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

Фурман Ирина

Академическая группа C3100

Принято

Ментор Вершинин Владислав

Санкт-Петербург

2024

Введение

Цель лабораторной работы – исследование алгоритмического подхода к задаче поиска подмассивов размера K в массиве размера N, сумма элементов которых равна нулю. В ходе выполнения работы необходимо реализовать алгоритм полного перебора всех возможных подмассивов заданного размера и оценить его эффективность. Также требуется на практике проанализировать временную сложность алгоритма, сравнить фактическое время выполнения с теоретическими расчетами и выявить влияние размера исходного массива на производительность алгоритма.

1. Теоретическая подготовка

Для выполнения лабораторной работы необходимо понимать структуру массивов и их индексацию. Важно ознакомиться с существующими алгоритмами для поиска подмассивов с заданными свойствами, включая метод скользящего окна и использование хеш-таблиц для оптимизации поиска. Нужно понимать принципы вычисления суммы элементов подмассива и методы проверки условия на равенство нулю. Также потребуется знание основ комбинаторики и принципов формирования комбинаций для определения всех возможных подмассивов заданного размера K. Необходимо знать принцип оценки времени исполнения программы и расчет оценки сложности алгоритма.

1. Реализация

В процессе реализации были использованы стандартная библиотека iostream для вывода результата, vector для работы со структурой данных типа вектор, ctime для подсчета времени выполнения программ.

Задача решается в несколько этапов. Основой алгоритма является цикл с количеством итераций N - K + 1.

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

Цикл итерирует от 0 до N - K, включительно. Это гарантирует, что мы не выйдем за пределы массива при обращении к элементам от i до i + K - 1. Далее мы обнуляем переменную sum и создаем вектор indices, который будет хранить индексы текущего подмассива. Вложенный цикл отвечает за суммирование элементов подмассива, начиная с arr[i] до arr [i + K - 1] и одновременно сохраняет индексы в indices.

int sum = 0;

std::vector<int> indices;

for (int j = 0; j < K; ++j) {

sum += arr[i + j];

indices.push\_back(i + j);

}

Таким образом мы получаем сумму элементов подмассива, которую будем проверять и в зависимости от ее значения выводить ответы. Если сумма равна нулю (sum == 0), внутренний цикл выводит индексы элементов текущего подмассива. Используется std::cout для вывода, при этом добавляется запятая для разделения индексов, кроме последнего.

if (sum == 0) {

for (size\_t k = 0; k < indices.size(); ++k) {

std::cout << indices[k];

if (k < indices.size() - 1) {

std::cout << ", ";

}

}

std::cout << std::endl;

}

В функции main определены несколько тестовых случаев. Каждый случай представляет собой вектор целых чисел и размер подмассива K. Вызывается функция findSubarraysWithZeroSum, чтобы найти подмассивы с нулевой суммой.

int main() {

    // Тестовые случаи

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

    int K1 = 2;

    std::cout << "Test Case 1:\n";

    findSubarraysWithZeroSum(testCase1, K1);

...

}

1. 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=4 – размер подмассива.

Рассмотрим результирующий вектор комбинаций result:

Каждая комбинация состоит из K индексов. Для N=25 и K=4, общее количество комбинаций индексов, которые нужно проверить, можно посчитать по формуле биномиального коэффициента С(25,4):

C(25,4)==12 650 комбинаций

Следовательно размер result равен 12 650\*k\*4=202 400 байт

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

Таким образом, общая память составляет:

40 + 202 400+4 = 202 444 байта ≈ 0.192 МБ

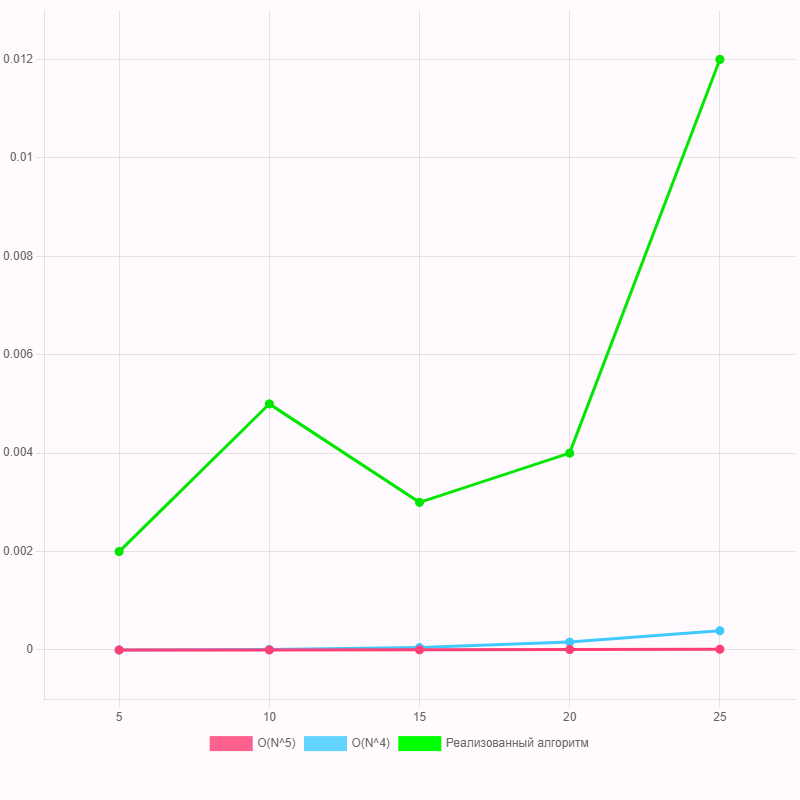
Подсчёт асимптотики:

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(k\*C(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, с | 0,002 | 0,005 | 0,003 | 0,004 | 0,012 |
| O(N^K), K=4, с | 0,000000625 | 0,00001 | 0,000050625 | 0,00016 | 0,000390625 |
| O(N^K), K=5, с | 0,000000125 | 0,000001 | 0,000003375 | 0,000008 | 0,000015625 |

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



Изображение №1 - График работы алгоритма

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

Заключение

В ходе выполнения лабораторной работы мной был реализован алгоритм поиска подмассивов размера K в массиве размера N, сумма элементов которых равна нулю. Для достижения поставленной цели я использовала метод полного перебора, который позволил рассмотреть все возможные комбинации значений массива. В процессе тестирования алгоритма на различных массивах с разнообразными числами и комбинациями входных значений были получены результаты, которые подтвердили теоретические оценки сложности алгоритма.  
Работа не ставила целью оптимизацию алгоритма, а, напротив, позволила мне на практике оценить ресурсоемкость решений задач с высокой вычислительной сложностью. Это дало возможность лучше понять влияние размера входных данных на время выполнения алгоритма.

ПРИЛОЖЕНИЕ А

Листинг кода файла lab04.cpp

#include <iostream>

#include <vector>

#include <ctime>  // Для использования clock()

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

    int N = arr.size();

    // Перебираем все возможные подмассивы размером K

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

        int sum = 0;

        std::vector<int> indices;

        // Считаем сумму подмассива и сохраняем индексы

        for (int j = 0; j < K; ++j) {

            sum += arr[i + j];

            indices.push\_back(i + j);

        }

        // Если сумма равна 0, выводим индексы

        if (sum == 0) {

            for (size\_t k = 0; k < indices.size(); ++k) {

                std::cout << indices[k];

                if (k < indices.size() - 1) {

                    std::cout << ", ";

                }

            }

            std::cout << std::endl;

        }

    }

}

int main() {

    // Тестовые случаи

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

    int K1 = 2;

    // Измерение времени выполнения

    clock\_t start = clock();

    std::cout << "Test Case 1:\n";

    findSubarraysWithZeroSum(testCase1, K1);

    clock\_t end = clock();

    double timeTaken = double(end - start) / CLOCKS\_PER\_SEC;

    std::cout << "Time taken: " << timeTaken << " seconds.\n";

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

    int K2 = 3;

    start = clock();

    std::cout << "Test Case 2:\n";

    findSubarraysWithZeroSum(testCase2, K2);

    end = clock();

    timeTaken = double(end - start) / CLOCKS\_PER\_SEC;

    std::cout << "Time taken: " << timeTaken << " seconds.\n";

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

    int K3 = 2;

    start = clock();

    std::cout << "Test Case 3:\n";

    findSubarraysWithZeroSum(testCase3, K3);

    end = clock();

    timeTaken = double(end - start) / CLOCKS\_PER\_SEC;

    std::cout << "Time taken: " << timeTaken << " seconds.\n";

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

    int K4 = 2;

    start = clock();

    std::cout << "Test Case 4:\n";

    findSubarraysWithZeroSum(testCase4, K4);

    end = clock();

    timeTaken = double(end - start) / CLOCKS\_PER\_SEC;

    std::cout << "Time taken: " << timeTaken << " seconds.\n";

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

    int K5 = 2;

    start = clock();

    std::cout << "Test Case 5:\n";

    findSubarraysWithZeroSum(testCase5, K5);

    end = clock();

    timeTaken = double(end - start) / CLOCKS\_PER\_SEC;

    std::cout << "Time taken: " << timeTaken << " seconds.\n";

    return 0;

}