

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ — ФУНДАМЕНТ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММЫ

АЛЕКСЕЙ ПАНКРАТЬЕВ,
ВЕДУЩИЙ ПРОГРАММИСТ

СЕНТЯБРЬ 2019

ЗАГАДКА



Ваша лаборатория ищет средство от смертельной болезни.

Вам на испытание пришла партия из 1 000 пробирок с лекарством, но в одной из них по ошибке отправили вместо лекарства ядовитый реактив. Внешне он ничем не отличается от медикамента.

Вам нужно как можно скорее передать пробирки в больницы для запуска клинического теста, но отправлять отравленную пробирку нельзя: погибнут люди. Тесты всех пробирок займут месяцы, это очень долго. Но у вас есть лабораторные мыши. Вы знаете, что лекарство безвредно для них, а даже капля яда их убьёт за сутки. Но у вас только 10 мышей, а пробирок — 1 000.

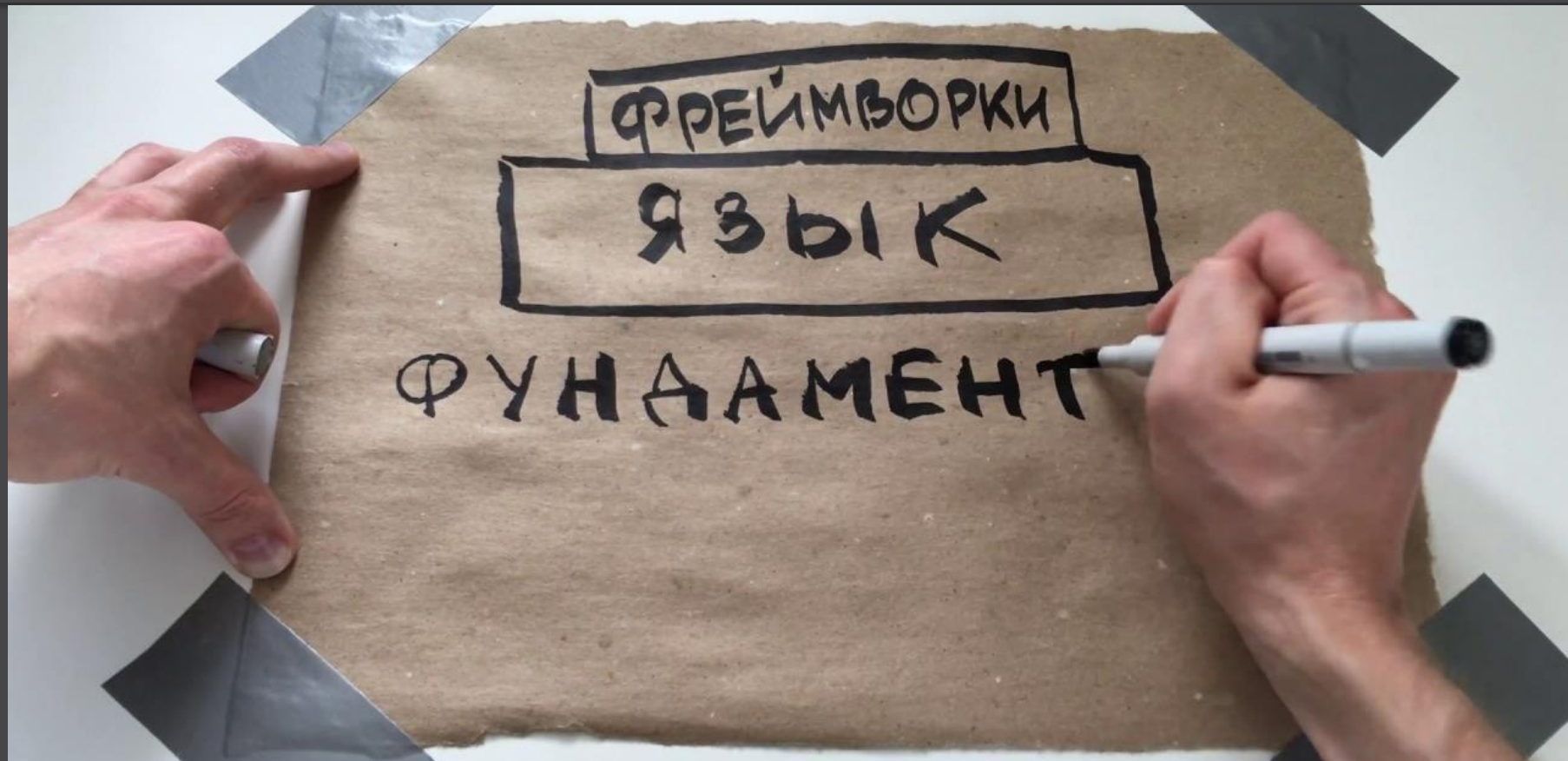
Как определить, где яд, как можно быстрее? За какое время можно гарантированно найти пробирку с ядом?

О СЕБЕ

- Работаю в ЕРАМ с 2016 на проекте, посвященном разработке решений для страховых компаний.
- До этого более 10 лет занимался разработкой в сфере геоинформационных технологий и инженерных изысканий.
- С начала 2018 г. регулярно участвую в OpenSource проектах.



О ЧЕМ ЭТОТ ДОКЛАД



ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ

Для описания вычислительной сложности используются разные методы («нотации»):

- $O(f(n))$ — (Big-O) — верхняя граница, «не хуже чем»
- $o(f(n))$ — (Little-o) — верхняя граница, «лучше чем»
- $\Omega(f(n))$ — (Omega) — нижняя граница, «не лучше чем»
- $\Theta(f(n))$ — (Theta) — точная оценка.

На практике проще всего использовать O-большое как приблизительную оценку сверху, «время работы нашего алгоритма растет не быстрее чем такая-то функция по мере роста объема входных данных».

ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ

$O(1)$ — константная сложность

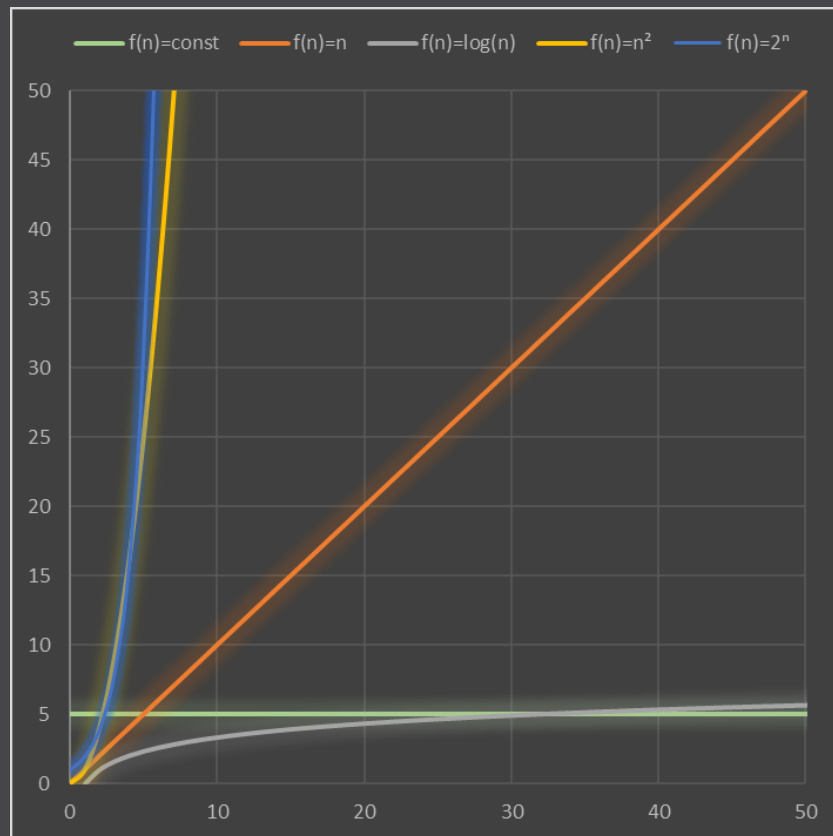
$O(\ln(n))$ — логарифмическая сложность

$O(n)$ — линейная сложность

$O(n^2)$ — квадратичная сложность

$O(2^n)$ — экспоненциальная сложность

и т.д.



МЫШИ И ПРОБИРКИ. НАИВНОЕ РЕШЕНИЕ

```
var laboratory = new Laboratory();
IMedicine[] samples = laboratory.GimmeSamples(1000);
IMouse[] mice = laboratory.GimmeMice(10);

var mouse = mice[0];

int i = -1;
do
{
    i++;
    if (i >= samples.Length)
        throw new InvalidOperationException("Not fair! All samples are clear!");

    mouse.TakeMedicine(samples[i]);
} while (mouse.IsAlive);

Console.WriteLine($"Sample number {i} (starting from zero) is poisoned");
```

МЫШИ И ПРОБИРКИ. ПАРАЛЛЕЛИЗМ

```
var laboratory = new Laboratory();
var samples = new ConcurrentQueue<IMedicine>(laboratory.GimmeSamples(1000));
IMouse[] mice = laboratory.GimmeMice(10);


IMedicine poisonedSample = null;

Parallel.ForEach(mice, mouse => {
    while (poisonedSample == null && samples.TryDequeue(out var sample))
    {
        mouse.TakeMedicine(sample);
        if (!mouse.IsAlive)
        {
            poisonedSample = sample;
            break;
        }
    }
});


Console.WriteLine($"Sample number {poisonedSample.Index} is poisoned");
```


МЫШИ И ПРОБИРКИ. ОПТИМИЗАЦИЯ

День 1

1..100 — мышь 1,
101..200 — мышь 2,
201..300 — мышь 3,
301..400 — 
401..500 — мышь 5,
501..600 — мышь 6,
601..700 — мышь 7,
701..800 — мышь 8,
801..900 — мышь 9,
901..1000 — мышь 10

День 2

301..311 — мышь 1,
312..322 — мышь 2,
323..333 — мышь 3,
334..344 — мышь 5,
345..355 — мышь 6,
356..366 — мышь 7,
367..377 — 
378..388 — мышь 9,
389..399 — мышь 10,
400 — в запасе

День 3

367 — мышь 1,
368 — мышь 2,
369 — мышь 3,
370 — мышь 5,
371 — мышь 6,
372 — мышь 7,
373 — мышь 9,
374 — мышь 10,
375..377 — в запасе

День 4

375 — мышь 1
376 — 
377 — в запасе

МЫШИ И ПРОБИРКИ. ОПТИМИЗАЦИЯ

```
do
{
    var sessions = DistributeSamplesByMice(suspiciousSamples, miceAlive);
    Parallel.ForEach(sessions, session =>
    {
        foreach (var sample in session.Samples)
            session.Mouse.TakeMedicine(sample);
        if (!session.Mouse.IsAlive)
        {
            suspiciousSamples = session.Samples.ToArray();
            if (suspiciousSamples.Length == 1)
                poisonedSample = suspiciousSamples[0];
        }
    });
    if (miceAlive.All(m => m.IsAlive))
        poisonedSample = sessions.AdditionalSample;
    miceAlive = miceAlive.Where(m => m.IsAlive).ToArray();
} while (poisonedSample == null && miceAlive.Length > 0);

if (poisonedSample == null && miceAlive.Length == 0)
    throw new InvalidOperationException("We ran out of mice");

Console.WriteLine($"Sample number {poisonedSample.Index} is poisoned");
```

МЫШИ И ПРОБИРКИ. ОПТИМИЗАЦИЯ



- Сложность — $O(\log(n))$: увеличение числа пробирок в 10 раз приведет к добавлению одной проверки.
- Почти на каждой проверке погибает одна мышь.
- Масштабируемость ограничена: количество пробирок не должно быть больше удвоенного факториала числа мышей (7 257 600).

ПРОБЛЕМА 1

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА

Дано:

- На сайте ГУ МВД выложен и регулярно обновляется список недействительных паспортов
- Формат данных — CSV, упакованный в архив
- > 120 миллионов строк, 1.5 ГБ несжатых данных

Необходимо:

- Загрузить данные в БД
- Регулярно обновлять данные в БД (добавлять и удалять).



НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА

	PASSP_SERIES	PASSP_NUMBER
1	0100	362424
2	0100	362426
3	0100	362427
4	0100	362428
5	0100	362429
6	0100	362430
7	0100	362431
8	0100	362432
9	0100	362433
10	0100	362434
11	0100	362435
12	0100	362439
13	0100	362442
14	0100	362443
15	0100	362447
16	0100	362448
17	0100	362450
18	0100	362458
19	0100	362460
20	0100	362461

Формат хранения зафиксирован в техническом задании: таблица в БД с полями типа текстового типа: PASSP_SERIES, PASSP_NUMBER.

Первичная загрузка происходит через BULK INSERT, что-то принципиально улучшить тут не получится.

Время первичной загрузки 3-5 часов (в зависимости от железа).

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА. НАИВНОЕ РЕШЕНИЕ

```
foreach (var passport in passports)
{
    if (!db.PassportExists(passport)) ← 50 мс
        db.InsertPassport(passport); ← 70 мс
}
```

- Не позволяет определить, какие данные следует удалить
- Неприемлемо медленно

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА. С ЧИСТОГО ЛИСТА

```
db.DeleteAllPassports();  
db.BulkInsert(passports);
```

- Если транзакция не закоммичена, во время загрузки невозможно проверять контрагентов
- Если коммитить по ходу загрузки, сервис будет выдавать неверные результаты
- Лог транзакций растет огромными темпами
- База «пухнет», т.к. очищенные страницы не переиспользуются сразу

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА. ~~USE THE FORCE~~ LUKE SQL

```
SELECT
    ISNULL(existing.PASSP_SERIES, new.PASSP_SERIES) PASSP_SERIES,
    ISNULL(existing.PASSP_NUMBER, new.PASSP_NUMBER) PASSP_NUMBER,
    IIF(existing.PASSP_SERIES IS NULL, 'DELETE', 'INSERT')
FROM PASSPORT existing
FULL OUTER JOIN TMP_PASSPORT new ON
    new.PASSP_SERIES = existing.PASSP_SERIES AND
    new.PASSP_NUMBER = existing.PASSP_NUMBER
WHERE existing.PASSP_SERIES IS NULL OR
    new.PASSP_SERIES IS NULL
```

- Требуется вдвое больше места для хранения
- Те же проблемы с неконтролируемым ростом БД

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА. I WILL WRITE MY OWN MERGE JOIN!



```
var existingPassports = db.GetAllPassports() ⬅ ~8 мин.  
    .OrderBy(p => p.Series)  
    .ThenBy(p => p.Number);
```

```
var newPassports = File.ReadAllLinesAsEnumerable(filePath) ⬅ ~8 мин.  
    .Select(line => Passport.FromCsvLine(line));
```

```
var newPassportsSorted = newPassports ⬅ ~30 мин. (сортировка)  
    .OrderBy(p => p.Series)  
    .ThenBy(p => p.Number);
```

```
var result = Merge(existingPassports, newPassports);
```

Уже гораздо лучше, но

- Сравнительно большое потребление памяти (> 6 ГБ)
- Сортировка занимает большую часть времени.

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА. BITMAP SORT

```
public class InvalidPassportsCollection
{
    private const int TOTAL_SERIES = 10_000;
    private const int TOTAL_NUMBERS= 1_000_000;

    private readonly BitArray[] _invalidPassports = new BitArray[TOTAL_SERIES];
    private readonly List<string> _incorrectRecords = new List<string>();

    public void Add(string series, string number)
    {
        int s, n;
        if (int.TryParse(series, out s) && int.TryParse(number, out n))
        {
            if (_invalidPassports[s] == null)
            {
                _invalidPassports[s] = new BitArray(TOTAL_NUMBERS);
            }
            _invalidPassports[s][n] = true;
        }
        else
        {
            _incorrectRecords.Add(string.Format("{0},{1}", series, number));
        }
    }
}
```

НЕДЕЙСТВИТЕЛЬНЫЕ ПАСПОРТА. BITMAP SORT USAGE

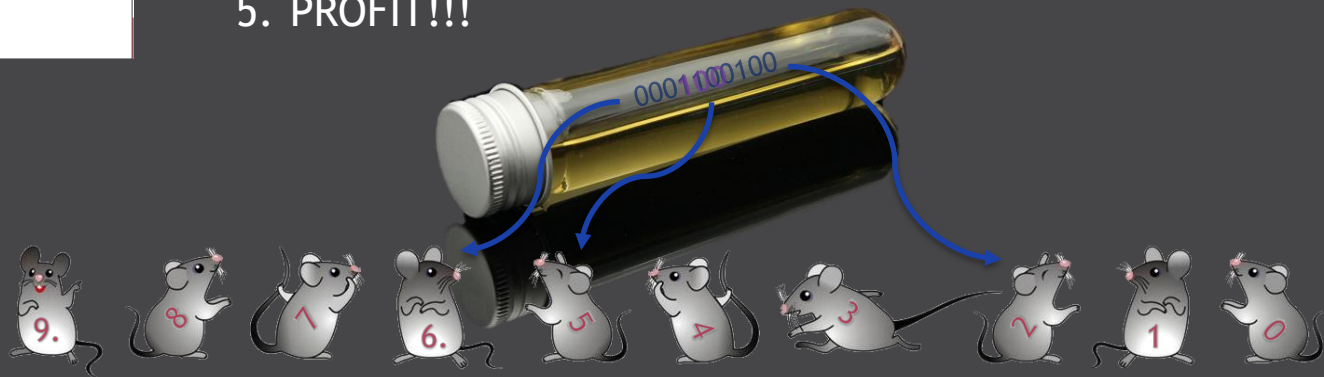
```
InvalidPassportsCollection existingPassports = null;  
InvalidPassportsCollection newPassports = null;  
Task.WaitAll(  
    Task.Run(() => { existingPassports = GetExistingPassports(); }),  
    Task.Run(() => { newPassports = GetNewPassports(); })  
);  
  
var diff = newPassports.GetDiff(existingPassports);
```

- Сложность — линейная, $O(n)$
- Обновление паспортов стало выполняться за 10 минут
- Потребление памяти — умеренно большое (~1.2 ГБ)
- Сервис остается доступен во время обновления данных
- Нет избыточной нагрузки на СУБД, нет неконтрольного роста БД и журнала транзакций.

МЫШИ И ПРОБИРКИ. ПРОГРАММИСТСКИЙ ПОДХОД

Номер пробирки	Номер мыши									
	9	8	7	6	5	4	3	2	1	0
0										
1										
...										
100										
...										
255										
...										
512										
...										
999										

1. Пронумеруем пробирки и мышей
2. Преобразуем номер каждой пробирки в двоичную запись
3. «Угостим» каждую мышь из тех пробирок, где стоит единица
4. ???
5. PROFIT!!!



МЫШИ И ПРОБИРКИ. РЕЗУЛЬТАТ

- Нахождение яда за одни сутки!
- Сложность — $O(1)$: мы всегда делаем одинаковое количество проверок.
- В худшем случае погибают 9 мышей.
- Масштабируемость сильно ограничена: каждое удвоение числа пробирок требует одну дополнительную мышь.

<https://github.com/Pankraty/laboratory-demo>



ПРОБЛЕМА 2

ОБЪЕДИНЕНИЕ РЕГИОНОВ

ОБЪЕДИНЕНИЕ РЕГИОНОВ. О ПРОЕКТАХ CLOSEDXML.*



ClosedXML.Report

<https://github.com/ClosedXML/ClosedXML.Report>

Библиотека для генерации отчетов в формате XLSX, базирующаяся на ClosedXML.

Автор — Алексей Рожков (a.k.a. b0bi79), Санкт-Петербург, Россия



ClosedXML

<https://github.com/ClosedXML/ClosedXML>

Библиотека общего назначения для чтения и редактирования файлов XLSX, базирующаяся на OpenXML.

«Хозяин» репозитория и один из авторов — Francois Botha (a.k.a. igitur), Cape Town, SA

ОБЪЕДИНЕНИЕ РЕГИОНОВ. ОПИСАНИЕ ПРОБЛЕМЫ

Conditional Formatting Rules Manager

Show formatting rules for: **This Worksheet**

Rule (applied in order shown)	Format	Applies to	Stop If True
Formula: =\$G15="Visa"	AaBbCcYyZz	=E\$15:G\$15	<input checked="" type="checkbox"/>
Formula: =\$G14="Visa"	AaBbCcYyZz	=E\$14:G\$14	<input checked="" type="checkbox"/>
Formula: =\$G13="Visa"	AaBbCcYyZz	=E\$13:G\$13	<input checked="" type="checkbox"/>
Formula: =\$G12="Visa"	AaBbCcYyZz	=E\$12:G\$12	<input checked="" type="checkbox"/>

Buttons: New Rule..., Edit Rule..., Delete Rule, OK, Close, Apply

Worksheet Data:

Customer: **Tom Sawyer Diving Centre**

Address: **&"{{Addr1}}"&" {{Addr2}}**

City: **{{City}}**

Order No: **Order No** | Sale date: **Sale date** | Ship date: **Ship date**

Phone: **{{Phone}}**

State	Country	Zip
St. Croix	US Virgin Island	00820

Addr1	Addr2	Payment method	Items total	Tax rate	Amount paid
		Visa	4,807.00	0.00	4,807.00
		Visa	3,065.00	0.00	3,065.00
		Visa	6,935.00	0.00	6,935.00
		Visa	31,219.95	0.00	31,219.95
		Credit	9,634.00	0.00	9,634.00
		Credit	3,640.00	0.00	3,640.00
		Credit	4,317.75	0.00	4,317.75
		Cash	3,596.00	0.00	3,596.00
6 Sugarloaf	Suite 103	Cash	2,150.00	8.50	2,150.00
			Total		

Phone: **504-798-3022** | Fax: **504-798-7772**

ОБЪЕДИНЕНИЕ РЕГИОНОВ. ПОСТАНОВКА ЗАДАЧИ

Дано:

- Коллекция прямоугольных регионов, которые могут соприкасаться, пересекаться или располагаться независимо

Необходимо

- Сформировать новую коллекцию регионов, покрывающих те же ячейки, без пересечений

ОБЪЕДИНЕНИЕ РЕГИОНОВ. ПОСТАНОВКА ЗАДАЧИ

	A	B	C	D	E
1	B2:C3, C2:D4, B4:B5, B6:E6, C5:D8, B7:B9				
2					
3					
4					
5					
6					
7					
8					
9					
10					

	A	B	C	D	E
1	B2:D8, E6:E6, B9:B9				
2					
3					
4					
5					
6					
7					
8					
9					
10					

ОБЪЕДИНЕНИЕ РЕГИОНОВ. РЕШЕНИЕ

	A	B	C	D	E	F	G	H	I	J
1						Номер строки	Битовая карта			
2						2	1	1	1	0
3						3	1	1	1	1
4										
5										
6						6	1	1	1	1
7						7	0	1	1	1
8										
9										
10										
11						11	0	0	0	0



<https://github.com/ClosedXML/ClosedXML/blob/develop/ClosedXML/Excel/Ranges/XLRangeConsolidationEngine.cs>

ЗАКЛЮЧЕНИЕ

Программу можно ускорять разными способами:

- микрооптимизациями — на десятки процентов
- распараллеливанием — в разы
- применяя более оптимальный алгоритм — на порядки

Если нет возможности заниматься этим на проекте — добро пожаловать в Open Source!



<https://hacktoberfest.digitalocean.com>

ССЫЛКИ

1. Задача про мышей https://zen.yandex.ru/media/code/zadacha-o-dvoichnoi-myshi-i-tysiache-probirok-5d284eacf2df2500adc95b35#comment_85044304
2. Оценка алгоритмической сложности <https://www.intuit.ru/studies/courses/683/539/lecture/12149?page=3>
3. Алгоритмы и структуры данных https://www.youtube.com/playlist?list=PLrCZzMib1e9pDxHYzmEzMmnMMUK-dz0_7
4. Библиотека ClosedXML <https://github.com/ClosedXML/ClosedXML>
5. Библиотека ClosedXML.Report <https://github.com/ClosedXML/ClosedXML.Report>
6. Я на гитхабе <https://github.com/Pankraty>
7. Hacktoberfest <https://hacktoberfest.digitalocean.com/>

**ПРИ ПОДГОТОВКЕ
ДОКЛАДА НИ ОДНА МЫШЬ
НЕ ПОСТРАДАЛА**



ПАНКРАТЪЕВ АЛЕКСЕЙ

ALEKSEI_PANKRATEV@EPAM.COM