Лабораторная работа № 8
РАЗРАБОТКА РЕКУРСИВНЫХ АЛГОРИТМОВ
Краткая теория

Цель работы. Разработка программ, реализующих различные рекурсивные алгоритмы, и оценка их временной и пространственной сложности.

Понятие рекурсии и примеры ее использования

В математике под рекурсией понимается способ организации вычислений, при котором функция вызывает сама себя с другим аргументом. Большинство современных языков высокого уровня поддерживают механизм рекурсивного вызова. Ввиду отсутствия в языке Java понятия функции рассматриваются рекурсивно вызываемые методы.

Таким образом, в языке Java **рекурсия** – это вызов метода из самого себя непосредственно (**простая рекурсия**) или через другие методы (**сложная** или **косвенная рекурсия**). Примером сложной рекурсии является случай, когда метод A вызывает метод B, а метод B – метод A.

Количество вложенных вызовов методов называется **глубиной рекурсии.**

Реализация рекурсивных вызовов опирается на механизм стека вызовов. Адрес возврата и локальные переменные метода записываются в стек, благодаря чему каждый следующий рекурсивный вызов этого метода пользуется своим набором локальных переменных. Обычно рекурсия реализуется в два прохода:

1) формирование в стеке последовательности аргументов;
2) собственно вычисления (реализация метода), выполняемые над данными, помещенными в стек.

На каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, т.е. для рекурсии необходимо оценивать емкостную сложность. При чрезмерно большой глубине рекурсии может наступить переполнение стека, и возникнуть исключительная ситуация **StackOverfowError** (переполнение стека).

Графическое представление цепочки рекурсивных вызовов, порождаемой данным алгоритмом, называется деревом рекурсивных вызовов.

Можно заметить, что некоторая ветвь дерева рекурсивных вызовов обрывается при достижении такого значения передаваемого параметра, при котором функция может быть вычислена непосредственно. Таким образом, рекурсия эквивалентна конструкции цикла, в котором каждый проход есть выполнение рекурсивной функции с заданным параметром.

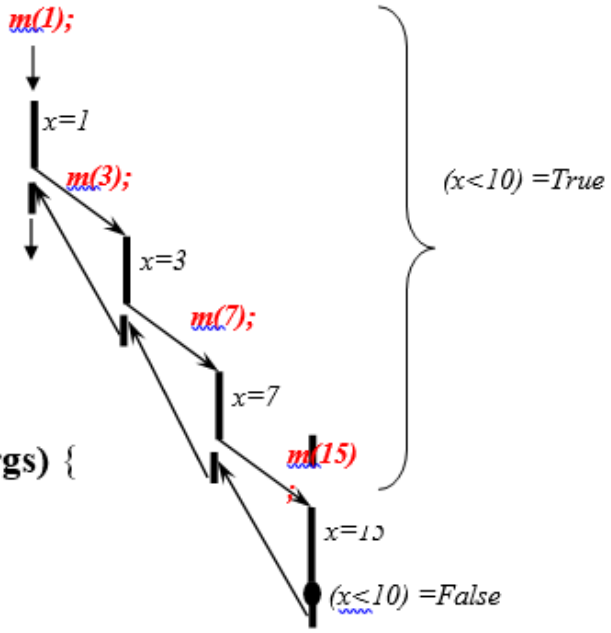
Рассмотрим несколько примеров простой рекурсии, когда метод вызывает сам себя. Проиллюстрируем их рисунком, представляющим дерево вызовов.

Пример 1. Для заданного параметра **x** вывести последовательность числового ряда в соответствии со следующими требованиями:

- очередной элемент x= 2*x+1 (новое значение вычисляется с использованием старого);
- x<10.

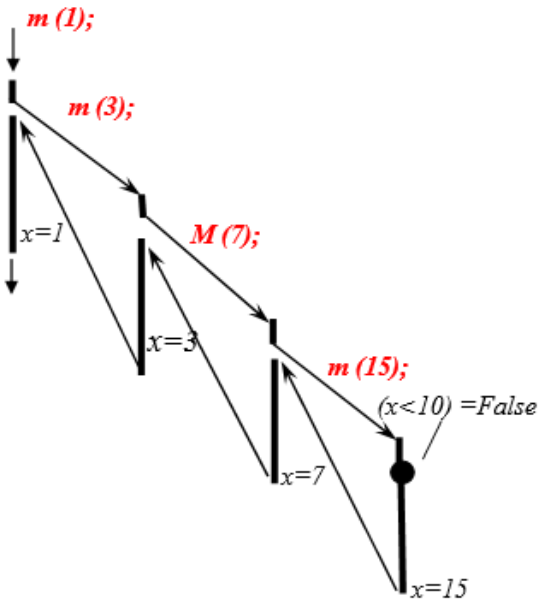
Ниже приведен метод *m*, вычисляющий элемент ряда, и фрагмент основной программы, которая его вызывает. Рядом с текстом метода представлено дерево вызовов.

```
public class Rec1 {
   public static void m(int x) {
      System.out.println("x="+x);
      if (x<10) {
        m(2*x+1);
      }
   }

public static void main(String[] args) {
      m(1);
   }
  }
```



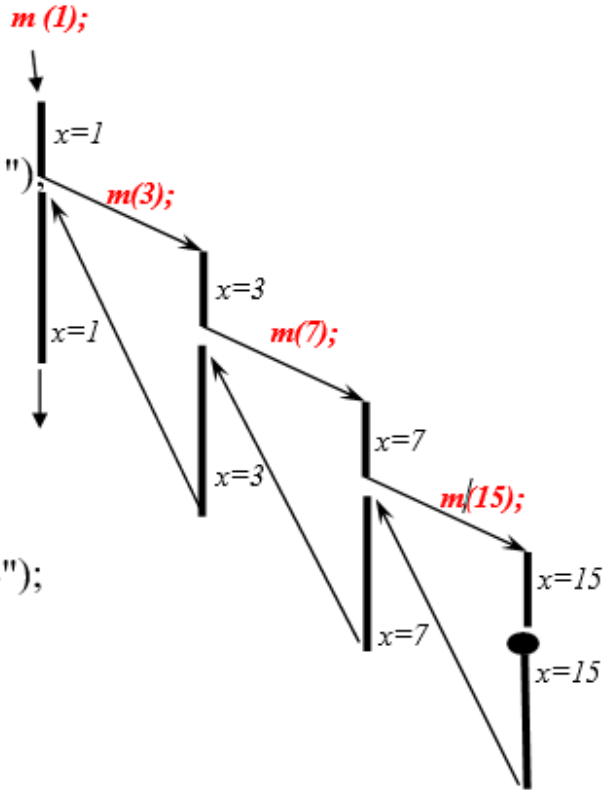
Пример 2. Вывести последовательность из примера 1 в обратном порядке.

```
public class Rec2 {
   public static void m(int x) {
      if (x<10) {
        m(2*x+1);
      }
      System.out.println("x="+x);
     }

   public static void main(String[]
args) {
      m(1);
      }
      }
```



Пример 3. Для примера 1 вывести параметр перед вхождением в рекурсию и после него.

```
public class Rec3 {
   private static int step=0;
   public static void m(int x)
       {space();
       System.out.println(""+x+"-> ");
       step++;
      if (x<10) {
        m(2*x+1);
      }
       step--;
       space();
      System.out.println(""+x+ " <-");
   }

   public static void space() {
      for (int i = 0; i < step; i++) {
           System.out.print("  ");
      }
   }

   public static void main(String[] args) {
     m(1);
   }
}
```



Следует отметить, что рекурсию всегда можно заменить циклом.

Классическим примером рекурсивных алгоритмов считаются вычисление факториала и чисел Фибоначчи. Рассмотрим примеры записи соответствующих методов.

Пример 3. Вычислить факториал числа n с использованием

рекурсии.

Факториал числа *n* (обозначается *n*!) — произведение всех натуральных чисел от 1 до n включительно, т.е. n!=1*2*…*n. Например, 5!=1*2*3*4*5 =4!*5 . В общем случае формулу для нахождения факториала можно записать так

n!=(*n*-1)!**n*, если *n* > 1 и

1, если *n* = 1.

```
public static  int fact(int n){
    int result;
    if (n==1)
       return 1;
    else{
       result=fact(n-1)*n;
       return result;
        }
       }
```

Пример 4. Подсчитать ряд чисел Фибоначчи до заданного значения *x*.

Последовательность Фибоначчи формируется так: первый и второй члены последовательности равны единицам, а каждый следующий — сумме двух предыдущих:

$$f(k) = f(k\text{-}1) + f(k\text{ -}2),$$

$$f(0) = 0,\ f(1) = 1.$$

№ числа	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*…*	*20*
число	*0*	*1*	*1*	*2*	*3*	*5*	*8*	*13*	*21*	*…*	*6765*

```
public static int f(int x){
if (x==0){
      return 0;
}else
    if (x==1){
        return 1;
    } else {
          return f(x-2)+f(x-1);
       }
}
```

Таким образом, трудоемкость рекурсивных алгоритмов зависит как от количества операций, выполняемых при одном вызове функции, так и от количества таких вызовов. Так как рекурсию можно заменить циклом, то порядок временной сложности соответствующего алгоритма можно считать равным числу повторений цикла.

Порядок выполнения лабораторной работы

Работа предполагает выполнение следующих этапов.

1. Знакомство со всеми разделами руководства.

2. Получение у преподавателя задания на разработку программы для рекурсивных алгоритмов (см. прил. 3).

3. Разработка и отладка заданной программы.

4. Получение верхней и экспериментальной оценки времени выполнения заданного алгоритма и программы.

5. Нахождение предельной оценки емкости памяти, необходимой для выполнения разработанной программы.

Содержание отчета о выполненной работе

Отчет о выполненной работе должен содержать.

1. Название и цель работы.
2. Словесное описание заданных рекурсивных алгоритмов.
3. Тексты программ.
4. Формулы верхней оценки временной и емкостной сложности заданных алгоритмов.
5. Результаты экспериментальной оценки временной и емкостной сложности заданных алгоритмов.

Контрольные вопросы

1. Что такое рекурсия и для чего она нужна?
2. Чем отличается простая рекурсия от сложной?
3. Какие классические рекурсивные алгоритмы Вы знаете?
4. Чем можно заменить рекурсию?
5. Какими характеристиками сложности описывается рекурсия?
6. Что такое глубина рекурсии?
7. Что представляет собой дерево рекурсии?
8. Как можно вычислить факториал без использования рекурсивного алгоритма?
9. Как можно вычислить числа Фибоначчи без использования рекурсивного алгоритма?
10. Как можно оценить емкостную сложность рекурсивного алгоритма?

Индивидуальные задания

Разработать следующие алгоритмы и программы с использованием рекурсии.

1. Линейного поиска целочисленного значения ключа в заданном массиве и вывода этого массива.
2. Линейного поиска слова в заданном массиве и вывода этого массива.
3. Дихотомического поиска целочисленного значения ключа в заданном массиве и вывода этого массива.
4. Интерполирующего поиска целочисленного значения ключа в заданном массиве и вывода этого массива.
5. Вычисления целой степени целого числа.
6. Вычисления целой степени вещественного числа.
7. Перевода целого числа, введенного с клавиатуры, в двоичную систему счисления.
8. Перевода целого числа, введенного с клавиатуры, в систему счисления с основанием q.
9. Ввода одномерного массива и линейного поиска целочисленного значения ключа в нем.
10. Ввода одномерного массива слов и линейного поиска заданного слова в нем.
11. Ввода одномерного массива и дихотомического поиска целочисленного значения ключа в нем.
12. Ввода одномерного массива и интерполирующего поиска целочисленного значения ключа в нем.