Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и интеллектуальных систем

03. «ОТ ДУШИ, РОДНОЙ, ДЕЛАЙ КРАСИВО!»

Лекции по информатике для студентов первого курса Высшей школы ИТИС
2019 год

МИХАИЛ АБРАМСКИЙ
старший преподаватель
Высшая школа ИТИС КФУ

Построение алгоритмов

- ~~Красота~~ Надежность написания
    - Code Conventions + Best Practices
- Эффективность алгоритма
- Эффективность построения

Code conventions

- Как правильно оформлять java-код
    - Правильный почерк java-разработчика

Пример – вычисление силы тяжести

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Команды

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Одна строка – одна команда (;) !

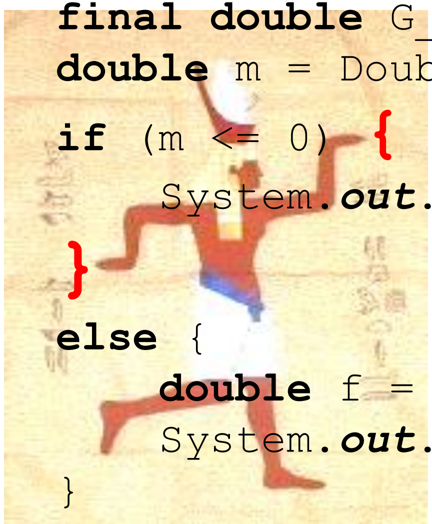
Скобки – Egyptian Style

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Открывающая
На той же строке, что и заголовок конструкции

Закрывающая
На новой строке, на одном уровне с заголовком

Скобки – Egyptian Style



```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Пробелы

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Пробелы

У скобок, у операторов

Нет пробела

У вызова методов

Отступы

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Вложенность – 4 пробела от того, во что вложено.

клавиша TAB – но не всегда (в обычном блокноте TAB = 8 пробелов)

Названия (идентификаторы)

Классы – UpperCamelCase

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Названия (идентификаторы)

Методы и переменные – lowerCamelCase

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Названия (идентификаторы)

Константы – UPPERCASE with Underscores

```
public class MyGravityClass {

    public static void main(String[] args) {

        // Ищем силу тяжести для заданной массы
        final double G_ACCELERATION = 9.86;
        double m = Double.parseDouble(args[0]);
        if (m <= 0) {
            System.out.println("Mass should be positive");
        }
        else {
            double f = m * G;
            System.out.println("Force of gravity is: " + f);
        }
    }
}
```

Проверяем

- OneTwoThree –
- oneTwoThree –
- oneTwoThree() –
- ONE_TWO_THREE –

Ответ

- OneTwoThree – класс
- oneTwoThree – переменная
- oneTwoThree(...) – метод
- ONE_TWO_THREE – константа

Такие правила позволяют читать и понимать код быстро:

- по названию ясна суть идентификатора

ЛАКОНИЧНОСТЬ КОДА

Нагромождение переменных

- Иногда это проблема вовсе не для теории сложности

- Пример: подсчитать (a – 1) * b + (b – 5) * d

```
int x = a - 1;
int y = x * b;
int z = b - 5;
int u = z * d;
int v = y + u;
```

Дублирование кода (влияющее на сложность)

```
if (x * x - 2 * x > 5 - x) {
    System.out.println(x * x - 2 * x);
}
else {
    System.out.println(5 - x);
}
```

Разумеется сложность не сильно увеличилась, но все же..
Лучше:

```
int a = x * x - 2 * x;
int b = 5 - x;
int max = a > b ? a : b; // Math.max
System.out.println(max);
```

Дублирование кода (не влияющее на сложность)

- n!! = 1 * 3 * 5 * … * n (если n – нечетное),
- и 2 * 4 * 6 * … * n (если четное).

Типичное решение:

```
if (n % 2 == 1) {
    int p = 1;
    for (int i = 1; i <= n; i+=2) {

    p *= i;
    }
}
else  {
    int p = 1;
    for (int i = 2; i <= n; i+=2) {
        p *= i;
    }
}
```

Причесываем

```
int p = 1;
for (int i = 2 - n % 2; i <= n; i+=2) {
    p *= i;
}
```

Еще проще

- Отнимая от n двойку, мы всегда получаем все четные или все нечетные числа.

```
int p = 1;
while (n >= 1) {
    p *= n;
    n -= 2;
}
```

ПРИВЕТ МАТЕМАТИЧЕСКОМУ АНАЛИЗУ

Вещественные числа

- В чем прелесть работы с ними на компьютере?
  - См. реальный мир.

Вещественные числа

- В чем прелесть работы с ними на компьютере?
  – См. реальный мир.

- Их необходимо считать до определенной точности (дальше не нужно)
    - Например, калькулятор на экране имеет место только для 10 символов – так зачем считать дальше 9го знака после запятой?

Предел последовательности

$$\forall \varepsilon > 0 \exists n = n(\varepsilon) \in N \forall \mathrm{n} > \mathrm{n}(\varepsilon)(|a_n - A| < \varepsilon)$$

Тогда А – предел последовательности a_n

Но что это реально означает?

$$\forall \varepsilon > 0 \exists n = n(\varepsilon) \in N \forall \mathrm{n} > \mathrm{n}(\varepsilon)(|a_n - A| < \varepsilon)$$

Раскроем модуль, получим:

$$A - \varepsilon < a_n < A + \varepsilon$$

Простым языком – элементы последовательности a_n находится недалеко от А (**примерно ему равны с точностью ε**)

Примерно ?
Для любого ε !?

$$\forall \varepsilon > 0 \exists n = n(\varepsilon) \in N \forall \mathrm{n} > \mathrm{n}(\varepsilon)(|a_n - A| < \varepsilon)$$

Возьмем простую последовательность

- $a_n = \frac{1}{n}$
- $\lim\limits_{n \to \infty} a_n = 0$

Вычислим

- $a_1 = 1$
- $a_2 = 0{,}5$
- $a_3 = 0{,}3333333333333$
- $a_4 = 0{,}25$
- $a_5 = 0{,}2$
- $a_6 = 0{,}1666666666667$
- $a_7 = 0{,}1428571428571429$
- $a_8 = 0{,}125$
- $\ldots$

- $\ldots$
- $a_{100} = 0{,}01$
- $\ldots$
- $a_{10000} = 0{,}0001$
- $\ldots$
- $a_{1000000} = 0{,}000001$
- $\ldots$
- $a_{1000000000000} = 0{,}000000000001$

Кто похож на 0?

Вспомним снова определение

$(|a_n - A| < \varepsilon)$

- …
- $a_{100} = 0{,}01 - 0 = 0{,}01$
- …
- $a_{10000} = 0{,}0001 - 0 = 0{,}0001$
- …
- $a_{1000000} = 0{,}000001 - 0 = 0{,}000001$
- …
- $a_{1000000000000} = 0{,}000000000001 - 0 = 0{,}000000000001$

Кто есть кто

$(|a_n - A| < \varepsilon)$

- …
- $a_{100} = 0{,}01 - 0 = \mathbf{0{,}01}$
- … $a_n - A$
- $a_{10000} = 0{,}0001 - 0 = \mathbf{0{,}0001}$
- … $a_n - A$
- $a_{1000000} = 0{,}000001 - 0 = \mathbf{0{,}000001}$
- … $a_n - A$
- $a_{1000000000000} = 0{,}000000000001 - 0 = \mathbf{0{,}000000000001}$

$a_n - A$

Это $\boldsymbol{\varepsilon}$

Кто есть кто

$(|a_n - A| < \varepsilon)$

- ...
- $a_{100} = 0{,}01 - 0 = \mathbf{0{,}01}$
- ...

$a_n - A$

- $a_{10000} = 0{,}0001 - 0 = \mathbf{0{,}0001}$
- ...

$a_n - A$

- $a_{1000000} = 0{,}000001 - 0 = \mathbf{0{,}000001}$
- ...

$a_n - A$

- $a_{1000000000000} = 0{,}000000000001 - 0 = \mathbf{0{,}000000000001}$

$a_n - A$

это ε

это n(ε)

$\forall n > n(\varepsilon)$

МАТАН VS ИНФОРМАТИКА

- Математический анализ говорит **о любом ε** – какое бы малое мы не взяли, все равно будет последовательность «стремиться» (быть ближе) к числу А, быть похожей на него.
  - Ну и там на бесконечности будет «равна А»

- В программировании – а зачем нам эта бесконечность? Мы наоборот фиксируем ε – и это наша точность вычислений. И считаем до тех пор, пока она не начнет соблюдаться.

Основа приближенного подсчета

Если нам нужно подсчитать что-то бесконечное (предел, бесконечную сумму, ряд функции), то мы

- Фиксируем точность
- Считаем до тех пор, пока не начнет выполняться точность (в смысле, указанном выше)

Но ведь предел изначально неизвестен

Что делать?

А вот что:

«Если все элементы будут похожи на предел, то они будут **похожи друг на друга»**

Фундаментальная последовательность

Любая сходящаяся последовательность является фундаментальной

- Мы не знаем предел заранее, не с чем сравнивать.
- Но можем пользоваться фундаментальностью, если знаем, что последовательность имеет предел

Ряды

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$$

$$shx = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{x^{2n-1}}{(2n-1)!} + \ldots$$

$$chx = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + \frac{x^{2n}}{(2n)!} + \ldots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + \ldots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + (-1)^{n-1}\frac{x^{2n-2}}{(2n-2)!} + \ldots$$

$$(1+x)^m = 1 + \frac{m}{1!}x + \frac{m(m-1)}{2!}x^2 + \frac{m(m-1)(m-2)}{3!}x^3 + \ldots$$

$$\frac{1}{1+x} = 1 - x + x^2 - x^3 + \ldots + (-1)^{n-1}x^n + \ldots$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots + (-1)^{n-1}\frac{x^n}{n} + \ldots$$

$$arctgx = x - \frac{x^3}{3} + \frac{x^5}{5} - \ldots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)} + \ldots$$

$$\arcsin x = x + \frac{1}{2}\frac{x^3}{3} + \frac{1\cdot 3}{2^2 2!}\frac{x^5}{5} + \frac{1\cdot 3\cdot 5}{2^3 3!}\frac{x^7}{7} + \ldots$$

$$tgx = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \ldots + \frac{2n-2}{(2n-1)!}x^{2n-1} + \ldots$$

http://promatematiku.ru/referati/vichislit-opredelenniy-integral-s-tochnosty-do-0001.html

Пример

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$$

http://promatematiku.ru/referati/vichislit-opredelenniy-integral-s-tochnosty-do-0001.html

Вычисление e^1

$a0 = 1$

$s0 = a0 = \mathbf{1}$

$a1 = 1$

$s1 = s0 + a1 = 1 + 1 = \mathbf{2}$

$a2 = 1/2! = 0,5$

$s2 = s1 + a2 = 2 + 0,5 = \mathbf{2,5}$

$a3 = 1/3! = 1/6 = 0,16666667$

$s3 = \mathbf{2,6666666667}$

$a4 = 1/4! = 1/24 = 0,0416666666666667$

$s4 = s3 + a4 = \mathbf{2,70833333333333}$

$a5 = 1/5! = 1/120 = 0,00833333333333333$

$s5 = \mathbf{2,71666666666666}$

$a6 = 1/6! = 1/720 = 0,00138888888888888$

$s6 = \mathbf{2,71805555555555555}$

Вычисление e

a(3) = 2.370370
a(3) – a(2) = 0.120370 > eps
a(4) = 2.441406
a(4) – a(3) = 0.071036 > eps
a(5) = 2.48832
a(5) – a(4) = 0.046914 > eps
…
a(100) = 2.704811
a(101) = 2.704946
a(101) – a(100) = 0.000135
…
a(200) = 2.711517
a(201) = 2.711550
a(201) – a(200) = 0.000033
…
a(300) = 2.713765
a(301) = 2.713780
a(301)-a(300) = 0.000015

Наблюдения:

Точность сходства элементов друг с другом выше, чем сходство с пределом

- Элементы похожи уже до 4го знака
- *А с числом e до какого?*

Оцените сложность (примерно)
Выгоднее считать или заранее хранить e? =)))

Math.pow(x, y)

- $x^y = e^{\ln(x^y)} = e^{y\,ln(x)}$
- *Math.exp(y * Math.ln(x));*
- Когда вы вызываете Math.exp или Math.log, там происходят такие же вычисления.
- *Поэтому Math.pow(x, 2) – very, very bad*
  - *В 100 раз медленней x * x*

ЭФФЕКТИВНОСТЬ АЛГОРИТМА

Эффективный алгоритм

- Делает то, что нужно!
  - Вычисляет именно ту функцию, для которой и был написан
    - Хотя и это не всегда (! вероятностные алгоритмы)
- Оптимально использует ресурсы
  - *А что есть ресурсы?*

Ресурсы алгоритма

- Время работы алгоритма
  - «Количество шагов»
    - *Длина трассы*
- Память, требуемая для работы
  - «Количество ячеек памяти, необходимых для работы»

КАК ИЗМЕРЯТЬ ОБЪЕМ РЕСУРСОВ?

Измеряем точно

- Замеряем время работы алгоритма в микро- (нано-) секундах
- Замеряем точный размер использованной памяти (все переменные)

- *Почему не годится?*

От чего зависят используемые ресурсы?

- Понятно, что от самой задачи.
- *А конкретнее?*

От входных данных

- А точнее – от их размера!

- *Что есть размер входных данных?*

От входных данных

- А точнее – от их размера!

- *Что есть размер входных данных?*
  - **Количество ячеек, занимаемое входом.**

Количество ячеек

- Вообще говоря, понятие относительное
  - Если речь об алгоритме, обрабатывающем массив, то **размер входа = размер массива** = количество чисел в нем (одно число – одна ячейка)
  - Если об алгоритме обработки целого числа, то **размер входа – количество цифр числа** (одна цифра – одна ячейка)
  - *Почему такой неоднородный подход нас не беспокоит?*

Почему такой подход нас не беспокоит?

- *А зачем нам сравнивать между собой алгоритм обработки массива и алгоритм обработки числа?*
  - Нужно сравнивать между собой алгоритмы, решающие одну задачу.

- Итак, ресурсы зависят от размера входа?
  - Раз зависят, то тогда это…

Сложность вычислений

- Computational Complexity

- **T(n) – временнАя сложность** (сложность по времени)
- **S(n) – пространственная сложность** (сложность по памяти)

- *Какая важнее, как думаете?*

Измерение сложности

- Раз это функция, то она как-то явно зависит от размера входа n.
  - КАК?
  - Можно выразить формулой:
    - Например: $T(n) = n$.

- **Внимание: нам не нужно задавать точную формулу зависимости**
  - Нам важно знать порядок зависимости от n – насколько сильно растет сложность при увеличении размера входа.
    - *Почему?*
    - *Как нам это облегчает жизнь?*

Дисклеймер

- Проходить более подробно разные меры сложности вы будете на алгоритмах и структурах данных

- Мы воспользуемся только одной.

- Как думаете, какой вид сложности нам полезен:
    a) Сложность в лучшем случае
    b) Сложность в худшем случае
    c) Сложность в среднем случае

Дисклеймер

- Проходить более подробно разные меры сложности вы будете на алгоритмах и структурах данных

- Мы воспользуемся только одной.

- Как думаете, какой вид сложности нам полезен:
    a) Сложность в лучшем случае
    b) Сложность в худшем случае
    c) Сложность в среднем случае

О - символика

«О-большое» (не путать с «о-маленьким»)

$f = O(g)$, если есть константа C, что

$$f(x) <= C \cdot\ g(x)$$

«Оценка сверху»
«Ну точно будет не больше, чем $g(x)$»

Асимптотическая оценка

Свойства O(f)

Верны только слева направо:

$$f \cdot O(g) = O(f \cdot g)$$
$$C \cdot O(f) = O(f)$$
$$O(f) + O(g) = O(max(f, g))$$

Пример:

$$O(n^2) + O(n) = O(n^2)$$
$$n \cdot O(n) = O(n^2)$$

Какие бывают сложности

- **Полиномиальная – O(n^k)**
  - Частные случаи – линейная и константная
- **Экспоненциальная – O(k^n)**
- **Логарифмическая – O($\log n$)**
  - Не важно основание логарифма
    - *Надо понимать, почему!*

(n – размер входа, k - константа)

Максимум массива

Ввод массива

```
int max = a[0];
for (int x : a) {
        if (x > max)
                max = x;
}
```

Какая сложность?

Сортировка

```
for (int i = 0; i < n - 1; i++) {
    m = i;
    for (int j = i + 1; j < n; j++) {
        if (a[j] > a[m]) {
            m = j;
        }
    }
    h = a[i];
    a[i] = a[m];
    a[m] = h;
}
```

Сложность?

Циклы по всем элементам массива – источник полиномиальной сложности

- Количество вложенности – порядок полиномиальной сложности

- Два соседних цикла – это какая сложность?

```
for (int i = 1; i <= n; i++) {
    //...
}
for (int i = 1; i <= n; i++) {
    //...
}
```

- А еще что может дать полиномиальную сложность?

Cruel World

- Было бы классно, если бы все алгоритмы имели полиномиальную сложность.

- Но:
  - *Перебрать все числа из 4х цифр*
    - *Сколько сочетаний?*

Cruel World

- Было бы классно, если бы все алгоритмы имели полиномиальную сложность.

- Но:
  - *Перебрать все числа из N цифр*
    - *Сколько сочетаний?*
    - *Вот такая и сложность – экспоненциальная!*

P и NP

- класс P – задачи, решаемые детерминированной машиной Тьюринга за полиномиальное время

- класс NP – задачи, решаемые недетерминированной машиной Тьюринга за полиномиальное время.
- класс NP – задачи, решаемые детерминированной машиной Тьюринга за экспоненциальное время.

- Понятно, что задача из P входит и в NP
  – Но можно ли задачу из NP решить за полиномиальное время

Who knows?

- Вроде очевидно.
  - Но никто не доказал
  - Лучший результат: Александр Разборов (1990-е)
    - Ввел natural proofs, показал, что в современной математике пока нет инструментов, способных доказывать такие теоремы.

- Экспоненциальная сложность – вроде плохо (есть медленные алгоритмы)
  - Но и хорошо – *нельзя взломать цифровые системы перебором – времени не хватит*.

Если P = NP

Летят все системы авторизации, аутентификации, банковские системы, электронные средства оплаты, криптографические системы, военные тайны, блокчейн и т.д.

Скорее всего не равно. Но