

# Математическая логика и теория алгоритмов

## Лекция 9

### Введение в теорию алгоритмов

Куценко Дмитрий Александрович

Белгородский государственный технологический университет  
имени В. Г. Шухова

Институт информационных технологий и управляющих систем

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

11 апреля 2013 г.

# Исторический ракурс

До середины XIX в. математика имела дело в основном с числами и вычислениями (исключение — геометрия).

# Исторический ракурс

До середины XIX в. математика имела дело в основном с числами и вычислениями (исключение — геометрия). Понятие алгоритма отождествлялось с понятием метода вычисления.

# Исторический ракурс

До середины XIX в. математика имела дело в основном с числами и вычислениями (исключение — геометрия).

Понятие алгоритма отождествлялось с понятием метода вычисления.

При этом понятие метода вычисления считалось изначально ясным и не нуждалось в специальных исследованиях.

# Исторический ракурс

До середины XIX в. математика имела дело в основном с числами и вычислениями (исключение — геометрия). Понятие алгоритма отождествлялось с понятием метода вычисления.

При этом понятие метода вычисления считалось изначально ясным и не нуждалось в специальных исследованиях.

Во 2-й половине XIX в. появились новые разделы математики, оперирующие с нечисловыми объектами (теория множеств, теория групп, неевклидовы геометрии и т. д.).

# Исторический ракурс

До середины XIX в. математика имела дело в основном с числами и вычислениями (исключение — геометрия). Понятие алгоритма отождествлялось с понятием метода вычисления.

При этом понятие метода вычисления считалось изначально ясным и не нуждалось в специальных исследованиях.

Во 2-й половине XIX в. появились новые разделы математики, оперирующие с нечисловыми объектами (теория множеств, теория групп, неевклидовы геометрии и т. д.).

Было показано, что некоторые вполне естественные рассуждения в рамках этих теорий приводят к неразрешимым противоречиям (например, парадоксы теории множеств).

# Исторический ракурс

До середины XIX в. математика имела дело в основном с числами и вычислениями (исключение — геометрия). Понятие алгоритма отождествлялось с понятием метода вычисления.

При этом понятие метода вычисления считалось изначально ясным и не нуждалось в специальных исследованиях.

Во 2-й половине XIX в. появились новые разделы математики, оперирующие с нечисловыми объектами (теория множеств, теория групп, неевклидовы геометрии и т. д.).

Было показано, что некоторые вполне естественные рассуждения в рамках этих теорий приводят к неразрешимым противоречиям (например, парадоксы теории множеств).

Всё это потребовало точного изучения принципов математических рассуждений и привело к появлению формальных систем и теории алгоритмов.

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г.



# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма.

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты понятия алгоритма были разработаны в 30-х годах XX века:

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты понятия алгоритма были разработаны в 30-х годах XX века:

- Алан Тьюринг предложил [машину Тьюринга](#).

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты понятия алгоритма были разработаны в 30-х годах XX века:

- Алан Тьюринг предложил [машину Тьюринга](#).
- Эмиль Пост — [машину Поста](#).

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты понятия алгоритма были разработаны в 30-х годах XX века:

- Алан Тьюринг предложил **машину Тьюринга**.
- Эмиль Пост — **машину Поста**.
- Алонзо Чёрч —  **$\lambda$ -исчисление Чёрча**.

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты понятия алгоритма были разработаны в 30-х годах XX века:

- Алан Тьюринг предложил **машину Тьюринга**.
- Эмиль Пост — **машину Поста**.
- Алонзо Чёрч —  **$\lambda$ -исчисление Чёрча**.
- Стивен Клини — **рекурсивные функции**.

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты понятия алгоритма были разработаны в 30-х годах XX века:

- Алан Тьюринг предложил **машину Тьюринга**.
- Эмиль Пост — **машину Поста**.
- Алонзо Чёрч —  **$\lambda$ -исчисление Чёрча**.
- Стивен Клини — **рекурсивные функции**.

Они оказались эквивалентными друг другу.

# Появление теории алгоритмов

Развитие теории алгоритмов начинается с доказательства Гёделем теорем о неполноте формальных систем, включающих арифметику, первая из которых была доказана в 1931 г. Возникшее тогда предположение об отсутствии алгоритмов решения многих математических проблем вызвало необходимость стандартизации понятия алгоритма. Первые стандартизованные варианты понятия алгоритма были разработаны в 30-х годах XX века:

- Алан Тьюринг предложил **машину Тьюринга**.
- Эмиль Пост — **машину Поста**.
- Алонзо Чёрч —  **$\lambda$ -исчисление Чёрча**.
- Стивен Клини — **рекурсивные функции**.

Они оказались эквивалентными друг другу.

Одним из наиболее удачных стандартизованных вариантов алгоритма является введённое в конце 1940-х гг.

А. А. Марковым понятие **нормального алгоритма**.



# Лица



Алан Матисон Тьюринг  
(1912—1954) — английский  
математик, логик,  
криптограф



Эмиль Леон Пост  
(1897—1954) — американский  
математик и логик



Алонзо Чёрч (1903—1995) —  
американский математик  
и логик



Стивен Коул К्लीни (Клёйни)  
(1909—1994) — американский  
математик и логик



Андрей Андреевич Марков  
(младший) (1903—1979) —  
советский математик

# Теория алгоритмов. Определение. Задачи

**Теория алгоритмов** — наука, изучающая общие свойства и закономерности алгоритмов и разнообразные формальные модели их представления.

# Теория алгоритмов. Определение. Задачи

**Теория алгоритмов** — наука, изучающая общие свойства и закономерности алгоритмов и разнообразные формальные модели их представления.

**Задачи теории алгоритмов:**

# Теория алгоритмов. Определение. Задачи

**Теория алгоритмов** — наука, изучающая общие свойства и закономерности алгоритмов и разнообразные формальные модели их представления.

**Задачи теории алгоритмов:**

- 1 формальное доказательство алгоритмической неразрешимости задач;

# Теория алгоритмов. Определение. Задачи

**Теория алгоритмов** — наука, изучающая общие свойства и закономерности алгоритмов и разнообразные формальные модели их представления.

**Задачи теории алгоритмов:**

- 1 формальное доказательство алгоритмической неразрешимости задач;
- 2 асимптотический анализ сложности алгоритмов;

# Теория алгоритмов. Определение. Задачи

**Теория алгоритмов** — наука, изучающая общие свойства и закономерности алгоритмов и разнообразные формальные модели их представления.

**Задачи теории алгоритмов:**

- 1 формальное доказательство алгоритмической неразрешимости задач;
- 2 асимптотический анализ сложности алгоритмов;
- 3 классификация алгоритмов в соответствии с классами сложности;

# Теория алгоритмов. Определение. Задачи

**Теория алгоритмов** — наука, изучающая общие свойства и закономерности алгоритмов и разнообразные формальные модели их представления.

**Задачи теории алгоритмов:**

- 1 формальное доказательство алгоритмической неразрешимости задач;
- 2 асимптотический анализ сложности алгоритмов;
- 3 классификация алгоритмов в соответствии с классами сложности;
- 4 разработка критериев сравнительной оценки качества алгоритмов и т. п.

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:



# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

Данное описание весьма расплывчато:

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

Данное описание весьма расплывчато:

- Какими средствами может выражаться «рецепт»?

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

Данное описание весьма расплывчато:

- Какими средствами может выражаться «рецепт»?
- Кто судит об «однозначности»?

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

Данное описание весьма расплывчато:

- Какими средствами может выражаться «рецепт»?
- Кто судит об «однозначности»?
- Является ли тот или иной процесс алгоритмом?

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

Данное описание весьма расплывчато:

- Какими средствами может выражаться «рецепт»?
- Кто судит об «однозначности»?
- Является ли тот или иной процесс алгоритмом?

Можно дать и такое определение, снимающее эти вопросы:

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

Данное описание весьма расплывчато:

- Какими средствами может выражаться «рецепт»?
- Кто судит об «однозначности»?
- Является ли тот или иной процесс алгоритмом?

Можно дать и такое определение, снимающее эти вопросы:

**Алгоритм** — это программа на любом универсальном языке программирования (например, на «Паскале»).

# Алгоритм

Рассмотрим следующее **интуитивное понятие алгоритма**:

**Алгоритм** — рецепт достижения результата с помощью однозначной последовательности действий.

Данное описание весьма расплывчато:

- Какими средствами может выражаться «рецепт»?
- Кто судит об «однозначности»?
- Является ли тот или иной процесс алгоритмом?

Можно дать и такое определение, снимающее эти вопросы:

**Алгоритм** — это программа на любом универсальном языке программирования (например, на «Паскале»).

Тогда критерием алгоритмичности процесса является возможность его запрограммировать.



# Общие требования к алгоритмам

- 1 **Дискретность** — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов (каждый шаг выполняется за конечный отрезок времени).

# Общие требования к алгоритмам

- 1 **Дискретность** — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов (каждый шаг выполняется за конечный отрезок времени).
- 2 **Детерминированность** — в каждый момент времени следующий шаг работы однозначно определяется состоянием системы.

# Общие требования к алгоритмам

- 1 **Дискретность** — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов (каждый шаг выполняется за конечный отрезок времени).
- 2 **Детерминированность** — в каждый момент времени следующий шаг работы однозначно определяется состоянием системы.
- 3 **Понятность** — алгоритм должен включать только те команды, которые может выполнить исполнитель.

# Общие требования к алгоритмам

- 1 **Дискретность** — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов (каждый шаг выполняется за конечный отрезок времени).
- 2 **Детерминированность** — в каждый момент времени следующий шаг работы однозначно определяется состоянием системы.
- 3 **Понятность** — алгоритм должен включать только те команды, которые может выполнить исполнитель.
- 4 **Конечность** — при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.

# Общие требования к алгоритмам

- ❶ **Дискретность** — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов (каждый шаг выполняется за конечный отрезок времени).
- ❷ **Детерминированность** — в каждый момент времени следующий шаг работы однозначно определяется состоянием системы.
- ❸ **Понятность** — алгоритм должен включать только те команды, которые может выполнить исполнитель.
- ❹ **Конечность** — при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.
- ❺ **Массовость** — алгоритм должен быть применим к разным наборам исходных данных.

# Общие требования к алгоритмам

- ❶ **Дискретность** — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов (каждый шаг выполняется за конечный отрезок времени).
- ❷ **Детерминированность** — в каждый момент времени следующий шаг работы однозначно определяется состоянием системы.
- ❸ **Понятность** — алгоритм должен включать только те команды, которые может выполнить исполнитель.
- ❹ **Конечность** — при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.
- ❺ **Массовость** — алгоритм должен быть применим к разным наборам исходных данных.
- ❻ **Результативность** — алгоритм должен завершаться определёнными результатами.

# Данные

Заметим, что любой алгоритм применяется к **исходным (входным) данным** и выдаёт **результат (выходные данные)**, а в процессе работы алгоритма появляются **промежуточные данные**.

# Данные

Заметим, что любой алгоритм применяется к **исходным (входным) данным** и выдаёт **результат (выходные данные)**, а в процессе работы алгоритма появляются **промежуточные данные**.

Если мы собираемся уточнить понятие алгоритма, то необходимо уточнить понятие **данных**, т. е. указать, каким требованиям они должны удовлетворять, чтобы алгоритмы могли с ними работать.



# Данные

Заметим, что любой алгоритм применяется к **исходным (входным) данным** и выдаёт **результат (выходные данные)**, а в процессе работы алгоритма появляются **промежуточные данные**.

Если мы собираемся уточнить понятие алгоритма, то необходимо уточнить понятие **данных**, т. е. указать, каким требованиям они должны удовлетворять, чтобы алгоритмы могли с ними работать.

По аналогии с формальными теориями в теории алгоритмов фиксируют конкретные конечные наборы исходных **объектов** (называемых **элементарными**) и конечный набор **средств построения** других объектов из элементарных.

# Алфавит и средства построения

Набор элементарных объектов образует конечный **алфавит** исходных **символов** (букв, цифр и т. д.), из которых образуются другие объекты.

# Алфавит и средства построения

Набор элементарных объектов образует конечный **алфавит** исходных **символов** (букв, цифр и т. д.), из которых образуются другие объекты.

Конечная (возможно, пустая) последовательность символов алфавита называется **словом**.

# Алфавит и средства построения

Набор элементарных объектов образует конечный **алфавит** исходных **символов** (букв, цифр и т. д.), из которых образуются другие объекты.

Конечная (возможно, пустая) последовательность символов алфавита называется **словом**.

Типичным средством построения являются **индуктивные определения**, указывающие, как строить новые объекты из уже построенных.

# Алфавит и средства построения

Набор элементарных объектов образует конечный **алфавит** исходных **символов** (букв, цифр и т. д.), из которых образуются другие объекты.

Конечная (возможно, пустая) последовательность символов алфавита называется **словом**.

Типичным средством построения являются **индуктивные определения**, указывающие, как строить новые объекты из уже построенных.

Как правило, не всякое слово является допустимым исходным данным для алгоритма.

# Алфавит и средства построения

Набор элементарных объектов образует конечный **алфавит** исходных **символов** (букв, цифр и т. д.), из которых образуются другие объекты.

Конечная (возможно, пустая) последовательность символов алфавита называется **словом**.

Типичным средством построения являются **индуктивные определения**, указывающие, как строить новые объекты из уже построенных.

Как правило, не всякое слово является допустимым исходным данным для алгоритма.

В этом случае обычно основному алгоритму предшествуют вспомогательные алгоритмы, проверяющие допустимость исходных данных.

# Алфавит и средства построения

Набор элементарных объектов образует конечный **алфавит** исходных **символов** (букв, цифр и т. д.), из которых образуются другие объекты.

Конечная (возможно, пустая) последовательность символов алфавита называется **словом**.

Типичным средством построения являются **индуктивные определения**, указывающие, как строить новые объекты из уже построенных.

Как правило, не всякое слово является допустимым исходным данным для алгоритма.

В этом случае обычно основному алгоритму предшествуют вспомогательные алгоритмы, проверяющие допустимость исходных данных.

Такая проверка называется **синтаксическим анализом**.

# Память

Данные для своего размещения требуют **памяти**.



# Память

Данные для своего размещения требуют **памяти**.

Память обычно считается **однородной** и **дискретной**, т. е. состоит из одинаковых ячеек, каждая из которых может содержать один символ алфавита данных.

# Память

Данные для своего размещения требуют **памяти**.

Память обычно считается **однородной** и **дискретной**, т. е. состоит из одинаковых ячеек, каждая из которых может содержать один символ алфавита данных.

При этом память может быть бесконечной.

# Вычислимые функции

Если существует алгоритм для вычисления значений числовой функции, то эта функция называется **эффективно вычислимой**, или **алгоритмически вычислимой**, или просто **вычислимой**.

# Вычислимые функции

Если существует алгоритм для вычисления значений числовой функции, то эта функция называется **эффективно вычислимой**, или **алгоритмически вычислимой**, или просто **вычислимой**.

Т. о. если входные и выходные данные алгоритма представляют собой числа (обычно имеются в виду целые неотрицательные), то алгоритм определяет вычислимую функцию.

# Вычислимые функции

Если существует алгоритм для вычисления значений числовой функции, то эта функция называется **эффективно вычислимой**, или **алгоритмически вычислимой**, или просто **вычислимой**.

Т. о. если входные и выходные данные алгоритма представляют собой числа (обычно имеются в виду целые неотрицательные), то алгоритм определяет вычислимую функцию.

Напомним, что алгоритмы, которые сводят решение поставленной задачи к арифметическим действиям над числами, называют **численными**.

# Вычислимые функции

Если существует алгоритм для вычисления значений числовой функции, то эта функция называется **эффективно вычислимой**, или **алгоритмически вычислимой**, или просто **вычислимой**.

Т. о. если входные и выходные данные алгоритма представляют собой числа (обычно имеются в виду целые неотрицательные), то алгоритм определяет вычислимую функцию.

Напомним, что алгоритмы, которые сводят решение поставленной задачи к арифметическим действиям над числами, называют **численными**.

Доказано, что все алгоритмы могут быть сведены к определениям вычислимых функций.

# Вычислимые функции

Если существует алгоритм для вычисления значений числовой функции, то эта функция называется **эффективно вычислимой**, или **алгоритмически вычислимой**, или просто **вычислимой**.

Т. о. если входные и выходные данные алгоритма представляют собой числа (обычно имеются в виду целые неотрицательные), то алгоритм определяет вычислимую функцию.

Напомним, что алгоритмы, которые сводят решение поставленной задачи к арифметическим действиям над числами, называют **численными**.

Доказано, что все алгоритмы могут быть сведены к определениям вычислимых функций.

Также отметим, что одна и та же вычислимая функция может определяться различными алгоритмами.

# Вычислимые функции

Если существует алгоритм для вычисления значений числовой функции, то эта функция называется **эффективно вычислимой**, или **алгоритмически вычислимой**, или просто **вычислимой**.

Т. о. если входные и выходные данные алгоритма представляют собой числа (обычно имеются в виду целые неотрицательные), то алгоритм определяет вычислимую функцию.

Напомним, что алгоритмы, которые сводят решение поставленной задачи к арифметическим действиям над числами, называют **численными**.

Доказано, что все алгоритмы могут быть сведены к определениям вычислимых функций.

Также отметим, что одна и та же вычислимая функция может определяться различными алгоритмами.

Таким образом,

**Вычислимая функция** — это множество алгоритмов, дающих на любом допустимом входном слове  $n$  один и тот же результат.



# Неопределённые вычислимые функции

Введённое определение алгоритма как программы на языке программирования допускает **любые программы**, в том числе синтаксически неправильные, закливающие на всех или некоторых данных.

# Неопределённые вычислимые функции

Введённое определение алгоритма как программы на языке программирования допускает **любые программы**, в том числе синтаксически неправильные, зацкливающиеся на всех или некоторых данных.

Поэтому среди вычислимых функций есть **неопределённые** на всех или каких-либо аргументах.

# Неопределённые вычислимые функции

Введённое определение алгоритма как программы на языке программирования допускает **любые программы**, в том числе синтаксически неправильные, зацкливающиеся на всех или некоторых данных.

Поэтому среди вычислимых функций есть **неопределённые** на всех или каких-либо аргументах.

Неопределённость  $f(n) = ?$  возникает в двух случаях:

# Неопределённые вычислимые функции

Введённое определение алгоритма как программы на языке программирования допускает **любые программы**, в том числе синтаксически неправильные, закливающиеся на всех или некоторых данных.

Поэтому среди вычислимых функций есть **неопределённые** на всех или каких-либо аргументах.

Неопределённость  $f(n) = ?$  возникает в двух случаях:

- 1 алгоритм работает безостановочно;

# Неопределённые вычислимые функции

Введённое определение алгоритма как программы на языке программирования допускает **любые программы**, в том числе синтаксически неправильные, зацкливающиеся на всех или некоторых данных.

Поэтому среди вычислимых функций есть **неопределённые** на всех или каких-либо аргументах.

Неопределённость  $f(n) = ?$  возникает в двух случаях:

- 1 алгоритм работает безостановочно;
- 2 алгоритм останавливается, не зная, что дальше делать.

# Неопределённые вычислимые функции

Введённое определение алгоритма как программы на языке программирования допускает **любые программы**, в том числе синтаксически неправильные, зацкливающиеся на всех или некоторых данных.

Поэтому среди вычислимых функций есть **неопределённые** на всех или каких-либо аргументах.

Неопределённость  $f(n) = ?$  возникает в двух случаях:

- 1 алгоритм работает безостановочно;
- 2 алгоритм останавливается, не зная, что дальше делать.

Эти ситуации могут нарушить требование результативности, поэтому их нужно как-то предусмотреть.

# Описание, механизм и процесс реализации алгоритма

Необходимо различать:

- 1 описание алгоритма (инструкцию или программу);

# Описание, механизм и процесс реализации алгоритма

Необходимо различать:

- 1 **описание алгоритма** (инструкцию или программу);
- 2 **механизм реализации алгоритма** (например, компьютер), включающий средства пуска, остановки, реализации элементарных шагов, управления ходом вычисления и выдачи результатов;



# Описание, механизм и процесс реализации алгоритма

Необходимо различать:

- ❶ **описание алгоритма** (инструкцию или программу);
- ❷ **механизм реализации алгоритма** (например, компьютер), включающий средства пуска, остановки, реализации элементарных шагов, управления ходом вычисления и выдачи результатов;
- ❸ **процесс реализации алгоритма**, т. е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным данным.

# Описание, механизм и процесс реализации алгоритма

Необходимо различать:

- ❶ **описание алгоритма** (инструкцию или программу);
- ❷ **механизм реализации алгоритма** (например, компьютер), включающий средства пуска, остановки, реализации элементарных шагов, управления ходом вычисления и выдачи результатов;
- ❸ **процесс реализации алгоритма**, т. е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным данным.

Будем предполагать, что описание и механизм реализации алгоритма **конечны** (память может быть бесконечной, но она не включается в механизм).

# Описание, механизм и процесс реализации алгоритма

Необходимо различать:

- ❶ **описание алгоритма** (инструкцию или программу);
- ❷ **механизм реализации алгоритма** (например, компьютер), включающий средства пуска, остановки, реализации элементарных шагов, управления ходом вычисления и выдачи результатов;
- ❸ **процесс реализации алгоритма**, т. е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным данным.

Будем предполагать, что описание и механизм реализации алгоритма **конечны** (память может быть бесконечной, но она не включается в механизм).

Требование конечности реализации совпадает с требованием результативности.

# Пример 1

Рассмотрим функцию

# Пример 1

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если десятичная запись числа } \pi \text{ содержит} \\ & \text{ровно } n \text{ подряд идущих нулей;} \\ 0, & \text{в противном случае.} \end{cases}$$

# Пример 1

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если десятичная запись числа } \pi \text{ содержит} \\ & \text{ровно } n \text{ подряд идущих нулей;} \\ 0, & \text{в противном случае.} \end{cases}$$

Алгоритм реализации этой функции заключается в последовательном вычислении цифр десятичного разложения числа

$\pi = 3,141592653589793238462643383279502884197169399 \dots$

# Пример 1

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если десятичная запись числа } \pi \text{ содержит} \\ & \text{ровно } n \text{ подряд идущих нулей;} \\ 0, & \text{в противном случае.} \end{cases}$$

Алгоритм реализации этой функции заключается в последовательном вычислении цифр десятичного разложения числа

$\pi = 3,141592653589793238462643383279502884197169399 \dots$

Если на каком-то шаге появляется  $n$  соседних нулей, процесс останавливается и  $f(n) = 1$ .

# Пример 1

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если десятичная запись числа } \pi \text{ содержит} \\ & \text{ровно } n \text{ подряд идущих нулей;} \\ 0, & \text{в противном случае.} \end{cases}$$

Алгоритм реализации этой функции заключается в последовательном вычислении цифр десятичного разложения числа

$\pi = 3,141592653589793238462643383279502884197169399 \dots$

Если на каком-то шаге появляется  $n$  соседних нулей, процесс останавливается и  $f(n) = 1$ .

Но если  $f(n) = 0$ , то вычислительный процесс никогда не остановится.



# Пример 1

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если десятичная запись числа } \pi \text{ содержит} \\ & \text{ровно } n \text{ подряд идущих нулей;} \\ 0, & \text{в противном случае.} \end{cases}$$

Алгоритм реализации этой функции заключается в последовательном вычислении цифр десятичного разложения числа

$\pi = 3,141592653589793238462643383279502884197169399 \dots$

Если на каком-то шаге появляется  $n$  соседних нулей, процесс останавливается и  $f(n) = 1$ .

Но если  $f(n) = 0$ , то вычислительный процесс никогда не остановится.

Поэтому неясно, вычислима эта  $f(n)$  или нет.

## Пример 2

Рассмотрим функцию

## Пример 2

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если } x^n + y^n = z^n \text{ разрешимо в целых числах;} \\ 0, & \text{в противном случае,} \end{cases}$$

## Пример 2

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если } x^n + y^n = z^n \text{ разрешимо в целых числах;} \\ 0, & \text{в противном случае,} \end{cases}$$

Поскольку великая теорема Ферма доказана (в 1995 г.), то эта функция вычислима и  $f(n) = 0$  при  $n > 2$ .

## Пример 2

Рассмотрим функцию

$$f(n) = \begin{cases} 1, & \text{если } x^n + y^n = z^n \text{ разрешимо в целых числах;} \\ 0, & \text{в противном случае,} \end{cases}$$

Поскольку великая теорема Ферма доказана (в 1995 г.), то эта функция вычислима и  $f(n) = 0$  при  $n > 2$ .

Но точнее сказать, что

$$f(n) = \begin{cases} 1, & \text{если } n \leq 2; \\ 0, & \text{если } n > 2, \end{cases}$$

не упоминая никакую теорему Ферма.

# Перечислимые множества

Одна из задач вычислительного характера — перечисление элементов множества.

# Перечислимые множества

Одна из задач вычислительного характера — перечисление элементов множества.

Множество считается **перечислимым**, если существует алгоритм порождения его элементов.

# Перечислимые множества

Одна из задач вычислительного характера — перечисление элементов множества.

Множество считается **перечислимым**, если существует алгоритм порождения его элементов.

Множество считается **разрешимым**, если существует алгоритм для выяснения принадлежности любого элемента  $n$  этому множеству. Говорят также, что разрешимое множество **распознаваемо**.



# Перечислимые множества

Одна из задач вычислительного характера — перечисление элементов множества.

Множество считается **перечислимым**, если существует алгоритм порождения его элементов.

Множество считается **разрешимым**, если существует алгоритм для выяснения принадлежности любого элемента  $n$  этому множеству. Говорят также, что разрешимое множество **распознаваемо**.

Иначе говоря, если есть такой алгоритм, который для данного  $x$  выдаёт 1 в случае, когда  $x \in X$ , то множество  $X$  является перечислимым (при этом, если  $x \notin X$ , то алгоритм может и не завершиться).

# Перечислимые множества

Одна из задач вычислительного характера — перечисление элементов множества.

Множество считается **перечислимым**, если существует алгоритм порождения его элементов.

Множество считается **разрешимым**, если существует алгоритм для выяснения принадлежности любого элемента  $n$  этому множеству. Говорят также, что разрешимое множество **распознаваемо**.

Иначе говоря, если есть такой алгоритм, который для данного  $x$  выдаёт 1 в случае, когда  $x \in X$ , то множество  $X$  является перечислимым (при этом, если  $x \notin X$ , то алгоритм может и не завершиться).

А если есть такой алгоритм, который для данного  $x$  выдаёт 1 в случае, когда  $x \in X$ , и 0, когда  $x \notin X$ , то множество  $X$  является разрешимым.

## Перечислимые множества (продолжение)

Таким образом, любое конечное множество разрешимо.

## Перечислимые множества (продолжение)

Таким образом, любое конечное множество разрешимо.

Можно ввести более строгие определения:

## Перечислимые множества (продолжение)

Таким образом, любое конечное множество разрешимо.

Можно ввести более строгие определения:

Множество  $X$  **перечислимо**, если оно есть область значений либо область определения вычислимой функции.

## Перечислимые множества (продолжение)

Таким образом, любое конечное множество разрешимо.

Можно ввести более строгие определения:

Множество  $X$  **перечислимо**, если оно есть область значений либо область определения вычислимой функции.

Множество  $X$  **разрешимо**, если его характеристическая функция вычислима.

# Теоремы о разрешимости и перечислимости

## Теорема Поста

Для разрешимости  $X$  необходимо и достаточно, чтобы  $X$  и его дополнение  $X^c$  были перечислимы.

# Теоремы о разрешимости и перечислимости

## Теорема Поста

Для разрешимости  $X$  необходимо и достаточно, чтобы  $X$  и его дополнение  $X^c$  были перечислимы.

## Теорема

Существует перечислимое, но неразрешимое множество положительных целых чисел.



# Теоремы о разрешимости и перечислимости

## Теорема Поста

Для разрешимости  $X$  необходимо и достаточно, чтобы  $X$  и его дополнение  $X^c$  были перечислимы.

## Теорема

Существует перечислимое, но неразрешимое множество положительных целых чисел.

## Утверждение

Для вычислимости  $f(x)$  необходимо и достаточно перечислимости множества пар  $(x, f(x))$ .

# Эффективно вычислимые функции

Аналогом разрешимого множества является понятие **эффективно вычислимой функции**  $f(n)$ , которая вычислима и определена при любом  $n$ .

# Эффективно вычислимые функции

Аналогом разрешимого множества является понятие **эффективно вычислимой функции**  $f(n)$ , которая вычислима и определена при любом  $n$ .

## Теорема

Множество эффективно вычислимых функций неперечислимо.