ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 3

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

Выполнил работу

Бровкин Аким

Академическая группа №J3110

Принято

Ментор, Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Цель лабораторной работы — на практике изучить влияние высокой сложности NP-полных задач на время выполнение.

Задачи – изучить решение NP-полных задач, реализовать комбинаторный алгоритм, создать измерить время работы алгоритма, написать отчёт.

1. Теоретическая подготовка

Для реализации этой лабораторной работы потребовалось

* знание контейнеров vector
* знание файловых и строковых потоков
* знание библиотеки численных алгоритмов
* знание библиотеки измерения времени
* знание передачи функций и элементов по ссылке

1. Реализация

* Реализация алгоритма нахождения подмассивов, сумма элементов которых равна нулю
* Реализация алгоритма генерации следующей комбинации индексов
* Реализация тестирующей системы

1. Экспериментальная часть

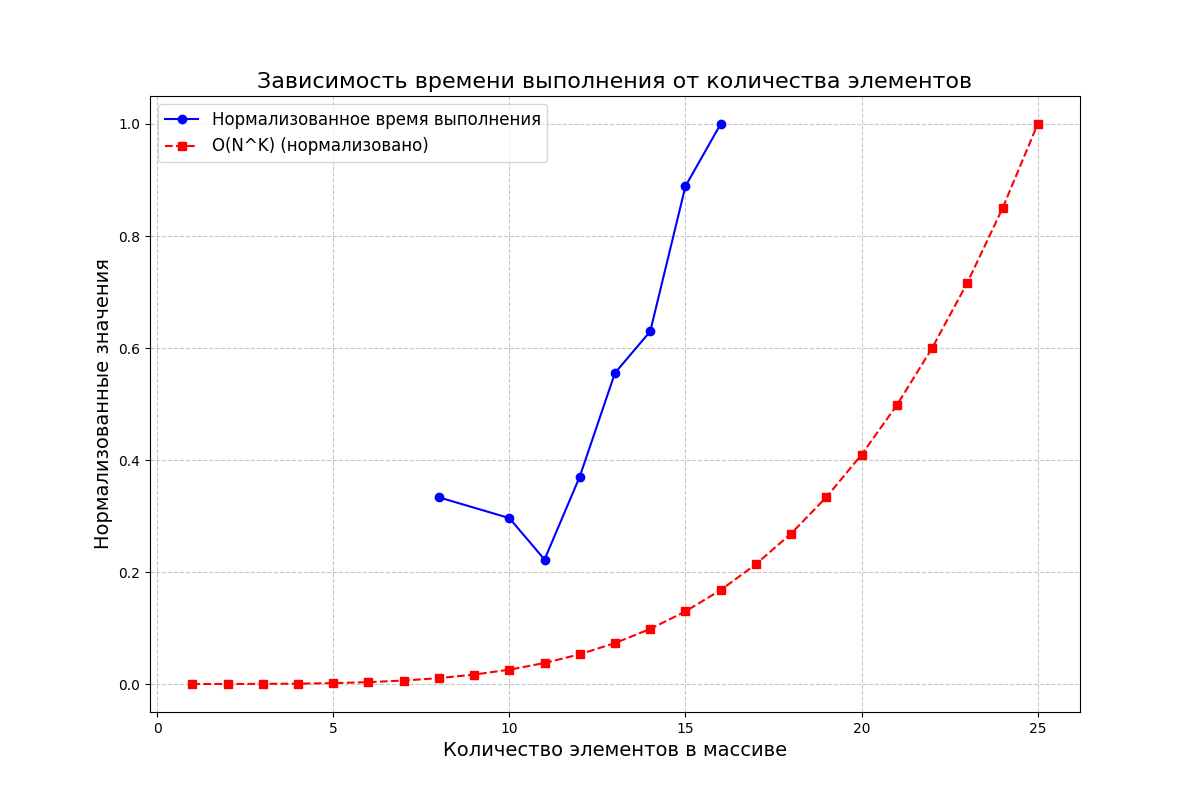
Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Требуемая сложность алгоритма, согласно заданию, должна быть больше N^K. Однако, даже реализовав алгоритм полного перебора, сложность вышла гораздо меньше.

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

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 8 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Время выполнения программы, с | 9e-5 | 8e-5 | 6e-5 | 1e-4 | 1,5e-4 | 1,7e-4 | 2,4e-4 | 2,7e-4 |
| O(25^4) | 152 | 95367 | 2384185 | 59604644 | 1490116119 | 37252902984 | 931322574615 | 23283064365386 |

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



Изображение 1 - График работы алгоритма

По графикам заметно, что время работы алгоритма не совпадает с условленной алгоритмической сложностью в виду отсутствия возможности реализации такого алгоритма с хотя бы приближенной сложностью.

Заключение.

В ходе выполнения работы мною был реализован алгоритм поиска подмассивов, сумма элементов которых равна 0, с заданным размером. В задании были противоречивые требования одновременной высокой алгоритмической сложности и самого алгоритма.

Полный перебор был реализован за счет генерации всех возможных подмассивов размера K.

Алгоритм и тестирующая система были реализованы на языке C++.

Листинг кода

#include <iostream>

#include <vector>

#include <numeric>

#include <algorithm>

using namespace std;

// Функция для вывода двумерного вектора

void print2DVector(const vector<vector<int> > &vec) {

for (size\_t i = 0; i < vec.size(); ++i) { // `size\_t i`: 8 байт, выделяется один раз

cout << "[";

for (size\_t j = 0; j < vec[i].size(); ++j) { // `size\_t j`: 8 байт, выделяется один раз

cout << vec[i][j];

if (j < vec[i].size() - 1) cout << ", ";

}

cout << "]";

if (i < vec.size() - 1) cout << ", ";

}

}

// Функция для генерации следующей комбинации индексов

bool next\_combination(vector<int>::iterator first, vector<int>::iterator last, int n) {

int k = last - first; // `int k`: 4 байта

vector<int>::reverse\_iterator rfirst = vector<int>::reverse\_iterator(last);

vector<int>::reverse\_iterator rlast = vector<int>::reverse\_iterator(first); // Итераторы rfirst и rlast: 8 байт каждый (16 байт в сумме)

// Ищем следующий элемент для комбинации

for (int i = 0; i < k; ++i) { // `int i`: 4 байта

if (rfirst[i] < n - i - 1) {

++rfirst[i];

for (int j = i - 1; j >= 0; --j) rfirst[j] = rfirst[j + 1] + 1; // `int j`: 4 байта, перезаписывается каждый раз

return true;

}

}

return false;

}

// Функция для поиска подмассивов с нулевой суммой размером K

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

vector<vector<int> > result; // `result` - двумерный вектор, служебные 24 байта + O(m \* K \* 4) байт на данные

int n = arr.size(); // `int n`: 4 байта

vector<int> indices(K); // `indices`: служебные 24 байта + K \* 4 байта на хранение данных

iota(indices.begin(), indices.end(), 0);

do {

if (indices.back() < n) {

int subarraySum = 0; // `subarraySum`: 4 байта

for (int idx: indices) {

subarraySum += arr[idx]; // `idx`: 4 байта, перезаписывается на каждом проходе цикла

}

if (subarraySum == 0) {

result.push\_back(indices); // При добавлении каждого подмассива памяти требуется 24 байта (служебная) + K \* 4 байта (данные)

}

}

} while (next\_combination(indices.begin(), indices.end(), n));

return result; // Итоговая память: 24 байта + 4 \* K \* m байт для всех подмассивов, где `m` — число подмассивов

}

int test\_passed = 0; // `int test\_passed`: 4 байта

int test\_failed = 0; // `int test\_failed`: 4 байта

// Функция для проверки равенства значений с выводом результата

void assertEqual(bool condition, const std::string &testName) {

if (condition) {

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

test\_passed++;

} else {

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

test\_failed++;

}

}

// Функция для отчета о результатах тестов

void report() {

std::cout << "\nQuantity of passed tests: " << test\_passed << "\n";

std::cout << "\nQuantity of not passed tests: " << test\_failed << "\n";

}

int main() {

// Примеры тестов

vector<int> arr1 = {1, -1, 2, -2, 3, -3, 4, -4}; // `arr1`: 24 байта (служебные) + 8 \* 4 байт (данные)

vector<vector<int> > expected1 = {

{0, 1, 2, 3}, {0, 1, 4, 5}, {0, 1, 6, 7}, {0, 3, 5, 6},

{1, 2, 4, 7}, {2, 3, 4, 5}, {2, 3, 6, 7}, {4, 5, 6, 7}

}; // `expected1`: служебные 24 байта + O(m \* K \* 4) байт на данные

assertEqual(findZeroSumSubarrays(arr1, 4) == expected1, " Test 1"); // Проверка теста 1

vector<int> arr2 = {3, -3, 6, -6, 9, -9, 0, 0, 0, 0}; // `arr2`: 24 байта (служебные) + 8 \* 4 байт (данные)

vector<vector<int> > expected2 = {

{0, 1, 2, 3}, {0, 1, 4, 5}, {0, 1, 6, 7}, {0, 1, 6, 8},

{0, 1, 6, 9}, {0, 1, 7, 8}, {0, 1, 7, 9}, {0, 1, 8, 9},

{0, 2, 5, 6}, {0, 2, 5, 7}, {0, 2, 5, 8}, {0, 2, 5, 9},

{1, 3, 4, 6}, {1, 3, 4, 7}, {1, 3, 4, 8}, {1, 3, 4, 9},

{2, 3, 4, 5}, {2, 3, 6, 7}, {2, 3, 6, 8}, {2, 3, 6, 9},

{2, 3, 7, 8}, {2, 3, 7, 9}, {2, 3, 8, 9}, {4, 5, 6, 7},

{4, 5, 6, 8}, {4, 5, 6, 9}, {4, 5, 7, 8}, {4, 5, 7, 9},

{4, 5, 8, 9}, {6, 7, 8, 9}

}; // `expected2`: служебные 24 байта + O(m \* K \* 4) байт на данные

assertEqual(findZeroSumSubarrays(arr2, 4) == expected2, " Test 2"); // Проверка теста 2

// Входной массив 3

vector<int> arr3 = {5, -5, 10, -10, 8000, 15789, -26786780, -3}; // `arr3`: 24 байта (служебные) + 8 \* 4 байт (данные)

vector<vector<int> > expected3 = {{0, 1, 2, 3}}; // `expected3`: служебные 24 байта + O(m \* K \* 4) байт на данные

assertEqual(findZeroSumSubarrays(arr3, 4) == expected3, " Test 3"); // Проверка теста 3

vector<int> arr4 = {7, 8, 9, 10, 11, 12}; // `arr4`: 24 байта (служебные) + 8 \* 4 байт (данные)

vector<vector<int> > expected4 = {}; // `expected4`: служебные 24 байта + O(m \* K \* 4) байт на данные

assertEqual(findZeroSumSubarrays(arr4, 4) == expected4, " Test 4"); // Проверка теста 4

vector<int> arr5 = {0, 0, 0, 0, 0, 0}; // `arr5`: 24 байта (служебные) + 8 \* 4 байт (данные)

vector<vector<int> > expected5 = {

{0, 1, 2, 3}, {0, 1, 2, 4}, {0, 1, 2, 5},

{0, 1, 3, 4}, {0, 1, 3, 5}, {0, 1, 4, 5},

{0, 2, 3, 4}, {0, 2, 3, 5}, {0, 2, 4, 5},

{0, 3, 4, 5}, {1, 2, 3, 4}, {1, 2, 3, 5},

{1, 2, 4, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}

}; // `expected5`: служебные 24 байта + O(m \* K \* 4) байт на данные

assertEqual(findZeroSumSubarrays(arr5, 4) == expected5, " Test 5"); // Проверка теста 5

report();

return 0;

}