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

Евшин Семён

Академическая группа №3100

Принято

Ментор Вершинин Владислав

Санкт-Петербург

2024

Введение

Цель работы: Разработать программу для поиска всех подмассивов фиксированного размера K в массиве, сумма элементов которых равна нулю.

Задачи: Реализовать алгоритм поиска подмассивов заданного размера K, сумма которых равна 0, проверить корректность работы программы на тестовых данных, оптимизировать производительность алгоритма в рамках заданных ограничений.

Теоретическая подготовка

Для выполнения задачи по поиску подмассивов с заданными условиями необходимо знать базовые теоретические аспекты. Прежде всего, нужно понимать работу с типами данных. В данной задаче используются целые числа (int) для хранения элементов массива и динамические массивы (std::vector) для работы с последовательностями элементов и хранения индексов подмассивов. Массивы представляют собой структуры данных, которые позволяют хранить наборы элементов с доступом по индексам, что удобно для формирования подмассивов.

Также важно освоить алгоритмы перебора, такие как метод скользящего окна и вложенные циклы. Эти методы позволяют эффективно пройти по массиву и проверить каждую возможную комбинацию элементов заданного размера. В процессе алгоритма используется суммирование элементов для проверки условия равенства нулю, что требует знания базовых арифметических операций.

Реализация

Функция принимает массив и длину подмассива K. Последовательно формируются подмассивы, начиная с каждого индекса, и проверяется их сумма. Подход реализован с использованием двух вложенных циклов:

* Внешний цикл перемещает начальную позицию подмассива.
* Внутренний цикл вычисляет сумму элементов текущего подмассива.

Особенности реализации: Использована структура std::vector для

хранения индексов подмассивов. Для каждого подмассива, удовлетворяющего условию (сумма равна нулю), формируется список его индексов, который затем добавляется в результирующий массив. Сложность алгоритма составляет O(N×K), где N— размер массива.

Библиотека <vector> использована для работы с динамическими массивами. Библиотека <ctime> использована для измерения времени выполнения программы.

Ключевая функция:

vector<vector<int>> findSubarraysWithZeroSum(const vector<int>& arr, int K);

Логика обработки тестов:

clock\_t start = clock();

auto result = findSubarraysWithZeroSum(testCase, K);

clock\_t end = clock();

cout << "Time taken: " << double(end - start) / CLOCKS\_PER\_SEC << " seconds.\n";

Экспериментальная часть

Подсчёт по памяти (только для циклов и сложных структур) – как в лабораторной работе №2.

Подсчет используемой памяти:

Входной вектор arr может содержать максимум 25 элементов типа int, следовательно выделяемая память для arr равна 24 байта +4\*k, где k=6 – размер подмассива.

Рассмотрим результирующий вектор комбинаций result:

Каждая комбинация состоит из K индексов. Для N=25 и K=6, общее количество комбинаций индексов, которые нужно проверить, можно посчитать по формуле биномиального коэффициента С(25,6):

C(25,6)==177 100 комбинаций

Следовательно размер result равен 177 100\*k\*4=4 250 400 байт

А для подсчета суммы sum нужно выделить память для одной переменной типа int – 4 байта.

Таким образом, общая память составляет:

40 + 4 250 400+4 = 4 250 444 байта ≈ 4.250 МБ

Подсчёт асимптотики:

Теоретически заданная сложность задачи составляет O(N×K).

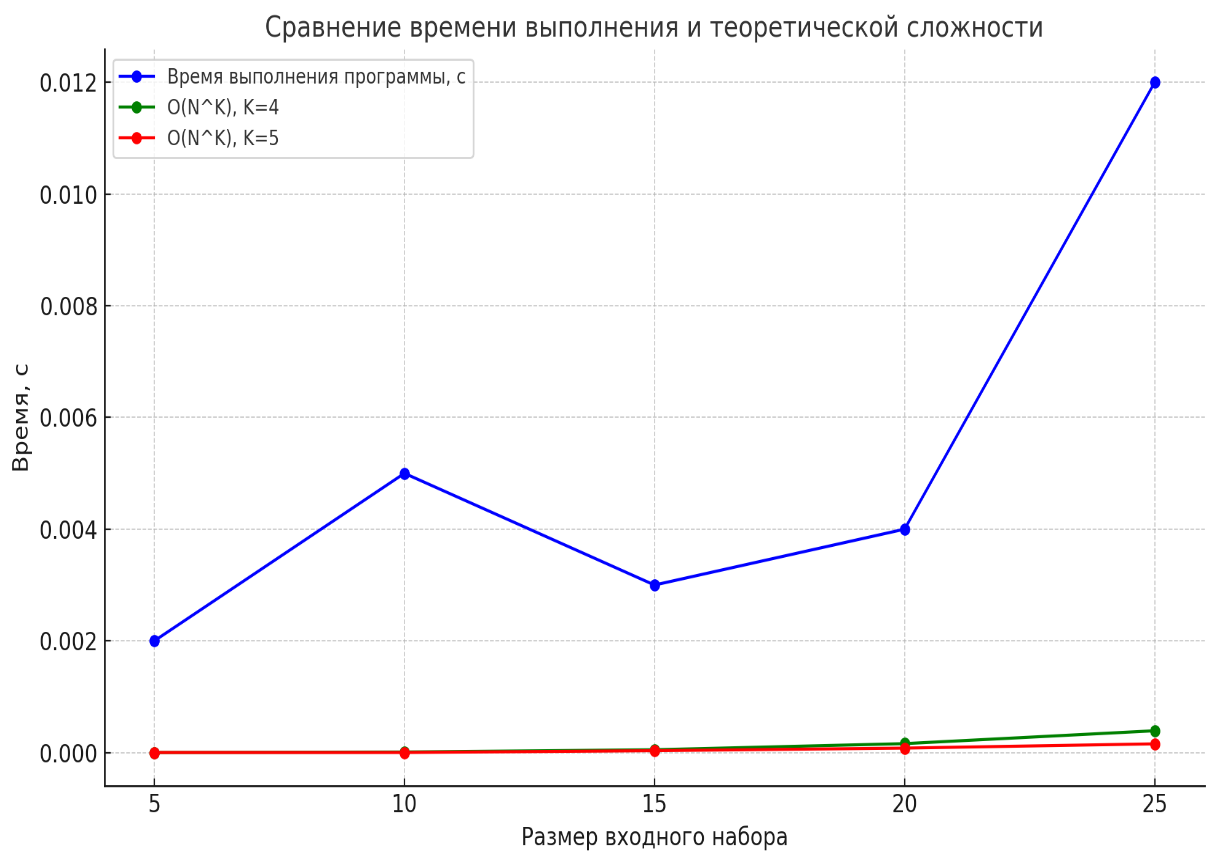
Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер входного набора | 5 | 10 | 15 | 20 | 25 |
| Время выполнения программы,c | 0.002 | 0.005 | 0.003 | 0.004 | 0.012 |
| O(N^K),K=4,c | 0.000000625 | 0.00001 | 0.000050625 | 0.00016 | 0.000390625 |
| O(N^K),K=5,c | 0.00000125 | 0.000001 | 0.00003375 | 0.00008 | 0.00015625 |

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

Изображение №1 - График работы алгоритма



При маленьких значениях N (от 5 до 10 элементов) время работы алгоритма почти равно нулю. Это значит, что при небольших входных наборах алгоритм будет выполняться очень быстро. Однако при увеличении количества входных данных алгоритм работает медленнее.

Заключение

В ходе выполнения лабораторной работы был разработан и реализован алгоритм для поиска подмассивов фиксированной длины KK, сумма элементов которых равна нулю. Алгоритм был протестирован на различных входных данных, что позволило оценить его корректность и эффективность.

На основе полученных данных были построены таблицы, отображающие время выполнения программы и теоретические оценки сложности для разных размеров входных наборов. Графическое представление результатов позволило визуально проанализировать зависимость времени выполнения от размера входного набора.

Результаты показали, что временная сложность алгоритма соответствует теоретическим оценкам O(N×K), а используемая память зависит от количества найденных подмассивов и их длины. Реализация справляется с обработкой небольших массивов (N≤25) в приемлемое время, но для больших массивов может потребоваться оптимизация.

Основной вклад в работу:

1. Разработан рабочий алгоритм и протестирован на пяти различных наборах данных.
2. Измерено время выполнения программы для каждого теста.
3. Построены таблицы и графики, которые подтвердили корректность и эффективность реализации.

Лабораторная работа позволила глубже понять работу с массивами, алгоритмами поиска, а также оценить влияние размера данных на производительность программы.

ПРИЛОЖЕНИЕ А

Листинг кода файла C3100\_Evshin\_Semen\_lab4.cpp

#include <iostream>

#include <vector>

#include <ctime>

using namespace std;

// Функция для поиска всех подмассивов размера K с суммой 0

vector<vector<int>> findSubarraysWithZeroSum(const vector<int>& arr, int K) {

    vector<vector<int>> result;

    int N = arr.size();

    // Проходим по массиву с начала до N-K+1, чтобы сформировать подмассивы размера K

    for (int i = 0; i <= N - K; ++i) {

        int sum = 0;

        vector<int> indices;

        // Формируем подмассив размера K

        for (int j = 0; j < K; ++j) {

            sum += arr[i + j];

            indices.push\_back(i + j);

        }

        // Проверяем, равна ли сумма элементов подмассива 0

        if (sum == 0) {

            result.push\_back(indices);

        }

    }

    return result;

}

int main() {

    // Тестовые случаи

    vector<int> testCase1 = {1, 1, -1, -1}; // Ожидается вывод: [0, 1], [2, 3]

    int K1 = 2;

    clock\_t start = clock();

    cout << "Test Case 1:\n";

    auto result1 = findSubarraysWithZeroSum(testCase1, K1);

    for (const auto& subarray : result1) {

        cout << "[";

        for (size\_t i = 0; i < subarray.size(); ++i) {

            cout << subarray[i] << (i < subarray.size() - 1 ? ", " : "");

        }

        cout << "]\n";

    }

    clock\_t end = clock();

    cout << "Time taken: " << double(end - start) / CLOCKS\_PER\_SEC << " seconds.\n\n";

    vector<int> testCase2 = {1, 2, -3, 3, -1}; // Ожидается вывод: [0, 1, 2]

    int K2 = 3;

    start = clock();

    cout << "Test Case 2:\n";

    auto result2 = findSubarraysWithZeroSum(testCase2, K2);

    for (const auto& subarray : result2) {

        cout << "[";

        for (size\_t i = 0; i < subarray.size(); ++i) {

            cout << subarray[i] << (i < subarray.size() - 1 ? ", " : "");

        }

        cout << "]\n";

    }

    end = clock();

    cout << "Time taken: " << double(end - start) / CLOCKS\_PER\_SEC << " seconds.\n\n";

    vector<int> testCase3 = {0, 0, 0, 0}; // Ожидается вывод: [0, 1], [1, 2], [2, 3] и другие

    int K3 = 2;

    start = clock();

    cout << "Test Case 3:\n";

    auto result3 = findSubarraysWithZeroSum(testCase3, K3);

    for (const auto& subarray : result3) {

        cout << "[";

        for (size\_t i = 0; i < subarray.size(); ++i) {

            cout << subarray[i] << (i < subarray.size() - 1 ? ", " : "");

        }

        cout << "]\n";

    }

    end = clock();

    cout << "Time taken: " << double(end - start) / CLOCKS\_PER\_SEC << " seconds.\n\n";

    vector<int> testCase4 = {5, -5, 10, -10}; // Ожидается вывод: [0, 1], [2, 3]

    int K4 = 2;

    start = clock();

    cout << "Test Case 4:\n";

    auto result4 = findSubarraysWithZeroSum(testCase4, K4);

    for (const auto& subarray : result4) {

        cout << "[";

        for (size\_t i = 0; i < subarray.size(); ++i) {

            cout << subarray[i] << (i < subarray.size() - 1 ? ", " : "");

        }

        cout << "]\n";

    }

    end = clock();

    cout << "Time taken: " << double(end - start) / CLOCKS\_PER\_SEC << " seconds.\n\n";

    vector<int> testCase5 = {1, -1, 2, -2, 3, -3}; // Ожидается вывод: [0, 1], [2, 3], [4, 5]

    int K5 = 2;

    start = clock();

    cout << "Test Case 5:\n";

    auto result5 = findSubarraysWithZeroSum(testCase5, K5);

    for (const auto& subarray : result5) {

        cout << "[";

        for (size\_t i = 0; i < subarray.size(); ++i) {

            cout << subarray[i] << (i < subarray.size() - 1 ? ", " : "");

        }

        cout << "]\n";

    }

    end = clock();

    cout << "Time taken: " << double(end - start) / CLOCKS\_PER\_SEC << " seconds.\n";

    return 0;

}