Грокаем алгособесы: Скользящее окно

Решаем задачи на подмассивы и подстроки за линейное время

Что такое паттерн «Скользящее окно»?

Определение

Техника, при которой мы храним и обновляем состояние непрерывного участка данных (подмассива или подстроки), передвигая его по структуре инкрементально — без пересчёта с нуля.

Представьте себе окно, которое скользит по массиву данных. Вместо того чтобы каждый раз заново считать характеристики каждого подмассива, мы просто обновляем информацию о том, что вошло в окно и что из него вышло. Это позволяет достичь значительного улучшения производительности.

Этот паттерн является фундаментальным инструментом в арсенале каждого разработчика и часто встречается на технических собеседованиях в ведущих IT-компаниях.

Два основных варианта

Фиксированное окно

Ширина задана заранее (например, «подмассив длины k»)

Переменное окно

Ширина растёт и сжимается под условие (например, «все символы уникальны»)

Принцип работы: Фиксированное окно

Инициализация окна

Формируем окно на первых K элементах структуры данных, вычисляем начальное значение

Сдвиг окна

Перемещаем окно на один элемент вправо: исключаем левый крайний элемент, добавляем новый справа

Обновление результата

Повторяем процесс, эффективно обновляя результат без полного пересчёта всех элементов

Визуализация работы алгоритма. Фиксированное окно

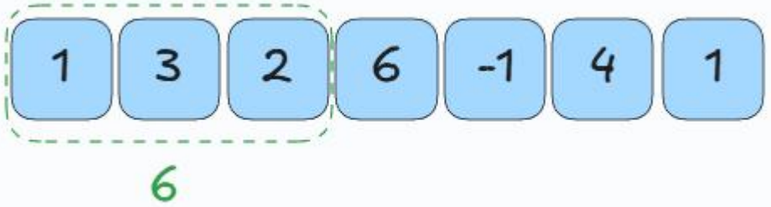
Рассмотрим массив **[1, 3, 2, 6, -1, 4, 1]** с окном размера **K=3**. Окно сдвигается вправо на один элемент на каждом шаге, захватывая новые данные.

Визуализация работы алгоритма. Фиксированное окно

01

Начальная позиция

Окно [1, 3, 2] — инициализация, сумма = 6



Визуализация работы алгоритма. Фиксированное окно

01

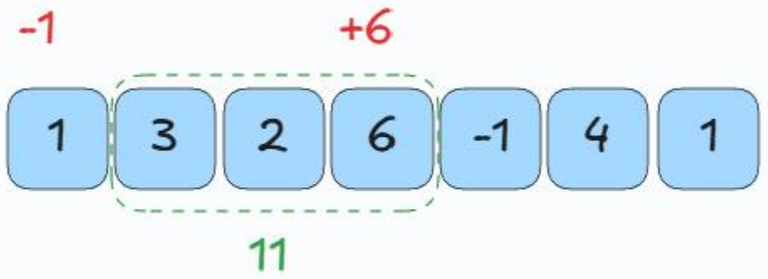
Начальная позиция

Окно [1, 3, 2] — инициализация, сумма = 6

02

Первый сдвиг

Окно [3, 2, 6] — убрали 1, добавили 6, сумма = 11



Визуализация работы алгоритма. Фиксированное окно

01

Начальная позиция

Окно [1, 3, 2] — инициализация, сумма = 6

02

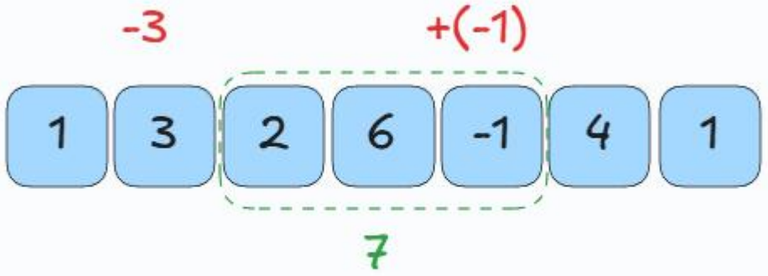
Первый сдвиг

Окно [3, 2, 6] — убрали 1, добавили 6, сумма = 11

03

Второй сдвиг

Окно [2, 6, -1] — убрали 3, добавили -1, сумма = 7



Визуализация работы алгоритма. Фиксированное окно

01

Начальная позиция

Окно [1, 3, 2] — инициализация, сумма = 6

02

Первый сдвиг

Окно [3, 2, 6] — убрали 1, добавили 6, сумма = 11

03

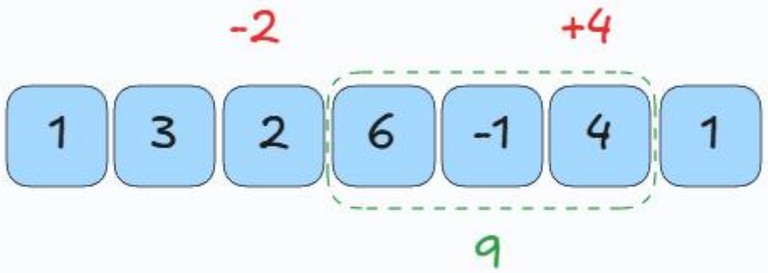
Второй сдвиг

Окно [2, 6, -1] — убрали 3, добавили -1, сумма = 7

04

Третий сдвиг

Окно [6, -1, 4] — убрали 2, добавили 4, сумма = 9



Визуализация работы алгоритма. Фиксированное окно

01

Начальная позиция

Окно [1, 3, 2] — инициализация, сумма = 6

02

Первый сдвиг

Окно [3, 2, 6] — убрали 1, добавили 6, сумма = 11

03

Второй сдвиг

Окно [2, 6, -1] — убрали 3, добавили -1, сумма = 7

04

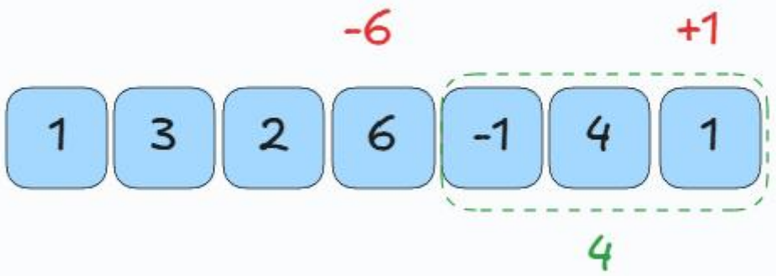
Третий сдвиг

Окно [6, -1, 4] — убрали 2, добавили 4, сумма = 9

05

Продолжение

И так далее до конца массива — каждый шаг за O(1)



Принцип работы. Переменное окно

01

Инициализация указателей

Два указателя: **left** и **right**, оба начинают с начала массива

02

Расширение окна

right расширяет окно → добавляем `arr[right]` в состояние

03

Проверка условия

Как только условие нарушается → сдвигаем **left**, пока не восстановится

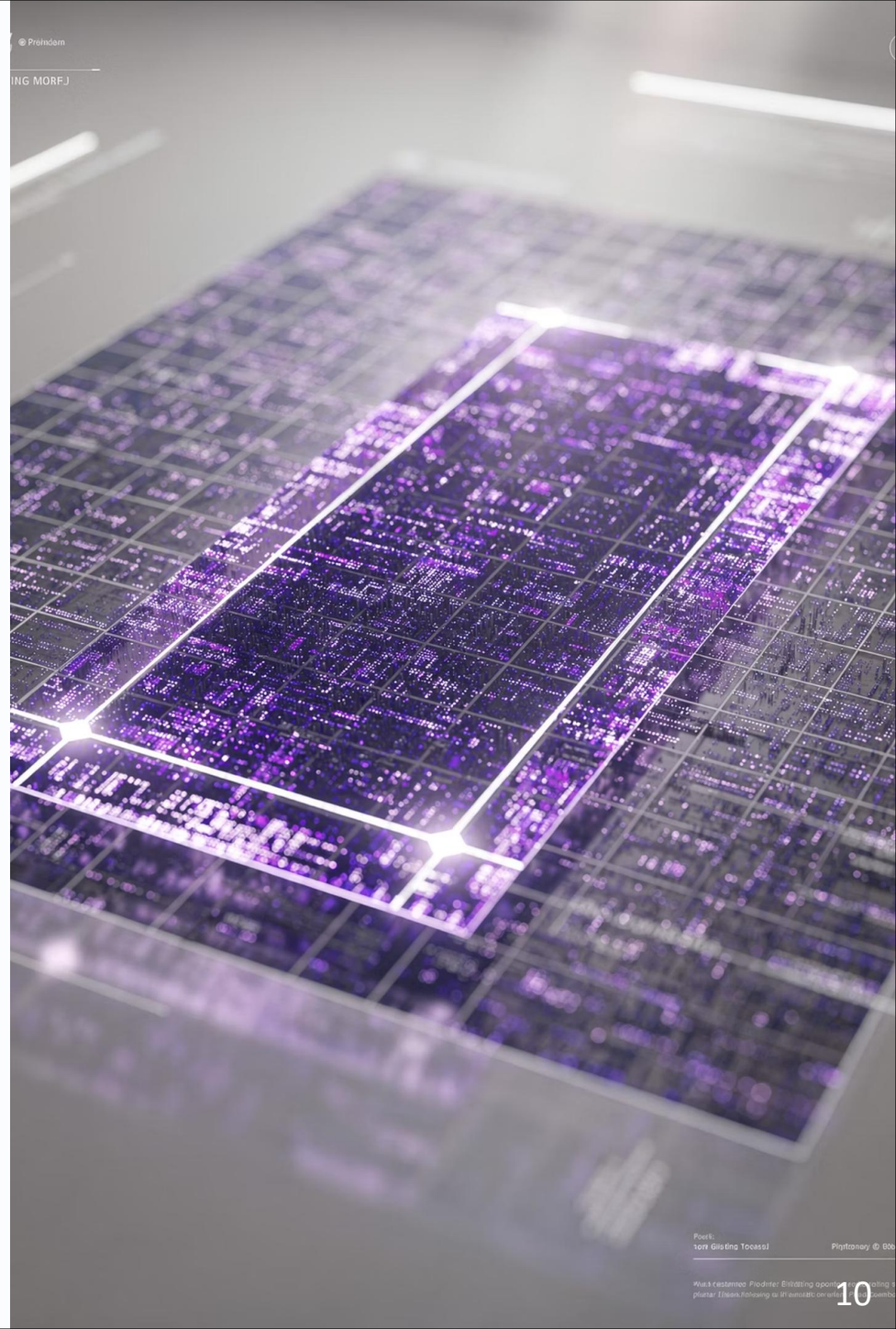
04

Обновление метрики

На каждом шаге обновляем метрику (сумму, длину, частоты и т.д.)

Ключевая идея

Мы **не пересчитываем всё**, а лишь корректируем окно на основе того, что вышло и вошло. Это фундаментальное отличие от наивного подхода с вложенными циклами.



Визуализация работы алгоритма. Переменное окно

Рассмотрим массив **[1, 3, 2, 6, -1, 4, 1]** с целью найти минимальную длину подмассива с суммой ≥ **6**. Окно динамически расширяется и сжимается в зависимости от условия.

Визуализация работы алгоритма. Переменное окно

01

Инициализация

left=0, right=0, окно [1], сумма=1 < 6 — расширяем окно

Визуализация работы алгоритма. Переменное окно

01

Инициализация

left=0, right=0, окно [1], сумма=1 < 6 — расширяем окно

02

Расширение окна

right=1, окно [1, 3], сумма=4 < 6 — продолжаем расширять

Визуализация работы алгоритма. Переменное окно

01

Инициализация

left=0, right=0, окно [1], сумма=1 < 6 — расширяем окно

02

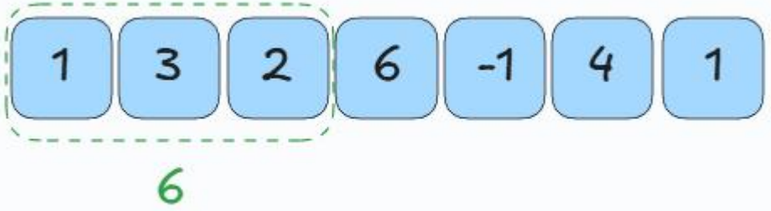
Расширение окна

right=1, окно [1, 3], сумма=4 < 6 — продолжаем расширять

03

Расширение окна

right=2, окно [1, 3, 2], сумма=6 ≥ 6 — условие выполнено! Длина=3



Визуализация работы алгоритма. Переменное окно

01

Инициализация

left=0, right=0, окно [1], сумма=1 < 6 — расширяем окно

02

Расширение окна

right=1, окно [1, 3], сумма=4 < 6 — продолжаем расширять

03

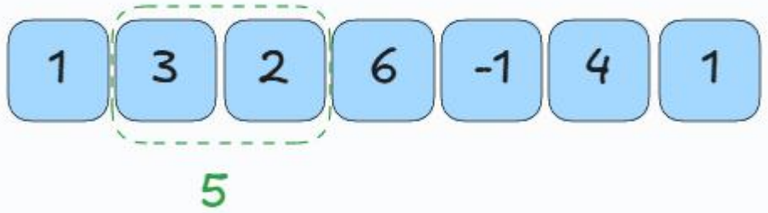
Расширение окна

right=2, окно [1, 3, 2], сумма=6 ≥ 6 — условие выполнено! Длина=3

04

Сжатие окна

left=1, окно [3, 2], сумма=5 < 6 — снова расширяем



Визуализация работы алгоритма. Переменное окно

01

Инициализация

left=0, right=0, окно [1], сумма=1 < 6 — расширяем окно

02

Расширение окна

right=1, окно [1, 3], сумма=4 < 6 — продолжаем расширять

03

Расширение окна

right=2, окно [1, 3, 2], сумма=6 ≥ 6 — условие выполнено! Длина=3

04

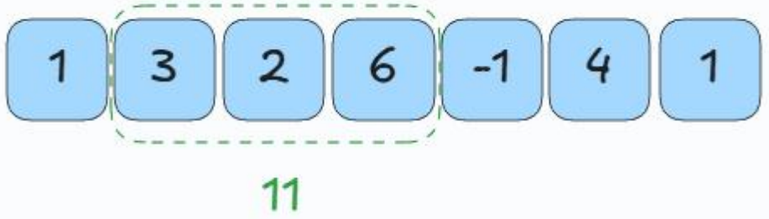
Сжатие окна

left=1, окно [3, 2], сумма=5 < 6 — снова расширяем

05

Расширение и сжатие

right=3, окно [3, 2, 6], сумма=11 ≥ 6 — пробуем сжать; left=2, окно [2, 6], сумма=8 ≥ 6 — длина=2



Визуализация работы алгоритма. Переменное окно

01

Инициализация

left=0, right=0, окно [1], сумма=1 < 6 — расширяем окно

02

Расширение окна

right=1, окно [1, 3], сумма=4 < 6 — продолжаем расширять

03

Расширение окна

right=2, окно [1, 3, 2], сумма=6 ≥ 6 — условие выполнено! Длина=3

04

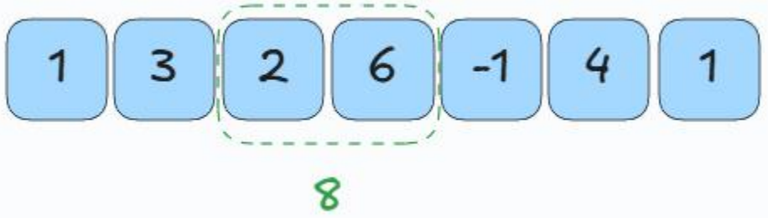
Сжатие окна

left=1, окно [3, 2], сумма=5 < 6 — снова расширяем

05

Расширение и сжатие

right=3, окно [3, 2, 6], сумма=11 ≥ 6 — пробуем сжать; left=2, окно [2, 6], сумма=8 ≥ 6 — длина=2



Визуализация работы алгоритма. Переменное окно

01

Инициализация

left=0, right=0, окно [1], сумма=1 < 6 — расширяем окно

02

Расширение окна

right=1, окно [1, 3], сумма=4 < 6 — продолжаем расширять

03

Расширение окна

right=2, окно [1, 3, 2], сумма=6 ≥ 6 — условие выполнено! Длина=3

04

Сжатие окна

left=1, окно [3, 2], сумма=5 < 6 — снова расширяем

05

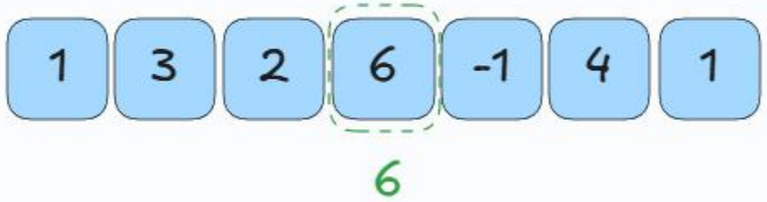
Расширение и сжатие

right=3, окно [3, 2, 6], сумма=11 ≥ 6 — пробуем сжать; left=2, окно [2, 6], сумма=8 ≥ 6 — длина=2

06

Оптимальное решение

left=3, окно [6], сумма=6 ≥ 6 — минимальная длина=1 найдена!



Ключевые преимущества подхода

O(n)

Временная сложность

Только один проход по массиву данных

O(k)

Пространственная сложность

Константная память или словарь для символов

100%

Оптимальность

Невозможно решить быстрее для данного класса задач

Почему это эффективно?

- Только один проход по данным
- Без вложенных циклов
- Без хранения всех возможных подмассивов
- Инкрементальное обновление состояния

Сравнение подходов

Наивный подход с двумя вложенными циклами дал бы нам сложность $O(n^2)$ или даже $O(n^3)$ в некоторых случаях.

Паттерн Sliding Window превращает это в линейное время $O(n)$ — великолепное улучшение для больших наборов данных.

Когда применять этот паттерн?

✓ Применяется, когда:

Непрерывность данных

Нужно найти **непрерывный подмассив или подстроку** в исходных данных

Есть ограничение

Условие типа: сумма ≤ X, длина = k, все символы уникальны, не более k различных элементов

Оптимизация

Требуется **максимизировать, минимизировать** или найти все подходящие варианты

✗ Не подходит, если:

Несвязность

Подмассив **не обязан быть непрерывным** (тогда используйте динамическое программирование)

Глобальные свойства

Задача про перестановку без учёта порядка элементов

Историчность

Требуется **отслеживать всю историю** предыдущих значений или состояний

Понимание этих критериев поможет вам быстро определить, когда паттерн Sliding Window является оптимальным выбором для решения задачи.

Maximum Sum Subarray of Size K. Фиксированное окно

Условие задачи

Дан массив целых чисел и число k. Найти подмассив длины k с максимальной суммой элементов.

Решение с Sliding Window O(n)

```
window_sum = sum(arr[:k])
max_sum = window_sum

for i in range(k, len(arr)):
    window_sum += arr[i] - arr[i-k]
    max_sum = max(max_sum, window_sum)
```

✓ O(n) времени

Только один проход

✓ O(1) памяти

Константное использование

Мы вычисляем сумму первого окна один раз, а затем для каждого следующего окна просто вычитаем элемент, который вышел слева, и прибавляем элемент, который вошёл справа.

Наивный подход O(n×k)

```
for i in range(len(arr) - k + 1):
    current_sum = sum(arr[i:i+k])
    max_sum = max(max_sum, current_sum)
```

Этот подход пересчитывает сумму для каждого окна с нуля, что приводит к избыточным вычислениям.

Пример задачи — Minimum Size Subarray Sum №209

Условие задачи

Дан массив положительных целых чисел и целевое значение target. Найти минимальную длину непрерывного подмассива, сумма элементов которого больше или равна target. Если такого подмассива нет, вернуть 0.

Решение с Sliding Window O(n)

```
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        left = 0
        window_sum = 0
        min_len = float('inf')

        for right in range(len(nums)):
            window_sum += nums[right]           # расширяем окно
            while window_sum >= target:         # пока условие выполнено — сжимаем
                min_len = min(min_len, right - left + 1)
                window_sum -= nums[left]
                left += 1

        if min_len == float('inf'): # приведем ответ к условию задачи
            min_len = 0
```

✓ O(n) времени

Каждый элемент посещается максимум дважды

✓ O(1) памяти

Только указатели и счётчики

Используем динамическое окно: расширяем его вправо, добавляя элементы, и сжимаем слева, когда условие выполнено. Это позволяет найти минимальную длину за один проход.

Другие популярные задачи с этим паттерном

1

Longest Substring Without Repeating Characters

LeetCode №3 — Найти длину самой длинной подстроки без повторяющихся символов. Используем переменное окно и словарь для отслеживания последних позиций символов.

2

Minimum Window Substring

LeetCode №76 — Найти минимальное окно в строке S, которое содержит все символы из строки T. Сложная задача, требующая двух словарей для сравнения частот.

3

Find All Anagrams in a String

LeetCode №438 — Найти все начальные индексы анаграмм строки p в строке s. Используем фиксированное окно размером len(p) и сравнение словарей частот.

4

Fruit Into Baskets

LeetCode №904 — Собрать максимальное количество фруктов, имея только две корзины (каждая для одного типа). По сути, задача на максимальный подмассив с не более чем двумя различными элементами.

Эти задачи представляют различные вариации паттерна Sliding Window и помогают развить глубокое понимание его применения в различных контекстах.

Математика за паттерном

Инкрементальное обновление

При сдвиге окна на **1 элемент**, достаточно выполнить только две операции:

Вычесть

Удалить вклад уходящего элемента

Прибавить

Добавить вклад входящего элемента

Это свойство **инкрементальности** является ключом к эффективности паттерна. Вместо пересчёта всей метрики окна (что требовало бы O(k) операций), мы выполняем константное число операций O(1).

Работа с частотами

В задачах с уникальностью символов или ограничением на количество различных элементов используется словарь частот `freq`.

Инвариант: `len(freq) == число_уникальных_внутри_окна`

Это позволяет за O(1) проверять, сколько различных элементов содержится в текущем окне, что критично для многих задач.

Интересный факт

Sliding Window — это **жадный алгоритм с локальной коррекцией**, который тем не менее гарантирует нахождение глобального оптимума.

Это возможно благодаря **непрерывности** данных: мы знаем, что оптимальное решение должно быть непрерывным подмассивом, и наш алгоритм проверяет все возможные непрерывные подмассивы ровно один раз.

Заключение и следующие шаги

✓ Практикуйся регулярно

Решай задачи на LeetCode по тегу **«Sliding Window»**. Начни с простых задач уровня Easy, постепенно переходя к Medium и Hard. Регулярная практика — ключ к мастерству.

✓ Понимай логику, а не шаблон

Умение адаптировать паттерн к новым условиям — ключ к успеху на собеседованиях. Не заучивай решения механически, а стремись понять, почему алгоритм работает именно так.

✓ Другие паттерны

Изучи паттерн **Two Pointers** (не путать с Fast/Slow Pointers!) — когда указатели движутся **навстречу друг другу** с противоположных концов массива.

Рекомендации по практике

1. Начни с задач на фиксированное окно
2. Переходи к задачам с переменным окном
3. Решай задачи на строки и массивы вперемешку
4. Анализируй чужие решения для углубления понимания

Признаки готовности

Ты готов к собеседованиям, когда можешь:

- Быстро распознать задачу на Sliding Window
- Выбрать подходящий вариант (фиксированное/переменное окно)
- Написать рабочий код за 15-20 минут
- Объяснить временную и пространственную сложность
- Обсудить граничные случаи и оптимизации

Подписывайся и оставайся на связи

Присоединяйся к сообществу, где разбирают не только задачи, но и контекст:

— как выбирать технологии в продакшене,

— как думать на собеседованиях,

— и как не выгорать, оставаясь профессионалом.

Telegram-канал

t.me/marat_notes

Не только про код:

- Алгособесы и data-инженерия (Airflow, Spark, Kafka)
- Разборы когнитивных искажений в IT (**«Ошибки мышления»**)
- Разборы интересных выступлений с конференций и статей (**«ITИнсайты»**)
- Мысли о балансе работы, отдыха и профессионального роста

Репозиторий с кодом

github.com/MaratNotes/marat_notes

Код ко всем разборам — с пояснениями и production-подходом.

Видео

youtube.com/@marat_notes

https://vkvideo.ru/@club231048746

Две рубрики:

- «Грокаем алгособесы» — паттерны, LeetCode, логика на интервью
- «Как работают данные» — продакшен-кейсы из реальных проектов

Удачи на собеседованиях! 🚀

Помни: постоянная практика и глубокое понимание фундаментальных концепций — вот что отличает успешных кандидатов. Не сдавайся, если что-то не получается сразу. Каждая решённая задача делает тебя лучше!