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

Ковров Евгений

Академическая группа №J3110

Принято

Должность, звание Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Цель работы: создать программное решение, реализующее поиск подмассивов размера K в массиве размера N, сумма элементов которых равна нулю.

Задачи:

1. Решить задачу алгоритмически
2. Реализовать алгоритм на языке C++
3. Протестировать код
4. Подготовить отчёт

Теоретическая подготовка

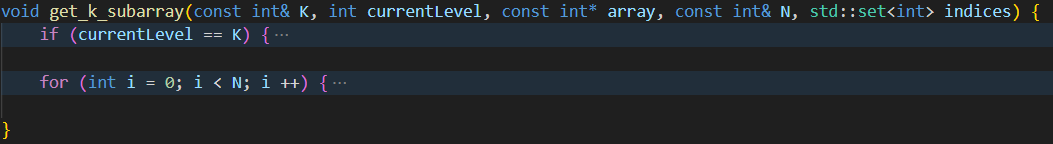
Основной тип данных в реализации – set. Данная структура данных была выбрана за счёт скорости поиска (O(log N)) и удобства хранения уникальных элементов, что в точности соответствует поставленной задаче.

В общих чертах алгоритм выглядит следующим образом:

* Перебираем все еще не использованные элементы исходного массива
* Для каждого элемента рекурсивно вызываем ту же функцию
* Повторяем алгоритм до тех пор, пока не достигнем требуемой глубины K

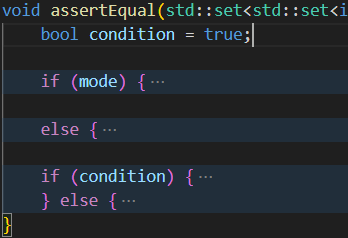
1. Реализация

Прежде всего, была реализована логика рекурсивной функции.



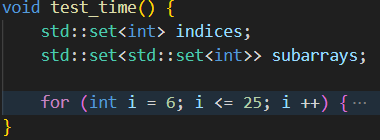
Изображение №1 – Логический облик рекурсивной функции

Затем были созданы тесты.



Изображение № 2 - Логический облик тестирующей функции

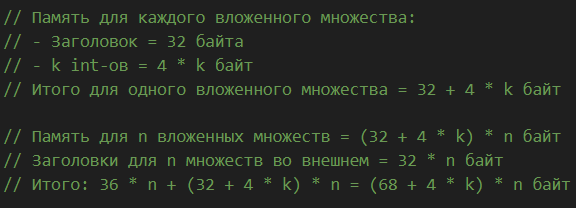
Последним этапом стало измерение времени в зависимости от числа элементов исходного массива.



Изображение № 3 - Логический облик функции, измеряющей время работы алгоритма

1. Экспериментальная часть

Подсчёт по памяти:



Изображение №4 – Подсчёт памяти 1



Изображение №5 – Подсчёт памяти 2



Изображение №6 – Подсчёт памяти 3

Подсчёт асимптотики:



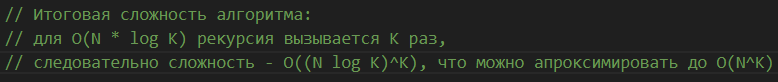
Изображение №7 – Подсчёт асимптотики 1



Изображение №8 – Подсчёт асимптотики 2



Изображение №9 – Подсчёт асимптотики 3



Изображение №10 – Подсчёт асимптотики 4

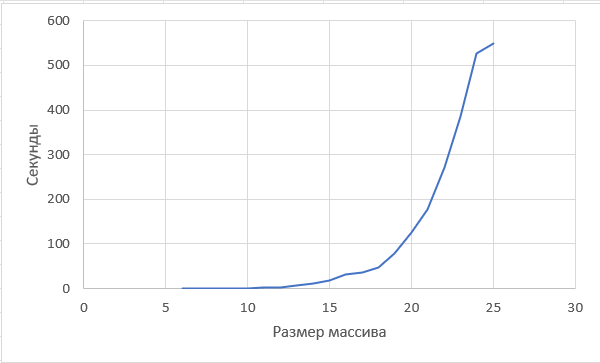
График зависимости времени от числа элементов:

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

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |
| --- | --- |
| Размер  массива | Время выполнения, с |
| 6 | 0.006494 |
| 7 | 0.035347 |
| 8 | 0.11746 |
| 9 | 0.341444 |
| 10 | 0.892427 |
| 11 | 1.69125 |
| 12 | 3.09677 |
| 13 | 5.60297 |
| 14 | 12.1333 |
| 15 | 18.2959 |
| 16 | 31.7406 |
| 17 | 36.6949 |
| 18 | 47.7722 |
| 19 | 77.7983 |
| 20 | 125.467 |
| 21 | 176.257 |
| 22 | 270.459 |
| 23 | 384.745 |
| 24 | 527.713 |
| 25 | 549.54 |

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №11.



Изображение №11 - График работы алгоритма

Из данных видно, что полученный на практике график довольно точно описывает теоретический график степенной функции. Некоторые отклонения могут быть вызваны “хорошими” случаями. Самое характерное из них наблюдается при 25 элементах.

1. Заключение

В ходе выполнения работы мною был реализован алгоритм поиска подмассивов размера K в массиве размера N, сумма элементов которых равна нулю. Цель работы была достигнута путём тестирования на массивах с различным количеством элементов и целевой длиной подмассива. Полученные результаты достаточно точно совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти, а также незначительно снизить асимптотику за счёт использования более подходящих структур данных.

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла k\_subarray.cpp

#include <iostream>

#include <vector>

#include <set>

#include <chrono>

#include <fstream>

#include <sstream>

#include <string>

void read\_csv(const std::string& filename, std::vector<int>& intArray) { // Функция чтения чисел из csv в массив

    std::ifstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file: " << filename << std::endl;

        return;

    }

    std::string line;

    while (std::getline(file, line)) {

        std::stringstream ss(line);

        std::string value;

        while (std::getline(ss, value, ';')) { // Split by comma

            try {

                int number = std::stoi(value); // Convert to int

                intArray.push\_back(number); // Add to the vector

            } catch (const std::invalid\_argument& e) {

                std::cerr << "Invalid number: " << value << std::endl;

            } catch (const std::out\_of\_range& e) {

                std::cerr << "Number out of range: " << value << std::endl;

            }

        }

    }

    file.close();

}

// Основная рекурсивная функция

void get\_k\_subarray(const int& K, int currentLevel, const int\* array, std::set<std::set<int>>& subarrays, const int& N, std::set<int> indices) {

    // Рекурсия вызывается K раз

    if (currentLevel == K) { // Если достигли нужной глубины рекурсии

        int summ = 0;

        for (const int& index : indices) // Проверяем сумму полученного подмассива

            summ += array[index]; // O(1) \* K - взятие по индексу

        if (summ == 0 && subarrays.find(indices) == subarrays.end()) // O(log N) - поиск по значению

            subarrays.insert(indices); // Если сумма 0 и такого подмассива ещё не было, добавляем к результату

        return;

    }

    //Для каждого не использованного до этого момента значения запускаем свою "ветвь" рекурсии

    for (int i = 0; i < N; i ++) {

        if (indices.find(i) == indices.end()) { // O(log K) \* N - поиск по значению

            std::set<int> next\_indices = indices; // (32 (заголовок) + 4 \* k (int-ы)) \* n байт

            next\_indices.insert(i);

            get\_k\_subarray(K, currentLevel + 1, array, subarrays, N, next\_indices);

        }

    }

    // Итоговая сложность алгоритма:

    // для O(N \* log K) рекурсия вызывается K раз,

    // следовательно сложность - O((N log K)^K), что можно апроксимировать до O(N^K)

    // Суммарные затраты памяти:

    // (68 + 4 \* k) \* n + (32 + 4 \* k) \* k \* n = (68 + 36 \* k + 4 \* k^2) \* n байт

}

int test\_passed = 0;

int test\_failed = 0;

// Главная тестирующая функция

void assertEqual(const int\* array, std::set<std::set<int>> real\_result, std::set<std::set<int>> expected\_result, const int& k, const bool mode, const std::string& testName) {

    bool condition = true;

    if (mode) { // Строгая проверка ответа

        condition = (real\_result == expected\_result); // Проверяем полное совпадение реального и ожидаемого результата

    }

    else { // Мягкая проверка ответа

        for (const auto& inner\_set : real\_result) {

            int tmp\_sum = 0;

            for (int value : inner\_set)

                tmp\_sum += array[value];

            if (tmp\_sum != 0 || inner\_set.size() != k) { // Проверяем равенство 0 суммы каждого подмассива и длину всех подмассивов

                condition = false;

                break;

            }

        }

    }

    if (condition) {

        std::cout << "[PASSED]" << testName << "\n";

        test\_passed++;

    } else {

        std::cout << "[NOT PASSED]" << testName << "\n";

        test\_failed++;

    }

}

// Вывод отчёта о тестах

void report() {

    std::cout << "\nTests passed: " << test\_passed << "\n";

    std::cout << "Tests not passed: " << test\_failed << "\n";

}

// Функция вывода множества

void display\_set(std::set<std::set<int>> target\_set) {

     for (const auto& innerSet : target\_set) {

        std::cout << "{ ";

        // Iterate over the inner set

        for (const auto& elem : innerSet) {

            std::cout << elem << " ";

        }

        std::cout << "}" << std::endl; // Close the brace for the inner set

    }

}

// Простые тесты из примера

void trivial\_tests() {

    std::set<int> indices;

    std::set<std::set<int>> subarrays;

    // Память для каждого вложенного множества:

    // - Заголовок = 32 байта

    // - k int-ов = 4 \* k байт

    // Итого для одного вложенного множества = 32 + 4 \* k байт

    // Память для n вложенных множеств = (32 + 4 \* k) \* n байт

    // Заголовки для n множеств во внешнем = 32 \* n байт

    // Итого: 36 \* n + (32 + 4 \* k) \* n = (68 + 4 \* k) \* n байт

    int arr1[4] = {1, -1, 2, -2};

    int k = 2;

    int N = sizeof(arr1) / sizeof(arr1[0]);

    get\_k\_subarray(k, 0, arr1, subarrays, N, indices);

    std::set<std::set<int>> expected\_result\_1;

    expected\_result\_1.insert({0, 1});

    expected\_result\_1.insert({2, 3});

    assertEqual(arr1, subarrays, expected\_result\_1, k, true, "Test: 1");

    subarrays.clear();

    int arr2[4] = {1, -1, 2, -3};

    k = 3;

    get\_k\_subarray(k, 0, arr2, subarrays, N, indices);

    std::set<std::set<int>> expected\_result\_2;

    expected\_result\_2.insert({0, 2, 3});

    assertEqual(arr2, subarrays, expected\_result\_2, k, true, "Test: 2");

    subarrays.clear();

    int arr3[4] = {1, 1, -1, -1};

    k = 2;

    get\_k\_subarray(k, 0, arr3, subarrays, N, indices);

    std::set<std::set<int>> expected\_result\_3;

    expected\_result\_3.insert({0, 2});

    expected\_result\_3.insert({0, 3});

    expected\_result\_3.insert({1, 2});

    expected\_result\_3.insert({1, 3});

    assertEqual(arr3, subarrays, expected\_result\_3, k, true, "Test: 3");

    subarrays.clear();

    report();

}

// Большой тест на данных из generate\_dataset

// (работает очень долго)

void big\_test() {

    std::vector<int> numbers;

    read\_csv("../data/numbers.csv", numbers);

    int arr[numbers.size()];

    for (int i = 0; i < numbers.size(); i ++)

        arr[i] = numbers[i];

    std::set<int> indices;

    std::set<std::set<int>> subarrays;

    int k = 6;

    int N = sizeof(arr) / sizeof(arr[0]);

    get\_k\_subarray(k, 0, arr, subarrays, N, indices);

    // display\_set(subarrays);

    std::set<std::set<int>> expected\_result\_4;

    assertEqual(arr, subarrays, expected\_result\_4, k, false, "Test: 4");

}

// Вычисление времени работы алгоритма

void test\_time() {

    std::set<int> indices;

    std::set<std::set<int>> subarrays;

    for (int i = 6; i <= 25; i ++) { // Проверяем для всех длин массива от 6 до 25

        int arr[i];

        for (int j = 0; j < i; j ++) { // Заполняем массив

            if (j % 2 == 0)

                arr[j] = 1;

            else

                arr[j] = -1;

        }

        int k = 6;

        int N = sizeof(arr) / sizeof(arr[0]);

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        get\_k\_subarray(k, 0, arr, subarrays, N, indices); // Запускаем алгоритм

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

        // Calculate the duration

        std::chrono::duration<double> duration = end - start;

        std::set<std::set<int>> tmp\_expected\_result;

        assertEqual(arr, subarrays, tmp\_expected\_result, k, false, "Test: " + std::to\_string(i)); // Проверяем корректность результата

        std::cout << "Execution time for " << i << " : " << duration.count() << " seconds" << std::endl; // Выводим время работы

    }

}

int main() {

    //trivial\_tests();

    big\_test();

    //test\_time();

    return 0;

}