотрицательных корней у данного уравнения нет, то функция $f(x_1, x_2, \dots, x_n)$ считается неопределённой при соответствующем наборе значений переменных $x_1, x_2, \dots, x_n$.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
5. Конструктивные приёмы (5)

ПРИМЕР

Доказать примитивную рекурсивность функции суммирования

$$f(x, y) = x + y.$$

$f(x, 0) = x + 0 = x$; $g(x) = x$; (это тождественная функция $I_1^1(x) = x$)

$f(x, 1) = x + 1 = h(x, 0, x)$ (данная функция непосредственного следования $h(x, y, z) = z + 1$);

$f(x, 2) = x + 1 = h(x, 1, x + 1) = x + 2$;

…

$f(x, y - 1) = h(x, y - 2, x + y - 2) = x + y - 1$;

$f(x, y) = h(x, y - 1, x + y - 1) = x + y$.

Таким образом, функция $f(x, y) = x + y$ построена примитивной рекурсией.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
6. Частично-рекурсивные функции (1)

Арифметические функции, которые могут быть построены с помощью операций

- суперпозиции,
- примитивной рекурсии,
- минимизации

называются частично-рекурсивными.

Если частично-рекурсивная функция всюду определена, то она называется **общерекурсивной**.

Общерекурсивная функция *свободна от ограничений* на

- состав,
- порядок и
- количество

применения операций.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
6. Частично-рекурсивные функции (2)

Частично-рекурсивная функция обладает свойствами:

1. каждая стандартно заданная частично-рекурсивная функция вычислима путём определенной процедуры механического характера;
2. какие бы классы алгоритмов не строились доныне, во всех случаях числовые функции, вычисляемые с их помощью, являются частично рекурсивными;
3. класс всех частично-рекурсивных функций совпадает с классом вычисляемых на ЭВМ частичных числовых функций.

Последнее свойство называется гипотезой(тезисом) Чёрча (Church Alonzo). Тезис **не доказан** и **не опровергнут**.

Понятием частично-рекурсивной функции пользуются для доказательства алгоритмической разрешимости или неразрешимости какой-либо задачи.

АЛГОРИТМЫ И ИСЧИСЛЕНИЯ
6. А.Н. Колмогоров о числах

Изучая числовые характеристики алгоритмов (длина программы, число операций и т.д.) академик А.Н. Колмогоров предложил классификацию.

1. Число *n* называется малым, если практически (для человека либо ЭВМ) возможно перебрать все релейно-контактные схемы из *n* элементов или выписать для них все булевы функции, общим числом $N = 2^{2^n}$.
2. Число *n* называется средним, если перебрать все схемы невозможно, но можно перебрать сами элементы или выработать систему обозначений (меток) для любой схемы из *n* элементов.
3. Число *n* называется большим, если невозможен перебор такого числа элементов, но можно установить систему обозначений для этих элементов.
4. Если невозможно реализовать условие 3, то число *n* называется сверхбольшим.