# Лекция\_1 Рекурсия: Базовые алгоритмы.

Ведение

1. Прямая рекурсия без ветвления

1.1 Простой пример

1.2 Факториал: итеративное вычисление

1.3 Факториал: рекурсивное вычисление

2. Косвенная рекурсия без ветвления

2.1 Вычисление четности/нечетности числа

3. Прямая рекурсия c ветвлениями

3.1 Функция Фибоначчи: итеративное вычисление

3.2 Функция Фибоначчи: рекурсивное вычисление

4. Косвенная рекурсия c ветвлениями

4.1 Преобразование в строку (Object/Object[])

**Рекурсия в программировании** - это когда метод вызывает сам себя. В таком случае метод называют рекурсивным.

(Рекурсией называется метод (функция), которая внутри своего тела вызывает сама себя.)

Классический пример использования рекурсии, который показывают во всех учебниках по программированию - вычисление факториала числа.

Факториал числа N - это произведение всех целых чисел от 1 до N. Например, возьмём число 3 и вычислим его факториал. У нас получится 1 \* 2 \* 3 = 6. Теперь напишем метод на Java в отдельном классе:

class Factorial {

// рекурсивный метод

int fact(int n) {

int result;

if (n == 1)

return 1;

result = fact(n - 1) \* n;

return result;

}

}

При вызове метода **fact()** с аргументом, равным 1, вернётся 1. Тут пока понятно. При других числах возвращается **fact(n - 1) \* n**. Получается, что нужно ещё раз вызвать метод **fact()**. И так происходит до тех пор, пока не дойдёт до единицы. При этом промежуточные значения умножаются.

Когда метод вызывает сам себя, новым локальным переменным и параметром выделяется место в стеке и код метода выполняется с этими новыми начальными значениями. При каждом возврате из рекурсивного вызова старые локальные переменные и параметры удаляются из стека, и выполнение продолжается с момента вызова внутри метода.

Следует помнить, что рекурсивные методы требуют больше ресурсов для выполнения и даже может вызвать переполнение памяти при слишком больших значениях.

Рекурсивные методы часто используют в сортировке, а также в алгоритмах, связанных с искусственным интеллектом. В обычной практике рекурсия используется редко.

При использовании рекурсивных методов нужно смотреть, чтобы в программе был оператор **if** для выхода из рекурсивного метода без выполнения рекурсивного вызова. Иначе метод никогда не выполнит возврат.

## 1. Прямая рекурсия без ветвления

Рекурсия бывает:  
– прямая и косвенная  
– без ветвления и с ветвлением

### 1.1 Простой пример

Рассмотрим несколько примеров прямой рекурсии без ветвления.  
Печатаем “на входе” (прямой порядок):

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **f(1);**  **}**    **public static void f(int arg) {**  **System.out.print(" " + arg);**  **if (arg < 7) {**  **f(2 \* arg);**  **}**  **}**  **}**  **Печатает:**  **>> 1 2 4 8** |

**Печатаем “на выходе” (обратный порядок):**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **f(1);**  **}**    **public static void f(int arg) {**  **if (arg < 7) {**  **f(2 \* arg);**  **}**  **System.out.print(" " + arg);**  **}**  **}**  **Печатает:**  **>> 8 4 2 1** |

**Печатаем и “на входе” и “на выходе”:**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **f(1);**  **}**    **public static void f(int arg) {**  **System.out.print(" " + arg);**  **if (arg < 7) {**  **f(2 \* arg);**  **}**  **System.out.print(" " + arg);**  **}**  **}**  **Печатает:**  **>> 1 2 4 8 8 4 2 1** |

### Тест 1-1

Проверьте, понимаете ли Вы как работает рекурсия на простом примере (уровень – simple).

### 1.2 Факториал: итеративное вычисление

Факториал – это функция целочисленного положительного аргумента, которая возвращает целое число равное произведению всех целых чисел от 1 до аргумента.  
Пример:  
factorial(5) = 1 \* 2 \* 3 \* 4 \* 5 = 120

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.out.println(factorial(5));**  **}**    **public static int factorial(int arg) {**  **int result = 1;**  **for (int k = 1; k <= arg; k++) {**  **result \*= k;**  **}**  **return result;**  **}**  **}**  **Печатает:**  **>> 120** |

**Для больших значений можно столкнуться с переполнением int:**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.out.println(factorial(17));**  **}**    **public static int factorial(int arg) {**  **int result = 1;**  **for (int k = 1; k <= arg; k++) {**  **result \*= k;**  **}**  **return result;**  **}**  **}**  **Печатает:**  **>> -288522240** |

**Факториал растет очень быстро, возможно, стоит использовать не примитивные типы, а BigInteger:**

|  |  |
| --- | --- |
|  | import java.math.BigInteger;    **public class App {**  **public static void main(String[] args) {**  **System.out.println(factorial(40));**  **}**    **public static BigInteger factorial(int arg) {**  **BigInteger result = BigInteger.ONE;**  **for (int k = 1; k <= arg; k++) {**  **result = result.multiply(BigInteger.valueOf(k));**  **}**  **return result;**  **}**  **}**  **Печатает:**  **>> 815915283247897734345611269596115894272000000000** |

### 1.3 Факториал: рекурсивное вычисление

Однако, для расчет факториала можно делать и рекурсивно:

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          System.out.println(factorial(5));      }        public static int factorial(int arg) {          if (arg == 1) {              return 1;          } else {              return arg \* factorial(arg - 1);          }      }  }    >> 120 |

Или же с использованием тернарного условного оператора даже короче (хотя и загадочнее):

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.out.println(factorial(5));**  **}**    **public static int factorial(int arg) {**  **return (arg == 1) ? 1 : arg \* factorial(arg - 1);**  **}**  **}**    **>> 120** |

### Тест 1-2

Проверьте, понимаете ли Вы примеры реализации одной арифметической функции через другую (уровень – middle).

## 2. Косвенная рекурсия без ветвления

### 2.1 Вычисление четности/нечетности числа

Пример #1: вычисление четности (isEven) и нечетности (isOdd) числа.  
Мы знаем, что 0 – четное число и НЕ НЕчетное число. Для чисел больше нуля – переходим к аргументу меньшему на 1

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          System.out.println("10 - нечетное число: " + isOdd(10));          System.out.println("10 - четное число:   " + isEven(10));      }        // проверка на четность      public static boolean isEven(int k) {          if (k == 0) {              return true; // 0 - четное число          } else {              return isOdd(k - 1);          }      }        // проверка на НЕчетность      public static boolean isOdd(int k) {          if (k == 0) {              return false; // 0 - НЕ НЕчетное число          } else {              return isEven(k - 1);          }      }  }    >> 10 - нечетное число: false  >> 10 - четное число:   true |

## 3. Прямая рекурсия c ветвлениями

Рассмотрим функцию Фибоначчи, определяемую следующим образом:  
fib(0) = 1  
fib(1) = 1  
fib(2) = fib(0) + fib(1)  
fib(3) = fib(1) + fib(2)  
fib(4) = fib(2) + fib(3)  
…  
Вот значения функции Фибоначчи от аргументов 0, 1 … 10

|  |  |
| --- | --- |
|  | -------------------------------------------------------  | arg     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|  -------------------------------------------------------  | fib(arg)| 1 | 1 | 2 | 3 | 5 | 8 | 13| 21| 34| 55| 89|  ------------------------------------------------------- |

### 3.1 Функция Фибоначчи: итеративное вычисление

Эту функцию можно вычислять итеративно (int[] result – это “память”, в которой хранятся значения вычисленные на предыдущих шагах):

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          for (int k = 0; k < 10; k++) {              System.out.println("fib(" + k + ") = " + fib(k));          }      }      public static int fib(int arg) {          if (arg == 0)  return 1;          if (arg == 1)  return 1;          int[] result = new int[arg + 1];          result[0] = 1;          result[1] = 1;          for (int k = 2; k < result.length; k++) {              result[k] = result[k - 2] + result[k - 1];          }          return result[arg];      }  }    >> fib(0) = 1  >> fib(1) = 1  >> fib(2) = 2  >> fib(3) = 3  >> fib(4) = 5  >> fib(5) = 8  >> fib(6) = 13  >> fib(7) = 21  >> fib(8) = 34  >> fib(9) = 55 |

Можно эти значения не запоминать, а хранить только последние два значения и через них вычислять следующее (используем идиому обмена значениями через дополнительную переменную)

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          for (int k = 0; k < 10; k++) {              System.out.println("fib(" + k + ") = " + fib(k));          }      }        public static int fib(int arg) {          if (arg < 2) {              return 1;          } else {              int prev = 1;              int next = 1;              for (int k = 2; k <= arg; k++) {                  int tmp = prev;                  prev = next;                  next = next + tmp;              }              return next;          }      }  }    >> fib(0) = 1  >> fib(1) = 1  >> fib(2) = 2  >> fib(3) = 3  >> fib(4) = 5  >> fib(5) = 8  >> fib(6) = 13  >> fib(7) = 21  >> fib(8) = 34  >> fib(9) = 55 |

### 3.2 Функция Фибоначчи: рекурсивное вычисление

Однако мы можем вычислить эту функцию и рекурсивно

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          for (int k = 0; k < 10; k++) {              System.out.println("fib(" + k + ") = " + fib(k));          }      }        public static int fib(int arg) {          if (arg == 0) {              return 1;          } else if (arg == 1) {              return 1;          } else {              return fib(arg - 2) + fib(arg - 1);          }      }  }    >> fib(0) = 1  >> fib(1) = 1  >> fib(2) = 2  >> fib(3) = 3  >> fib(4) = 5  >> fib(5) = 8  >> fib(6) = 13  >> fib(7) = 21  >> fib(8) = 34  >> fib(9) = 55 |

Или даже в еще более короткой форме – в одну строчку

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          for (int k = 0; k < 10; k++) {              System.out.println("fib(" + k + ") = " + fib(k));          }      }        public static int fib(int arg) {          return arg < 2 ? 1 : fib(arg - 2) + fib(arg - 1);      }  }    >> fib(0) = 1  >> fib(1) = 1  >> fib(2) = 2  >> fib(3) = 3  >> fib(4) = 5  >> fib(5) = 8  >> fib(6) = 13  >> fib(7) = 21  >> fib(8) = 34  >> fib(9) = 55 |

Вы действительно поняли рекурсию, если можете без запуска программы сказать, что выведут следующие программы

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          fib(5);      }      public static int fib(int arg) {          System.out.print(" " + arg);          return arg < 2 ? 1 : fib(arg - 2) + fib(arg - 1);      }  }    >> 5 3 1 2 0 1 4 2 0 1 3 1 2 0 1 |

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          fib(5);      }      public static int fib(int arg) {          int result = arg < 2 ? 1 : fib(arg - 2) + fib(arg - 1);          System.out.print(" " + arg);          return result;      }  }    >> 1 0 1 2 3 0 1 2 1 0 1 2 3 4 5 |

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          fib(5);      }        public static int fib(int arg) {          return arg < 2 ? 1 : fib(arg - 2) + \_(arg) + fib(arg - 1);      }        public static int \_(int arg) {          System.out.print(" " + arg);          return 0;      }  }    >> 3 2 5 2 4 3 2 |

В данном случае мы обходим дерево в глубину слева-направо

|  |  |
| --- | --- |
|  | 5      /  \     /    \    3      4   /\     /  \  1  2   2     3    /\  /\    /\   0 1  0 1  1 2                /\               0 1 |

1 0 1 2 3 0 1 2 1 0 1 2 3 4 5

Если же мы заменим fib(k) = fib(k-2) + fib(k-1) на fib(k) = fib(k-1) + fib(k-2), то РЕЗУЛЬТАТ вычисления функции не изменится, однако ПРОЦЕСС будет другой (обход ЗЕРКАЛЬНОГО по отношению к предыдущему дерева в глубину слева-направо)

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          fib(5);      }        public static int fib(int arg) {          System.out.print(" " + arg);          return arg < 2 ? 1 : fib(arg - 1) + fib(arg - 2);      }  }    >>  5 4 3 2 1 0 1 2 1 0 3 2 1 0 1 |

Обходим теперь вот такое дерево

|  |  |
| --- | --- |
|  | 5          /   \         /     \        4        3       / \      / \      3   2     2  1     /\   /\   /\    2  1  1 0  1 0   /\  1  0 |

### Тест 1-3

Проверьте, понимаете ли Вы “обход дерева вызовов” при рекурсивном вычислении чисел Фибоначчи.

## 4. Косвенная рекурсия c ветвлениями

### 4.1 Преобразование в строку (Object/Object[])

Пример #2:  
Если toStrObj(Object) обнаруживает, что ее аргумент – массив, то вызывает toStrArr(Object[]).  
А toStrArr(Object[]) для каждого аргумента вызывает toStrObj(Object).

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          System.out.println(toStrObj(42));          System.out.println(toStrObj(new Object[] {"Hello", 42}));          System.out.println(toStrObj(new Object[] {"Hello",42,                  new Object[] {"Hello",42,                          new Object[] {"Hello", 42, new Object[0]}}}));      }        public static String toStrObj(Object arg) {          if (arg.getClass().isArray()) {              return toStrArr((Object[]) arg);          } else {              return arg.toString();          }      }        public static String toStrArr(Object[] array) {          String result = "[";          for (int k = 0; k < array.length; k++) {              result += toStrObj(array[k]);              if (k < array.length - 1) {                  result += ", ";              }          }          return result + "]";      }  }    >> 42  >> [Hello, 42]  >> [Hello, 42, [Hello, 42, [Hello, 42, []]]] |

### Тест 1-4

Проверьте, понимаете ли Вы косвенную рекурсию.

## Литература

**Stack/Method invocation в JVM**  
….- JVM Spec: [Java Virtual Machine Stacks](http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5.2)  
….- JVM Spec: [Frames](http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.6)  
….- JVM Spec: [Local Variables](http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.6.1)  
….- JVM Spec: [Receiving Arguments](http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-3.html#jvms-3.6)  
….- JVM Spec: [Invoking Methods](http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-3.html#jvms-3.7)  
**Алгоритмическая сторона вопроса**  
– Вирт, “Алгоритмы и структуры данных”  
….- глава 3, “Рекурсивные алгоритмы”  
– Макконелл, “Анализ алгоритмов”  
….- Глава 1.5. “Алгоритмы вида «разделяй и властвуй»”  
– Кормен, “Алгоритмы. Построение и анализ”  
….- Глава 4. Рекуррентные соотношения