ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Поиск подмассивов размера K в массиве размера N, сумма элементов которых равна нулю»

Выполнил работу

Фёдоров Егор

Академическая группа J3111

Принято

Ментор, Вершинин Владислав

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель работы: написать программу, которая будет возвращать все подмассивы размера 6, сумма элементов которых равна 0.

Задачи:

— Решить задачу комбинаторным способом

— Проанализировать затрачиваемую память и время работы программы

— Построить диаграмму зависимости времени от входных данных

1. Теоретическая подготовка

Используемые типы данных:

— int для целых чисел, которые являются индексами элементов массива

— double для дробных чисел, которые являются элементами массива

— std::vector для хранения множеств элементов

Используемые алгоритмы:

— Получение массива с индексами элементов сумма, которых равна 0

Для получения данного массива создается пустой вектор, с помощью нескольких циклов находятся 6 индексов элементов, после чего с помощью “vector.push\_back” индексы добавляются в вектор

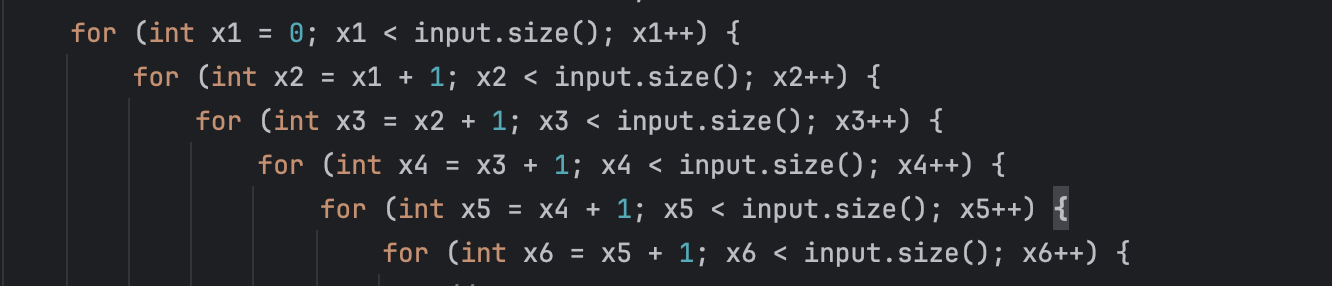
1. Реализация

3.1 Анализ задачи, поиск решения, удовлетворяющего критериям.

На данном этапе проводился анализ поставленной задачи. По условию, решение должно быть комбинаторным, поэтому использовался алгоритм, который выполняет полный перебор всех возможных комбинаций элементов. Это было осуществлено с помощью вложенных циклов.

3.2 Написание программы

3.2.1 На данном этапе, был реализован алгоритм, по перебору возможных комбинаций индексов элементов, он изображен на рисунке 1. Этот алгоритм реализован с помощью нескольких вложенных циклов.

Рисунок 1 - Алгоритм по перебору комбинаций

Цикл, состоящий из шести вложенных циклов, используется для перебора всех возможных комбинаций индексов из шести элементов. Каждый из этих циклов отвечает за выбор одного элемента из вектора. Первый цикл начинается с индекса 0 и проходит до последнего элемента, выбирая первый индекс. Второй цикл начинается с текущего значения первого цикла, что гарантирует, что индексы не будут повторяться. Третий цикл продолжает эту логику, начиная с индекса второго цикла, и так далее до шестого цикла.

Таким образом, каждый следующий цикл имеет возможность выбирать только те индексы, которые идут после предыдущего выбранного индекса. Это приводит к тому, что каждая комбинация индексов будет уникальной и не будет содержать одинаковых значений.

3.2.2. Реализация выполнения условия

На каждой итерации последнего (шестого) цикла происходит проверка условия, сумма выбранных элементов равна 0. (Рисунок 2.) Если условия выполнены, комбинация добавляется в результирующий вектор. Этот процесс продолжается до тех пор, пока все возможные комбинации не будут перебраны. В результате, такой подход позволяет находить все уникальные сочетания элементов заданного размера выполняющие условия.

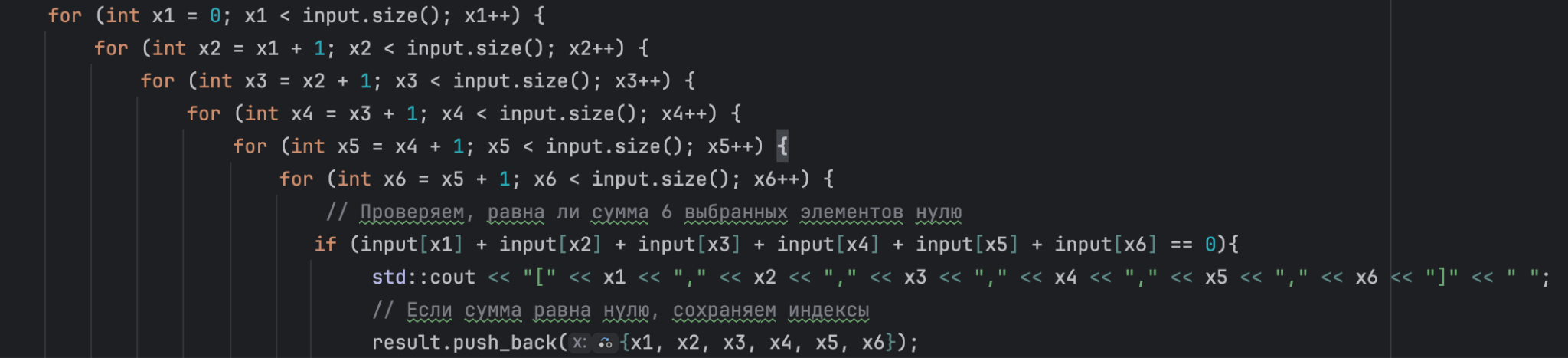


Рисунок 2 - Проверка выполнения условия

3.2.3 Проверка кода с помощью тестов

На данном этапе были написаны тесты для проверки корректности программы. Тесты проверяли работу для следующих случаев:

— Пример, где ожидается несколько комбинаций

— Пример, где нет комбинаций удовлетворяющим условиям

— Пример, где будет всегда одна комбинация

— Пример, где используются дробные числа

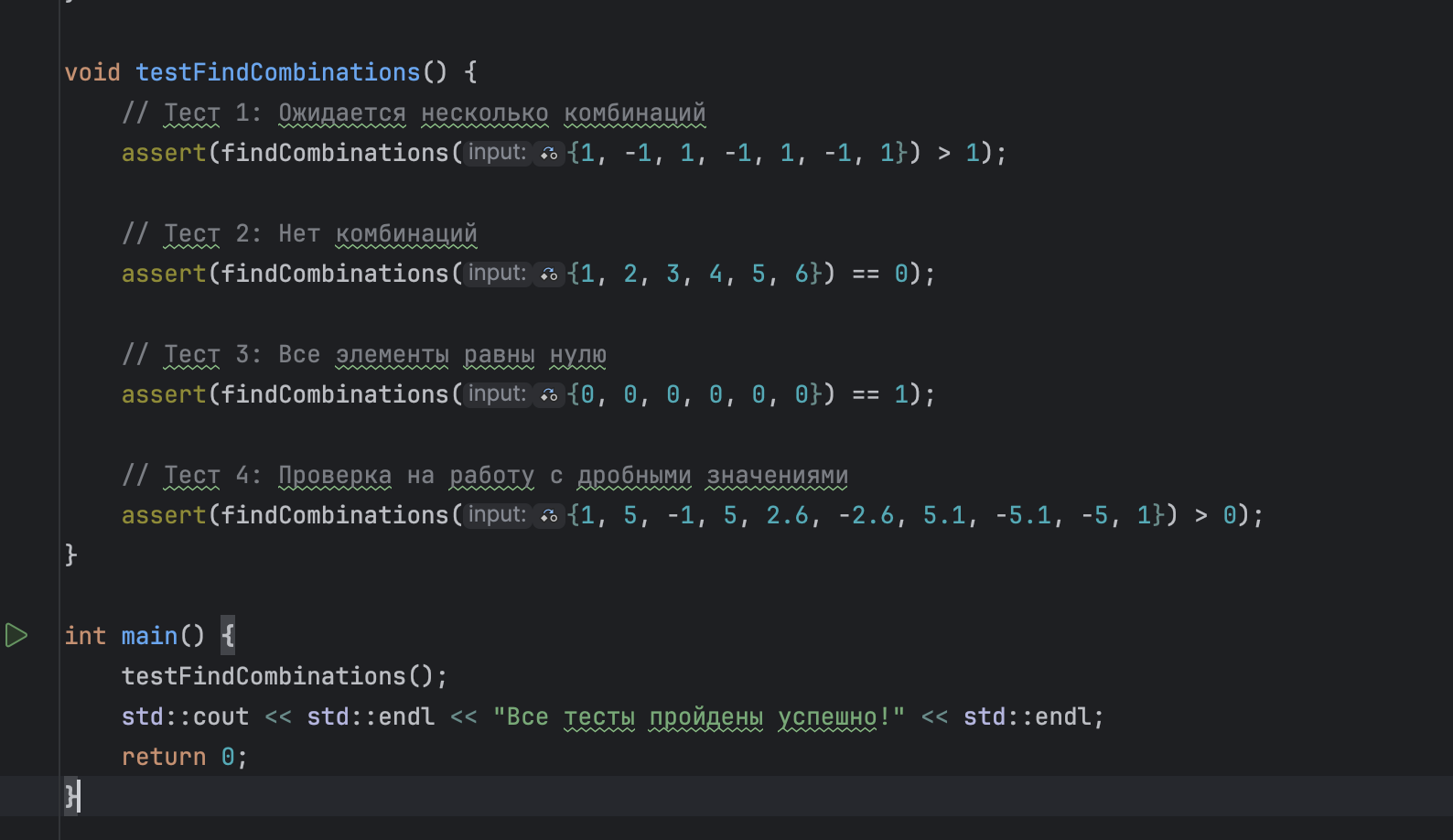
Реализация тестов представлена на Рисунке 3. В результате выводится ошибка, если какой-либо тест не пройден, в ином случае выводится надпись “Все тесты успешно пройдены”

Рисунок 3 - Реализация Тестов

1. Экспериментальная часть

На данном этапе производилась оценка затрачиваемой памяти и асимптотики. Также проводились замеры времени работы программы для разных входных данных. Для замера времени использовалась библиотека chrono. На основе этого был создан график, для удобного анализа зависимости роста времени работы от входных данных.

Подсчёт памяти  
Рассмотрим приблизительную оценку:

— Вектор `input` = n \* ‘sizeof(double)’ байт. Например, если n равно 10, то размер `input` будет 80 байт (10 \* 8 байт, где 8 байт - размер `double`).

— Вектор `result`: В худшем случае, когда все комбинации из 6 элементов дают сумму 0, вектор result будет содержать `n! / (6! \* (n - 6)!)` комбинаций. Каждая комбинация в result - это вектор из 6 целых чисел (типа `int`).

— Размер одной комбинации: `6 \* sizeof(int)` байт.

\* Общий размер result = n! / (6! \* (n - 6)!) \* 6 \* 4 байт.

Пример: Если `input.size()` равно 10, то:

— `input` будет занимать 80 байт.

— `result` в худшем случае будет содержать 210 комбинаций, что составит `210 — 6 \* 4` байт (где 4 байта - размер `int`).

Полная оценка: 80 + 210 \* 6 \* 4 = 5120 байт.

Подсчет Асимптотики  
Код состоит из шести вложенных циклов, каждый из которых проходит по элементам вектора input. Рассмотрим, как это влияет на временную сложность:

1. Первый цикл (x1): проходит от 0 до n-1, где n — размер вектора.

2. Второй цикл (x2): проходит от x1 + 1 до n-1, что в среднем дает примерно (n-1)/2 итераций.

3. Третий цикл (x3): проходит от x2 + 1 до n-1, что также будет давать примерно (n-2)/3 итераций.

4. Четвертый цикл (x4): проходит от x3 + 1 до n-1.

5. Пятый цикл (x5): проходит от x4 + 1 до n-1.

6. Шестой цикл (x6): проходит от x5 + 1 до n-1.

Таким образом, общее количество итераций всех шести вложенных циклов можно оценить как: ∑ₖ₌₀⁵ n-k / 6! ≈ n⁶ / 720

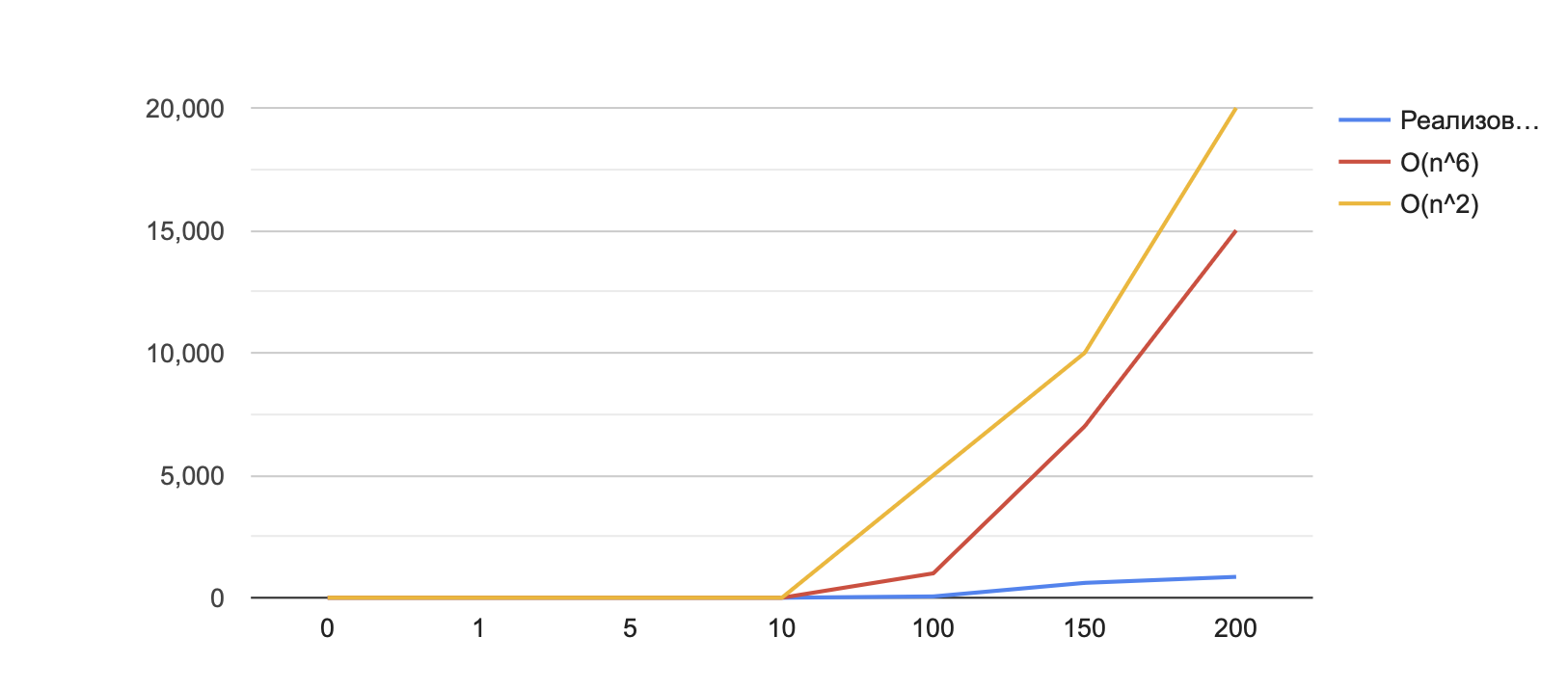
Следовательно общая сложность алгоритма = O(n⁶)

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(N^6) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

| Размер входного набора | 1 | 5 | 10 | 100 | 150 | 200 |
| --- | --- | --- | --- | --- | --- | --- |
| Время выполнения программы, с | 1\*e^-9 | 0.000001 | 0.001 | 53.59 | 606.7 | 852.5 |
| O(N\*6), с | 1\*e^-9 | 0.000015 | 0.0001 | 996 | 7100 | 950400 |
| O(N^2), с | 0,0008 | 0,0128 | 0.001 | 5,07 × 10^20 | очень много | ++очень много |

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №4.



Изображение №4 - График работы алгоритма

Анализируя график, представленный на изображении №4, можно увидеть, что теоретически сложность была определена корректна, но практическая реализация алгоритма, получилось не такой какой ожидалось. Стоить отметить, что должно было получиться наоборот ведь 25^6 > 2 ^ 25 Также она соответствует требованию быть не менее O(2^N) , Однако лишь на “Теоретических сведениях”

1. Заключение

В ходе выполнения лабораторной работы мною был реализован алгоритм поиска подмассивов длиной 6, сумма элементов которых равна 0. Полученные результаты немного отличаются с теоретическими оценками сложности алгоритма. Это можно связать с отличием в мощности используемого оборудования.

На примере этой работы можно заметить, что при высокой сложности алгоритма, применения его на больших данных неудобно, из-за больших затрат по времени.

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла main.cpp

#include <iostream>

#include <vector>

#include <cassert>

int findCombinations(const std::vector<double>& input) {

std::vector<std::vector<int>> result;

for (int x1 = 0; x1 < input.size(); x1++) {

for (int x2 = x1 + 1; x2 < input.size(); x2++) {

for (int x3 = x2 + 1; x3 < input.size(); x3++) {

for (int x4 = x3 + 1; x4 < input.size(); x4++) {

for (int x5 = x4 + 1; x5 < input.size(); x5++) {

for (int x6 = x5 + 1; x6 < input.size(); x6++) {

// Проверяем, равна ли сумма 6 выбранных элементов нулю

if (input[x1] + input[x2] + input[x3] + input[x4] + input[x5] + input[x6] == 0){

std::cout << "[" << x1 << "," << x2 << "," << x3 << "," << x4 << "," << x5 << "," << x6 << "]" << " ";

// Если сумма равна нулю, сохраняем индексы

result.push\_back({x1, x2, x3, x4, x5, x6});

}

}

}

}

}

}

}

std::cout << std::endl;

return result.size(); // Возвращаем количество найденных комбинаций

}

void testFindCombinations() {

// Тест 1: Ожидается несколько комбинаций

assert(findCombinations({1, -1, 1, -1, 1, -1, 1}) > 1);

// Тест 2: Нет комбинаций

assert(findCombinations({1, 2, 3, 4, 5, 6}) == 0);

// Тест 3: Все элементы равны нулю

assert(findCombinations({0, 0, 0, 0, 0, 0}) == 1);

// Тест 4: Проверка на работу с дробными значениями

assert(findCombinations({1, 5, -1, 5, 2.6, -2.6, 5.1, -5.1, -5, 1}) > 0);

}

int main() {

testFindCombinations();

std::cout << std::endl << "Все тесты пройдены успешно!" << std::endl;

return 0;

}