

Алгоритм структурной числовой симметрии(СЧС)

Титульная страница

Автор: Ющенко Михаил Юрьевич

Дата: 19.05.2025 год

Аннотация

Настоящий проект посвящен исследованию и разработке оригинального алгоритма структурной числовой симметрии (СЧС), который представляет собой универсальный подход к оперированию числами и выявлению закономерностей в их структуре. Предлагаемый алгоритм позволяет проводить эффективное деление числа на части, производить расчеты и объединять результаты, обеспечивая высокую степень точности и надежности.

Основные положения и этапы алгоритма включают:

Разбиение натурального числа на части с соблюдением близости по разрядности.

Последующее умножение каждой части на единое натуральное число.

Объединение полученных результатов в единую конструкцию.

Анализ полученной величины и сравнение с традиционным методом умножения.

Практическое применение данного алгоритма возможно в различных областях, включая теорию чисел, информатику, биологию, экономику, химию, музыку и литературу (всего свыше 10 дисциплин). Проведенные экспериментальные исследования подтвердили стабильность и устойчивость алгоритма, что открывает новые перспективы для дальнейших исследований и внедрения в практику.

Проект разработан Ющенко Михаилом Юрьевичем и опубликован под лицензией All Rights Reserved.

Любое копирование или/и распространение без прямого согласия автора, строго запрещено!

Содержание:

1. История возникновения Структурной числовой симметрии (СЧС)
 2. История возникновения Алгоритма СЧС
 3. Формулировка алгоритма СЧС
 4. Пояснения терминов к формулировки СЧС
 5. Примеры по каждому термину
 6. Обоснование научности
 7. Принцип работы алгоритма
 8. Статистика проверок
 9. Примеры по каждому правилу.
 10. Возможные применения в реальных задачах
 11. Лицензия и контакты.
-
-

1. История возникновения идеи

Как родилась гипотеза о СЧС (Структурной Числовой Симметрии),которая позже стала явлением в связи её доказательством :

В 2025 году, 4 мая, я, Ющенко Михаил Юрьевич столкнулся с задачей:

Вычислить вручную ($999 \wedge 9999$) * 3. Это число оказалось слишком большим для прямого умножения. Тогда я попробовал разбить его на части, умножить каждую часть на коэффициент, а потом объединить результаты. Но данный подход выдал колоссальное кол-во ошибок, но я продолжал вытаскивать из этого числа всё новые и новые числа, эти все неудачные попытки, привели меня к выведению

собственной гипотезы. Так родилась гипотеза структурной числовой симметрии, суть которой заключается в следующем:

В рамках десятичной системы счислений, для любого целого числа $N \geq 10$, если его разбить на $m \geq 2$ частей, максимально близких по разрядности, чтобы количество разрядов всех частей отличались не более чем на единицу, при этом $m \in \mathbb{N}$, но не больше разрядности числа N , умножить каждую часть на одно и то же натуральное число k , а затем объединить результаты как десятичное число PQ , то:

Возможно полное совпадение: $PQ = N * k$

Может совпадать начало или конец

Может наблюдаться частичная симметрия: совпадает начало или конец

Совпадают и начало, и конец, при разных разрядностях

Вопрос: Может ли существовать такое число N , для которого ни одно из правил не выполняется?

Она была проверена программно на миллионах чисел. см. файл:

Структурная_числовая_симметрия.jl Ни одного случая без совпадений найдено не было.

Потом я задался вопросом, а что если расширить данную гипотезу для всех натуральных чисел? т.е. для тех, при которых $N < 10$, то тогда я получил более обобщённую формулировку:

В рамках десятичной системы счисления, для любого натурального числа N , если: N разбивается на $m \geq 2$ натуральных частей, максимально близких по разрядности (разница в количестве разрядов между любыми двумя частями не превышает единицу), $m \in \mathbb{N}$, но m при этом не должно превышать количество разрядов числа N , Если $N < 10$, то перед числом дописываются ведущие нули, чтобы его длина стала равна m . Затем каждая часть умножается на одно и то же натуральное число k . Результаты объединяются как строковое представление в десятичное число PQ . Тогда выполняется хотя бы одно из следующих условий: Полное совпадение : $PQ = N * k$. Совпадает начало или конец. Совпадают и начало, и конец, но при этом PQ и $N * k$ имеют разную разрядность.

Вопрос: Может ли существовать такое число N , для которого ни одно из правил не выполняется?

Ответ: Нет, такого числа нет! См.доказательство

Таким образом, гипотеза стала явлением.

2. История возникновения алгоритма для Структурной числовой симметрии.

Ну так, поскольку само N стремиться к бесконечности, а помимо всего прочего совпадения только по началу так и не было найдено, но зато я пришёл к выводу, что если совпадает начало то так же совпадает конец между оператором PQ и классическим произведением $N * k$, при этом PQ и $N * k$ обязательно имеют разные разрядности, это ключевое отличие от полного совпадения. Но при этом обратной зависимости нет. А ещё одно явление - во всех случаях, будь то полное совпадение или совпадение по началу и концу, ну а так же совпадение только по концу, наблюдается следующее явление, а именно, везде и всегда совпадение по концу, так возникла идея о алгоритме для Структурной числовой симметрии.

3. Сам алгоритм:

Алгоритм Структурной Численной Симметрии, который работает, следующем образом: В рамках десятичной системы счисления, для любого натурального числа N , если: N разбивается на $m \geq 2$ натуральных частей, максимально близких по разрядности (разница в количестве разрядов между любыми двумя частями не превышает единицу), $m \in \mathbb{N}$, но не больше разрядности самого числа N . Если $N < 10$, то перед числом дописываются ведущие нули, чтобы его длина стала равна m . Затем каждая часть умножается на одно и то же натуральное число k , результаты объединяются как строковое представление в десятичное число PQ . Тогда обязательно выполняется одно из следующих условий: полное совпадение : $PQ = N * k$, совпадает конец, совпадают и начало и конец, но при этом PQ и $N * k$ имеют разную разрядность.

4. Подробные пояснения к терминам:

- 1.«В рамках десятичной системы счисления».

Это значит, что мы работаем с числами так, как они записаны в привычной нам форме : от 0 до 9, слева направо, с учётом позиции цифры.

- 2.«Натуральное число N ».

Это означает:

N — любое положительное целое число: 1, 2, 3, ..., 100, ..., 1000000.

Не требуется, чтобы оно было простым, чётным или большим.

- 3.«N разбивается на $m \geq 2$ частей».

Это говорит о том, что:

Мы не можем разбить число на 1 часть — минимальное количество частей: 2.

Можно разбивать на 2, 3, 4... части, но не больше, чем количество разрядов в числе N.

- 4.«Максимально близкие по разрядности».

Это значит:

При разбиении все части должны быть очень похожи по разрядности.

Разница в количестве цифр между частями не превышает 1.

- 5.« $m \in \mathbb{N}$, но не больше разрядности самого числа N».

То есть:

Если $N = 1234$ (4 разряда), то m может быть: 2 и 4.

Нельзя разбивать на 5 частей, потому что в числе всего 4 цифры. Нельзя разбивать на 3 части, потому это противоречит принципу работы СЧС, где разница в разрядах не превышает 1.

- 6.«Если $N < 10$, то перед числом дописываются ведущие нули, чтобы его длина стала равна m ».

Это нужно, чтобы малые числа тоже можно было обработать по тем же правилам, что и большие.

- 7.«Каждая часть умножается на одно и то же натуральное число k ».

Значит:

Все части умножаются на один и тот же множитель.

Множитель может быть любым натуральным числом: 1, 2, 3, ...

- 8.«Результаты объединяются как строковое представление в десятичное число PQ».

Это означает:

Умноженные части соединяются как строка, а не как математическая сумма.

Это не просто математическое умножение , а структурное преобразование.

- 9.«Тогда обязательно выполняется одно из следующих условий».

После всех действий всегда:

Совпадает всё число целиком.

Или конец.

Или и начало, и конец , даже если разрядность чисел разная.

Этот принцип пока ни разу не был нарушен за миллионы проверок!

5. Примеры по каждому пояснению из терминов:

1.) Это значит:

Мы работаем с числами так, как мы привыкли — в привычной форме.

Числа записываются цифрами от 0 до 9.

Позиция цифры важна (например, $123 \neq 321$, потому что позиции разные).

Пример:

- $N = 123456789$

- $m = 3$

- $k = 7$

разбиение $\rightarrow ["123", "456", "789"]$

умножение частей $\rightarrow ["861", "3192", "5523"]$ построчное объединение $\rightarrow PQ = "86131925523"$

сравнение частей:

$$N * K = 864197523 \text{ и } PQ = 86131925523$$

Совпадают начало (8) и конец (3)

2.) Не нужно, чтобы число было простым или чётным.

Подходит любое положительное целое число.

Пример:

- $N = 13$

- $m = 2$

- $k = 7$

разбиение $\rightarrow ["1", "3"]$

умножение частей $\rightarrow ["7", "21"]$ построчное объединение $\rightarrow PQ = "721"$

$$N * K = 91$$

сравнение частей:

- $PQ = 721$

- $N * k = 91$

Совпадает только конец (1)

3.) Минимум на 2 части

Можно больше: 3, 4, 5... но не больше длины числа

Пример:

- $N = 101$

- $m = 2$

- $k = 7$

Разбиение $\rightarrow ["10", "1"]$

Умножение частей $\rightarrow ["70", "7"]$ построчное объединение $\rightarrow PQ = "707"$

Сравниваем:

$$N * K = 707 \text{ и } PQ = 707$$

Полное совпадение.

4.) Все части имеют очень похожую разрядность.

Разница между частями не больше одной цифры.

Пример:

- $N = 12345$

- $m = 3$

построчное разбиение числа $(N, m) \rightarrow ["12", "34", "5"]$

Части: 2, 2, 1 → разница не превышает единицу.

Условие выполнено.

5.) Если число имеет 4 цифры, то нельзя делить на 5 частей.

Можно делить на 2 и 4 части.

Пример:

$N = 1234$

кол-во разрядов(N) = 4

- $m = 5 \rightarrow$ Недопустимое значение.
- $m = 3 \rightarrow$ Недопустимое значение, потому что, нарушается правило "не больше одного 1 разряда"
- $m = 2 \rightarrow$ Допустимое значение.

6.) Так можно использовать малые числа в полной системе проверки.

Пример:

- $N = 3$
- $k = 7$
- $m = 2$
- \rightarrow разрядность(N) = 1 \rightarrow дополняем до 2 разрядов \rightarrow "03"

разбиение \rightarrow ["0", "3"]

умножение частей \rightarrow ["0 \times 7=0", "3 \times 7=21"] построчное объединение \rightarrow PQ = "021"

- $N * K = 21$
- После очистки от нулей: PQ = 21 \rightarrow $N * k = 21 \rightarrow$ Полное совпадение

7.) Это важно для сохранения структуры.

где k - натуральное число, может быть любым: 1, 2, 3, ..., 99999999

Пример:

- $N = 1234$
- $m = 2$

- $k = 4$

разбиение → ["12", "34"]

умножение частей → ["48", "136"] построчное объединение → $PQ = "48136"$

$$N * K = 1234 \times 4 = 4936$$

$$PQ = 48136$$

сравниваем PQ и $N * k$

Совпадают начало ("4") и конец ("6")

8.) Это не просто математическое умножение.

Это цифровое преобразование : части умножаются, затем соединяются как строки.

- $N = 899766$

- $m = 2$

- $k = 4$

разбиение → ["899", "766"]

построчное умножение частей → ["3596", "3064"] построчное объединение в число → $PQ = "35963064"$

сравнение PQ и $N * k$

$$PQ = 35963064$$

$$N * K = 899766 \times 4 = 3599064$$

Совпадают начало ("3") и конец ("4")

9.) Всегда будет совпадать:

Полное совпадение

Только конец

И начало, и конец

Ни одного случая без совпадений не найдено

Пример:

- $N = 11$

- $m = 2$

- $k = 7$

разбивание $\rightarrow ["1", "1"]$

умножение частей $\rightarrow ["7", "7"]$ построчное объединение $\rightarrow PQ = "77"$

- $PQ = 77$

- $N * K = 77$

- Сравниваем PQ с $N * k$

- Полное совпадение!

6. Научная значимость.

- Алгоритм работает для всех натуральных чисел, включая простые, составные, большие степени
- Для него есть формальное доказательство, и к тому же он эмпирически проверен на миллионах чисел
- Он имеет структурную инвариантность:
- Если совпадает начало \rightarrow обязательно совпадёт и конец
- Но если совпадает только конец \rightarrow начало может не совпадать
- Доказательство, постольку он был выведен из гипотезы, а в данный момент явления СЧС (Структурной числовой симметрии) , то и доказательство его лежит в рамках СЧС См.доказательство

Это говорит о глубокой закономерности, которая может быть использована в:

- Теории чисел
- Информатике
- Биологии
- Физике
- Химии
- Экономике
- Медицине

- Астрономии
- Музыке
- Литературе
- Истории
- Логистики
- Теории игр
- Психологии
- Философии
- и в других областях

7. Принцип работы алгоритма

Как это работает? (Простыми словами)

- 1.В рамках десятичной системы счисления, берём любое натуральное число N
- 2.Разбиваем его на $m \geq 2$ натуральных частей, близких по разрядности
- 3.Если число маленькое ($N < 10$), дополняем его нулями до нужной длины
- 4.Каждую часть умножаем на натуральное число k
- 5.Результаты соединяем как строку \rightarrow получаем PQ
- 6.Сравниваем PQ с классическим произведением $NK = N \times k$
- 7.Всегда будет хотя бы частичное совпадение!

Примеры работы алгоритма:

N			Разбиение	PQ	NK	Результат
101			["10", "1"]	"707"	"707"	Полное совпадение
135			["13", "5"]	"9135"	"945"	Совпадают начало и конец
1			["1",	"721"	"91"	Совпадает только конец

N		Разбиение	PQ	NK	Результат
3		"3"]	"	"	
1 2 3 4		["12", "34"]	"481 36"	"49 36"	Совпадают начало и конец
1 0 0 1		["10", "01"]	"700 7"	"70 07"	Полное совпадение

Все эти примеры показывают, что начало и конец чисел сохраняют связь, даже если середина меняется.

8. Статистика проверок

Диапазон: от 1 до 10000000, m=2, k=7

- ✓ Полных совпадений: 1430758
- ↔ Совпадают начало и конец: 8560838
- ↔ Совпадает только начало: 0
- ↔ Совпадает только конец: 8404
- ✗ Без совпадений: 0

[Скачать данные](#)

То же самое, но при k=99999999

- 📊 Сводная статистика:
- ✓ Полных совпадений: 10999
- ↔ Совпадают начало и конец: 9969075
- ↔ Совпадает только начало: 0
- ↔ Совпадает только конец: 19926
- ✗ Без совпадений: 0

[Скачать данные](#)

9. Примеры по каждому правилу.

Полное совпадение:

- $N = 101$

- $m = 2$

- $k = 7$

разбиение \rightarrow ["10", "1"]

умножение частей \rightarrow ["70", "7"] построчное объединение в число $\rightarrow PQ = 707$

Сравнение частей

$$NK = 101 \times 7 = 707 \text{ и } PQ = 707$$

Результат: Полное совпадение

Совпадают начало и конец:

- $N = 899766$

- $m = 2$

- $k = 4$

разбиение \rightarrow ["899", "766"]

умножение частей \rightarrow ["3596", "3064"] построчное объединение в число $\rightarrow PQ = 35963064$

сравниваем:

$$NK = 899766 \times 4 = 3599064 \text{ и } PQ = 35963064$$

- Совпадают: "3" и "4"

- Совпадают начало и конец

Совпадает только конец:

- $N = 13$

- $m = 2$

- $k = 7$

разбиение → ["1", "3"]

умножение частей → ["7", "21"] построчное объединение в число → PQ = "721"

сравниваем:

$N * K = 13 \times 7 = 91$ и $PQ = 721$

Совпадает только конец ("1")

10. Возможные применения в реальных задачах

1. В теории чисел

2. В информатике

3. В биологии

4.1 В физике(закон сохранения энергии)

4.2 В физике(специальной теории относительности)

4.3 В физике(квантовой физики)

5. В химии

6. В экономике

7. В медицине

8. В астрономии

9. В музыке

10. В литературе

11. В истории

12. В логистике

13. В теории игр

14. В психологии

15. В философии

11. Лицензия и контакты:

Автор: ([Михаил])([<https://github.com/Misha0966>]) почта для
связи: misha0966.33@gmail.com

Сайт: <https://structuralnumericalsymmetry.ru>

Данный проект распространяется под лицензией All Rights Reserved.

Любое копирование или/и распространение без прямого согласия автора, строго запрещено!

using Printf # Подключает модуль для форматированного вывода (например, @printf)

using CSV # Подключает библиотеку для работы с CSV-файлами

using DataFrames # Подключает тип данных DataFrame для табличного хранения результатов

using Base.Threads # Включает поддержку многопоточности

using ProgressMeter # Позволяет отображать прогресс выполнения циклов

Функция разбивает число N на m частей максимально близких по длине

function split_number_str(N::Integer, m::Integer)

s = string(N) # Преобразует число N в строку

if N < 10 # Если число меньше 10 — дополняем ведущими нулями до длины m

s = lpad(s, m, '0') # Добавляем слева нули до длины m

end

len = length(s) # Определяем общую длину строки

base_len = div(len, m) # Базовая длина части

remainder = len % m # Остаток при делении — сколько частей будут длиннее на 1 символ

parts = String[] # Массив для хранения частей числа

idx = 1 # Текущая позиция в строке

for i in 1:m # Цикл по количеству частей

current_len = base_len + (i <= remainder ? 1 : 0) # Вычисляем длину текущей части

push!(parts, s[idx:idx+current_len-1]) # Добавляем часть в массив


```
idx += current_len # Сдвигаем индекс начала следующей части  
end
```

```
return parts # Возвращаем массив частей числа  
end
```

```
# Умножает часть числа, сохраняя его длину
```

```
function multiply_preserve_length(part::String, k::Integer)
```

```
num = parse{BigInt, part} * k # Преобразуем часть в число и умножаем на k
```

```
result = string(num) # Обратно в строку
```

```
return lpad(result, length(part), '0') # Сохраняем исходную длину, добавляя нули  
слева
```

```
end
```

```
# Удаляет ведущие нули из строки
```

```
function remove_leading_zeros(s::String)
```

```
if all(c -> c == '0', s) # Если вся строка состоит из нулей
```

```
return "0" # Возвращаем "0"
```

```
else
```

```
idx = findfirst(c -> c != '0', s) # Находим первый не-нулевой символ
```

```
return s[idx:end] # Возвращаем строку без ведущих нулей
```

```
end
```

```
end
```

```
# Сравнивает PQ и NK по началу и концу
```

```
function compare_pq_nk(pq::String, nk::String)
```

```
if pq == nk # Полное совпадение
```

```
return "✅ Полное совпадение"
```

```
end
```

```
min_len = min(length(pq), length(nk)) # Минимальная длина строк
```

```
prefix_match = 0 # Счётчик совпадений спереди
```

```
for i in 1:min_len # Цикл сравнения символов с начала
```

```
pq[i] == nk[i] ? prefix_match += 1 : break # Увеличиваем счётчик или выходим
```

```
end
```

```
suffix_match = 0 # Счётчик совпадений с конца
```

```
for i in 1:min_len # Цикл сравнения символов с конца
```

```
pq[end - i + 1] == nk[end - i + 1] ? suffix_match += 1 : break # Увеличиваем или выходим
```

```
end
```

```
if prefix_match > 0 && suffix_match > 0 # Совпадают начало и конец
```

```
return "🔄 Совпадают начало и конец"
```

```
elseif prefix_match > 0 # Только начало
```

```
return "🔄 Совпадает только начало"
```

```
elseif suffix_match > 0 # Только конец
```

```
return "🔄 Совпадает только конец"
```

```
else # Нет совпадений
```

```
return "❌ Нет совпадений"
```

```
end
```

```
end
```

```
# Проверка алгоритма для одного числа
```

```

function check_algoritм(N::Integer, m::Integer, k::Integer)

N_str = string(N) # Преобразуем N в строку
nk_str = string(N * k) # Умножаем N на k и преобразуем в строку


parts_str = split_number_str(N, m) # Разбиваем N на m частей
multiplied_parts_str = [multiply_preserve_length(p, k) for p in parts_str] # Умножаем
каждую часть
pq_str = join(multiplied_parts_str) # Объединяем части обратно


# Удаление ведущих нулей перед сравнением
pq_clean = remove_leading_zeros(pq_str) # Чистим PQ
nk_clean = remove_leading_zeros(nk_str) # Чистим NK


result = compare_pq_nk(pq_clean, nk_clean) # Сравниваем PQ и NK


return ( # Возвращаем именованный кортеж (NamedTuple) с результатами
проверки СЧС
N = N, # Исходное число N
m = m, # Число частей, на которое было разбито N
k = k, # Множитель, на который умножались части числа
parts = string(parts_str), # Строковое представление разбиения числа на части
multiplied_parts = string(multiplied_parts_str), # Строковое представление
умноженных частей
PQ = pq_clean, # Результат конкатенации умноженных частей (очищенный от
ведущих нулей)
NK = nk_clean, # Результат умножения всего числа на k ( $N * k$ ) (очищенный от
ведущих нулей)

```

```
result = result # Результат сравнения строк PQ и NK (полное совпадение, начало,
конец и т.д.)
```

```
) # Итоговый NamedTuple содержит все данные по проверке для одного числа N
end
```

```
# Параллельная проверка диапазона чисел
```

```
function run_tests_parallel(start_N::Integer, stop_N::Integer, m::Integer, k::Integer)
```

```
results_df = DataFrame( # Создаём DataFrame для хранения результатов
```

```
N = Int[], # Поле "N" — целые числа
```

```
m = Int[], # Поле "m" — целые числа
```

```
k = Int[], # Поле "k" — целые числа
```

```
parts = String[], # Поле "parts" — строки (части исходного числа)
```

```
multiplied_parts = String[], # Поле "multiplied_parts" — строки (умноженные части)
```

```
PQ = String[], # Поле "PQ" — строка результата после умножения частей
```

```
NK = String[], # Поле "NK" — строка  $N * k$ 
```

```
result = String[] # Поле "result" — строка с оценкой совпадения
```

```
)
```

```
count_full = Atomic{Int}(0) # Счётчик полных совпадений
```

```
count_partial_start = Atomic{Int}(0) # Только начало
```

```
count_partial_end = Atomic{Int}(0) # Только конец
```

```
count_partial_both = Atomic{Int}(0) # И начало, и конец
```

```
count_none = Atomic{Int}(0) # Нет совпадений
```

```
@showprogress "🚀 Проверяем  $N \in [start\_N, stop\_N]$ ,  $m = m$ ,  $k = k$ " for N in
start_N:stop_N # Отображаем прогресс
```

```
res = check_algoritm(N, m, k) # Выполняем проверку для конкретного N
```

```
Threads.atomic_add!(count_full, res.result == "✅ Полное совпадение" ? 1 : 0) #  
Обновляем счётчики
```

```
Threads.atomic_add!(count_partial_start, res.result == "🔄 Совпадает только начало" ?  
1 : 0) # Увеличиваем счётчик частичных совпадений (только начало)
```

```
Threads.atomic_add!(count_partial_end, res.result == "🔄 Совпадает только конец" ? 1 :  
0) # Увеличиваем счётчик частичных совпадений (только конец)
```

```
Threads.atomic_add!(count_partial_both, res.result == "🔄 Совпадают начало и  
конец" ? 1 : 0) # Увеличиваем счётчик частичных совпадений (начало и конец)
```

```
Threads.atomic_add!(count_none, res.result == "❌ Нет совпадений" ? 1 : 0) #  
Увеличиваем счётчик случаев без совпадений
```

```
push!(results_df, [ # Добавляем результаты по текущему числу N в DataFrame
```

```
res.N # Исходное число N
```

```
res.m # Количество частей m
```

```
res.k # Множитель k
```

```
res.parts # Строковое представление разбиения на части
```

```
res.multiplied_parts # Строковое представление умноженных частей
```

```
res.PQ # Результат PQ после объединения (очищенный)
```

```
res.NK # Результат NK = N * k (очищенный)
```

```
res.result # Результат сравнения: полное или частичное совпадение / нет
```

```
])
```

```
end
```

```
full = count_full[] # Получаем финальное значение счётчика полных совпадений
```

```
partial_start = count_partial_start[] # Получаем финальное значение счётчика  
совпадений только начала
```

```
partial_end = count_partial_end[] # Получаем финальное значение счётчика  
совпадений только конца
```

```
partial_both = count_partial_both[] # Получаем финальное значение счётчика  
совпадений начала и конца
```

```
none = count_none[] # Получаем финальное значение счётчика отсутствия  
совпадений
```

```
println("\n💾 Сохраняю результаты в CSV...") # Сохранение статистики в файл  
CSV.write("results.csv", results_df) # Записываем таблицу результатов в файл CSV
```

```
open("statistics.txt", "w") do io # Открываем файл для записи статистики в режиме  
перезаписи
```

```
write(io, "📊 Структурная числовая симметрия\n")
```

```
write(io, "=====\n")
```

```
write(io, "Диапазон N: [$start_N, $stop_N]\n")
```

```
write(io, "Количество частей m = $m\n")
```

```
write(io, "Множитель k = $k\n")
```

```
write(io, "-----\n")
```

```
write(io, "✅ Полных совпадений: $full\n")
```

```
write(io, "🔄 Совпадают начало и конец: $partial_both\n")
```

```
write(io, "🔄 Совпадает только начало: $partial_start\n")
```


```
write(io, "🔄 Совпадает только конец: $partial_end\n")
```


```
write(io, "❌ Без совпадений: $none\n")
```


```
write(io, "📄 Результаты по каждому числу — в 'results.csv'\n")
```


```
end
```


```
println("\n📊 Сводная статистика:") # Вывод статистики в терминал
```


@printf("  Полных совпадений: %d\n", full) # Печатаем количество полных совпадений


@printf("  Совпадают начало и конец: %d\n", partial_both) # Печатаем количество совпадений начала и конца

@printf("  Совпадает только начало: %d\n", partial_start) # Печатаем количество совпадений только начала

@printf("  Совпадает только конец: %d\n", partial_end) # Печатаем количество совпадений только конца

@printf("  Без совпадений: %d\n", none) # Печатаем количество отсутствующих совпадений

println("\n  Статистика сохранена в 'statistics.txt') # Выводим в консоль сообщение о сохранении статистики

println("  Результаты сохранены в 'results.csv') # Выводим сообщение о сохранении результатов

return results_df # Возвращаем заполненную таблицу результатов (DataFrame)
end

Пользовательские параметры

start_N = 1 # Начальное число диапазона проверки

stop_N = 10000000 # Конечное число диапазона проверки

m = 2 # Число, на которое разбивается исходное число

k = 7 # Множитель, на который умножаются части числа

Запуск тестов

run_tests_parallel(start_N, stop_N, m, k) # Вызываем основную функцию для параллельной проверки СЧС

Допустим последняя цифра PQ всегда совпадает с NK.

Явление СЧС(Структурной числовой симметрии):

Для любого натурального числа $N \geq 1$, разбитого на $m \geq 2$ натуральных частей, и умноженного на натуральное число k :

Либо $PQ = NK$ (полное совпадение)

Либо совпадают начало и конец

Либо совпадает только конец

Но ни в одном случае нет полного несовпадения

Эмпирический вывод:

Везде и всегда, для любого типа совпадения, совпадает конец

Это ключевой инвариант явления

Даже в диапазоне от 30 до 40 миллионов встречается только полное совпадение или совпадение начала и конца , но конец всё равно всегда совпадает .

Важное наблюдение:

Если последняя цифра всегда совпадает , то проверка по концу является необходимым условием для выполнения всех других типов совпадений .

Теорема о конце (Last Digit Invariance Theorem)

Формулировка: для любого натурального числа $N \geq 1$, разбитого на $m \geq 2$ натуральных частей, и умноженного на натуральное число k , результат объединения умноженных частей как строки PQ всегда имеет ту же последнюю цифру , что и классическое произведение $NK = N * k$.

То есть:

$\text{last_digit}(PQ) = \text{last_digit}(NK)$

Доказательство:

Пусть $N = A_1, A_2 \dots A_m$ Где N — натуральное число, представленное строкой

A_1, A_2, \dots, A_m — его части после разбиения

При умножении по частям получаем:

$PQ = \text{string}(A_1 * k) * \text{string}(A_2 * k) \dots \text{string}(A_m * k)$

Обозначим $B = A \bmod 10$ где B — последняя часть исходного числа.

После умножения:

$$Bk = B * k$$

Так как при объединении частей последняя цифра PQ определяется именно Bk , то:

$$\text{last_digit}(PQ) = (B * k) \bmod 10$$

Рассмотрим классическое умножение:

$$NK = N * k = (a * 10 + B) * k = a * 10 * k + B * k$$

$$a * 10 * k \text{ — оканчивается на ноль} \Rightarrow \text{last_digit}(NK) = (B * k) \bmod 10 = \text{last_digit}(PQ)$$

Следовательно:

Для любого натурального N и любого натурального числа k :

Последняя цифра PQ всегда равна последней цифре $N * K$

Теперь вернёмся к вопросу о невозможном совпадении только по началу:

За всю историю проверок(между PQ и $N * k$):

- Было много случаев полного совпадения
- Было ещё больше совпадений по началу и концу
- Было много совпадений только по концу
- Но ни одного случая только по началу
- Это говорит о том, что конец — более стабильная и устойчивая часть , чем начало.

Значит, можно сделать следующий акцент:

Конец числа инвариантен относительно явления СЧС (Структурной числовой симметрии).

Начало может отличаться между PQ и $N * k$, но конец — нет.

Доказательство явления структурной числовой симметрии через теорему о конце

Можно сделать следующее утверждение:

Поскольку конец PQ всегда совпадает с концом NK , то:

Совпадение только по началу невозможно , потому что тогда нарушается инвариантность конца

*Все три типа совпадений :

- Полное совпадение
- Совпадают начало и конец
- Совпадает только конец

Выводятся из одного фундаментального факта : конец числа сохраняется после преобразования

Математически.

Если:

$$PQ = \text{string}(A1 * k) \cdot \text{string}(A2 * k) \dots \text{string}(A_m * k)$$

$$NK = N * k$$

И:

$$\text{last_digit}(PQ) = \text{last_digit}(N * K)$$

То тогда:

- Совпадение только по началу невозможно
- Всегда будет либо полное совпадение, либо совпадение по концу, либо по обоим.

Что и требовалось доказать!

using Printf # Используется для форматированного вывода (например, @printf)

Функция разбивает число N как строку на m частей

function split_number_str(N::String, m::Integer)

len = length(N) # Длина строки N

base_len = div(len, m) # Базовая длина каждой части при делении на m

remainder = len % m # Остаток от деления длины N на m

parts = String[] # Массив для хранения частей

idx = 1 # Текущий индекс начала следующей части

for i in 1:m # Цикл по количеству частей

current_len = base_len + (i <= remainder ? 1 : 0) # Добавляем +1 к длине первым "remainder" частям

push!(parts, N[idx:idx+current_len-1]) # Добавляем подстроку в массив частей

idx += current_len # Сдвигаем индекс на начало следующей части

end

return parts # Возвращаем массив строковых частей

end

Умножает часть числа, сохраняя длину

function multiply_preserve_length(part::String, k::Integer)

num = parse(BigInt, part) * k # Преобразуем строку в BigInt и умножаем на k

result = string(num) # Обратно преобразуем в строку

```
return lpad(result, max(length(part), length(result)), '0') # Если результат короче
исходной части — дополняем нулями слева
```

```
end
```

```
# Проверка СЧС для одного числа
```

```
function check_full_match_for_one_number(N::BigInt, m::Integer, k::Integer)
```

```
N_str = string(N)          # Преобразуем N в строку
```

```
parts = split_number_str(N_str, m)    # Разбиваем N на m частей
```

```
pq_parts = [multiply_preserve_length(part, k) for part in parts] # Умножаем каждую
часть и сохраняем как строки с сохранением длины
```

```
pq_str = join(pq_parts) # Объединяем умноженные части — это PQ
```

```
nk_str = string(N * k) # Умножаем всё число целиком — это NK
```

```
is_full_match = pq_str == nk_str # Удаляем ведущие нули из pq_str и nk_str для
корректного сравнения (если нужно)
```

```
@printf("🔢 N = %s\n", N_str) # Выводим N
```

```
@printf("📐 m = %d\n", m) # Выводим количество частей
```

```
@printf("📏 k = %d\n", k) # Выводим коэффициент умножения
```

```
if is_full_match # Если совпадают:
```

```
@printf("🔧 Разбиение:\n")
```

```
for (i, part) in enumerate(parts) # Перечисляем все части
```

```

@printf("  Часть %d: \"%s\"\n", i, part)
end

@printf("➡ Умноженные части:\n")
for (i, part) in enumerate(pq_parts) # Перечисляем умноженные части
@printf("  Часть %d: \"%s\"\n", i, part)
end

@printf("📌 PQ = %s\n", pq_str)
@printf("📌 NK = %s\n", nk_str)
@printf("✅ Результат: Полное совпадение найдено!\n")

filename = "full_match_N$(N_str[1:min(50, length(N_str))])_m$m.txt" # Генерируем
имя файла

open(filename, "w") do io # Открываем файл на запись
write(io, "📊 Структуральная числовая симметрия\n")
write(io, "=====\n")
write(io, "🔢 N = $N_str\n")
write(io, "📐 m = $m\n")
write(io, "📏 k = $k\n")
write(io, "-----\n")
write(io, "🔧 Разбиение:\n")
for (i, part) in enumerate(parts)
write(io, "  Часть $i: \"$part\", длина: $(length(part))\n")
end
write(io, "➡ Умноженные части:\n")
for (i, part) in enumerate(pq_parts)

```

```

write(io, "  Часть $i: \"$part\", длина: $(length(part))\n")
end
write(io, "📌 PQ = $pq_str\n")
write(io, "📌 NK = $nk_str\n")
write(io, "✅ Результат: Полное совпадение найдено.\n")
write(io, "=====\n")
end

```

```

println("\n📄 Результаты сохранены в файл: $filename")
else # Если нет совпадения:
@printf("❌ Нет совпадений для данного разбиения.\n")
end

```

```

return ( # Возвращаем структуру с результатом
N = N,
m = m,
k = k,
PQ = pq_str,
NK = nk_str,
result = is_full_match ? "Полное совпадение" : "Нет совпадения"
)
end

```

Пользовательский раздел

```

println("🔄 Вычисляем  $N = 99^{999999}$ ...) # Пример:  $N = 99^{999999}$  большое число

```

`N_bigint = big(99)^999999` # Можно заменить на `big"99"^big"999999"` для большего числа

`m = 2` # количество частей

`k = 3` # `k` — натуральное число

Запуск проверки

`check_full_match_for_one_number(N_bigint, m, k)`