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

Дубинина Диана

Академическая группа C3100

Принято

Ментор Вершинин Владислав

Санкт-Петербург

2024

Введение

Цель лабораторной работы — изучить алгоритмический подход к поиску подмассивов длины K в массиве размера N, для которых сумма элементов равна нулю. В процессе работы нужно разработать алгоритм, осуществляющий полный перебор всех возможных подмассивов указанного размера, и оценить его эффективность. Кроме того, необходимо экспериментально проанализировать временную сложность алгоритма, сопоставить теоретические расчёты с фактическим временем выполнения и определить, как размер массива влияет на производительность алгоритма.

1. Теоретическая подготовка

Для выполнения лабораторной работы требуется знание структуры массивов и принципов их индексации. Важно изучить существующие алгоритмы поиска подмассивов с заданными характеристиками, включая метод скользящего окна и использование хеш-таблиц для оптимизации процесса. Необходимо понимать, как вычисляется сумма элементов подмассива и каким образом проверяется условие равенства нулю. Кроме того, потребуется знание основ комбинаторики и методов формирования всех возможных подмассивов заданной длины KKK. Также важно владеть навыками оценки времени выполнения программы и анализа вычислительной сложности алгоритма.

1. Реализация

В процессе реализации были использованы стандартная библиотека iostream для вывода результата, vector для работы со структурой данных типа вектор, cmath используется для вычисления количества комбинаций N^K, chrono используется для измерения времени выполнения алгоритма, random используется для генерации случайных массивов в тестах. using namespace std упрощает использование стандартных библиотек, убирая необходимость писать std:: перед каждым вызовом, using namespace std::chrono упрощает вызов функций из chrono.

Задача решается в несколько этапов.

Этап 1: Инициализация и подсчёт всех возможных комбинаций

Подсчитывается общее количество комбинаций: totalCombinations=N^K

Здесь N - длина массива arr K - размер подмассива. Для этого используется функция pow из библиотеки <cmath>. Цикл проходит от 0 до totalCombinations, где каждое значение mask кодирует одну возможную комбинацию индексов.

Этап 2: Декодирование комбинации индексов из числа mask

Каждое значение mask интерпретируется как число в N-ичной системе счисления. Например: Для N=3,K=2, mask=5 соответствует индексу:

* + 5/3=1 (остаток 2) → Индекс 2.
  + 1/3=0 (остаток 1) → Индекс 1.
  + Результат: [1,2].

Реализация кодировки:

vector<int> indices;

int currentMask = mask;

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

int index = currentMask % N; // Остаток от деления даёт индекс

indices.push\_back(index);

currentMask /= N; // Переходим к следующему разряду

}

Этап 3: Проверка упорядоченности индексов

Проверяется, чтобы индексы в массиве indices были строго возрастающими (условие упорядоченности). Это нужно, чтобы исключить невалидные подмассивы (например, [2, 1, 3]).

Реализация кодировки:

bool isValid = true;

for (int i = 1; i < K; ++i) {

if (indices[i] <= indices[i - 1]) { // Проверяем порядок

isValid = false;

break;

}

}

if (!isValid) continue; // Пропускаем невалидную комбинацию

Если индексы неупорядочены, алгоритм пропускает эту комбинацию.

Этап 4: Подсчёт суммы элементов

Если комбинация индексов упорядочена, вычисляется сумма элементов массива arr, соответствующих этим индексам:

int sum = 0;

for (int index : indices) {

sum += arr[index]; // Суммируем элементы массива

}

Если сумма равна 0, комбинация добавляется в результирующий массив result:

if (sum == 0) {

result.push\_back(indices); // Добавляем комбинацию

}

Этап 5: Возврат результата

Когда все комбинации N^K обработаны, алгоритм возвращает результирующий массив result, содержащий все подходящие комбинации индексов.

Этап 6: тестирование и анализ времени

Тестирование реализуется через генерацию случайного массива определенного размера, значения в котором находятся в диапазоне [minValue,maxValue]

Установка фиксированных параметров:

const int K = 5; // Размер подмассива

vector<int> testSizes = {0, 5, 10, 15, 20, 25}; // Размеры массива для тестирования

int minValue = -10, maxValue = 10; // Диапазон значений элементов массива

K — это фиксированный размер подмассива, который ищется в алгоритме.

testSizes — это массив, содержащий размеры NNN, которые будут протестированы.

minValue и maxValue — минимальное и максимальное значение для элементов массива.

Основной цикл для тестирования реализуется через генерацию случайного массива, сразу после которого алгоритм находит время начала работы алгоритма. Далее вызывается функция findSubarraysWithZeroSum(arr, N, K); и после окончания работы находится время окончания. После алгоритм просчитывает длительность работы алгоритма и выводит результат в таблицу в соответствии с размером входного массива

auto duration = duration\_cast<milliseconds>(end - start).count();

cout << N << " | " << duration << " мс" << endl;

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=5 – размер подмассива.

Рассмотрим результирующий вектор комбинаций result:

Каждая комбинация состоит из K индексов. Для N=25 и K=5, общее количество комбинаций индексов, которые нужно проверить, можно посчитать по формуле биномиального коэффициента С(25,5):

C(25,5)== 53 130 комбинаций

Следовательно размер result равен 53 130\*k\*4=1 062 600 байт

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

Таким образом, общая память составляет:

44 + 1 062 600+4 = 1 062 648 байта ≈ 1.063 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 массива | Время выполнения программы, мс | O(N^K), K=4, мс | O(N^K), K=5, мс |
| 5 | 0,002 | 0,000625 | 0,000125 |
| 10 | 0,076 | 0,01 | 0,001 |
| 15 | 0,563 | 0,050625 | 0,003375 |
| 20 | 2,280 | 0,16 | 0,008 |
| 25 | 6,697 | 0,390625 | 0,015625 |

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 lab4.cpp

#include <iostream>

#include <vector>

#include <cmath>

#include <chrono>

#include <random>

using namespace std;

using namespace std::chrono;

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

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

vector<vector<int>> result;

// Генерация всех возможных комбинаций индексов размера K

int totalCombinations = pow(N, K); // Общее количество комбинаций (N^K)

for (int mask = 0; mask < totalCombinations; ++mask) {

vector<int> indices;

int currentMask = mask;

// Генерация индексов на основе текущей комбинации

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

int index = currentMask % N; // Получаем индекс из диапазона [0, N-1]

indices.push\_back(index);

currentMask /= N;

}

// Проверка на порядок индексов (они должны быть последовательными)

bool isValid = true;

for (int i = 1; i < K; ++i) {

if (indices[i] <= indices[i - 1]) {

isValid = false;

break;

}

}

// Если индексы неупорядочены, переходим к следующей комбинации

if (!isValid) continue;

// Подсчитываем сумму элементов для текущей комбинации индексов

int sum = 0;

for (int index : indices) {

sum += arr[index];

}

// Если сумма равна 0, добавляем индексы в результат

if (sum == 0) {

result.push\_back(indices);

}

}

return result;

}

// Функция для генерации случайного массива

vector<int> generateRandomArray(int size, int minValue, int maxValue) {

vector<int> arr(size);

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dis(minValue, maxValue);

for (int i = 0; i < size; ++i) {

arr[i] = dis(gen);

}

return arr;

}

int main() {

const int K = 5; // Размер подмассива

vector<int> testSizes = {0, 5, 10, 15, 20, 25}; // Размеры массива для тестирования

int minValue = -10, maxValue = 10; // Диапазон значений элементов массива

cout << "N (размер массива) | Время (мс)\n";

cout << "----------------------------------\n";

for (int N : testSizes) {

// Генерация случайного массива

vector<int> arr = generateRandomArray(N, minValue, maxValue);

// Замер времени

auto start = high\_resolution\_clock::now();

findSubarraysWithZeroSum(arr, N, K);

auto end = high\_resolution\_clock::now();

// Вычисление времени в миллисекундах

auto duration = duration\_cast<milliseconds>(end - start).count();

cout << N << " | " << duration << " мс" << endl;

}

return 0;

}