Лекция 2.3: Базовые алгоритмы / Рекурсивные алгоритмы.

1. Популярные задачи
... 1.1 Язык Дика (язык правильных скобочных выражений)
... 1.2 Получение из одного слова – другого слова путем вычеркивания букв
... 1.3 ???
... 1.4 ???
... 1.5 ???
2. Рекурсивные сортировки
... 2.1 Сортировка слиянием (merge sort)
... 2.2 Быстрая сортировка (quick sort)
Тесты по всей лекции
Видео
Лабораторные
Литература

Ваша успеваемость

Java Core	14.34%
1. Основы Java	88.07%
2. Базовые алгоритмы	24.13%
3. Исключения	31.21%
4. Ввод/вывод	0%
5. Многопоточность	0%
6. Коллекции	0%
7. ООП: Синтаксис	0%
8. ООП: Шаблоны	0%
9. Продвинутые возможности	0%
10. Java 8	0%

Язык Дика

Язык Дика – это язык правильных скобочных выражений. "((){}())" – это слово языка Дика, а ")(" – не слово языка Дика.

Рекурсивный генератор всех слов языка Дика длины 2 * N:

```
import java.util.ArrayList;
import java.util.List;

import static java.util.Collections.singletonList;

public class App {
    public static void main(String[] args) {
        System.out.println(next("", 3, 3));
    }

    public static List<String> next(String head, int open, int close) {
        if (open == 0 & close == 0) {
            return singletonList(head);
        } else if (open == 0 & close != 0) {
            return next(head + ')', open, close - 1);
        } else if (close == open) {
            return next(head + '(', open - 1, close);
        } else {
            List<String> result = new ArrayList<>();
            result.addAll(next(head + '(', open, close - 1));
            result.addAll(next(head + ')', open - 1, close));
            return result;
        }
    }
}

>> [()()(), ()(()), (())(), (()()), ((()))]
```

Рекурсивный распознаватель слов языка Дика

```
public class App {
    public static boolean recognize(String word) {
        return recognize(word, 0);
    }

    private static boolean recognize(String word, int open) {
        if (word.isEmpty()) {
            return open == 0;
        } else {
            switch (word.charAt(0)) {
                case '(':
                    return recognize(word.substring(1), open + 1);
                case ')':
                    return open > 0 && recognize(word.substring(1), open - 1);
                default:
                    throw new IllegalArgumentException();
            }
        }
    }

    public static void main(String[] args) {
        System.out.println(recognize("("));
        System.out.println(recognize(")("));
        System.out.println(recognize("()"));
        System.out.println(recognize("(())"));
        System.out.println(recognize("((()()(()))"));
    }
}

>> false
>> false
>> true
>> true
>> true
```

2.1 Сортировка слиянием (mergesort)

```
???

???

???
```

2.2 Быстрая сортировка (quicksort)

```
???

???

???
```

Рассмотрим следующую серию примеров
Рекурсивная реализация сложения через функцию прибавления единицы:

```
public class App {
    public static void main(String[] args) {
        System.out.println(sum(5, 3));
    }

    public static int sum(int a, int b) {
        return b == 0 ? a : sum(a, b - 1) + 1;
    }
}

>> 8
```

Рекурсивная реализация умножения через функцию сложения:

```
public class App {
    public static void main(String[] args) {
        System.out.println(mul(5, 3));
    }

    public static int mul(int a, int b) {
        return b == 1 ? a : mul(a, b - 1) + a;
    }
}

>> 15
```

Рекурсивная реализация возведения в степень через функцию умножения:

```
public class App {
    public static void main(String[] args) {
        System.out.println(pow(5, 3));
    }

    public static int pow(int a, int b) {
        return b == 1 ? a : pow(a, b - 1) * a;
    }
}

>> 125
```

А теперь все вместе:

```
public class App33 {
    public static void main(String[] args) {
        System.out.println(pow(5, 3));
    }

    public static int pow(int a, int b) {
        return b == 1 ? a : mul(pow(a, b - 1), a);
    }

    public static int mul(int a, int b) {
        return b == 1 ? a : sum(mul(a, b - 1), a);
    }

    public static int sum(int a, int b) {
        return b == 0 ? a : sum(a, b - 1) + 1;
    }
}

>> 125
```

Заметьте:
1. Мы реализовали возведение в степень используя исключительно операции +1, -1, рекурсивный вызов функции, тернарный условный оператор. Вопрос в том – можно ли все возможные целочисленные функции построить из этого набора операций?
2. Мы строили каждую следующую функцию единообразно из предыдущей: *1 -> sum -> mul -> pow. Вопрос – можем ли мы продолжить этот ряд функций в бесконечность, получая все более и более растущие функции. Скажем, какая функция следует за степенью?

Ответ на первый вопрос – ???

Ответ на второй вопрос – да, можем продолжить в бесконечность. Вот следующая функция:

```
import java.math.BigInteger;

public class App {
    public static void main(String[] args) {
        System.out.println(next(BigInteger.valueOf(5), BigInteger.valueOf(3)));
    }

    public static BigInteger next(BigInteger a, BigInteger b) {
        return b.equals(BigInteger.ONE) ?
                a
                :
                next(a, b.subtract(BigInteger.ONE)).pow(a.intValue());
    }
}

>> 298023223876953125
```

Эти идеи привели к развитию:
- ??? функция Аккермана
- ??? стрелочная нотация Кнута
- ???

Тесты по всей лекции

Тест, состоящий из случайных вопросов тестов этой лекции

??? ТЕСТОВ ПОКА НЕТ ???

Видео

Набор декабрь 2013
Набор октябрь 2013
Набор июль 2013
Набор апрель 2013
Набор февраль 2013
Набор январь 2013
Набор октябрь 2012

Лабораторные

??? пока нет

Литература

??? пока нет