ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7

«Жадный алгоритм»

Выполнил работу

Игнатенок Филипп

Академическая группа №J3111

Принято

Вершинин Владислав

Санкт-Петербург

2024

**Введение**

Цель работы: найти минимальное число «ходов» (перекачек платьев), необходимое для того, чтобы все машины содержали одинаковое количество платьев, или определить, что это невозможно (тогда вернуть -1).

При этом каждый «ход» подразумевает, что мы можем выбрать любое подмножество машин, и каждая из выбранных может передать ровно одно платье (если у неё оно есть) одной из соседних машин; все такие передачи внутри одного «хода» осуществляются параллельно.

**Теоретическая подготовка**

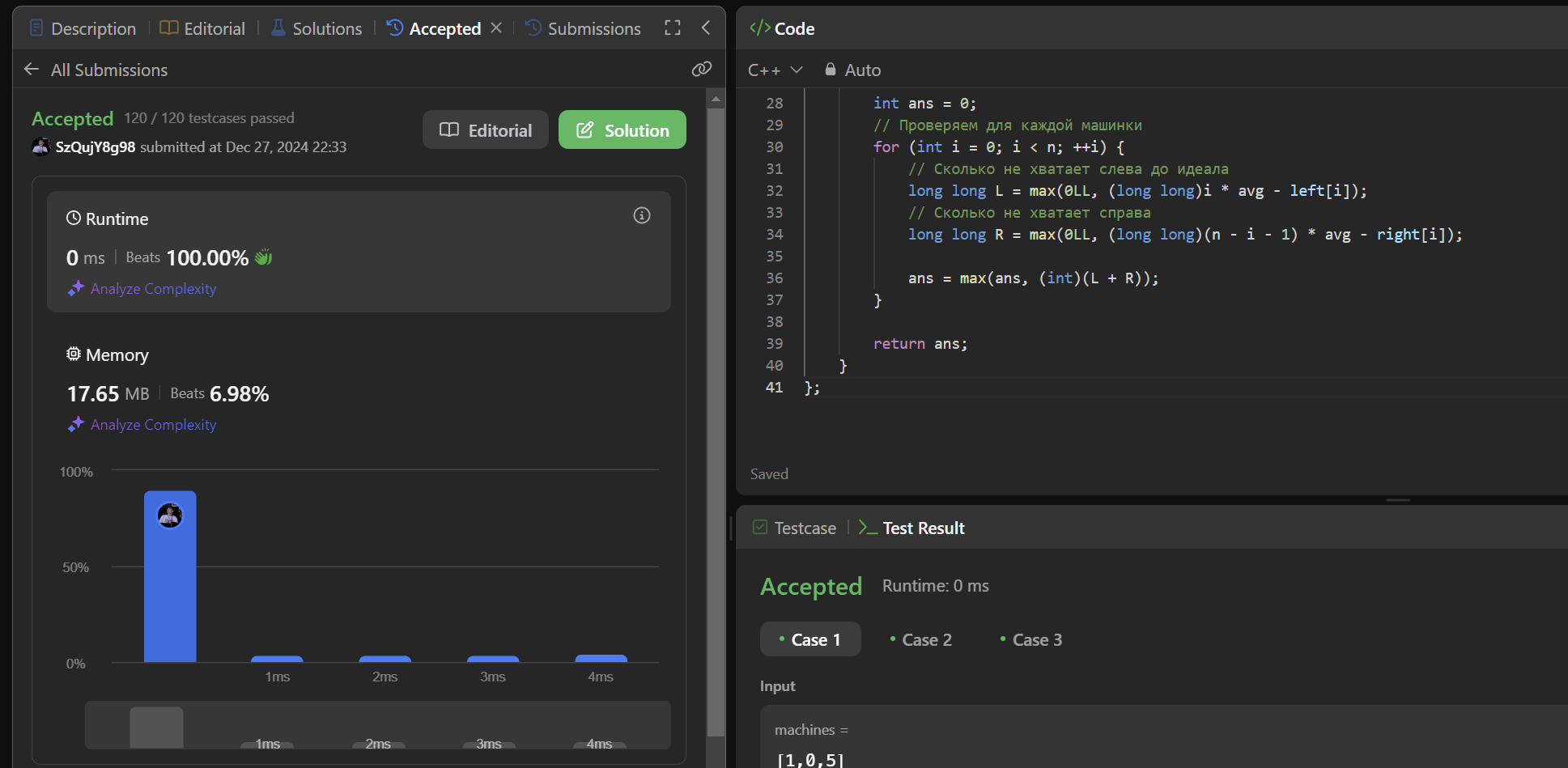
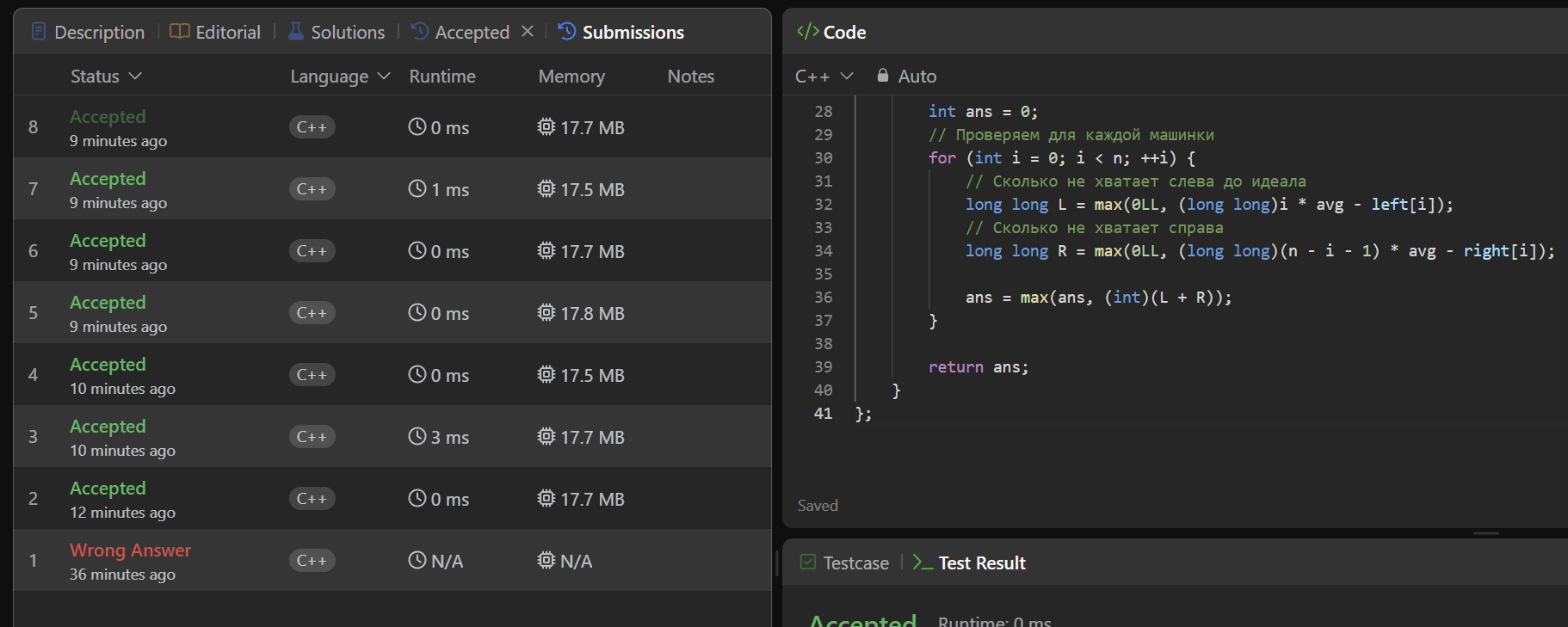
Жадный алгоритм используется, потому что:

1. Локальный оптимум: мы компенсируем дефицит или избыток платьев на месте, не откладывая решение проблемы на будущие шаги.
2. Параллельность передач: жадный подход учитывает, что передавать платья можно из нескольких машин одновременно, и поэтому сразу «переваривает» максимальный локальный дисбаланс.
3. Упрощение решения: благодаря локальному выравниванию мы не перебираем все варианты перестановок, а последовательно (но сразу «по максимуму») приводим систему к целевому состоянию, что упрощает логику и снижает количество шагов.

**Реализация**

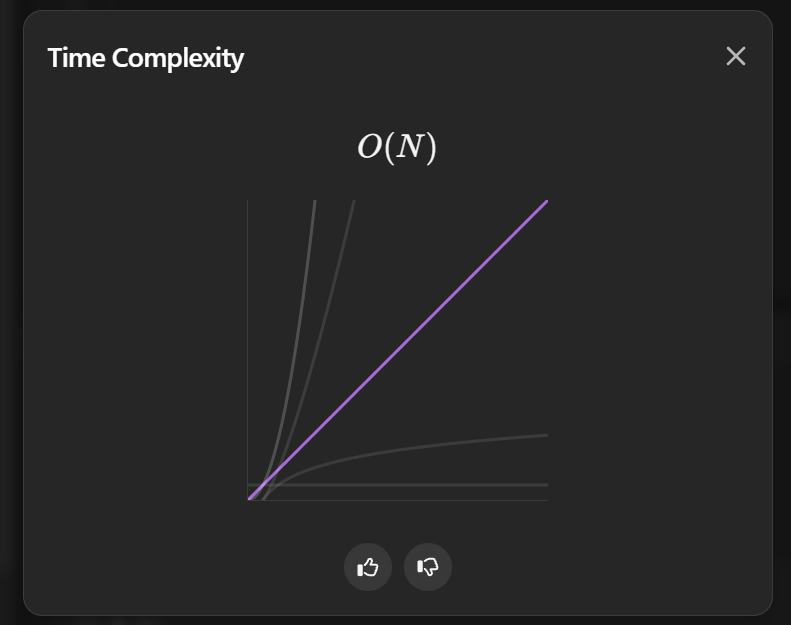
Реализация заключается в использовании жадного алгоритма, который последовательно рассматривает каждую машинку и вычисляет, сколько «платьев» ей нужно получить или отдать, чтобы соблюсти идеальное среднее, учитывая при этом параллельные передачи. Для этого создаются массивы префиксных и суффиксных сумм, позволяющие определить, насколько «нехватка» или «избыток» платьев существует слева и справа от данной машинки. Основная идея — для каждой машинки вычислить, сколько одновременно нужно перекачать слева и справа, и взять максимум по всем машинам как итоговое число ходов, используя жадный подход для мгновенного «выравнивания» локального дисбаланса.

**Тестовая часть**

**Заключение**

Предложенный жадный алгоритм решает задачу балансировки платьев в машинках за счёт параллельной передачи по одному платью от/к каждой машинке. Решение строится на вычислении префиксных и суффиксных сумм, позволяющих в O(n) определить «бутылочное горлышко» — место, где требуется максимальное количество передач. Алгоритм с таким подходом работает за время порядка O(n) и не требует дополнительного перебора вариантов.



**Приложения**

class Solution {

public:

/\*

Сложность по времени: O(n)

Сложность по памяти: O(n)

\*/

int findMinMoves(vector<int>& machines) {

// 1) Подсчёт суммы всех платьев — O(n) по времени

long long total = 0;

for (int d : machines) {

total += d; // Проходим по всем элементам

}

int n = (int)machines.size();

// 2) Проверка: если общее число платьев не делится без остатка на n, уравнять невозможно

// Операция проверки делимости — O(1)

if (total % n != 0) {

return -1;

}

// Среднее количество платьев, которое должно быть в каждой машине

long long avg = total / n;

// 3) Создаём массив префиксных сумм left — O(n) по времени, O(n) по памяти

// left[i] = сумма элементов machines[0..i-1]

vector<long long> left(n, 0);

for (int i = 1; i < n; ++i) {

left[i] = left[i - 1] + machines[i - 1];

}

// 4) Создаём массив суффиксных сумм right — O(n) по времени, O(n) по памяти

// right[i] = сумма элементов machines[i+1..n-1]

vector<long long> right(n, 0);

for (int i = n - 2; i >= 0; --i) {

right[i] = right[i + 1] + machines[i + 1];

}

// 5) Для каждой машины вычисляем, сколько «перекачать» слева и справа — O(n)

// и берём максимум

int ans = 0;

for (int i = 0; i < n; ++i) {

// Сколько не хватает «слева», чтобы привести машины 0..i-1 к идеалу

long long L = max(0LL, (long long)i \* avg - left[i]);

// Сколько не хватает «справа» для машин i+1..n-1

long long R = max(0LL, (long long)(n - i - 1) \* avg - right[i]);

// Максимум (L + R) за все i даёт ответ

ans = max(ans, (int)(L + R));

}

// 6) Возвращаем итоговый результат — O(1)

return ans;

}

};