

AI помощники и оптимизация кода: советы, ожидание и реальность

Владимир Ярославский
Исполнительный директор, Сбер



JPoint, Москва
29-30 апреля 2026

Владимир Ярославский

- Более 30 лет в любимой IT-отрасли
- Язык программирования — Java
- Первая версия JDK, с которой я познакомился, была 1.0.1
- В JDK и Android применяется моя сортировка Dual-Pivot Quicksort
- В NetBeans есть моя система печати
- Моя почта vlv.spb.ru@mail.ru



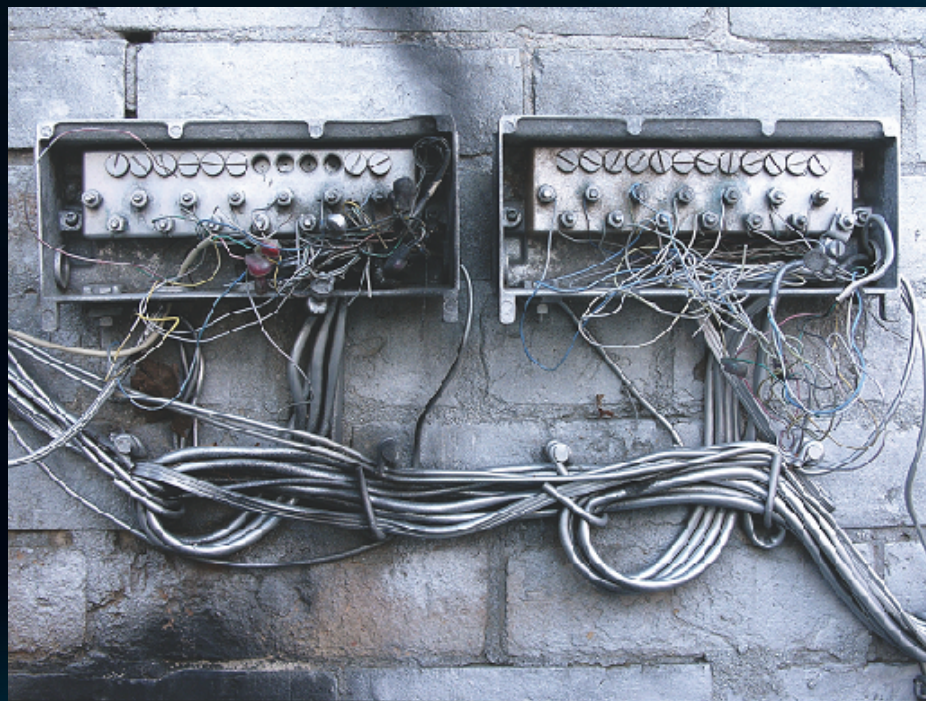
Текущий проект



Создание сберовской
платформы для
разработки

Что значит хороший код?

- читаемый
- ясный
- простой дизайн
- работает правильно
- **производительный**



На что обращаем внимание?

- размер занимаемой памяти
- сложность алгоритма
- глубина рекурсии
- количество потоков
- ...
- время выполнения



О чем поговорим

String [Builder]

float / int

Алгоритмы

Трюки с байтами

Типы данных

Математика

Коллекции

Параллелизм

Какие используем инструменты



IntelliJ IDEA



JMH
Java Microbenchmark Harness

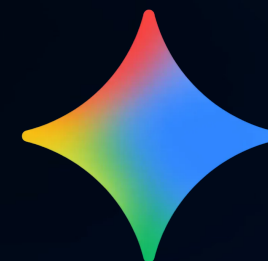
Какие используем инструменты



SourceCraft
Code Assistant



ChatGPT



Google Gemini

Java Microbenchmarking Harness (JMH)

<https://github.com/openjdk/jmh>

JMH is a Java harness for building, running, and analysing nano / micro / milli / macro benchmarks written in Java

```
<groupId>org.openjdk.jmh</groupId>  
<artifactId>jmh-core</artifactId>  
<version>1.37</version>
```

```
org.openjdk.jmh:jmh-core:1.37
```



Warmup

```
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MICROSECONDS)
@Warmup(iterations=5, time=2, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations=4, time=4, timeUnit = TimeUnit.SECONDS)

public class BenchmarkTest {
    @Benchmark
    public void test() {
        // сам тест...
    }
}

@Setup
void setup() {
    // инициализация...
}
```

Проверяйте версию JDK

```
@Benchmark
```

```
public void sortArray() {  
    Arrays.sort(a);  
}
```

Jdk 8	Jdk 25
610_654,946	554_227,643 us/op

```
@Benchmark
```

```
public void concat() {  
    for (int c = 0; ++c < 32000;) {  
        str += c;  
    }  
}
```

Jdk 8	Jdk 25
535_752,925	232_019,815 us/op

String [Builder]

String vs. StringBuilder

```
String text = "";  
for (int i = 0; i < 10_000; ++i) {  
    text += i;  
}
```

```
StringBuilder builder = new StringBuilder();  
for (int i = 0; i < 10_000; ++i) {  
    builder.append(i);  
}  
String text = builder.toString();
```

String vs. StringBuilder

The screenshot displays a code editor on the left and a chat window on the right. The code editor shows the following Java code:

```
1 package com.boot.ai;
2
3 public class MainAI1 {
4
5     public static void main(String[] args) {
6         String text = "";
7         for (int i = 0; i < 10_000; ++i) {
8             text += i;
9         }
10        System.out.println(text);
11    }
12 }
13
```

The chat window, titled "GigaCode Chat", shows a message from "You" with the text "Оптимизируй код". Below this, the chat displays the optimized code:

```
package com.boot.ai;

public class MainAI1 {

    public static void main(String[] args) {
        StringBuilder text = new StringBuilder();
        for (int i = 0; i < 10_000; ++i) {
            text.append(i);
        }
        System.out.println(text);
    }
}
```

Below the code, the chat provides an explanation in Russian:

Объяснение оптимизации:

1. Заменен String text = ""; на StringBuilder text = new StringBuilder();
2. Использован метод append() вместо оператора += для конкатенации строк
3. Снижено количество создаваемых объектов String с 10,000 до 1
4. Уменьшена нагрузка на garbage collector
5. Повышена производительность за счет использования изменяемого буфера StringBuilder

At the bottom of the chat window, there are buttons for "Think On", "Add context", and "MainAI1.java Current X".

String vs. StringBuilder

```
public static void main(String[] args) {  
  
    StringBuilder textBuilder = new StringBuilder();  
  
    for (int i = 0; i < 10_000; ++i) {  
        textBuilder.append(i);  
    }  
    System.out.println(textBuilder.toString());  
}
```

Оптимизация: использование StringBuilder

Ускорение выполнения на 100-1000 раз (в зависимости от JVM)



String vs. StringBuilder

AI, оптимизируй еще раз

```
public static void main(String[] args) {  
    // Оптимизация #1: Точное выделение памяти (38,890 символов)  
    StringBuilder textBuilder = new StringBuilder(38890);  
  
    // Оптимизация #2: Ясная переменная цикла вместо "i"  
    for (int num = 0; num < 10_000; ++num) {  
        textBuilder.append(num);  
    }  
    System.out.println(textBuilder);  
}
```



Дополнительная оптимизация:

Точное вычисление ёмкости буфера
Учитываем реальную длину всех чисел от 0 до 9999

1-значные: 10 чисел * 1 символ = 10

2-значные: 90 чисел * 2 символа = 180

3-значные: 900 чисел * 3 символа = 2700

4-значные: 9000 чисел * 4 символа = 36,000

Итого: 38,890 символов

Почему 50`000?



Числа от 0 до 9999 в сумме дают примерно 40–45 тысяч символов, так что 50000 — безопасный запас

String vs. StringBuilder

Benchmark	Score	Error	Units
StringBuilder	87,404 ±	0,933	us/op
String	10883,332 ±	1025,496	us/op

Разница в 125 раз!



ArrayList



LinkedList

ArrayList vs. LinkedList

Сравни ArrayList и LinkedList при добавлении элементов в начало, середину и конец



Сравнение ArrayList и LinkedList при добавлении элементов

Операция	ArrayList	LinkedList
Добавление в начало	$O(n)$ (требуется сдвиг всех элементов)	$O(1)$ (просто изменение указателей)
Добавление в середину	$O(n)$ (сдвиг элементов после указанной позиции)	$O(n)$ (поиск позиции + $O(1)$ для вставки)
Добавление в конец	$O(1)$ (в среднем, без ресайза)	$O(1)$ (изменение указателя последнего элемента)

Объяснение разницы

1. ArrayList (динамический массив):

- **Внутренняя реализация:** Хранит элементы в непрерывном блоке памяти.
- **Добавление в начало/середину:** Требуется сдвиг всех последующих элементов (`System.arraycopy`), что занимает $O(n)$ времени.
- **Добавление в конец:** Быстро ($O(1)$), если есть свободное место. При переполнении массива — $O(n)$ из-за ресайза.

2. LinkedList (двусвязный список):

- **Внутренняя реализация:** Хранит элементы в отдельных узлах с ссылками на соседние элементы.
- **Добавление в начало:** Быстро ($O(1)$), просто изменяются указатели головы.
- **Добавление в середину/конец:**
 - **Поиск позиции:** $O(n)$ (требуется пройти по списку).
 - **Сама вставка:** $O(1)$ (меняются только указатели соседних узлов).

ArrayList vs. LinkedList

Добавление 20 000 элементов (микросекунды)

	Начало	Середина	Конец
ArrayList	2,679	1,314	0,047
LinkedList	0,069	25,041	0,054

ArrayList vs. LinkedList

Добавление 20 000 элементов (микросекунды)

	Начало	Середина	Конец
ArrayList	2,679	1,314	0,047
LinkedList	0,069	25,041	0,054
ArrayDeque	0,049		0,048

ArrayList vs. LinkedList

Чтение 20 000 элементов (микросекунды)

	Начало	Середина	Конец
ArrayList	0,035	0,036	0,038
LinkedList	0,054	24,124	0,037

В 670 раз медленнее!

ArrayList vs. LinkedList

Производительность (на 10_000 элементов)



Операция	ArrayList (время в мс)	LinkedList (время в мс)
Добавление в начало	~15	~0.1
Добавление в середину	~7	~0.2
Добавление в конец	~0.05	~0.1

??

Примечание: LinkedList выигрывает при операциях в начале, но проигрывает в случайном доступе ($O(n)$ vs $O(1)$ у ArrayList)

Заключение

- Для частых вставок/удалений в начале/середине — LinkedList.
- Для быстрого доступа по индексу и добавления в конец — ArrayList.
- LinkedList имеет большую фрагментацию памяти, но лучше масштабируется при частых модификациях.



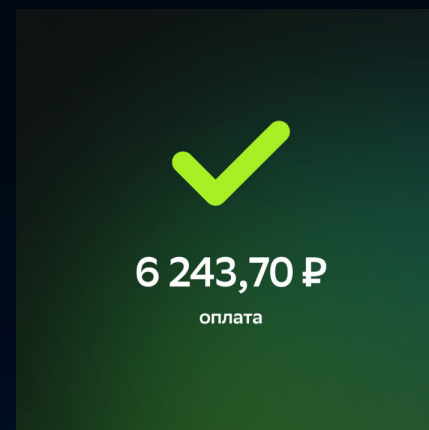
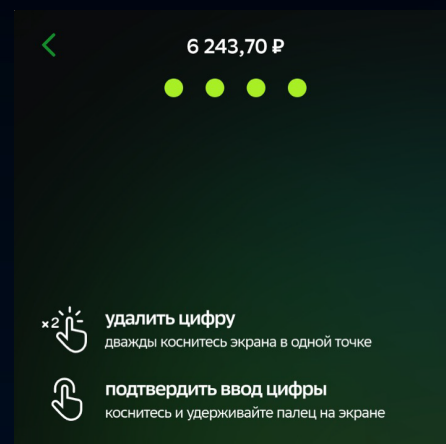
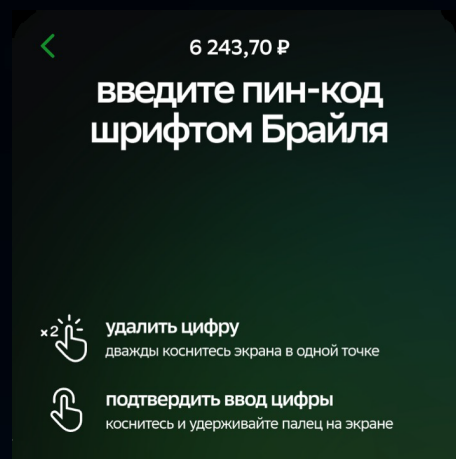
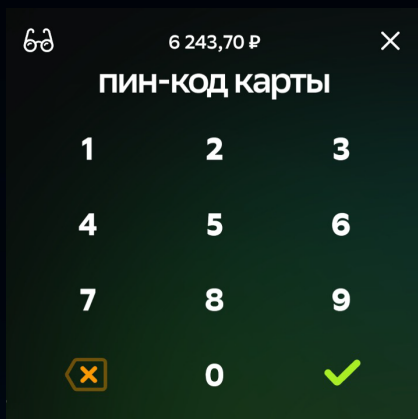
Но на прошлой неделе были другие цифры!?

	ArrayList	LinkedList
Добавление в начало	~1200 мс	~0.5 мс
Добавление в середину	~600 мс	~120 мс
Добавление в конец	~0.2 мс	~0.3 мс

??

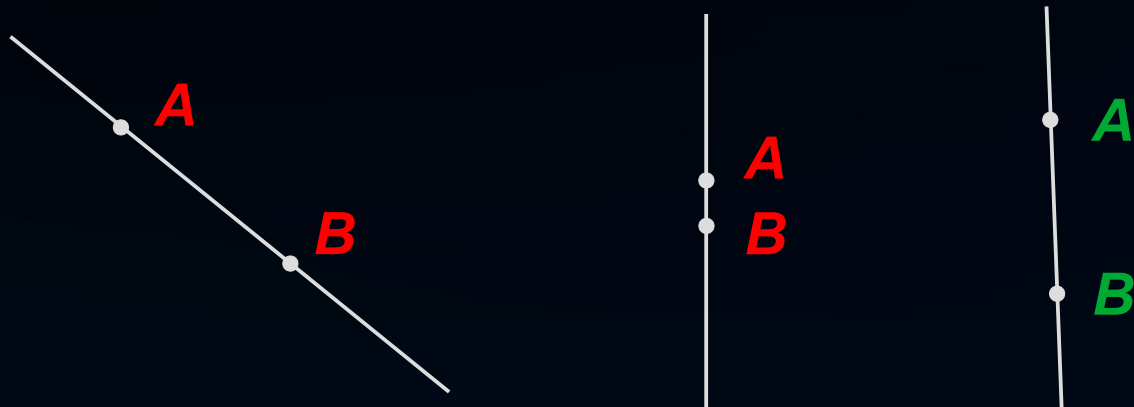
Математика (школьная геометрия)

Как незрячим вводить ПИН-код?



Постановка задачи

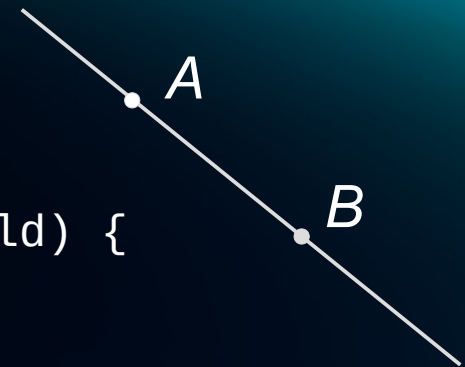
Определить, находятся ли две точки на почти вертикальной прямой и находятся ли они достаточно далеко друг от друга (координаты целочисленные)?



Просим AI нам помочь

AI, напиши код на Java, который определяет, находятся ли две точки далеко друг от друга?

```
boolean arePointsFarApart(Point p1, Point p2, double threshold) {  
    double distance = // Вычисляем евклидово расстояние  
        Math.sqrt(Math.pow(p2.x - p1.x, 2)  
            + Math.pow(p2.y - p1.y, 2));  
    return distance > threshold;  
}
```

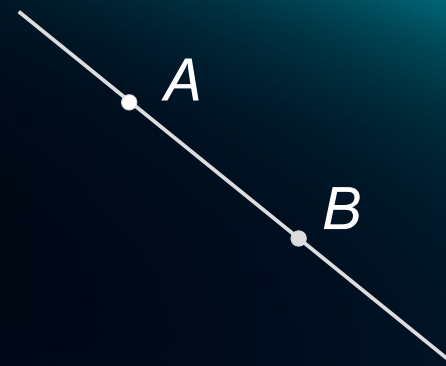


Школьная геометрия

1. **Далеко** ли расположены две точки друг от друга?

$$A(x_1, y_1) \quad B(x_2, y_2)$$

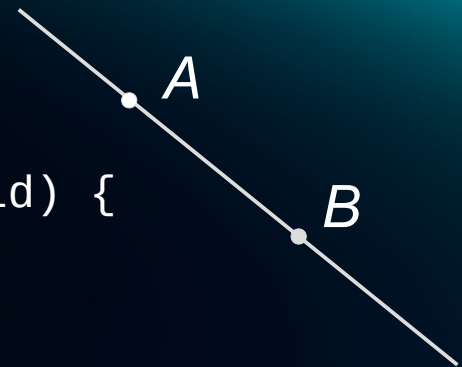
$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} > \text{min} \quad ?$$



Школьная геометрия

AI, оптимизируй

```
// Оптимизированный метод без вычисления квадратного корня
boolean arePointsFarApart(Point p1, Point p2, double threshold) {
    double dx = p2.x - p1.x; // Разница по оси X
    double dy = p2.y - p1.y; // Разница по оси Y
    double squaredDistance = dx * dx + dy * dy;
    // Сравнение с квадратом порога
    return squaredDistance > threshold * threshold;
}
```

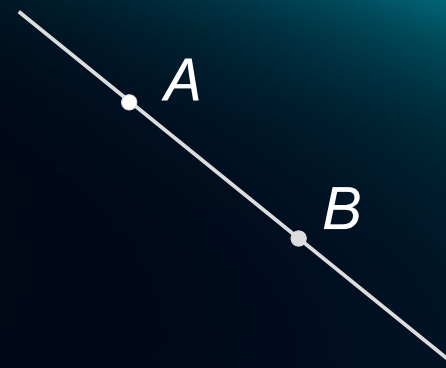


Школьная геометрия

1. Далеко ли расположены две точки друг от друга?

$$A(x_1, y_1) \quad B(x_2, y_2)$$

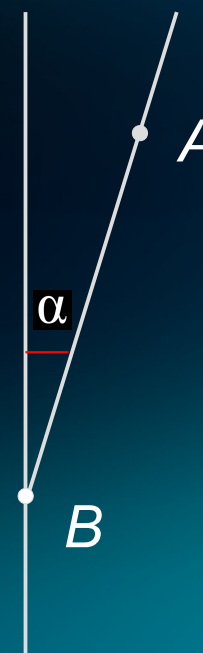
$$(x_2 - x_1)^2 + (y_2 - y_1)^2 > MIN \quad !$$



Школьная геометрия

AI, напиши кода на Java, который проверяет, находятся ли две точки на почти вертикальной прямой

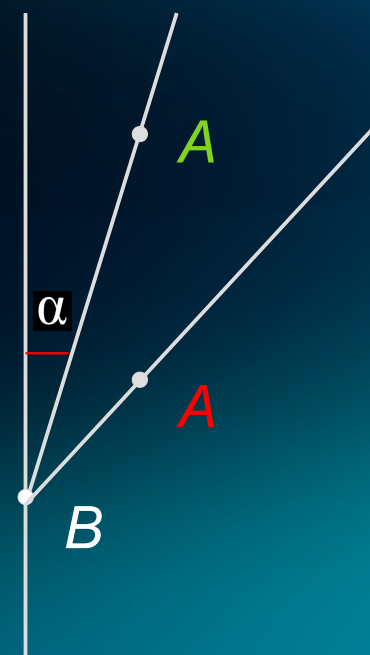
```
// Метод для проверки "почти вертикальности"  
boolean isAlmostVertical(double x1, double y1,  
    double x2, double y2, double epsilon  
) {  
    return Math.abs(x1 - x2) < epsilon;  
}
```



Школьная геометрия

AI, напиши кода на Java, который проверяет, находятся ли две точки на почти вертикальной прямой

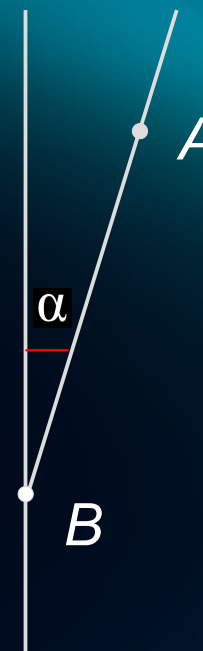
```
// Метод для проверки "почти вертикальности"  
boolean isAlmostVertical(double x1, double y1,  
    double x2, double y2, double epsilon  
) {  
    return Math.abs(x1 - x2) < epsilon; ?  
}
```



Школьная геометрия

2. Находятся ли две точки на почти вертикальной прямой?

Как найти угол между прямой и осью OY ?

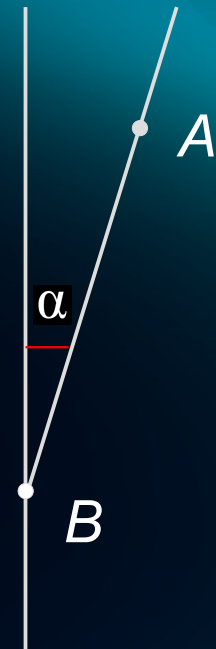


Школьная геометрия

2. Находятся ли две точки на почти вертикальной прямой?

$$A(x_1, y_1) \quad B(x_2, y_2)$$

$$\frac{|x_2 - x_1|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} < \sin(7^\circ) \quad ?$$



А можно ли проще?

Школьная геометрия

2. Находятся ли две точки на почти вертикальной прямой?

$$A(x_1, y_1) \quad B(x_2, y_2)$$

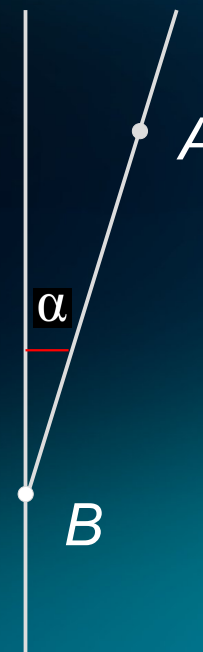
Тангенс проще синуса!



Школьная геометрия

$$A(x_1, y_1) \quad B(x_2, y_2)$$

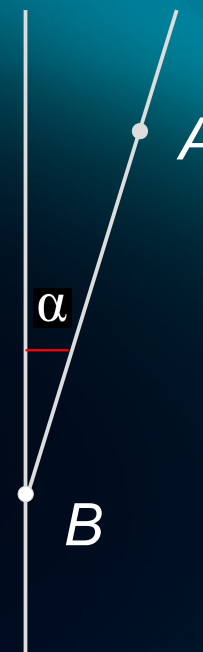
$$\left| \frac{x_2 - x_1}{y_2 - y_1} \right| < \operatorname{tg}(7^\circ) \quad ?$$



Школьная геометрия

$$A(x_1, y_1) \quad B(x_2, y_2)$$

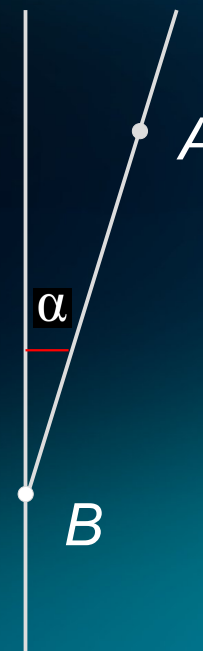
$$\frac{1}{\operatorname{tg}(7^\circ)} |x_2 - x_1| < |y_2 - y_1|$$



Школьная геометрия

$$A(x_1, y_1) \quad B(x_2, y_2)$$

$$8 \cdot |x_2 - x_1| < |y_2 - y_1| \quad !$$

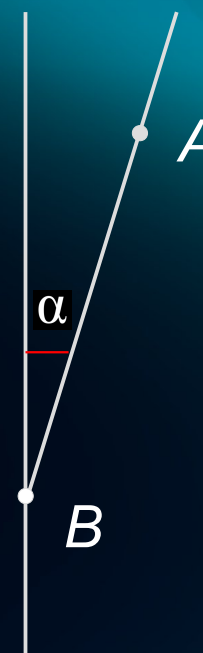


Школьная геометрия

Находятся ли две точки на почти вертикальной прямой и находятся ли они достаточно далеко друг от друга?

$$8 \cdot |x_2 - x_1| < |y_2 - y_1| \quad \&\&$$

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 > MIN$$



Школьная геометрия

Находятся ли две точки на почти вертикальной прямой и находятся ли они достаточно далеко друг от друга?

А можно ли еще проще, если допустить небольшую погрешность?



Школьная геометрия

Находятся ли две точки на почти вертикальной прямой
и находятся ли они достаточно далеко друг от друга?



Школьная геометрия

Находятся ли две точки на почти вертикальной прямой
и находятся ли они достаточно далеко друг от друга?

$$8 \cdot |x_2 - x_1| < |y_2 - y_1| \quad \&\&$$

$$\left(|x_2 - x_1| > \text{min} \quad || \quad |y_2 - y_1| > \text{min} \right)$$



Школьная геометрия

Находятся ли две точки на почти вертикальной прямой и находятся ли они достаточно далеко друг от друга?

Сможем ли еще упростить?

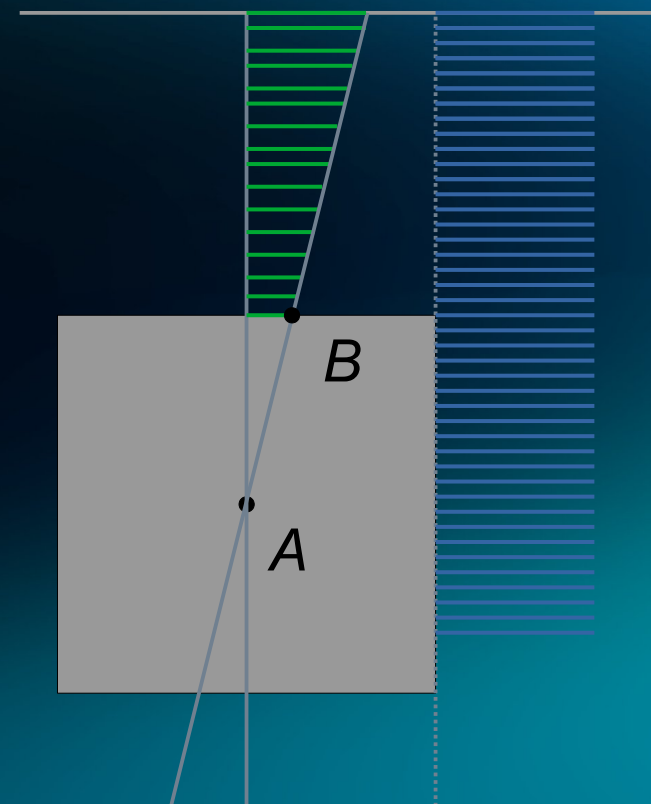


Школьная геометрия

Находятся ли две точки на почти вертикальной прямой и находятся ли они достаточно далеко друг от друга?

$$8 \cdot |x_2 - x_1| < |y_2 - y_1| \quad \&\&$$

$$(|x_2 - x_1| > \mathit{min} \quad || \quad |y_2 - y_1| > \mathit{min})$$



Школьная геометрия

Находятся ли две точки на почти вертикальной прямой и находятся ли они достаточно далеко друг от друга?

$$|y_2 - y_1| > \min \ \&\&$$

$$8 \cdot |x_2 - x_1| < |y_2 - y_1| \quad \text{!!!}$$

Школьная геометрия

Было

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} > \min$$

$$\frac{|x_2 - x_1|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} < \sin(7^\circ)$$

???

Стало

$$|y_2 - y_1| > \min \ \&\& \\ 8 \cdot |x_2 - x_1| < |y_2 - y_1|$$

!!!

Школьная геометрия

Benchmark	Score	Error	Units
Было	3,797	$\pm 0,035$	us/op
Стало	1,527	$\pm 0,014$	us/op

Разница в 2.5 раза!



Алгоритмические трюки

Алгоритмические трюки

```
int a = (int) (v * 0.381966);
```

Вычисление целочисленной части числа,
например, определить индекс при делении массива на части

Алгоритмические трюки

```
int a = (int) (v * 0.381966);
```

AI, оптимизируй

1. *// Константа для оптимизации вычислений*

```
private static final double GOLDEN_RATIO = 0.381966;  
int a = (int) (v * GOLDEN_RATIO);
```
2.

```
private static final double GOLDEN_RATIO =  
    (Math.sqrt(5) - 1) / 2;
```
3.

```
int a = (int) (v * 381966L / 1_000_000);
```



Алгоритмические трюки

```
int a = (int) (v * 0.381966);
```

$$0.381966 \approx \frac{1}{4} + \frac{1}{8} + \frac{1}{128} \quad \sum 1/2^k$$

Дешевая аппроксимация из сдвигов вправо

```
int a = (v >> 2) + (v >> 3) + (v >> 7);
```

Алгоритмические трюки

Benchmark	Score	Error	Units
shifts	3,716	$\pm 0,030$	us/op
division	5,060	$\pm 0,197$	us/op

Разница в 1.4 раза!

Алгоритмические трюки

Counting sort(short[] a) – сортировка подсчетом

$-32768 \leq a[i] \leq 32767,$

а индексы массива должны быть $i \geq 0$

```
for (int i = high; i > low; ++count[a[--i] + 32768]);
```

Алгоритмические трюки

Counting sort(short[] a) – сортировка подсчетом

$-32768 \leq a[i] \leq 32767,$

а индексы массива должны быть $i \geq 0$



```
for (int i = high; i > low; ++count[a[--i] + 32768]);
```

```
void test(int low, int high, int[] a, int[] count) {  
    int i = high; // начинаем с индекса 'high' и движемся к 'low'  
    while (i > low) {  
        // Уменьшаем 'i' перед доступом к массиву 'a'  
        int index = a[--i] + 32768;  
        // Инкрементируем соответствующий элемент в 'count'  
        ++count[index];  
    }  
}
```

Алгоритмические трюки

Counting sort(short[] a) – сортировка подсчетом

$-32768 \leq a[i] \leq 32767,$

а индексы массива должны быть $i \geq 0$

```
for (int i = high; i > low; ++count[a[--i] + 32768]);
```

но будет чуть быстрее

```
for (int i = high; i > low; ++count[a[--i] & 0xFFFF]);
```

Алгоритмические трюки

Benchmark	Score	Error	Units
sum	1,877	$\pm 0,049$	us/op
mask	0,663	$\pm 0,121$	us/op

Выигрыш в 2.8 раза!

Алгоритмические трюки

Замена нескольких `boolean` одним `int` с битовой маской (битовая упаковка)

```
boolean b1, b2, b3;  
result = b1 && b2 && b3;
```

```
int bits;  
result = (bits & MASK) == MASK;
```

```
bits:    *****010
```

```
MASK:    00000111      (MASK = 7)
```



```
// Самый вероятный false ставим первым  
return b3 && b2 && b1;
```

Алгоритмические трюки

Замена нескольких `boolean` одним `int` с битовой маской

	З/Л	Пн	Вт	Ср	Чт	Пт	Сб	Вс	
<code>schedule:</code>	0	0	0	1	0	0	1	0	(18)
<code>WEEKEND_MASK =</code>	0	0	0	0	0	0	1	1	(3)

Летает ли самолет по сб или вс:

`(schedule & WEEKEND_MASK) > 0`

Летает ли самолет по сб и вс:

`(schedule & WEEKEND_MASK) == WEEKEND_MASK`



Алгоритмические трюки

Benchmark	Score	Error	Units
boolean	1,264	$\pm 0,007$	us/op
mask	1,008	$\pm 0,016$	us/op

Разница составляет 20%!



Алгоритмические трюки

Является ли число n степенью двойки?

```
/**  
 * Проверяет, является ли число  $n$  степенью двойки.  
 * @param  $n$  Проверяемое число.  
 * @return true, если  $n$  – положительная степень двойки.  
 */  
public static boolean isPowerOfTwo(long n) {  
    return n > 0 && (n & (n - 1)) == 0;  
}
```



Объяснение логики

Проверка $n > 0$: Исключает отрицательные числа и ноль, которые не являются степенями двойки

Битовая операция $(n \& (n - 1)) == 0$:

для степеней двойки в двоичном виде только один бит установлен (1000...)

при вычитании 1 все биты после установленного становятся 1 (0111...)

битовое И между n и $n - 1$ даёт 0

Алгоритмические трюки

Является ли число n степенью двойки?

Действительно быстрее:

```
return n > 0 && (n & (n - 1)) == 0;
```



```
Integer.bitCount(n) == 1;
```

чем с циклом:

```
while (n > 1 && n % 2 == 0) {  
    n = n / 2;  
}  
return n == 1;
```

Benchmark	Score	Error	Units
isPowerWhile	1,403	± 0,005	ns/op
isPowerAnd	0,935	± 0,004	ns/op

Быстрее в 1.5 раза!

Алгоритмические трюки

Для поразрядной сортировки `float` как `int`

```
private static int f2i(float f) {  
    int x = Float.floatToRawIntBits(f);  
    return x < 0 ? ~x : x | (1 << 31);  
}  
  
return x ^ ((x >> 31) | 0x80000000);
```



Алгоритмические трюки

type	size	"if"	"xor"	speedup
RANDOM	3000	98,601	67,732	1.46
RANDOM	40000	1524,875	891,409	1.71
RANDOM	800000	32224,975	20047,253	1.61
RANDOM	5000000	209115,684	141213,287	1.48

Быстрее в 1.5+ раза!

Алгоритмические трюки

1. *Pair insertion sort*, вставляйте на нужные места сразу пару элементов вместо одного
2. Применяйте *Merging sort* для сортировки почти упорядоченных данных
3. Используйте *Radix sort* вместо Quicksort при сортировке большого количества случайных чисел
4. ...



Типы данных

Primitives vs. wrappers

Не используйте wrapper'ы

Не используйте wrapper'ы

```
Long value = getResult();
```

```
public void setSize(Integer size);
```

```
public Boolean isValid();
```

```
int - Integer
```

```
long - Long
```

```
short - Short
```

```
char - Character
```

```
byte - Byte
```

```
float - Float
```

```
double - Double
```

```
boolean - Boolean
```

Primitives vs. wrappers

Используйте примитивы

Не используйте wrapper-ы

```
long value = getResult();
```

```
public void setSize(int size);
```

```
public boolean isValid();
```

int - Integer

long - Long

short - Short

char - Character

byte - Byte

float - Float

double - Double

boolean - Boolean

Primitives vs. wrappers



Использование примитивных типов `int` вместо `Integer`:

нет автобоксинга (`Integer -> int`) и создания лишних объектов

ускоряем выполнение на ~20-30% по сравнению с версией с `Integer`



Primitives vs. wrappers



Сортировка 60 000 случайных чисел, %

Параллельные вычисления

Параллельные вычисления

Пакет `java.util.concurrent`

Классы

- * `RecursiveTask`
- * `RecursiveAction`
- * `CountedCompleter`
- ...

Методы

- * `compute()` — реализация алгоритма
- * `fork()`, `invoke()`, `invokeAll()`, `tryComplete()`

Параллельные вычисления

Массив

```
-----  
|a1|a2|a3| | | | | | | |an|  
-----
```

`processData(a, low, high)` —
обрабатывает данные в диапазоне [low..high)

Параллельные вычисления

```
class MyTask extends RecursiveAction
...
@Override protected void compute() {
    if (high - low < MIN_PARALLEL_SIZE) {
        processData(a, low, high);
    } else {
        int mid = (low + high) >>> 1;
        invokeAll( // !!!
            new MyTask(a, low, mid),
            new MyTask(a, mid, high)
        );
    }
}
```

```
new MyTask(...).invoke();
```

Параллельные вычисления

```
@Override protected void compute() {  
    if (high - low < MIN_PARALLEL_SIZE) {  
        int mid = low + (high - low) / 2; // [1] (low + high) >>> 1;  
        processData(a, low, mid); // [2] ??  
        processData(a, mid + 1, high); // [2] ??  
    } else {  
        int mid = low + (high - low) / 2; // [3] ??  
        MyTask leftTask = new MyTask(a, low, mid);  
        MyTask rightTask = new MyTask(a, mid + 1, high);  
        leftTask.fork();  
        rightTask.compute(); // выполняем сразу правую задачу  
        leftTask.join();  
    }  
}
```



Параллельные вычисления

Использование `fork()` вместо `invokeAll()`:

`leftTask.fork()` позволяет запускать задачу асинхронно, а `rightTask.compute()` выполняется сразу в текущем потоке, это уменьшает количество активных задач и улучшает использование ресурсов



Параллельные вычисления: еще вариант

```
class MyTask extends CountedCompleter
...
@Override public void compute() {
    while (high - low > MIN_PARALLEL_SIZE) {
        int mid = (low + high) >>> 1;
        addToPendingCount(1);
        new MyTask(this, a, low, mid).fork();
        low = mid; // обрабатываем правую часть
    }
    processData(a, low, high);
    tryComplete();
}
```

Количество
потоков меньше
в 2 раза



Non thread-safe классы

Используйте `ArrayList` вместо `CopyOnWriteArrayList`

Используйте `StringBuilder` вместо `StringBuffer`

Используйте `HashMap` вместо `ConcurrentHashMap`

...

в однопоточном приложении!

Non thread-safe классы

Чтение из списка `list.get(n/2);`

Benchmark	Score	Error	Units
ArrayList	0,039 ± 0,004		us/op
CopyOnWriteArrayList	0,529 ± 0,853		us/op

Разница в 13 раз!



Замена `ConcurrentHashMap` на `HashMap`

Если нет необходимости в потокобезопасной карте, `HashMap` работает быстрее и экономит память, в данном случае явно не используется многопоточность, поэтому `HashMap` — оптимальный выбор

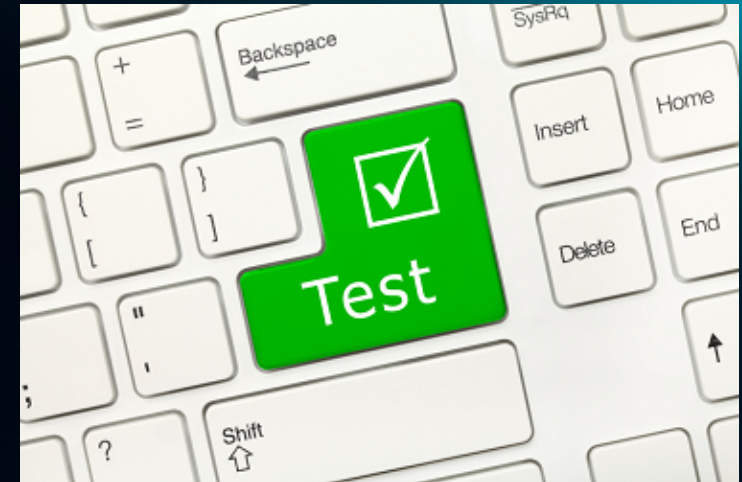
Тестирование

Типы тестов:

- тесты на корректность (unit tests)
- тесты на производительность (benchmarking)
- smoke тесты
- нагрузочные тесты (stress tests)
- тесты на каждый баг

Что важно:

- тестовое покрытие (test coverage)
- прогрев (warmup)

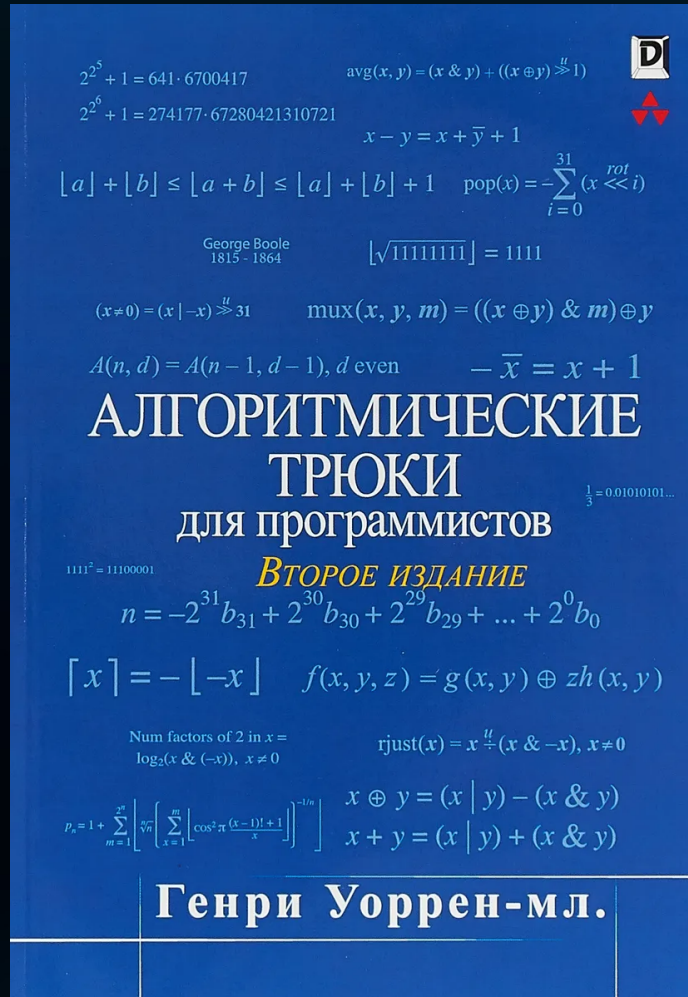


Итоги

Задача	Оценка (max 10)
Строки	10 ✓
Коллекции	2 ✗
Геометрия	3 ✗
Трюки с байтами	6 ±
Алгоритмы	3 ✗
Булевская логика	5 ±
Типы данных	10 ✓
Параллелизм	7 ✓



Что почитать



2²⁵ + 1 = 641 · 6700417 avg(x, y) = (x & y) + ((x ⊕ y) >> 1)

2²⁶ + 1 = 274177 · 67280421310721 x - y = x + \bar{y} + 1

$\lfloor a \rfloor + \lfloor b \rfloor \leq \lfloor a + b \rfloor \leq \lfloor a \rfloor + \lfloor b \rfloor + 1$ pop(x) = $\sum_{i=0}^{31} (x \ll i) \text{ rot}$

George Boole 1815 - 1864 $\lfloor \sqrt{11111111} \rfloor = 1111$

(x ≠ 0) = (x | -x) >> 31 mux(x, y, m) = ((x ⊕ y) & m) ⊕ y

A(n, d) = A(n - 1, d - 1), d even - \bar{x} = x + 1

АЛГОРИТМИЧЕСКИЕ ТРЮКИ для программистов

1111² = 11100001 $\frac{1}{3} = 0.01010101\dots$

ВТОРОЕ ИЗДАНИЕ

$$n = -2^{31}b_{31} + 2^{30}b_{30} + 2^{29}b_{29} + \dots + 2^0b_0$$

$\lceil x \rceil = -\lfloor -x \rfloor$ f(x, y, z) = g(x, y) ⊕ zh(x, y)

Num factors of 2 in x = log₂(x & (-x)), x ≠ 0 rjust(x) = x >> (x & -x), x ≠ 0

$$p_n = 1 + \sum_{m=1}^{2^n} \left[\sqrt[n]{\sum_{i=1}^m \left[\cos^2 \pi \frac{(i-1)+1}{x} \right]} \right]^{-1/n}$$

$x \oplus y = (x | y) - (x \& y)$
 $x + y = (x | y) + (x \& y)$

Генри Уоррен-мл.

- Деление на константы
- Округление к степени 2
- Вычисление остатков от деления
- Методы подсчета количества битов
- Перестановка битов и байтов
- Алгоритмы сжатия и расширения
- Коды с коррекцией ошибок
- Формулы для простых чисел

Выводы

Используйте ИИ-помощников

Применяйте алгоритмические трюки

Смотрите на задачу шире

Математика \neq программирование

Включайте голову!

Тесты спасают наш мир!

Спасибо!

Владимир Ярославский

vlv.spb.ru@mail.ru