# ОСНОВЫ ПРОГРАММИРОВАНИЯ ДЛЯ МАТЕМАТИКОВ. Часть I

А.М. Магомедов

МАХАЧКАЛА – 2014

Рецензенты:
Якубов А.З – доцент кафедры дискретной математики и информатики ДГУ, канд. физ.-мат. наук.,
Лугуев Т.С. – доцент кафедры дискретной математики и информатики ДГУ, канд. физ.-мат. наук

Одобрено учебно-методической комиссией ФМиКН ДГУ

Магомедов А.М.

Часть I курса лекций по основам программирования содержит материал первых пяти лекций, прочитанных в 2014-15 у. г. студентам 1-го курса факультета математики и компьютерных наук Дагестанского государственного университета, обучающихся по специальности «Фундаментальная информатика и информационные технологии».

Магомедов Абдулкарим Магомедович

Основы программирования для математиков

Курс лекций. Часть I

Оглавление

# Лекция 1. Системы счисления

Лекция содержит краткие сведения из истории систем счисления, рассмотрены позиционные и непозиционные системы, переводы чисел из одной системы в другую, представление целых чисел в памяти компьютера, затронуты вопросы машинной арифметики. Использованы источники [1] – [5].

## 1.1. Унарная система

Естественно предположить, что когда люди начали считать, у них появилась потребность в записи чисел. Находки археологов на стоянках первобытных людей свидетельствуют о том, что первоначально количество предметов изображали повторением одного знака: зарубки, черточки, точки.

Позже для облегчения счета эти значки стали группировать по три или по пять. Такая система записи чисел называется единичной или *унарной*, так как любое число в ней образуется путем повторения одного знака, символизирующего единицу. Отголоски единичной системы счисления встречаются и сегодня. Так, чтобы узнать, на каком курсе учится курсант военного училища, нужно сосчитать число полосок на его рукаве. Начинающий приверженец бразильского джиу-джитсу, обладатель белого пояса, получает пометки: одну, две, три и четыре, после чего он получает право на синий пояс (рис. 1).

Единичной системой счисления пользуются малыши, показывая на пальцах свой возраст, а счетные палочки используются для обучения учеников 1-го класса счету. Другие примеры: узлы, четки. Унарная система – пример непозиционной системы.

Рис. 1. Одному из спортсменов из числа программистов известный тренер Карлос Сиеверт завязывает синий пояс (г. Сиэтл, шт. Вашингтон).

### 1.2. Непозиционные системы

В непозиционной системе каждой цифре в любом месте записи числа соответствует одно и то же значение. Римская система счисления возникла более двух с половиной тысяч лет назад в Древнем Риме. Ее алфавит содержит 7 символов, они приведены в следующей таблице слева, справа – записи целых чисел от 1 до 10.

| | | | |
|---|---|---|---|
| I | 1 | I | 1 |
| V | 5 | II | 2 |
| X | 10 | III | 3 |
| L | 50 | IV | 4 (без одного пять) |
| C | 100 | V | 5 |
| D | 500 | VI | 6 |
| M | 1000 | VII | 7 |
| | | VIII | 8 |
| | | IX | 9 (без одного десять) |
| | | X | 10 |

Числа в римской системе записываются слева направо, от бóльших к меньшим. Например, XI = 11, XII = 12, XIII = 13.

В записи натуральных чисел цифры могут повторяться, но не в такой степени, как в унарной системе. Если при этом бóльшая цифра

стоит перед меньшей, то они складываются (*принцип сложения*), если же меньшая – перед бóльшей, то меньшая вычитается из большей (*принцип вычитания*). Последнее правило применяется во избежание четырёхкратного повторения одной и той же цифры.

**Замечание 1.** Левая цифра может быть меньше только на «порядок», т.е. для V и X – I, для L и C – X, для D и M – C.

Приведем примеры. XC – 90, XD – 490, M CM LXX IV – 1974, MMMM – 4000 (нет символа для обозначения числа выше 1000), MMMMMMMMMXM – 9990, MMMMMMMMMM – 10000.

Следующий пример изложим несколько более подробно:

MCMXCIX = 1000+(1000-100)+(100-10)+(10-1)=1999.

**Замечание 2.** Римская система не является полностью непозиционной, так как (см. выше) меньшая цифра, идущая перед бóльшей, вычитается из неё.

Римская нумерация преобладала в Италии до XIII в., а в других странах Западной Европы – до XVI в. Еще 200 лет назад в деловых бумагах числа должны были обозначаться римскими цифрами, считалось, что обычные арабские цифры легче подделать. С нею мы часто сталкиваемся и в повседневной жизни. Это номера глав в книгах, указание века, числа на циферблате часов и т. д.

Итак, римская система не является вполне позиционной. Древнеегипетская система счисления – пример «чисто» непозиционной десятичной системы, где величина числа не зависит от порядка расположения составляющих его знаков: их можно записывать сверху вниз, справа налево или вперемежку. Она возникла пример-

но в 3 тыс. лет до н.э. Для обозначения 1, 10, 100, 1000 и т.д. имелись специальные знаки: шест (1), дуга или путы для скота (100), свернутый лист пальмы или веревка (100), цветок лотоса (1000), палец (10000), головастик (100000), божество (1000000) и т.д.

Для сравнения: римская система не является десятичной; чтобы записать число, римляне раскладывали его на сумму тысяч, полутысяч, сотен, полусотен, десятков, пятерок и единиц.

В 595 году нашей эры в Индии появилась знакомая нам десятичная система счисления. Знаменитый персидский математик Аль-Хорезми выпустил в IX веке учебник, в котором изложил основы десятичной системы индусов. После перевода его с арабского языка на латынь и выпуска в XIII в. книги Леонардо Пизано (Фибоначчи) эта система счисления стала доступна европейцам, получив название арабской.

Непозиционные системы счисления имеют ряд существенных недостатков:

1. Существует постоянная потребность введения новых знаков для записи больших чисел.
2. Невозможно представлять дробные и отрицательные числа.
3. Сложно выполнять арифметические операции, так как не существует алгоритмов их выполнения.

### 1.3. Позиционные системы счисления

Система счисления называется позиционной, если одна и та же цифра имеет разные значения, которые определяются ее позицией в последовательности цифр, изображающих число.

Основные достоинства любой позиционной системы счисления – простота выполнения арифметических операций и ограниченное количество символов (цифр), необходимых для записи любых чисел.

Вавилонская 60-ричная система счисления появилась за 2000 лет до н.э., это одна из первых известных систем счисления, основанная на позиционном принципе. Ее следы заметны и в наши дни: мы делим один час на 60 минут, а минуту делим на 60 секунд, окружность – на 360 частей. В этой системе счисления числа составлялись из двух видов знаков: прямой клин использовался для обозначения единиц, а лежачий клин – для обозначения десятков.

| | | | |
|---|---|---|---|
|  |  |  |  |
| Ноль | Прямой клин – 1 | Лежащий клин – 10 | Запись в вавилонской системе |

Вавилоняне первыми придумали ноль, в вавилонской системе разряды – позиционные, но разряд – это не цифра, а непозиционная группа цифр. Мы не останавливаемся подробно на этой весьма развитой системе, отсылая интересующихся к статье [5].

Наиболее распространены десятичная и двоичная системы счисления (последняя была впервые предложена Г. В. Лейбницем в 1703 г.).

В современных позиционных системах в качестве основания выбирается целое положительное число $\alpha > 1$, и число изображается с помощью цифр из множества $\{0, 1, \dots, \alpha - 1\}$:

число

$$x_n x_{n-1} \dots x_2 x_1 x_0 . x_{-1} \dots x_{-k}$$

равно сумме

$$x_n \cdot \alpha^n + x_{n-1} \cdot \alpha^{n-1} + \cdots + x_1 \cdot \alpha^1 + x_0 \cdot \alpha^0 + x_{-1} \cdot a^{-1} + \cdots x_{-k} \cdot a^{-k}.$$

| 10 | 2 | 3 | 8 | 16 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 | 2 |
| 3 | 11 | 10 | 3 | 3 |
| 4 | 100 | 11 | 4 | 4 |
| 5 | 101 | 12 | 5 | 5 |
| 6 | 110 | | 6 | 6 |
| 7 | 111 | | 7 | 7 |
| 8 | 1000 | | 10 | 8 |
| 9 | 1001 | 100 | 11 | 9 |
| 10 | 1010 | 101 | 12 | A |
| 11 | 1011 | 102 | 13 | B |
| 12 | 1100 | 110 | | C |
| 13 | 1101 | 111 | 15 | D |
| 14 | 1110 | 112 | 16 | E |
| 15 | 1111 | 120 | 17 | F |
| 16 | 10000 | 121 | 20 | |

Рис. 2. К упражнению 4.

Таким образом, основание системы счисления определяет количество различных цифр, которых можно использовать для записи, а также – сколько раз меняется «вес» цифры при ее перемещении на одну позицию.

Упражнение. Решить самостоятельно:

1) $124.5_{10} = (\ )_2$. 2) $101_2 = (\ )_{10}$.

3) Пусть в системе счисления с основанием $\alpha$ $25 = 31_\alpha$. Найдите $\alpha$.

Решение: над 3 пишем 1, а над 1 пишем 0:

$$25 = 3 \cdot \alpha^1 + 1 \cdot \alpha^0.$$

Решим уравнение и получим, что $\alpha = 8$.

4) Заполните пустые клетки таблицы на рис. 2, где столбцы соответствуют системам счисления с основаниями 10, 2, 3, 8 и 16 слева-направо. Ответы (сверху вниз): 20, 21, 22, 14, 10.

### 1.4. Способы перевода чисел в систему с основанием $\alpha$

Способ 1 – для натурального числа. Деление «уголком» на $\alpha$ до получения в частном цифры, меньшей основания системы. После этой цифры в обратном порядке записать все остатки.

Способ 2 – для дробных чисел (<1). Последовательное умножение дробной части числа и дробной части произведений на $\alpha$. Дробная часть ответа записывается последовательными целыми частями полученных произведений.

Упражнение 5. Перевести устно число 141 в двоичную, 8-ричную и 16-ричную.

$141_{10} = 10001101_2 = 215_8 = 8D_{16}$.

Упражнение 6. Перевести устно число 3,25 в двоичную систему.

### 1.5. Представление в памяти целых отрицательных чисел. Элементы машинной арифметики

Из школьной информатики известно, что в Арифметико-Логическом Устройстве (АЛУ) аппаратно реализовано лишь сложение целых положительных чисел, для остальных же арифметических действий предусмотрена программная эмуляция. Чтобы понять, как выполняется вычитание целых чисел, рассмотрим представление целого отрицательного числа $-\alpha$ (например, -12) в памяти компьютера в *дополнительном коде* в предположении, что длина разрядной сетки равна $n = 8$.

Прежде всего, вычислим прямой код числа 12: 00001100; здесь добавлением лидирующих нулей двоичное представление 1100 дополнено до восьми двоичных цифр.

Затем вычислим обратный код заданного числа, для чего вычтем из $2^n - 1$ (в двоичной системе записывается с помощью $n$ единиц) прямой код: результат, очевидно, отличается от прямого кода заменой единиц на нули и наоборот: 11110011.

Для получения дополнительного кода остается прибавить «столбиком» 1 к обратному коду:

```
11110011
+       1
```

Складывая последние разряды, получим 10 (двоичную запись 2), запишем 0 в последнем разряде суммы, а 1 сохраним «в уме»:

```
        1
11110011
+       1
----------
        0
```

Складывая единицы в предпоследнем разряде, снова получим 10, поэтому снова сохраним 1 «в уме»:

```
       1
11110011
+       1
----------
       00
```

Далее легче: складываем 1 (которое «пошло на ум», как в известном фильме), с 0, что даст 1; остается перенести первые пять разрядов числа в результат:

```
       1
11110011
+       1
----------
11110100
```

Таким образом, дополнительный код числа -12 равен 11110100.

Вспоминая приведенное выше представление обратного кода, имеем, что он равен $2^n - \alpha$.

Рассмотрим теперь подробности выполнения действия 15-12. Прежде всего, действие будет рассматриваться АЛУ как сложение чисел 15 и (-12). После замены 15 прямым кодом, а -12 – дополнительным кодом:

```
 00001111
+
 11110100
------------
100000011
```

Лидирующая единица занимает $(n + 1)$-ю позицию и выпадает из разрядной сетки; таким образом, в результате ячейка сохраняет двоичное значение 00000011, т.е. значение 3.

### 1.6. Примеры программ вывода представления чисел в памяти

Представление целых чисел типа byte.

```
program Project1;
{$APPTYPE CONSOLE}
uses  SysUtils;
var i: 0..7; a: 0..256;
begin
  read (a);
  for i:=7 downto 0 do
          write ( (a shr i) and 1);
  readln;
end.
```

Вопрос: какую ошибку вы заметили? Опишите вывод при a=256.

Представление целых и вещественных 4-байтных чисел.

```
program Project1;
{$APPTYPE CONSOLE}
```

```
uses
  SysUtils;
type
  array4=array [0..3] of byte;
var
  i: 0..31; j: 0..3;
  sL: Longint;
  L: Longint=260;
S: Single=2.25;
  a: array4;
begin
 for i:=31 downto 0 do
    write ( (L shr i) and 1);
  a:=array4(S);
  writeln;
  for j:=3 downto 0 do
  begin
     for i:=7 downto 0 do write ( (a[j] shr i) and 1);
  end;
  readln;
end.
```

Вопрос: как сократить программу путем применения директивы компилятора Absolute?

Перевод чисел типа Extended (10 байтов).

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
  TbyteArray=array[0..9] of byte;
var
  a: extended=0.25;
  b: TbyteArray;
  i,j: byte;
begin
  b:=TbyteArray (a);
  for j:=9 downto 0 do
    for i:=7 downto 0 do
    write((b[j] shr i) and 1);
  readln;
```

end.

### 1.7. Функции Excel для представления чисел в различных системах счисления

Используя функции LOG (число; основание) и функцию ОКРУГЛВВЕРХ (число; количество разрядов), найдите количество разрядов в $x$-ричной записи числа 255 при $x = 10, 2, 3, 5, 8, 16$.

Указание: верхняя целая часть логарифма от числа по $x$.

Например, в случае $x = 2$:

=ОКРУГЛВВЕРХ (LOG (255;2);0)=8.

Для перевода числа в римскую систему предусмотрена функция вида:

=РИМСКОЕ (число до 3999).

Функции, предназначенные перевод целых чисел в двоичную систему (инженерные функции):

=ВОСЬМ.В.ДВ (число): конвертирование числа из восьмеричной системы счисления;

=ДЕС.В.ДВ (число): конвертирование числа из десятичной системы счисления;

=ШЕСТН.В.ДВ (число): конвертирование числа из шестнадцатеричной системы счисления.

Перевод чисел в десятичную систему:

=ДВ.В.ДЕС (число) – конвертирование числа из двоичной системы счисления;

=ВОСЬМ.В.ДЕС (число) – конвертирование числа из восьмеричной системы счисления;

=ШЕСТН.В.ДЕС (число) – конвертирование числа из шестнадцатеричной системы счисления.

Перевод чисел в восьмеричную систему:

=ДВ.В.ВОСЬМ (число) – конвертирование числа из двоичной системы счисления;

=ДЕС.В.ВОСЬМ (число) – конвертирование числа из десятичной системы счисления;

=ШЕСТН.В.ВОСЬМ (число) – конвертирование числа из шестнадцатеричной системы счисления.

Перевод чисел в шестнадцатеричную систему:

=ДВ.В.ШЕСТН (число) – конвертирование числа из двоичной системы счисления;

=ВОСЬМ.В.ШЕСТН (число) – конвертирование числа из восьмиричной системы счисления;

=ДЕС.В.ШЕСТН(число) – конвертирование числа из десятичной системы счисления.

### 1.8. Игра НИМ

Играют два игрока. Три кучи камней содержат $a$, $b$ и $c$ камней соответственно. Игроки по очереди выбирают какую-либо кучу и берут оттуда камни. Выигрывает игрок, которому достался последний камень. Спрашивается, как по значениям $a$, $b$ и $c$ узнать, кто выиграет при правильной игре? Какова выигрышная стратегия?

**Решение**.

1. Представления $a$, $b$ и $c$ в двоичной системе разместим так, чтобы младшие разряды располагались друг под другом справа, старшие также друг под другом левее. Те представления, где меньше цифр, чем в других, дополним слева лидирующими нулями. Длину представления обозначим $n$, а суммы по столбцам – через $S_i$,

$0 \le i \le n-1$ справа налево ($S_0$ – сумма младших разрядов и т.д.).

2. Утверждение. Если среди $S_i$ $(i = 0, \dots, n-1)$ имеется хотя бы одно нечетное значение, т.е. 1 или 3 («ситуация нечетности»), то начинающий обладает выигрышной стратегией игры; если же все суммы четные («ситуация четности»), то выигрышной стратегией обладает второй игрок.

3. Достаточно доказать, что *для любой ситуации нечетности имеется возможность совершить ход, преобразующий ее к ситуации четности, а любой ход в ситуации четности воссоздает ситуацию нечетности*. В самом деле, если утверждение будет доказано, то игрок, начинающий в ситуации нечетности, после конечного количества ходов создаст для оппонента ситуацию четности, где все разряды равны нулю, и выиграет.

4. **Пусть имеем ситуацию нечетности** и $k$ – самый левый из тех столбцов, в которых суммы имеют нечетные значения; тогда, очевидно, либо один элемент, либо все три элемента в столбце $k$ имеют значение 1. Обозначим через $j$ номер любой строки, содержащей 1 в столбце $k$, и заменим 1 на 0.

Далее, продвигаясь в строке $j$ вправо от столбца $k+1$ по направлению к меньшим разрядам, выполним действия: если сумма в текущем столбце четная, то разряд в строке $j$ сохраняется, если нечетная – меняется (с 0 на 1 или наоборот). В результате сумма элементов в каждом столбце станет четным.

5. **Пусть имеем ситуацию четности.** Т.к. любое взятие камней есть уменьшение элементов одной из трех строк, то хотя бы в одной позиции этой строки (пусть, для определенности, в позиции $k$) значе-

ние разряда будет изменено. Следовательно, в столбце $k$ сумма изменится на нечетное значение. Таким образом, утверждение, выделенное выше курсивом, доказано полностью.

**Литература:**

1. Системы счисления. URL: https://ru.wikipedia.org/wiki/ (Дата просмотра: 5 сентября 2014 г.).

2. Гашков С. Б. Системы счисления и их применение. – М.: МЦНМО, 2004.

3. Фомин С. В. Системы счисления. – М.: Наука, 1987. – 48 с.

4. Яглом И. Системы счисления // Квант. – 1970. – № 6. – С. 2-10.

5. Веселовский И. Н. Вавилонская математика // Труды Института истории естествознания и техники. – М.: Академия наук СССР, 1955. – В. 5. – С. 241-304.

**Вопросы по лекции:**

Какие системы называются непозиционными? Определение римской системы счисления.

Какие системы счисления называются позиционными?

Сформулируйте, какой смысл имеет основание системы счисления. Какие важные свойства записи числа оно характеризует?

По записи числа в заданной системе счисления написать развернутое представление в виде суммы (по степеням основания).

Перевод заданного натурального числа в двоичную систему и обратно.

Перевод заданного дробного (от нуля до единицы) числа в двоичную систему (с заданным числом цифр после запятой) и обратно.

Перевод заданного числа с целой и дробной частями в двоичную систему (с заданным числом цифр после запятой) и обратно.

Программа вывода на экран представления в памяти целого числа из диапазона от 0 до 255.

Программа вывода на экран представления в памяти целого отрицательного числа из диапазона от -256 до -1.

Пусть известно, что под целое число выделен байт памяти, содержащий восемь единиц. Что можно сказать о значении данного числа?

Перевод заданного двоичного числа (целого или дробного) в троичную систему.

Перевести устно целые числа 1, …, 16 в 2-ю, 3-ю, 8-ю и 16-ю систему.Игра НИМ.

## Лекция 2. Экономичность системы счисления

Сначала мы рассмотрим одно замечательное свойство троичной системы счисления, благодаря которому ей было отдано предпочтение в некоторых ранних ЭВМ. Затем приступим к изучению вопросов применения систем счисления в исследовании математических проблем. Это изучение найдет продолжение в следующей лекции. Использованы материалы [1]-[3].

### 2.1. Из истории троичной системы

Позиционной системе счисления с основанием 3 в литературе уделено больше внимания, нежели другим системам счисления (за исключением разве лишь двоичной). Наиболее часто в этих исследованиях упоминается симметричная троичная система с алфавитом {-1,0,+1}. Симметричная троичная система использовалась, например, в ЭВМ Сетунь. В отечественной литературе встречается также утверждение, что, работая в палате мер и весов, Д. И. Менделеев с учётом симметричной троичной системы счисления разработал цифровой ряд значений весов разновеса для взвешивания на лабораторных весах, который используется по сей день.

Объективности ради заметим, что позиционная симметричная троичная система счисления была предложена итальянским математиком Фибоначчи (1170 – 1250) для решения «задачи о гирях». Задачу о наилучшей системе гирь рассматривал также Лука Пачоли (XV в.), а частный случай этой задачи был изложен в 1612 г. в книге французского математика Клода Баше де Мезириака «Сборник

занимательных задач». Русский перевод книги «Игры и задачи, основанные на математике» вышел в Петербурге в 1877 г. Этой задачей занимался Леонард Эйлер. Но и Д. И. Менделеев, действительно, интересовался данной задачей. Вообще, симметричность при взвешивании на рычажных весах использовали с древнейших времён, добавляя гирю на чашу с товаром. Элементы троичной системы счисления присутствовали в системе счисления древних шумеров, в системах мер, весов и денег, в которых присутствовали единицы, равные 3. Эти сведения заимствованы из [1], где упоминается и о следующем замечательном свойстве: троичная позиционная показательная несимметричная система счисления по затратам числа знаков наиболее экономична из позиционных показательных несимметричных систем счисления. Доказательство этого факта обычно связывают с именем Джона фон Неймана.

### 2.2. Самая экономичная система счисления

**Лемма 1**. Число, написанное в 2-ичной системе с помощью $m$ единиц, равно $2^m$-1.

**Доказательство.** Для вычисления десятичного эквивалента значения $(11\ldots1)_2$, записанного $m$ единицами, просуммируем члены геометрической прогрессии по формуле

$$b_1 + b_1q + \cdots + b_1q^{m-1} = \frac{b_1 - b_1q^m}{1-q},$$

где $b_1 = 1,\ q = 2$. Получим:

$$\frac{1-2^m}{1-2} = 2^m - 1.$$

Пусть даны $n$ карточек, на которых предлагается написать цифры системы счисления с основанием $x$ так, чтобы представить любое целое число от 0 до $y$ при возможно бóльшем значении $y$. Понятно, что $y$ будет зависеть от $x$: $y = y(x)$. Систему счисления с основанием $x$ будем называть *самым экономичным*, если при данном основании системы $y(x)$ достигает наибольшего значения.

Если, например, $n = 30$, то при $x = 2$, написав на 15 карточках 0, на остальных 15 карточках – 1, можно записать любое целое число, состоящее из 15 двоичных цифр. Наибольшее из них – двоичное число, запись которого состоит из 15 единиц, – равно, согласно лемме 1, $2^{15}$- 1. Понятно, что данное значение совпадает с количеством всех представимых такими карточками чисел, начиная с 1. С учетом числа 0 получим всего $2^{15}$ чисел. Если же, например, написать меньше единиц (нулей), нежели 15, то нельзя будет с помощью карточек записать число из 15 единиц (соответственно, нельзя записать число из 15 нулей).

*Упражнение для самостоятельного решения*. Доказать, что наибольшее значение при выбранном основании получится, когда каждой цифре алфавита отведено одно и то же количество карточек. Таким образом, удается представить подряд идущие 15-значные числа:

0…00, 0…01, 0…10, 0…11, …., 1…11.

Выберем $m = 15$ в лемме 1 и получим, что $y(2) = 2^{15} - 1 + 1 = 2^{\frac{30}{2}}$. Рассуждая аналогичным образом при $n = 30$, $x = 3$, получим $y(3) = 2^{\frac{30}{3}}$.

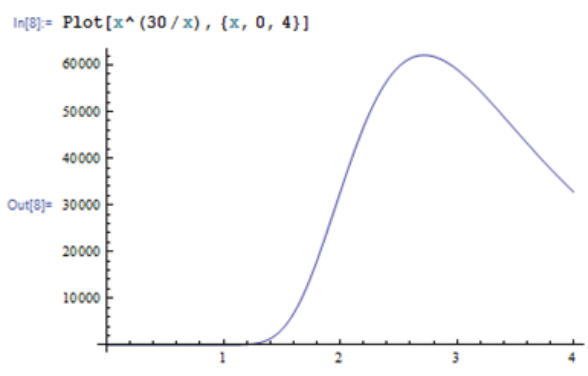
В общем случае $y(x) = x^{\frac{n}{x}}$. Исследуем эту функцию на максимум. Для вычисления производной от показательно-степенной функции $y(x) = u(x)^{v(x)}$ обычно применяют два метода. Первый заключается в том, что вычисляется производная как от степенной функции и складывается с производной, вычисленной как от показательной функции: $\left(u(x)^{v(x)}\right)' = v(x) \cdot u(x)^{v(x)-1} + u(x)^{v(x)} \cdot \ln u(x) \cdot v'(x)$. Мы используем второй способ, называемы «логарифмическим дифференцированием»:

$$y = x^{\frac{n}{x}};\ \ln y = \frac{n}{x}\ln x;\ \frac{y'}{y} = -\frac{n}{x^2}\ln x + \frac{n}{x^2};\ \frac{y'}{y} = \frac{n}{x^2}(1 - \ln x).$$

Отсюда $y' = y \cdot \frac{n}{x^2}(1 - \ln x) = x^{\frac{n}{x}} \cdot \frac{n}{x^2}(1 - \ln x)$.

Приравнивая производную к нулю, находим единственную критическую точку $x = e$. Поскольку значения производной слева от критической точки положительны, а справа – отрицательны, то в данной точке достигается наибольшее значение. Несложное исследование графика функции показывает, что $y(3) = 3^{10} > 2^{15} = y(2)$. В самом деле, $(3^2)^5 > (2^3)^5$.

На рис. 1 представлены графики функции $y(x) = x^{\frac{30}{x}}$, полученные с использованием системы компьютерной математики. Графики различаются лишь областью изменения $x$: (0,4) и (0, 12) соответственно.



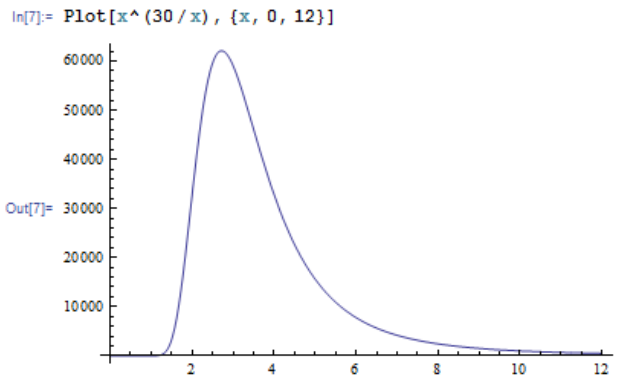




Рис. 1. Графики функции $y(x) = x^{\frac{30}{x}}$.

Из левого графика видно, что среди целых $x$ точкой максимума является $x = 3$, правый график приведен для усиления наглядности.

Таким образом, самой экономичной является 3-ичная система счисления.

Однако преимущества 2-ичной системы по технической реализуемости оказались решающими на весах человеческой практики.

### 2.3. Применение систем счисления в теории множеств

Для конечного множества понятие мощности служит синонимом числа элементов множества. Так, мощность множества {-1, 0, 1} равна трем. Два множества с одинаковым числом элементов называются *равномощными*. Понятно, что между элементами таких множеств можно установить взаимно-однозначное соответствие (биекцию). Последнее соображение позволяет распространить понятие мощности и на бесконечные множества.

Два множества (конечные или бесконечные) называются равномощными, если между их элементами можно установить биекцию. Множество называется *счетным*, если его элементы можно пронумеровать (всеми) натуральными числами: 1, 2, …

Очевидно, что множество четных чисел и множество целых чисел равномощны; в самом деле, каждому $x$ можно сопоставить $2x$.

Упражнение (для устного решения). Докажите, что множество натуральных чисел и множество целых чисел равномощны.

Как видно из данного утверждения, множество всех целых чисел счетно, т.е. для нумерации его элементов достаточно использовать часть его же элементов. Следующее утверждение еще удивительнее.

**Теорема 1.** Множество рациональных чисел счетно.

**Доказательство**. Построим матрицу, бесконечную вправо и вниз: выпишем в первом столбце все положительные рациональные числа со знаменателем 1, во втором – все положительные рациональные числа со знаменателем 2 и т.д., ограничиваясь несократимыми дробями. Затем пронумеруем их, как показано на рис. 2. Теорема доказана.


Рис. 2. Нумерация рациональных чисел.

С множеством всех действительных чисел ситуация обстоит иначе, даже если ограничиться множеством действительных чисел из промежутка (0; 1). И здесь не обойтись без двоичной системы.

**Теорема 2.** Существует биекция между множеством всех бесконечных двоичных последовательностей и множеством всех действительных чисел из промежутка (0; 1].


Георг Кантор (1845-1918).

**Доказательство.** Каждому $x \in (0; 1]$ поставим в соответствие двоичную последовательность следующим образом. Вначале последовательность пуста, в качестве промежутка (a; b] выберем (0; 1]. Разобьем промежуток (a; b] на два:

$(a, \frac{a+b}{2}]$ и $(\frac{a+b}{2}, b]$.

Если $x$ принадлежит левому из них, добавим к имеющейся двоичную последовательности 0, иначе – 1. Промежуток, содержащий $x$, переобозначим $(a;\ b]$ и снова разобьем на два промежутка; если $x$ принадлежит левому из них, добавим к последовательности 0, иначе – 1. Продолжив этот процесс до бесконечности, получим двоичную последовательность, которую и поставим в соответствие выбранному $x$.

Чтобы убедиться, что установленное таким способом отображение является биекцией, достаточно проследить процесс однозначного восстановления $x$ по двоичной последовательности с получением последовательности вложенных промежутков (содержащих искомое $x$) с длиной, стремящейся к нулю, и воспользоваться известной леммой из курса математического анализа о существовании и единственности общей точки такой последовательности промежутков.

Теорема доказана.

Покажем теперь, что действительные числа промежутка (0; 1) нельзя пронумеровать натуральными числами. Для доказательства применим знаменитую «диагональ Г. Кантора».

**Теорема 3.** Множество всех действительных чисел из промежутка (0; 1) несчетно.

**Доказательство**. Допустим противное: пусть нумерация действительных чисел промежутка (0; 1) возможна. Разместим каждую *i*-ю двоичную последовательность в *i*- й строке бесконечной матрицы (рис. 3).

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | … |
| 1 | 1 | 1 | 1 | 1 | 1 | … |
| 0 | 1 | 0 | 1 | 0 | 1 | … |
| 1 | 0 | 1 | 0 | 1 | 0 | … |
| 0 | 1 | 1 | 0 | 0 | 1 | … |
| 1 | 0 | 0 | 1 | 1 | 1 | … |
| … | … | … | … | … | … | … |
| | | | | | | |
| 1 | 0 | 1 | 1 | 1 | 0 | … |

Рис. 3. Диагональ Кантора.

Инвертированная диагональная двоичная последовательность отличается от первой последовательности своим 1-м разрядом, от второй – 2-м разрядом, от *k*- й – своим *k*-м разрядом. Т.е. этой последовательности нет ни в одной строке. Полученное противоречие доказывает теорему.

### Литература:

1. Троичная система счисления, 2014.
URL: https://ru.wikipedia.org/wiki/ (Дата просмотра: 5 октября 2014 г.).

2. Брусенцов Н. П., Маслов С. П. , Розин В. П. , Тишулина А. М. Малая цифровая вычислительная машина Сетунь. – М.: Издательство Московского университета, 1965.

3. Фомин С. В. Системы счисления. – М.: Наука, 1987. – 48 с.

### Вопросы к лекции:

Объясните понятие «самая экономичная система».

Какая функция исследуется в задаче поиска самой экономичной системы?

Точка максимума этой функции ближе к $x = 3$, нежели к $x = 2$. Можно ли из этого факта заключить, что троичная система экономичнее, нежели двоичная (или имеется иное основание для такого заключения)?

Понятие равномощности множеств.

Какое множество называется счетным?

Докажите, что множество четных чисел счетно.

Докажите, что множество рациональных чисел счетно.

Придумайте алгоритм, с помощью которого можно установить взаимно-однозначное соответствие между множеством действительных чисел промежутка $(0; 1)$ и множеством всех действительных чисел.

Каким свойством обладает множество всех двоичных последовательностей?

Докажите, что множество всех действительных чисел несчетно.

## Лекция 3. Связи с проблемами математики

В лекции рассмотрены некоторые проблемы математики, которые способствовали развитию компьютерных наук и, в свою очередь, испытали благотворное обратное воздействие этого развития. В частности, показано применение идей систем счисления к доказательству счетности множества алгебраических чисел, обозначены связи с одной из древнейших задач математики и выявлены компьютерные аспекты вопросов, относящихся к числам Ферма; указано на существенное различие между результатом о существовании объекта и разработкой алгоритма его построения; сформулированный в конце лекции вопрос о существовании алгоритма для проверки разрешимости уравнений в целых числах является фактически приглашением к уточнению понятия алгоритма, что явится темой следующей лекции. Использованы материалы из [1]-[3].

### 3.1. Алгебраические и трансцендентные числа

В изложении данного вопроса использованы материалы статьи [1].

Действительное число называется *алгебраическим*, если оно является корнем какого-нибудь алгебраического уравнения с целыми коэффициентами. В противном случае число называется *трансцендентным*.

Любое рациональное число $m/n$ – алгебраическое, т.к. является корнем уравнения $nx - m = 0$ с целыми коэффициентами $n$ и $-m$.

Отсюда видно, что множество $A$ всех алгебраических чисел бесконечно.

Число $\sqrt{2}$, не являясь рациональным, также относится к алгебраическим числам, т.к. является корнем уравнения второй степени $x^2 - 2 = 0$ с целыми коэффициентами 1 и -2. Вообще, все числа, получаемые из целых чисел с помощью арифметических операций и операции извлечения корня – алгебраические.

А существует ли хотя бы одно трансцендентное число? Если доказать, что множество $A$ счетно, то ввиду несчетности множества всех действительных чисел это означало бы, что множество всех трансцендентных чисел не только не пусто, но несчетно.

**Теорема 1**. Множество алгебраических чисел счетно.

**Доказательство.** Мы приведем видоизмененный вариант доказательства, приведенного Дедекиндом в письме Г. Кантору (1873 г.). Построение биекции между множеством алгебраических чисел $A$ и некоторым подмножеством множества рациональных чисел $Q$ (вместе с установленной выше бесконечностью множества $A$) означало бы, что $A$ счетно.

*Минимальным многочленом* для алгебраического числа $\alpha$ назовем многочлен наименьшей возможной степени с целыми коэффициентами, наименьший общий делитель которых равен 1, с положительным старшим коэффициентом и имеющий корень, равный $\alpha$. Можно доказать, что каждое алгебраическое число имеет ровно один минимальный многочлен. Из приведенных выше примеров видно, что, например, для рационального $\alpha$ степень минимального

многочлена равна 1, для $\alpha = \sqrt{2}$ степень минимального многочлена равна 2.

Количество действительных корней многочлена не больше, чем его степень, поэтому можно пронумеровать все действительные корни многочлена по возрастанию. Теперь всякое алгебраическое число $\alpha$ полностью определяется своим минимальным многочленом (т. е. набором коэффициентов минимального многочлена) и номером $k$ позиции $\alpha$ среди упорядоченных по возрастанию действительных корней этого многочлена:

$$\alpha \to (a_0, a_1, \dots, a_n, k). \qquad (1)$$

Итак, каждому алгебраическому числу $\alpha$ мы поставили в соответствие конечный набор целых чисел, причем по этому набору $\alpha$ восстанавливается однозначно. Если рассмотреть множество с элементами $(a_0, a_1, \dots, a_n, k)$ в 11-ричной системе счисления с алфавитом, который дополнительно к десяти символам десятичной системы включает и символ «запятая», то получим, что множество различных натуральных чисел – наборов вида (1) – не более чем счетно. С учетом замечания в начале доказательства отсюда заключаем, что А счетно. Теорема доказана.

**Следствие.** Т. к. множество всех действительных чисел $R$ несчетно, то множество трансцендентных чисел, дополняющее множество $A$ до $R$, несчетно.

Теорема не содержит рекомендаций, как определить, является ли данное число алгебраическим (а этот вопрос является весьма важным).

В 1882 г. немецкий математик К. Линдеман доказал, что число $\pi$ трансцендентно. Отсюда немедленно следует неразрешимость одной из знаменитых задач древности – о квадратуре круга.

### 3.2. Приложение результатов к древнейшим задачам математики

**Задача о квадратуре круга** [2]**.** При помощи циркуля и линейки построить квадрат, площадь которого равна площади заданного круга с радиусом 1.

Т.к. площадь этого круга равна $\pi$, построение искомого квадрата сводится к построению отрезка длины $\sqrt{\pi}$. Далее воспользуемся без доказательства известным результатом К. Гаусса. Если по заданному отрезку другой отрезок получен с помощью четырех арифметических операций и операции извлечения квадратного корня, говорят, что он выражается в «квадратных радикалах».

**Теорема 2** (К. Гаусс). Если задан отрезок длины 1, то с помощью циркуля и линейки можно построить такие и только такие отрезки, которые выражаются в квадратных радикалах.

Все такие числа являются алгебраическими. Например, при целых $m$ и $n$ число $\sqrt{\frac{m}{n}}$ является алгебраическим, т.к. является корнем уравнения $nx^2 - m = 0$ с целыми коэффициентами. Поскольку число $\pi$ трансцендентно, то и $\sqrt{\pi}$ трансцендентно: ведь если бы $\sqrt{\pi}$ являлось алгебраическим числом, то и число $\pi$, как произведение двух алгебраических чисел, было бы алгебраическим. Поэтому построить отрезок длины $\sqrt{\pi}$ при помощи циркуля и линейки невозможно.

Еще в древней Греции математики умели строить с помощью циркуля и линейки правильные $n$ -угольники, где

$$n = k \cdot 2^m, \qquad k = 1, 3, 5, 15;\ m = 0,1,2, \ldots$$

Подчеркнем, что здесь под линейкой понимается инструмент, который позволяет построить отрезки прямых, но не измерять длины отрезков. Задачу построения других правильных многоугольников (или доказательство невозможности таких построений) не могли решить в течение двух тысячелетий. Она была решена в 1796 г. К. Гауссом – тогда еще студентом филологического (!) факультета Геттингенского университета.

Сначала, используя первообразные корни из единицы, он доказал, что с помощью циркуля и линейки можно построить такие отрезки, длины которых выражаются в квадратных радикалах, и только их. Так как корни из единицы делят окружность на равные дуги, то задача построения правильного $n$ -угольника сводится к вопросу: при каких $n$ действительные и мнимые части корней $n$ -й степени из единицы выражаются в квадратных радикалах. Таким образом, геометрическая задача была сведена к чисто алгебраической.

В возрасте 18 лет К. Гаусс нашел способ выразить в квадратных радикалах сторону правильного 17-угольника и тем самым доказал, что с помощью циркуля и линейки можно построить правильный 17-угольник. К. Гаусс отводил этой своей юношеской работе первейшее место и завещал высечь на своей могильной плите правильный 17-угольник. Завещание не было выполнено, но в городе Брауншвейге памятник К. Гауссу стоит на 17-угольном постаменте.

Позднее Гаусс доказал, что с помощью циркуля и линейки построить правильный $n$-угольник можно тогда, когда

$$n = 2^m \cdot p_1 \cdot \ldots \cdot p_k,$$

где $m = 0, 1, 2, \ldots,$ а $p_1, p_2, \ldots, p_k$ – попарно различные простые числа вида $2^{2^i} + 1$, где $i$ – целое неотрицательное. В 1836 П. Ванцель доказал, что других правильных многоугольников, которых можно построить циркулем и линейкой, не существует.

**Замечание**. Числа вида $2^{2^i} + 1$, гд $i$ – целое неотрицательное, называются *числами Ферма*. П. Ферма ошибочно полагал, что все числа такого вида простые. Это действительно так, но … при $i = 0,\ 1,\ 2, 3, 4$ (это числа 3, 5, 17, 257, 65537). Недюжинные вычислительные способности позволили Л. Эйлеру найти нетривиальные делители числа Ферма при $i$=5: 641 и 6700417. До сих пор неизвестно, существуют ли простые числа Ферма при $i > 5$.

Упражнение. Используя «школьные» знания по программированию, убедитесь, что $2^{2^5} + 1$ имеет только два нетривиальных делителя.

Важно понять, что между доказательством существования объекта и разработкой алгоритма его построения порой пролегает трудный путь. Приведенный выше результат К. Гаусса относится только к принципиальной возможности построения правильного $n$-угольника. Конкретные реализации построения весьма трудоёмки. К. Гаусс не всегда публиковал свои достижения, и построение 17-угольника впервые было опубликовано К. Ф. фон Пфейдерером в 1802 г., а 257-угольника – М. Г. фон Пауккером в 1822 г. (по дру-

гим сведениям – Ф. Ришелло, который предложил решение на 80 страницах текста). Сообщается, что в библиотеке Геттингенского университета хранится рукопись, являющаяся итогом 10-летней работы О. Гермеса и содержащая метод построения 65537-угольника, изложенный на «необъятном» количестве страниц. По преданию, труд был предпринят по поручению научного руководителя, который отослал надоедливого аспиранта с соответствующим заданием в надежде (оказавшейся, увы, тщетной), что после этого аспирант навсегда исчезнет из поля зрения.

Приведенные сведения почерпнуты из статьи [3].

### 3.3. 10-я проблема Д. Гильберта

В предыдущем разделе мы убедились, что решение задачи о трансцендентности числа $\pi$ влечет решение знаменитой геометрической задачи. Следующий пример тесной внутренней связи между различными областями математики явится для нас ценным прологом к теме следующей лекции – уточнению понятия алгоритма.

Давид Гильберт (1862-1943).

**Определение**. Уравнение $P(x_1, x_2, \dots, x_n) = 0$, где $P$ – многочлен с целыми коэффициентами, называется *диофантовым*.

**10-я проблема Д. Гильберта**. Существует ли универсальный алгоритм проверки существования у диофантова уравнения решения в целых числах?

Результат Ю. Матиясевича (1970 г.) гласит, что 10-я проблема Д. Гильберта *алгоритмически неразрешима*. В следующей лекции нам предстоит ознакомиться с этим важным понятием.

**Литература:**

1. Болибрух А. А. Проблемы Гильберта 100 лет спустя, 2014. URL: http://www.mccme.ru/mmmf-lectures/books/books/books.php?book=2&page=3 (Дата обращения: 5 октября 2014 г.).

2. Построение правильного многоугольника с помощью циркуля и линейки, 2009. URL: http://fxdx.ru/page/postroenie-pravilnogo-mnogougolnika-s-pomoshhju-cirkulja-i-linejki/ (Дата обращения: 5 октября 2014 г.).

3. Теорема Гаусса-Ванцеля, 2014. URL: https://ru.wikipedia.org/ wiki/ (Дата обращения: 5 октября 2014 г.).

**Вопросы к лекции:**

Какое число называется алгебраическим? Приведите примеры алгебраических чисел.

Какое число называется трансцендентным? Приведите примеры.

Что можно сказать о конечности множества алгебраических чисел? А о счетности?

Докажите, что любое рациональное число $m/n$ – алгебраическое.

Докажите, что число $\sqrt{3}$ алгебраическое.

Докажите трансцендентность числа $\sqrt{\pi}$.

Что известно о счетности множества трансцендентных чисел?

Сформулируйте задачу о квадратуре круга и объясните, почему она не имеет решения.

Какое уравнение называется диофантовым?

Сформулируйте результат Ю. Матиясевича.

## Лекция 4. Машина Тьюринга и алгоритмически неразрешимые задачи

Лекция посвящена строгому определению алгоритма как программы для машины Тьюринга. Доказано утверждение о существовании алгоритмически неразрешимых задач. Использованы материалы из [1, гл. 1], [2, гл.1-2] и [3, гл. 8].

### 4.1. Описательное определение алгоритма

Обычно алгоритм определяют как конечную последовательность действий, которую должен выполнить исполнитель с целью получения определенного результата. Действия эти выполняются в соответствии с правилами, обладающими свойствами *массовости, детерминированности, результативности, дискретности* и *элементарности*. Разъясним смысл этих терминов.

*Массовость* – инвариантность алгоритма относительно входной информации. Например, алгоритм формулируется не для конкретного квадратного уравнения, а для всех уравнений вида $ax^2 + bx + c = 0$, где $a \neq 0$. *Детерминированность* – однозначность применения правил на каждом шаге; *результативность* – получение информации, являющейся результатом; *дискретность* означает, что следующий шаг начинается по времени после конца предыдущего шага; *элементарность* – отсутствие необходимости дальнейшей детализации правил.

Понятно, что описательное определение алгоритма не может соответствовать целям таких строгих наук, как математическая логика,

кибернетика и др. Уточнение понятия алгоритма возможно с помощью машины Тьюринга (для той же цели используют тезис Черча, машину Поста, алгоритм Маркова).

### 4.2. Машина Тьюринга


Алан Мэтисон Тьюринг (1912-1954).

Строгое определение алгоритма появилось в 1936 г. Английский математик А. Тьюринг описал схему абстрактной машины и предложил называть алгоритмом то, что́ умеет выполнять такая машина. При этом определении, если что-то не может быть выполнено машиной Тьюринга (МТ), это не алгоритм. Используя МТ, можно доказывать существование или отсутствие алгоритмов решения различных задач.

А.Тьюринг пришел к идее своей машины, исследуя математическую проблему, поставленную Д. Гильбертом (см. предыдущую лекцию). Проблемы математики привели к теоретической конструкции универсального вычислительного устройства и доказательству существования *алгоритмически неразрешимых* проблем.

Перечисление частей МТ начнем с бесконечной в обе стороны ленты, разбитой на ячейки. Каждая ячейка содержит символ из конечного алфавита $\{a_1, \dots, a_n, \Delta\}$, включающего «специальный» (или «пустой») символ $\Delta$.

В любой момент времени МТ находится в одном из конечного числа состояний $q_1, \ldots, q_m$. Иногда в множество состояний включают два выделенных состояния, обозначаемых $q_Y, q_N$ и называемых *терминальными*; переход в одно из этих состояний означает завершение выполнения программы. Когда задача, решаемая программой, сформулирована в форме, подразумевающей ответ «да» или «нет», достижение состояния $q_Y$ или $q_N$ соответствует ответу «да» или «нет» соответственно.

По ленте перемещается «автомат», который в каждый момент времени обозревает ровно одну ячейку и в зависимости от текущего состояния $q_i$ и обозреваемого символа $a_j$ выполняет действие из трех пунктов:

запись некоторого символа $a_k$ в обозреваемую ячейку,

переход в некоторое состояние $q_l$,

перемещение автомата на $t$ ячеек, $t \in \{-1,\ 0, +1\}$: соответственно на одну ячейку влево (-1), вправо (+1) или же на 0 (неподвижность).

Отображение $(q_i, a_j) \rightarrow (a_k, q_l, t)$ множества пар в множество троек обычно задается таблично и называется *программой для МТ*. Сначала на ленту записывается некоторое слово из непустых символов алфавита («входное слово»), по обе стороны которого ячейки до бесконечности заполнены специальным символом.

Затем выполняются действия в соответствии с программой. Работа программы может завершиться в состоянии $q_Y$ или $q_N$ (или когда

при выполнении действия автомат остается неподвижен и сохраняет символ и состояние), но программа может и зацикливаться.

### 4.3. Примеры машин Тьюринга

Пример 1. Если программа имеет вид:

| | Δ | 0 | 1 |
|---|---|---|---|
| $q_1$ | $\Delta, q_1, +1$ | $0,q_1,+1$ | $1,q_1,+1$ |

автомат будет беспрерывно передвигаться вправо и никогда не остановится.

Пример 2. Пусть программа задана следующим образом:

| | Δ | 0 | 1 |
|---|---|---|---|
| $q_1$ | $\Delta, q_1, 0$ | $\Delta,q_1,+1$ | $\Delta,q_1,+1$ |

Увидев пустую ячейку, автомат оставляет ее пустой и остается неподвижен. Увидев знак 0 или 1, стирает его и сдвигается вправо. Таким образом, он двигается вправо, опустошая ячейки, и останавливается, добравшись до первой непустой ячейки справа от «входного слова».

**Замечание.** Всюду в дальнейшем будем предполагать, что автомат находится в пределах входного слова.

Упражнение 1. Составить программу, которая прибавляет единицу к входному слову – натуральному числу. Предполагается, что в начальный момент времени автомат находится напротив самой правой цифры числа.

Схема программы может выглядеть так.

1. Пока обозреваемый символ является цифрой, то:

если цифра равна 0,…, 8, увеличить ее на 1 (здесь 1 – либо значение, добавляемое к последней цифре, либо значение, сохраненное «в уме») и перейти к пункту 2;

если это – цифра 9, то заменить ее на 0 (и тогда 1 – «в уме») и сместиться влево на одну ячейку.

2. Остановиться.

Каждый из этих двух пунктов может быть реализован одним состоянием МТ, поэтому нам понадобятся два состояния:

| | Δ | 0 | 1 | … | 8 | 9 |
|---|---|---|---|---|---|---|
| $q_1$ | 1,$q_2$,0 | 1,$q_2$,0 | 2,$q_2$,0 | | 9,$q_2$,0 | 0,$q_1$,-1 |
| $q_2$ | $\Delta, q_2, 0$ | $0, q_2, 0$ | $1, q_2, 0$ | | $8, q_2, 0$ | $9, q_2, 0$ |

Подробные записи в клетках последней строки таблицы, обозначающие завершение программы, можно было бы для краткости заменить условным обозначением, например, знаком «!». Иногда удобно данным знаком заменить некоторую часть записи в клетке таблицы.

Упражнение 2. Решите предыдущее упражнение без предположения о местонахождении автомата (но с учетом замечания, см. выше).

1. Найти самую правую ячейку входного слова. Для этого: пока обозреваемый символ не пуст, продвигаться вправо; обнаружив пустой символ, сделать шаг влево и перейти к пункту 2.

2. Пока обозреваемый символ является цифрой: при равенстве его 0,…, 8 выполнить увеличение на 1 и перейти к пункту 3; если символ равен 9, заменить на 0 и сместиться влево.

3. Остановиться.

Достаточно предусмотреть два состояния МТ:

| | $\Delta$ | 0 | 1 | … | 8 | 9 |
|---|---|---|---|---|---|---|
| $q_1$ | $\Delta$,$q_2$,-1 | 0,$q_1$,+1 | 1,$q_1$,+1 | | 8,$q_1$,+1 | 9,$q_1$,+1 |
| $q_2$ | 1,! | 1,! | 2,! | | 9,! | 0,$q_2$,$-1$ |

Упражнение 3. Задан следующий алфавит: $\{\Delta, 0, 1, \dots, 9\}$. Входное слово – некоторое натуральное число. Составьте программу его умножения на 2.

Очевидно, следует найти последний символ входного слова (состояние $q_1$) и продвигаться влево, последовательно умножая на 2 обозреваемую цифру. Будем различать

«простую замену» – замену обозреваемой цифры i на цифру (2i mod 10) – действие в состоянии $q_2$

и

«замену с увеличением» – замену обозреваемой цифры i на цифру (2i mod 10) +1 или же замену пустого символа на цифру 1 – действие в состоянии $q_3$.

Замена с увеличением требуется, если в предыдущем действии на 2 умножалась цифра из диапазона 5..9 с очевидным сохранением единицы «в уме». Следовательно, работа в состоянии $q_3$ проводит-

ся после действия над цифрами 5, 6, 7, 8 или 9 (выполненного в состоянии $q_2$ или в состоянии $q_3$).

Каждая замена цифры сопровождается сдвигом влево, а замена пустого символа завершает программу.

Напишем сказанное в виде схемы.

1. Найти правый конец слова на ленте: пока символ не пуст, продвигаться вправо; обнаружив пустой символ, сделать шаг влево и перейти к пункту 2.

2. Если обозреваемый символ *i* является цифрой 0, 1, 2, 3 или 4, выполнить обычную замену со смещением влево;

если обозреваемый символ *i* является цифрой 5, 6, 7, 8 или 9, то выполнить замену с увеличением со смещением влево и перейти к пункту 3;

если обозреваемый символ *i* является пустым, заменить его на 1 и остановиться.

3. Если обозреваемый символ *i* является цифрой, то выполнить замену с увеличением и смещением влево; при этом вернуться в состояние $q_0$, если *i* равно 0, 1, 2, 3 или 4.

Если обозреваемый символ является пустым, заменить его на 1 и остановиться.

| | Δ | 0 | 1 | … | 4 | 5 | … | 9 |
|---|---|---|---|---|---|---|---|---|
| $q_1$ | Δ,$q_2$,- | 0,$q_1$,+ | 1,$q_1$,+ | … | 4,$q_1$,+ | 5,$q_1$,+ | … | 9,$q_1$,+ |
| $q_2$ | Δ,! | 0,$q_2$,− | 2,$q_2$,− | … | 8, $q_2$,− | 0, $q_3$,− | … | 8, $q_3$,− |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $q_3$ | 1,! | $1,q_2,-$ | $3,q_2,-$ | … | $9,q_2,-$ | $1,q_3,-$ | … | $9,q_3,-$ |

Записи -1 и +1 сокращены до – и + соответственно.

Упражнение 4. Входное слово представлено набором звездочек. Нужно стереть все звездочки и написать их число в десятичной системе.

Будем формировать это число на ленте слева от звездочек. Схема программы:

1. Найти самый правый символ входного слова.

2. Если текущее слово заканчивается звездочкой (заметим, что в процессе работы во входное слово включаются и цифры формируемого числа, поэтому данное условие означает лишь то, что не все звездочки стерты), то стереть эту звездочку, в противном случае завершить работу.

3. Прибавить единицу к сформированному ранее числу и перейти к пункту 1.

В соответствии со схемой, последовательно выполняется действие по стиранию самой правой звездочки и прибавлению единицы к сформированному числу (при первом удалении звездочки выполняется запись единицы в ячейку слева от самой первой звездочки). Выполнение трех пунктов схемы программы повторяется до тех пор, пока не будет стерта последняя звездочка, после чего машина остановится (пункт 2).

Автомат может видеть на ленте цифры, написанные в процессе работы, и звездочки входного слова. В состоянии $q_1$ выполняется по-

иск самой правой ячейки слова и переход в состояние $q_2$. Находясь в состоянии $q_2$ и увидев звездочку, автомат стирает его, сдвигается влево и переходит в состояние прибавления единицы ($q_3$). Если же после перехода в состояние $q_2$ автомат видит цифру, то машина останавливается, т.к. это означает, что все звездочки уже стерты. В состоянии $q_3$ автомат двигается по ленте влево, минуя оставшиеся звездочки, пока не дойдет до числа, и прибавляет к числу единицу (аналогично тому, как это сделано в упражнении 2).

### 4.4. Самоприменимость алгоритма и алгоритмическая неразрешимость

Из соображений краткости изложение данного вопроса приведем в «популярном», нестрогом виде.

Описывая различные алгоритмы и доказывая реализуемость всевозможных композиций алгоритмов, А. Тьюринг убедительно показал необозримые возможности предложенной им конструкции. Это позволило ему выступить с тезисом, который на протяжении длительного времени вызывал ожесточенное сопротивление, но окончательно утвердился как основной тезис современной теории алгоритмов: *Всякий алгоритм может быть реализован соответствующей МТ.*

В теории алгоритмов известны задачи, относительно которых доказано, что не существуют алгоритмы для их решения. Одна из них – 10-я проблема Д. Гильберта, о которой сказано в предыдущей лекции.

Нетрудно заметить, что алфавит для МТ можно выбрать таким образом, чтобы его символами можно было записать и любое дей-

ствие в клетках таблицы, представляющей программы для МТ. Тогда вытянутые в одну линию строки таблицы (описания программы) можно рассматривать как входное слово.

**Определение**. Алгоритм называется *самоприменимым*, если, начав работу над собственным описанием, он рано или поздно останавливается. В противном случае алгоритм называется *несамоприменимым*.

**Теорема.** Задача построения алгоритма А, проверяющего самоприменимость любого заданного алгоритма, неразрешима.

**Доказательство**. Допустим противное: алгоритм А существует и применение его к произвольному алгоритму Р запишет на ленту букву Y, если Р – самоприменимый, и букву N – в противном случае:

$$A(P) = \begin{cases} Y, & \text{если P самоприменим,} \\ N, & \text{если P несамоприменим.} \end{cases}$$

Чтобы получить противоречие, нам понадобится алгоритм В, который зацикливается, если на ленте написано входное слово из единственного символа Y, и завершается, когда на ленте написано входное слово, состоящее из символа N. Запишем алгоритм В:

| | Y | N | Δ |
|---|---|---|---|
| $q_1$ | Δ,$q_1$,0 | N,$q_1$,0 | Y,$q_1$,0 |

Определим теперь алгоритм К=А*В как последовательное выполнение алгоритмов А и В. Мы допустили, что алгоритм А существует, В мы построили сами, поэтому его существование не вызывает сомнений. Докажем, что алгоритм К не может существовать, пока-

зав, что алгоритм К не является ни самоприменимым, ни несамоприменимым.

Действительно, рассмотрим применение алгоритма К к его собственной записи: сначала применяется А, затем к полученному результату (Y или N) применяется В.

1 случай. Допустим, что алгоритм К самоприменимый. Тогда

А (запись К) напишет на ленту Y. Но получив на входе букву Y, алгоритм В должен зациклиться. Значит, зацикливается композиция К=А*В, следовательно, алгоритм К оказывается несамоприменимым.

2 случай. Допустим, что алгоритм К несамоприменим, тогда

А (запись К) запишет на ленту символ N. Получив на входе N, алгоритм В остановится. Следовательно, алгоритм К является самоприменимым.

Итак, алгоритм К не может быть ни самоприменимым, ни несамоприменимым и поэтому не существует. Тогда не существует алгоритм А. Теорема доказана.

**Литература:**

1. Любимский Э. З., Мартынюк В. В., Трифонов Н. П. Программирование. – М.: Наука, Главная редакция физико-математической литературы, 1980.

2. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982.

3. Хопкрофт Д., Мотвани Р., Ульман Д. Введение в теорию автоматов, языков и вычислений – М.: Вильямс, 2002. – С. 528.

### Вопросы к лекции:

Описательное определение алгоритма.

Составные части МТ.

Программа для МТ.

Примеры программ для МТ.

Основной тезис теоремы алгоритмов.

Самоприменимый алгоритм.

Теорема о проверке самоприменимости.

## Лекция 5. Сложность алгоритмов. Полиномиальные и NP-полные задачи

Мы определим понятие входной длины задачи, которая не зависит от выбранной «разумной» схемы кодирования, и введем понятие *временно́й сложности*; дадим определение алгоритма *с полиномиальной временной сложностью* и рассмотрим класс задач NP и два ее подкласса – P (задач, разрешимых за полиномиальное время) и NPC (*NP-полных задач*, к которым полиномиально сводится любая задача из NP). В заключение сформулируем проблему тысячелетия: NP ≠ P. Использованы материалы из [1] и [2].

### 5.1. Массовая и индивидуальная задачи, входная длина

Как и в предыдущей лекции, начнем с квадратного уравнения. Если требуется найти действительные решения уравнения вида

$$ax^2 + bx + c = 0$$

с неопределенными значениями параметров *a, b, c*, то мы имеем целую «массу» задач. Присвоение всем параметрам конкретных значений приводит к «индивидуальной» задаче.

Термины «задача» и «массовая задача» рассматриваются как синонимы. Массовая задача П определяется списком всех своих параметров и формулировкой свойств, которым должно удовлетворять решение, а индивидуальная задача I получается из массовой задачи П, если всем параметрам присвоить конкретные значения.

На рис. 1 приведен пример индивидуальной задачи о коммивояжере – задаче кратчайшего обхода всех «городов» – вершин некоторого графа с заданными целочисленными расстояниями между городами.

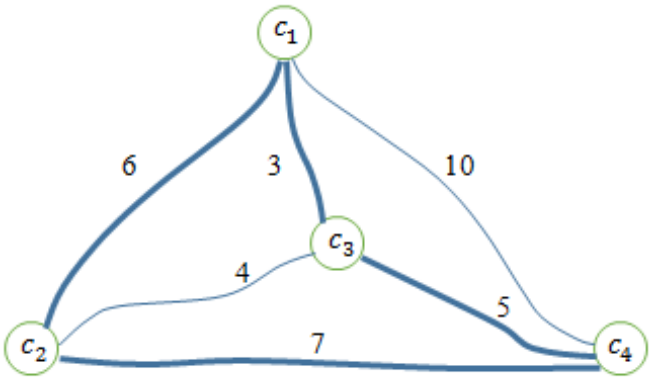



Рис. 1. Индивидуальная задача о коммивояжере и маршрут минимальной длины, равной 21.

Описание индивидуальной задачи можно передать машине Тьюринга как входное слово, составленное из символов алфавита. Например, можно выбрать алфавит {/,0,1,2,3,4,5,6,7,8,9} из 11 символов (этим уточнением мы хотим подчеркнуть, что запятая выступает в качестве разделителя, не является символом алфавита) и закодировать пример на рис. 1 в виде входного слова:

6/3/10//4/7//5.

Или же выбрать алфавит {/,0,1} и получить входное слово 110/11/1010//100/111/101.

Суть выбранной нами схемы кодирования массовой задачи П вполне разумна в том смысле, что лаконична, не содержит искусственных фрагментов, удлиняющих код. Заключается она в следующем: сначала до первой пары сдвоенных слэш перечисляются (с разделителем /) все расстояния от первого города до второго, третьего и т.д., затем до второго сдвоения слэш перечисляются расстояния от второго города до третьего, четвертого и т.д.

Более сложные индивидуальные задачи отображаются в цепочку символов входного слова для машины Тьюринга аналогичным способом.

**Определение.** Под *входной длиной* индивидуальной задачи I массовой задачи П понимается число символов во входном слове, полученном применением к индивидуальной задаче I некоторой разумной схемы кодирования для массовой задачи П.

В первом из двух приведенных выше примеров длина входа индивидуальной задачи равна 14.

**Упражнение**. Подсчитайте для выбранной выше схемы кодирования длину входа для индивидуальной задачи при $m = 10$, если известно, что расстояния между городами заданы целыми числами от 1 до 9.

Заметим, что до первой пары слэш перечислены $m - 1$ цифр, задающих соответственно расстояния от первого города до 2-го, …, $m$-го города, всего $m - 1$ цифр; разделителей на единицу меньше, т.е.

$m-2$; если учесть и две заключительные слэш этого промежутка, получим $(m-1)+(m-2)+2=2m-1$. Легко видеть, что в следующем промежутке на два символа меньше и т.д. Эта закономерность (арифметическая прогрессия) простирается вплоть до предпоследнего промежутка, т.к. заключительный промежуток содержит лишь единственную цифру – расстояние от $(m-1)$-го города до -го города.

Число членов в полученной таким образом арифметической прогрессии равно $m-2$, первый член равен $2m-1$, разность прогрессии составляет $-2$. По формуле

$$S_n = \frac{2a_1 + d(n-1)}{2} n$$

имеем:

$[2(2m-1)+(-2)(m-3)]/2 * (m-2) = (2m-1-m+3)(m-2) = (m+2)(m-2) = m^2-4$.

Остается учесть символ из последнего промежутка входного слова.

Ответ: $m^2-3$.

**Замечание**. Хотя при разных схемах кодирования входные длины задачи могут быть разными, при разумных схемах кодирования эти длины связаны «полиномиальным образом»: если входная длина при одной разумной схеме кодирования равна $n$, при другой может равняться, например, $n^2+n$, но не $2^n$ или $3^n$.

## 5.2. Временна́я сложность алгоритма, полиномиальные алгоритмы и труднорешаемые задачи

Понятно, что программист выбирает для решения своей задачи самый эффективный алгоритм.

Подсчитаем, сколько времени потребуется для переборного решения задачи о коммивояжере в случае $m = 100$. Маршрутов, начинающихся с 1-го города, всего $m - 1$; каждый из них может быть продолжен $m - 2$ способами. Таким образом, 2-звеньевых маршрутов с началом в городе 1 всего $(m - 1)(m - 2)$. Продолжая этот процесс, получим, что различных $(m - 1)$-звеньевых маршрутов (т.е. маршрутов обхода всех $m$ городов) всего

$$(m - 1)(m - 2) \ \ldots \ (m - (m - 1)) = (m - 1)!$$

Для персонального компьютера, способного просчитать 1 трлн. маршрутов в 1 сек., получим следующую продолжительность времени, исчисляемого в млрд. лет (напишем в виде команды для системы компьютерной математики Mathematica):

$100!/(10\text{^}12 * 3600 * 24 * 365 * 1000000000)$,

что равно $2.9 \times 10^{128}$ (млрд. лет).

Перефразируя слова героини известного произведения Л. Кэрролла скажем, что такие числа «наводят на мысли, но непонятно, на какие именно» (Льюис Кэрролл – английский писатель, математик, логик и философ. Настоящее имя – Чарльз Лютвидж Доджсон).

В широком смысле понятие эффективности связано со всеми вычислительными ресурсами компьютера, но на практике обычно эф-

фективность связывают с быстротой алгоритма: «самый эффективный – самый быстрый».

**Определение**. Временна́я сложность алгоритма решения массовой задачи – это функция, которая каждой входной длине $n$ ставит в соответствие максимальное (по всем индивидуальным задачам с длиной входа $n$) время, затрачиваемое алгоритмом на решение индивидуальных задач этой длины.

Говоря о времени, здесь и далее подразумевается, что на одну операцию машина Тьюринга тратит одну единицу времени.

**Обозначение**. Если для функции $f(n)$ существует константа $c$, такая, что

$$|f(n)| \leq c|g(n)|,$$

то будем писать $f(n) = O(g(n))$.

**Определение**. *Полиномиальным алгоритмом* (или *алгоритмом полиномиальной временно́й сложности*) называется алгоритм, у которого временна́я сложность равна $O(p(n))$, где $p(n)$ – некоторая полиномиальная функция, а $n$ – входная длина.

**Определение.** Задача называется *труднорешаемой*, если для ее решения не существует полиномиального алгоритма.

Понятие труднорешаемости оказывается, по существу, независимой от конкретной схемы кодирования и выбранной модели ЭВМ, используемых при определении временно́й сложности.

Временна́я сложность определена нами как мера поведения алгоритма в наихудшем случае. Например, утверждение, что алгоритм

имеет временну́ю сложность порядка $2^n$, надо понимать так, что по крайней мере одна индивидуальная задача с входной длиной $n$ потребует времени порядка $2^n$. В то же время может оказаться, что большинство индивидуальных задач (или даже – все индивидуальные задачи, кроме одной) довольствуется значительно меньшими затратами времени.

### 5.3. Класс NP и NP-полные задачи

Рассмотрим класс задач, для решения которых полиномиального алгоритма может и не существовать, но если решение нам известно (например, мы его угадали), то проверить его правильность возможно за полиномиальное время. Этот класс обозначим NP.

Обозначение NP берет происхождение от слов ***N****ondeterminictic* ***P****olynomial*. Объясняется это тем, что в отличие от обычной, «детерминированной» машины, машина Тьюринга, снабженная дополнительным модулем, способным угадывать ответ, называется *недетерминированной машиной Тьюринга*.

Вот некоторые примеры задач из класса NP: коммивояжер, самый длинный путь, самый короткий путь, разбиение множества целых чисел на два подмножества с равными суммами. Интересно, что упомянутая задача о кратчайшем пути принципиально отличается от других перечисленных задач в некотором, весьма важном для нас смысле. Чтобы понять это обстоятельство, необходимо ввести следующее важное понятие.

**Определение.** Задача из класса NP называется *NP-полной*, если любая задача класса NP допускает сведение к ней за полиномиальное

время, другими словами, – сведение с помощью алгоритма, выполняемого за полиномиальное время.

Уже сам факт существования хотя бы одной NP-полной задачи представляется замечательным результатом. Но … NP-полных задач оказывается «слишком много». В частности, из перечисленных выше задач все задачи, за исключением задачи о кратчайшем пути, являются NP-полными.

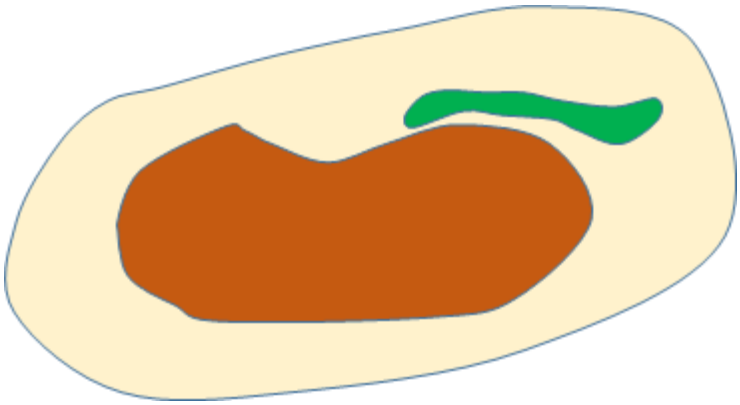


Рис. 2. Схема классов NP, P и NPC, приведенная в [1].

Чем это плохо? Ни для одной NP-полной задачи до сих пор не удалось отыскать полиномиальный алгоритм решения. И это – несмотря на то, что на поиски таких алгоритмов было затрачено в сотни раз больше усилий, чем на доказательство теоремы Ферма. Данное утверждение заимствовано из книги [1].

Внешний контур на рис. 2 ограничивает класс NP. Бóльший из вложенных контуров является символическим обозначением класса NP-полных задач (NPC), меньший – обозначением класса задач (P), разрешимых за полиномиальное время. Неизвестно, пересекаются ли эти два класса: P и NPC. Неизвестно также, содержит ли класс NP задачи, отличные от P и NPC. Если окажется, что какая-либо из

NP-полных задач решается за полиномиальное время, то это будет означать, что и все остальные задачи из этого класса эффективно разрешимы.

**Стивен Артур Кук** (г.р. 1939), лауреат премии Тьюринга.

В настоящее время принято считать, что NP-полные задачи за полиномиальное время решить нельзя, то есть $NP \neq P$. Из сказанного о классе NP видно, что проблему $NP \neq P$ в упрощенной формулировке можно трактовать так: если проверка решения задачи является простой процедурой, следует ли отсюда, что и решение задачи тоже является простой процедурой?

Для большинства задач из класса NP (а таких задач сформулировано несколько тысяч) либо уже построены эффективные алгоритмы решения, либо доказана их NP-полнота. Тем не менее, и в данном вопросе существуют исключения. Так, сложность задачи о дискретном логарифмировании или задачи о составном числе до сих пор не известна.

Основы теории NP-полных задач были заложены в работе С. Кука [2], опубликованной в 1971 г.

**Литература:**

1. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982.

2. Cook S. A. The complexity of theorem-proving procedures / Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, New York, 1971. – P. 151-158.

**Вопросы к лекции:**

Массовая и индивидуальная задачи. Примеры.

Формулировка задачи о коммивояжере.

Пример входного кода индивидуальной задачи о коммивояжере.

Независимость длины входного кода от выбора разумной схемы кодирования.

Временна́я сложность алгоритма.

Алгоритм полиномиальной временно́й сложности.

Труднорешаемая задача.

NP-полная задача.

Примеры NP-полных задач.

Схема классов NP, P и NPC.

Гипотеза NP ≠ P.