**Министерство науки и высшего образования Российской Федерации**  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет цифровых трансформаций**

**Дисциплина:**

«Алгоритмы и структуры данных»

**ПРАКТИЧЕСКАЯ РАБОТА №4**

«NP полные задачи»

**Выполнил:**

Ступичев Алексей, студент группы J3110

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил(а):**

ФИО преподавателя, должность преподавателя

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

Содержание

[Содержание 2](#_Toc181934659)

[Введение 3](#_Toc181934660)

[Теоретическая часть 4](#_Toc181934661)

[Реализация 5](#_Toc181934662)

[Экспериментальна часть 7](#_Toc181934663)

[Заключение 10](#_Toc181934664)

[Приложение 11](#_Toc181934665)

Введение

**Цель работы** – научится решать NP полные задачи на примере задачи: Поиск подмассивов размера K в массиве размера N, сумма элементов которых равна нулю.

Задачи:

1. Придумать алгоритм для решения задачи.
2. Реализовать алгоритм на языке c++.
3. Составить тесты для различных данных.
4. Определить асимптотику алгоритма.
5. Составить отчет по работе.

ТЕОРЕТИЧЕСКАЯ подготовка

В работе не использовались специфические типы данных. В работе используются только функции, прописанные внутри. Для определения времени работы алгоритма подключена библиотека chrono.

Реализация

Чтобы решить данную задачу, нужно перебрать все подмассивы длины K, найти их сумму и проверить равна ли она 0. Если бы K было фиксированным, например 2, то можно было бы написать два цикла, перебирающих все пары. Но для нефиксированного K, нужно перебрать эти пары как-то по-другому. Для этого я написал две функции new\_indexes и new\_inexes\_increasing. Им подается набор индексов длины K, а они находят ближайшие следующие новые индексы. New\_index прибавляет к последнему индексу 1 и проверяет превысил ли этот индекс максимальный возможный, если да, то прибавляет 1 к предыдущему, а текущий обнуляет. Функция new\_inexes\_increasing проверяет на то, идут ли данные индексы в порядке строгого возрастания, что гарантирует несовпадение с уже бравшимися до этого индексами. Так же в обеих функциях есть проверка на последний возможный набор индексов, которым является N-K, N-K+1 ... N-1.

Далее нужна функция, находящая сумму элементов из данного массива по выданным индексам. Это реализовано в функции is\_sum\_zero, которая для каждого индекса вызывает соответствующее значение из массива и суммирует их. Затем проверяет, равна ли сумма 0 и возвращает true в случае равенства, и false в обратном.

Последний шаг - делать перебрать все подмассивы. Это сделано в основной функции subarrays\_with\_sum\_zero, в которой пока индексы не достигнут последнего, просматривается сумма для данного подмассива, а затем берутся новые индексы. Если сумма подмассива равна нулю, то он записывается в результат.

Экспериментальна часть

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов.

Оценка памяти:

В начале выделяется память под данный массив длины N.

Затем под массив индексов длины K. В фунциях new\_indexes и new\_indexes\_increasing меняются элементы массива индексов, в них затраты на дополнительную память нет.

В функции is\_sum\_zero так же не выделяется памяти под что-то существенное.

В основной функции создается массив ответов, в который будут попадать массивы индексов длины K. Его размер - от 0 до кол-ва подмассивов длины K в массиве длины N.

Подсчёт асимптотики:

Функция new\_indexes максимально проходит два раза по массиву длины K.

Функция new\_indexes\_increasing вызывает new\_indexes и проходит один раз по массиву длины K асимптотика выполнения одной итерации цикла - 3\*K, так до тех пор, пока не получится новые верные индексы. Максимум это может происходить в случае, когда получается послений набор индексов, это займет (N-K+1)(N-K+2)...(N-1)\*3\*K - асимптотика (N-1)^K

Функция is\_sum\_zero складывает K элементов. Асимптотика - O(K).

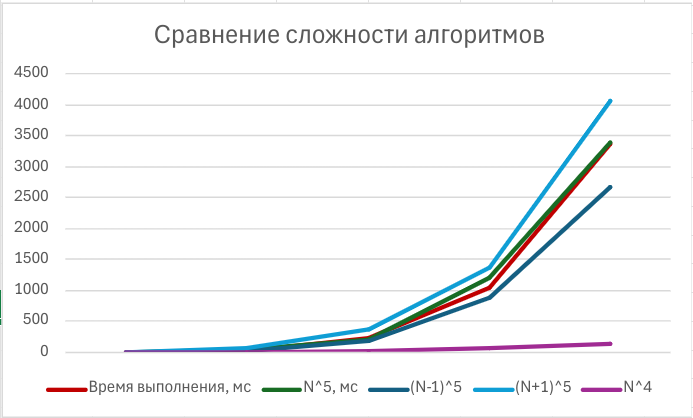
Основная функция идет по индексам из new\_index\_increasing, ее асимптотика - сумма асимптотик всех асимптотик new\_index\_increasing.

Это (N-K)(N-K+1)...(N-1) = O(N^K).

Теоретически заданная сложность задачи составляет O(N^K) с точностью до коэфицента и менее. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

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

Изображение №1 - График работы алгоритма

Анализ графика:

Время выполнения довольно точно совпадёт с теоретической оценкой. Это можно объяснить тем, что перебор элементов при одной и той же длине изначального массива одинаковый.

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. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения времени работы алгоритма, например улучшить нахождение новых индексов.

Приложение

Листинг кода файла main.cpp:

*//Поиск подмассивов размера K в массиве размера N, сумма элементов которых равна нулю, K = 5, N до 25*  
*#include <iostream>*  
*#include <vector>*  
*#include <chrono>*  
*using namespace std;*  
   
   
*vector<int> new\_indexes(vector<int> index, int N) {*  
    *// проверка на последний возможный вариант для индексов*  
    *bool flag = false;*  
    *if (index[0] == N - index.size()) {*  
        *flag = true;*  
        *for (int i = 1; i < index.size(); i++) {*  
            *if (index[i] != index[i - 1] + 1) {*  
                *flag = false;*  
            *}*  
        *}*  
    *}*  
   
    *if (flag) {*  
        *return vector(1, -1); // при последнем возможном варианте вернет vector(1,-1)*  
    *}*  
    *// конец проверки на последний возможный вариант для индексов*  
    *int i = index.size() - 1;*  
    *index[i]++;*  
    *while (index[i] == N) {*  
        *index[i] = 0;*  
        *i--;*  
        *index[i]++;*  
    *}*  
    *return index;*  
*}*  
   
*vector<int> new\_indexes\_increasing (vector<int> index, int N) {*  
    *bool flag = true;*  
    *while (flag){*  
        *index = new\_indexes(index, N);*  
        *if (index == vector(1,-1)) {*  
            *return vector(1,-1);*  
        *}*  
        *flag = false;*  
        *for (int i = 0; i < index.size() - 1; i++) {*  
            *if (index[i] >= index[i+1]) {*  
                *flag = true;*  
            *}*  
        *}*  
    *}*  
    *return index;*  
*}*  
   
   
   
*bool is\_sum\_zero(int arr[], vector<int> index, int K) {*  
    *long sum = 0;*  
    *for (int i = 0; i < K; i++) {*  
        *sum += arr[index[i]];*  
    *}*  
    *if (sum == 0) {*  
        *return true;*  
    *} else {*  
        *return false;*  
    *}*  
*}*  
   
   
   
*vector<vector<int>> subarrays\_with\_sum\_0(int arr[], int N, int K) {*  
    *vector<vector<int>> result;*  
    *if (K == 1* *&& arr[0] == 0) { // Проверь на пустой массив*  
        *result.push\_back({0});*  
    *}*  
    *vector<int> index = new\_indexes\_increasing(vector(K, 0), N);*  
    *while (index != vector(1, -1)) {*  
        *if (is\_sum\_zero(arr, index, K)) {*  
            *result.push\_back(index);*  
        *}*  
        *index = new\_indexes\_increasing(index, N);*  
    *}*  
    *return result;*  
*}*  
   
   
   
*int test\_passed = 0;*  
*int test\_failed = 0;*  
   
*void assertEqual(vector<vector<int>> condition, vector<vector<int>> need, const std::string& testName) {*  
    *if (condition == need) {*  
        *std::cout << "[PASSED]"* *<< testName << "\n";*  
        *test\_passed++;*  
    *} else {*  
        *std::cout << "[NOT PASSED]"* *<< testName << "\n";*  
        *test\_failed++;*  
    *}*  
*}*  
   
*void report() {*  
    *std::cout << "\nИтого тестов пройдено: "* *<< test\_passed << "\n";*  
    *std::cout << "\nИтого тестов не пройдено: "* *<< test\_failed << "\n";*  
*}*  
   
*void time(int arr[], int N, int K) {*  
    *auto* *start = chrono::high\_resolution\_clock::now();*  
    *vector<vector<int>> result = subarrays\_with\_sum\_0(arr, N, K);*  
    *auto* *end = chrono::high\_resolution\_clock::now();*  
    *auto* *duration = chrono::duration\_cast<chrono::milliseconds>(end - start);*  
    *cout << duration.count() << " milliseconds\n";*  
*}*  
   
   
*int main() {*  
    *int test = 0;*  
    *int arr1[] {1,-1,2,-2};*  
    *int arr2[] {1,-1,2,-3};*  
    *int arr3[] {1,1,-1,-1};*  
    *int arr4[] {0};*  
    *int arr5[] {0,1,1,1};*  
    *int arr6[] {1,2,3,-2,5,6,7,8,9,10,-1,12,13,-2,15,16,17,18,19,20,21,22,23,24,25};*  
    *int arr7[] {1,2,3,-2,5,6,7,8,9,10,-1,12,13,-2,15,16,17,18,19,20};*  
    *int arr8[] {1,2,3,-2,5,6,7,8,9,10,-1,12,13,-2,15};*  
    *int arr9[] {1,2,3,-2,5,6,7,8,9,10};*  
    *int arr10[] {1,2,3,-2,5};*  
    *assertEqual(subarrays\_with\_sum\_0(arr1, 4, 2),{{0,1},{2,3}}," test1 from task");*  
    *assertEqual(subarrays\_with\_sum\_0(arr2,4,3),{{0,2,3}}," test2 from task");*  
    *assertEqual(subarrays\_with\_sum\_0(arr3,4,2),{{0,2},{0,3},{1,2},{1,3}}," test3 from task");*  
    *assertEqual(subarrays\_with\_sum\_0(arr4, 1, 1),{{0},}," {0}");*  
    *assertEqual(subarrays\_with\_sum\_0(arr5, 4, 2),{}," without solutions");*  
    *assertEqual(subarrays\_with\_sum\_0(arr6, 25, 5),{{0,1,10,13,24},{2,3}}," test1 from task");*  
    *time(arr6, 25, 5);*  
    *time(arr7, 20, 5);*  
    *time(arr8, 15, 5);*  
    *time(arr9, 10, 5);*  
    *time(arr10, 5, 5);*  
    *report();*  
    *return 0;*  
   
*}*