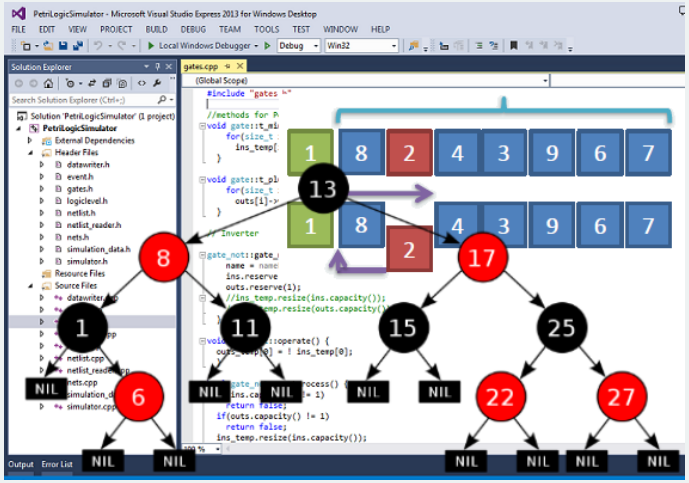
МИЭТ
Национальный исследовательский университет «МИЭТ»
Кафедра ПКИМС

Теория алгоритмов



Лекция 1

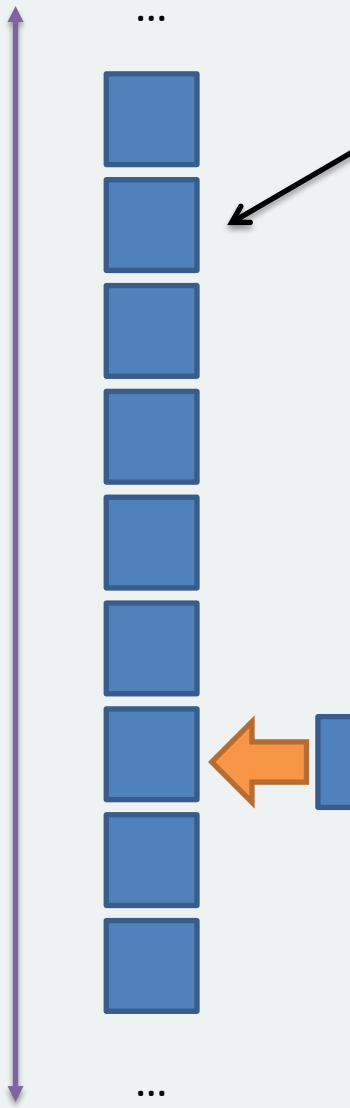
Общие сведения об алгоритмах

Алгоритм: история, определение

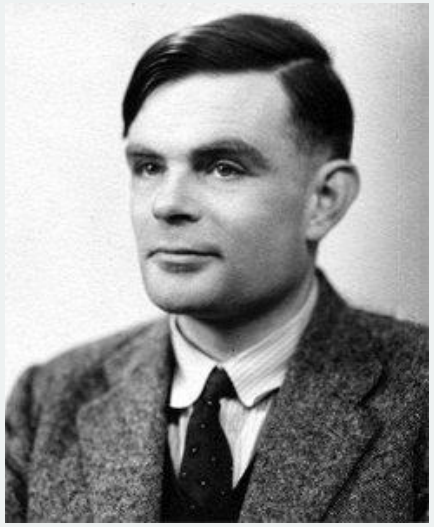
Слово «алгоритм», или «алгорифм» – от английского algorithm, происходит от имени учёного Абу Абдуллах Мухаммеда ибн Муса аль-Хорезми.

Алгоритм - это организованная последовательность действий, понятных для некоторого исполнителя, ведущая **за конечное число шагов** к решению поставленной задачи.

Машина Тьюринга



Ячейки, хранящие информацию «память»



Алан Тьюринг

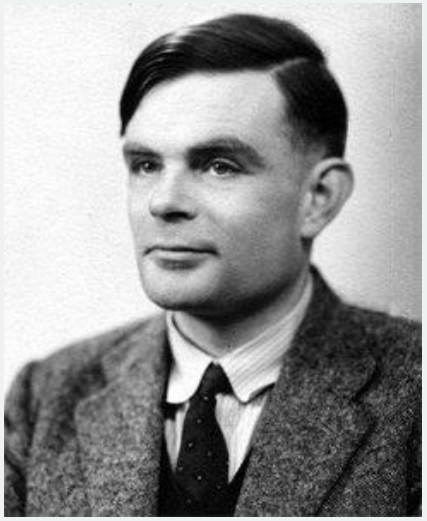
Перечень состояний

Правила перехода

Управляющая машина «процессор»

Тезисы Чёрча-Тьюринга

Любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга



Алан Тьюринг

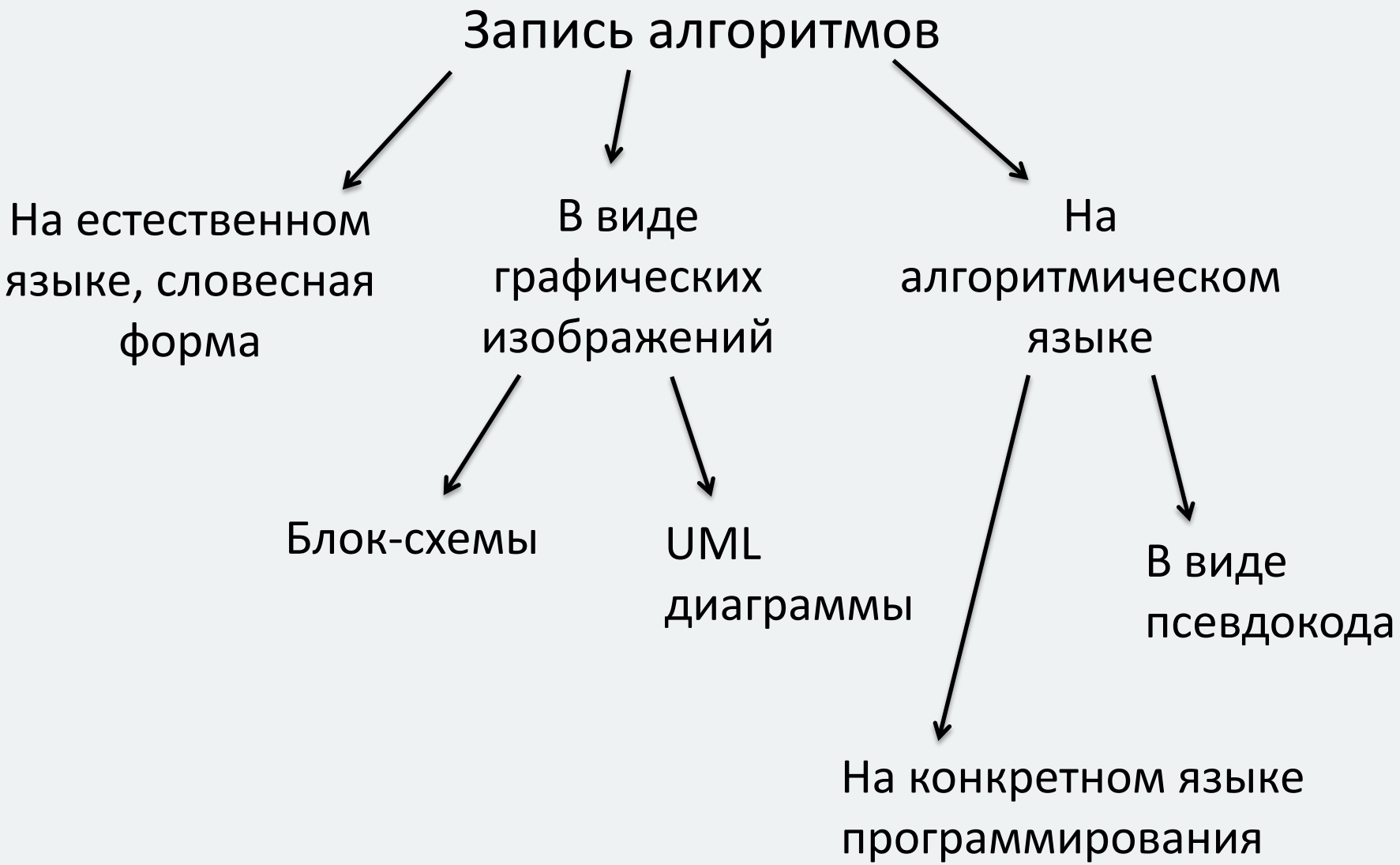
Алонзо Чёрч

Любой конечный физический процесс, не использующий аппарат, связанный с непрерывностью и бесконечностью, может быть вычислен физическим устройством

Свойства алгоритмов

- **Дискретность -** алгоритм состоит из последовательности отдельных шагов - элементарных действий, выполнение которых на заданном уровне абстракции не представляет сложности и вполне понятно.
- **Корректность -** если алгоритм создан для решения определенной задачи, то для всех исходных данных он должен всегда давать правильный результат и ни для каких исходных данных не будет получен неправильный результат.
- **Детерминированность -** при задании одних и тех же исходных данных несколько раз алгоритм будет выполняться абсолютно одинаково и всегда будет получен один и тот же результат.
- **Массовость** - с помощью алгоритма можно решать не одну конкретную задачу, а любую задачу из некоторого класса однотипных задач при всех допустимых значениях исходных данных.
- **Конечность** - последовательность элементарных действий алгоритма не может быть бесконечной, неограниченной, хотя может быть очень большой
- **Результативность** - выполнение алгоритма обязательно должно привести к решению поставленной задачи, либо к сообщению о том, что при заданных исходных величинах задачу решить невозможно.

Формы записи алгоритмов

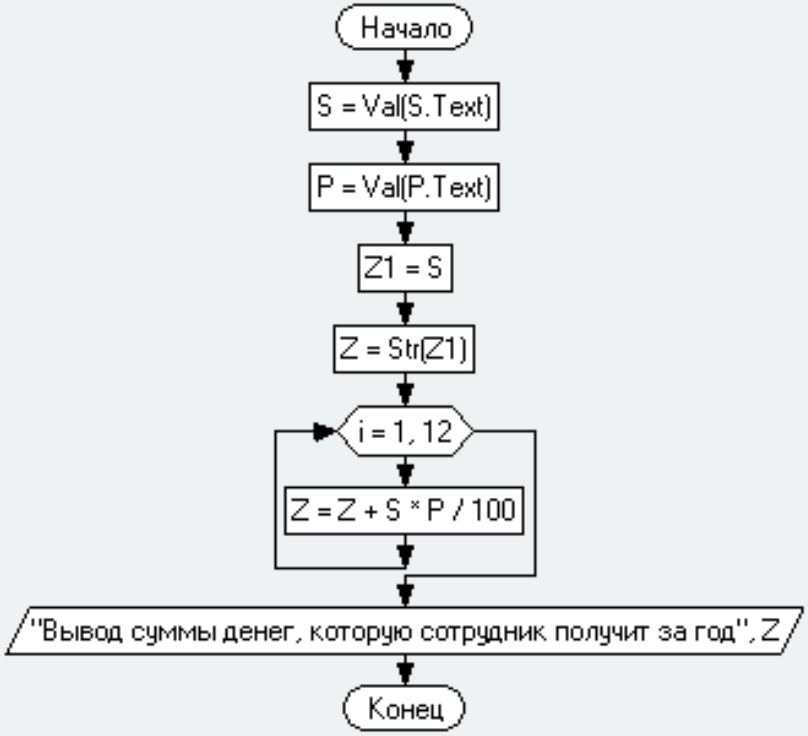


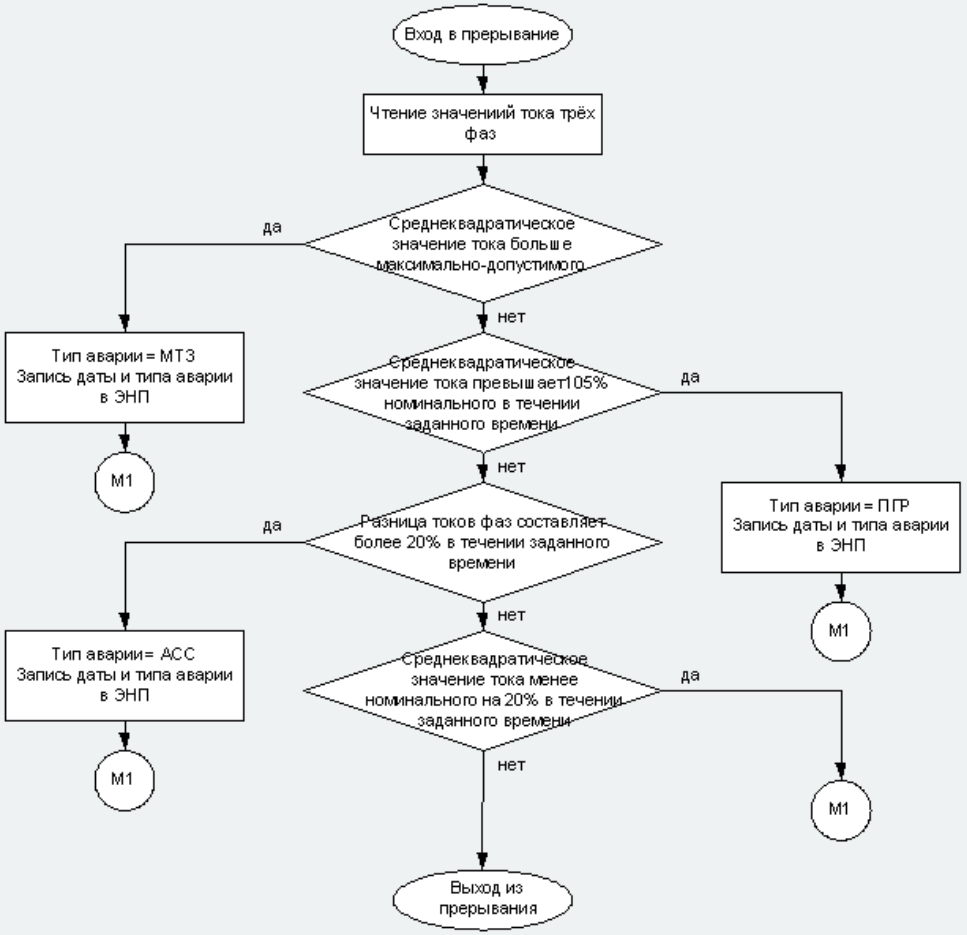
Формы записи алгоритмов: словесная

Программа сложения двух положительных чисел.

Сначала считываются с клавиатуры два числа, затем выполняется проверка того, что эти числа положительны. Если хотя бы одно число не положительно, выводится соответствующее сообщение и выполняется выход из программы. Если все числа …

Графическая форма записи: блок-схемы

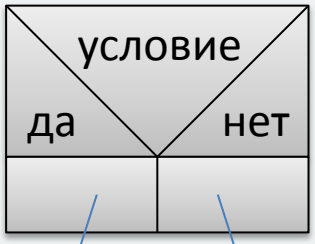




Графическая форма записи: диаграммы Насси-Шнейдермана

Функциональный блок
(действие)

Блок следования
(последовательность действий)



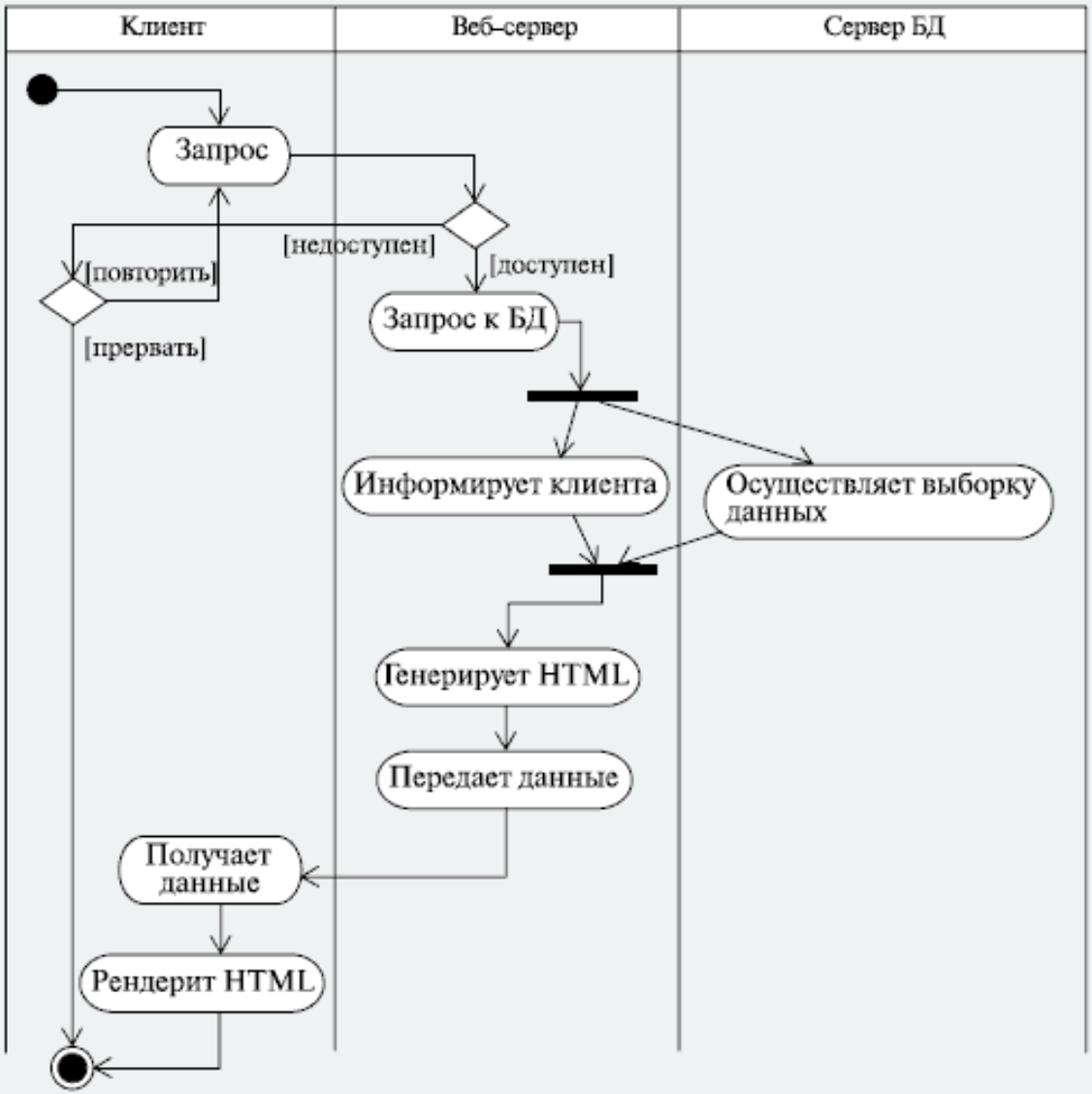
Блок решения
(условие)

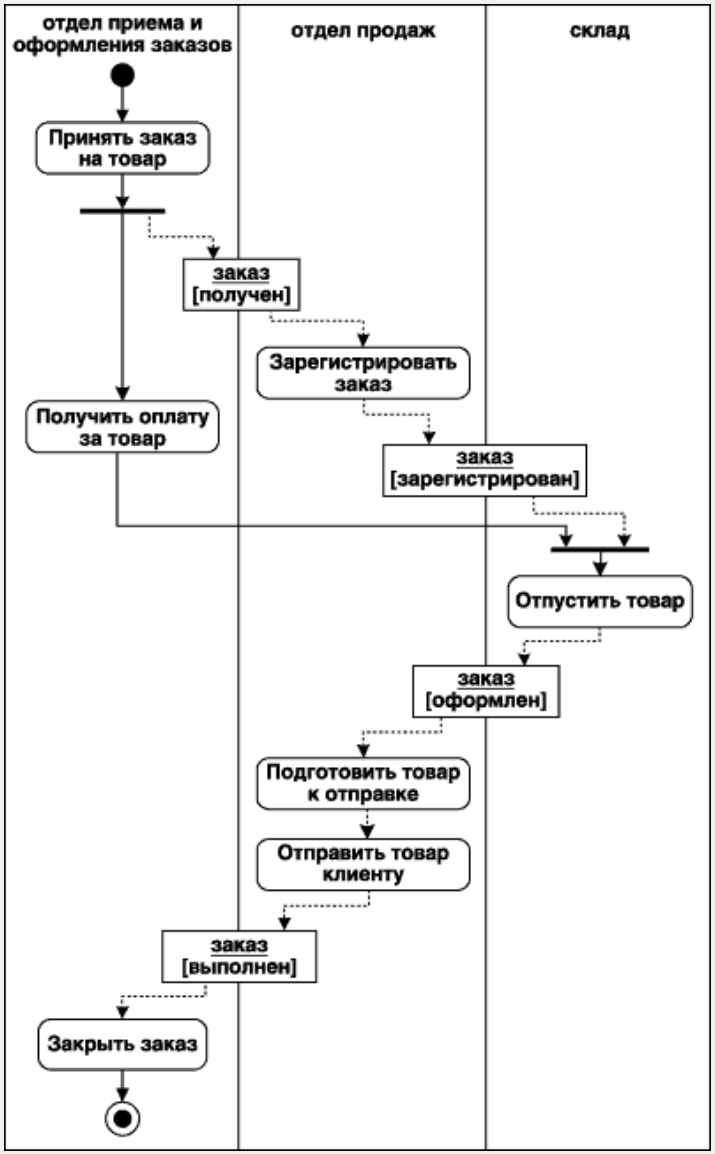
Блок «пока»
(цикл с
предусловием)

Действия,
если «да»

Действия,
если «нет»

Графическая форма записи: UML





Формы записи алгоритмов: код на языке программирования

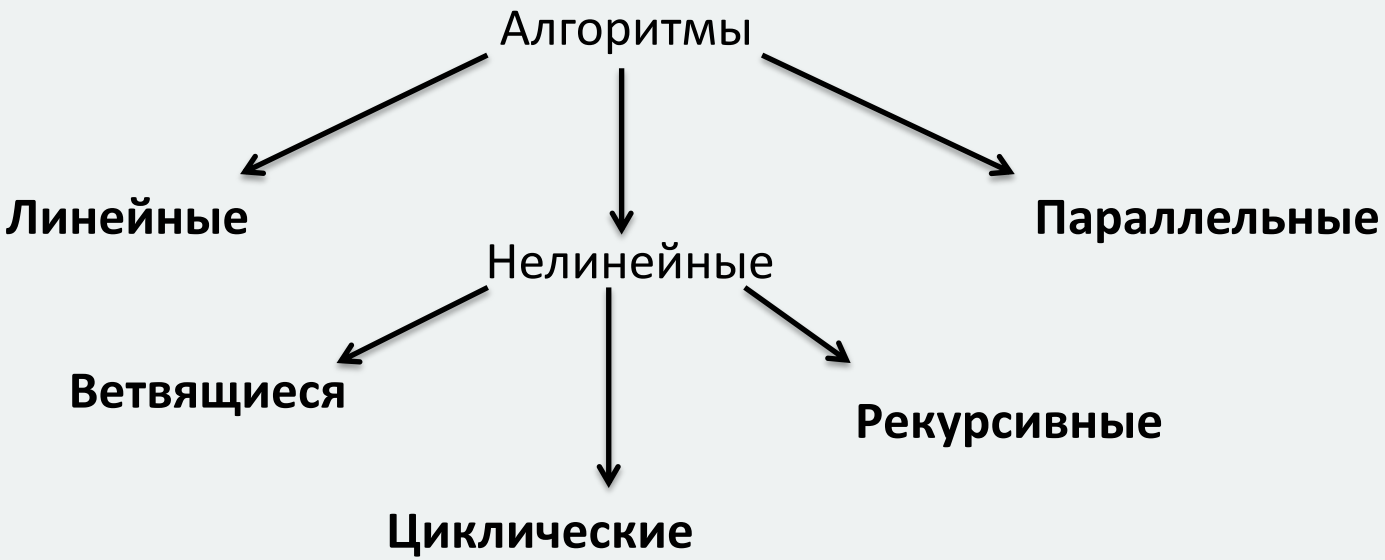
```
program diag;
uses crt;

var
  a, b: word;

function Nod(x, y: word): word;
begin
  if (x <> 0) then
    Nod := Nod(y mod x, x)
  else
    Nod := y;
end;

begin
  ClrScr;
  WriteLn('Введите длины сторон прямоугольника: ');
  Write('a = ');
  ReadLn(a);
  Write('b = ');
  ReadLn(b);
  WriteLn;
  WriteLn('Количество единичных квадратов: ', a + b - NOD(a, b));
  ReadKey;
end.
```

Формы записи алгоритмов: запись в виде псевдокода

```
for (int i = 0; i < N; ++i) {
   scanf("%d", &x);
   printf("%d", x + 2);
}
```

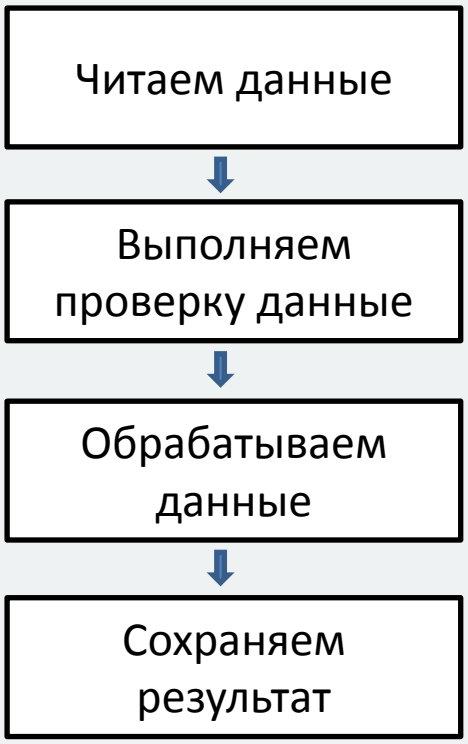
```
for i : 0 → N {
   read x
   print x + 2
}
```

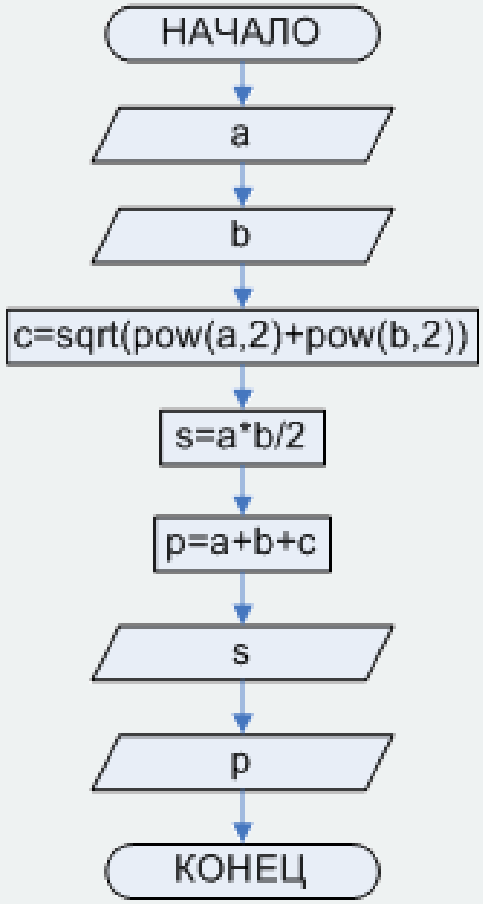
Виды алгоритмов с точки зрения их исполнения



Виды алгоритмов: линейные

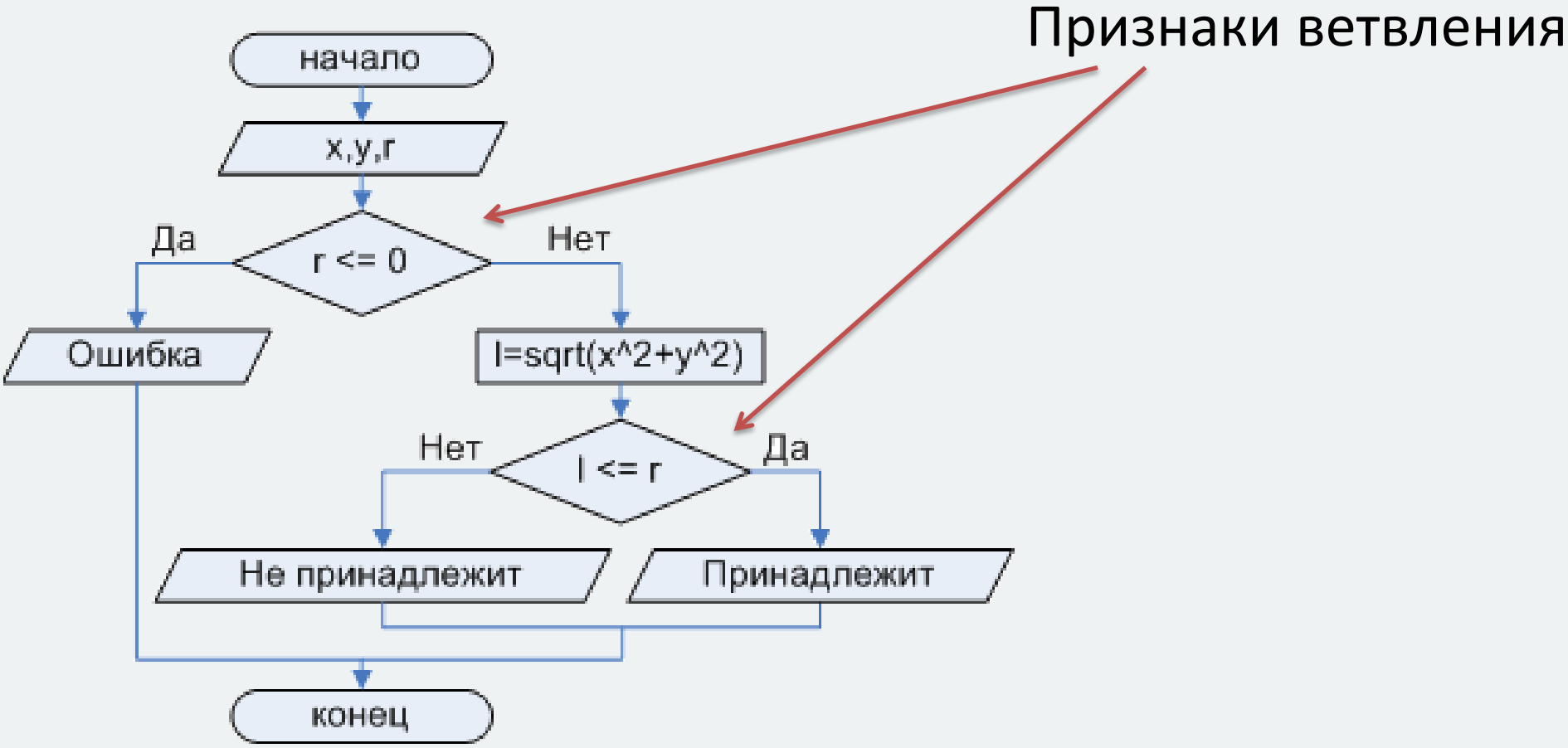
Линейные – на практике реально почти не применяются, но играют важную роль при разработке работы программы на высоком уровне абстракции.





Виды алгоритмов: ветвящиеся / разветвляющиеся

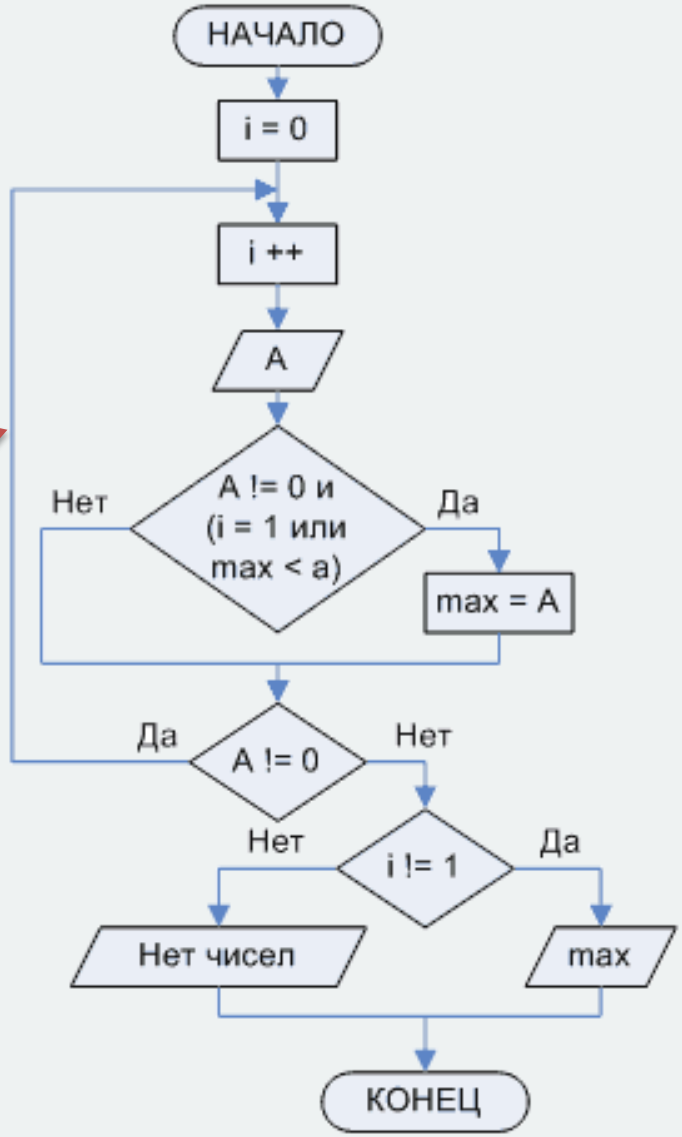
Разветвляющиеся алгоритмы – те, в которых есть проверка логических выражений.



Виды алгоритмов: циклические

Циклические алгоритмы – те, в которых используется многократное повторное выполнение одного и того же программного кода, реализованного в теле цикла.

Признак цикличности



Виды алгоритмов: рекурсивные (1)

Рекурсивные алгоритмы – те, в которых подпрограмма вызывается сама из себя.

Задача: считать с клавиатуры число, найти и вывести на экран число Фибоначчи, соответствующее этому числу.

Виды алгоритмов: рекурсивные (2)

```
long Fib_NoRec(long A) {
  long a = 0, b = 1, c = 0;
  while (A - 1) {
    c = a + b;
    a = b;
    b = c;
    --A;
  }
  return c;
}
```

Перебор

```
long Fib_Rec(long A) {
  if (A == 0 || A == 1)
    return A;

  return (Fib_Rec(A - 1) + Fib_Rec(A - 2));
}
```

Рекурсия

Виды алгоритмов по принципу работы

Детерминированные (жёсткие) алгоритмы – предполагают одну и ту же последовательность действий вне зависимости от входных данных.

Вероятностные (гибкие, адаптивные) алгоритмы – предполагают различное решение в зависимости от некоторых случайным образом генерируемых данных.

Эвристические (статистически верные) алгоритмы – не имеют чёткого обоснования, но дают верный ответ в большинстве случаев.

Парадигмы разработки алгоритмов

Линейные – алгоритмы, последовательно обрабатывающие данные.

Разделяй и властвуй – при использовании этой парадигмы разработки данные делятся на блоки и алгоритм обрабатывает только интересующий его блок данных, удовлетворяющий некоторым критериям, а не весь набор.

Алгоритмы динамического программирования – основная идея парадигмы состоит в учёте ранее посчитанных данных при последующих вычислениях.

Алгоритмы работающие в ширину и в глубину – разработка алгоритмов с без применения рекурсивного подхода и с применением рекурсии.

Чем определяется сложность алгоритма?

Сложность алгоритма определяется числом некоторых операций в зависимости от числа обрабатываемых элементов.

«О большое» – математическое обозначение для сравнения асимптотического поведения функций

$$f(n) = O(g(n))$$

Такая запись читается следующим образом: функция f(n) асимптотически ограничена сверху функцией g(n)

O(N)
O(N·logN)

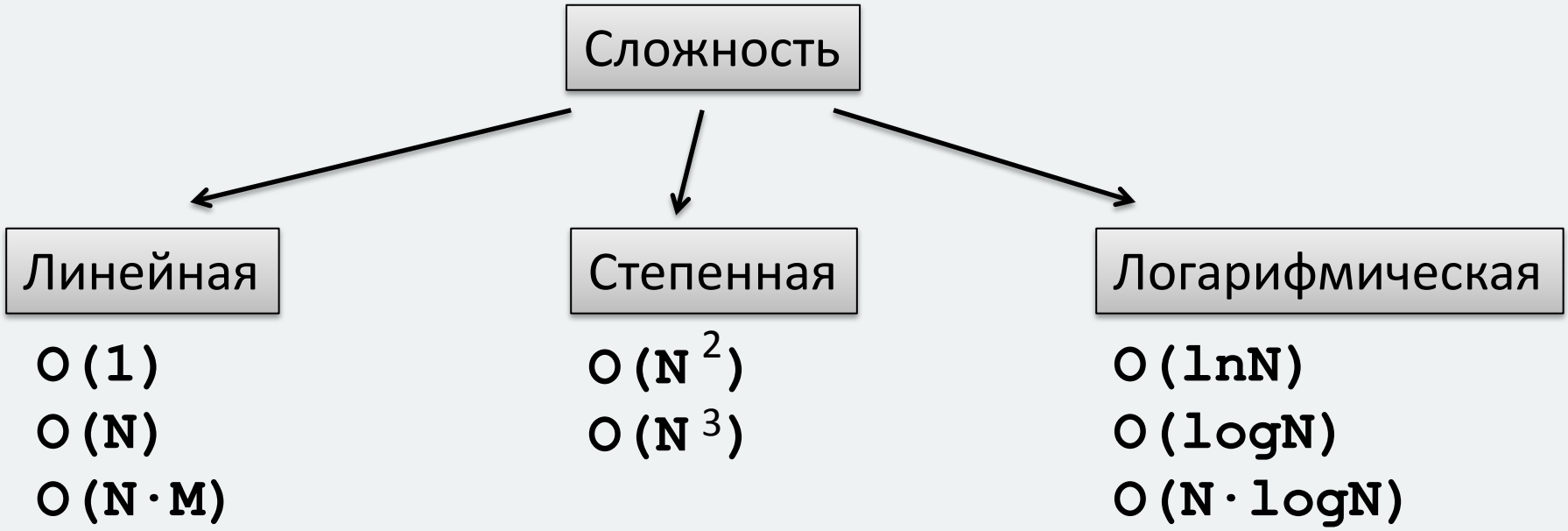
Алгоритмическая сложность может измеряться для:

- худшего случая;
- среднего случая;
- лучшего случая.

Алгоритмическая сложность может измеряться для:

- числа операций;
- требуемого объёма памяти.

Примеры обозначений сложностей алгоритмов



сложность доступа к элементу массива
O(1)

сложность доступа к элементу списка
O(N)

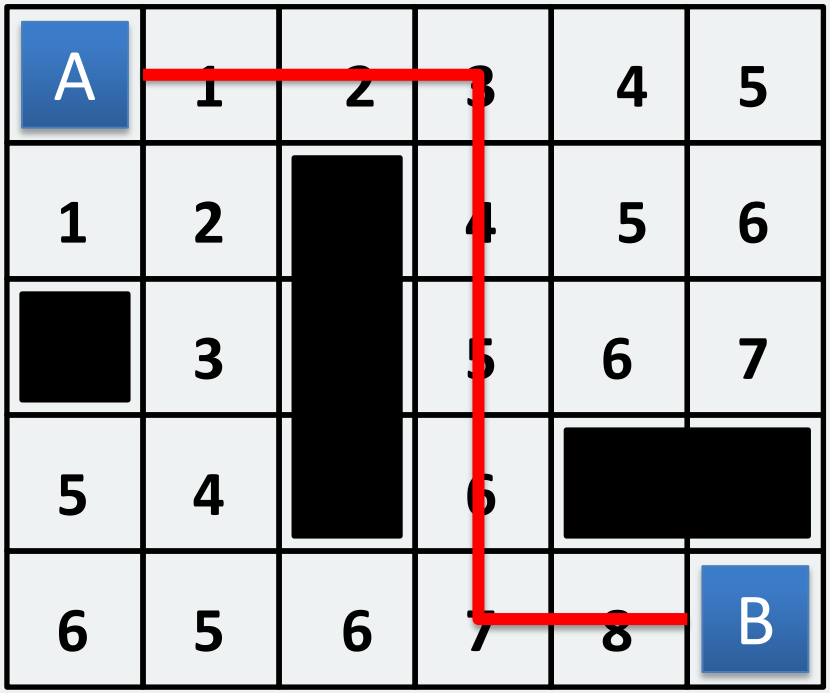
сложность обхода элементов матрицы
O(N^2)

сложность плохих алгоритмов сортировки
O(N^2)

сложность хороших алгоритмов поиска и сортировки
O(N logN)

Волновой алгоритм

Задача: найти кратчайшее расстояние между двумя точками на плоскости



Шаг 1:
нанести на поле сетку

Шаг 2:
проставить первую волну вокруг начальной точки

Шаг 3:
запустить цикл обхода всей матрицы, проставляя следующие номера волны вокруг текущего номера, пока не дойдём до финиша

Шаг 4:
находим обратный путь по наименьшим номерам от конечной точки