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

Филатова Элина

Академическая группа №группы

Принято

Практикант, Вершинин Владислав

Санкт-Петербург

2024

**Введение**

Цель работы – реализовать алгоритм для поиска подмассивов размера K (в частности K=4), сумма элементов которых равна 0, исходного массива размера N (до 25 элементов) и оценить производительность этого алгоритма посредством измерения времени выполнения программы.

Задачи:

Реализовать комбинированный алгоритм для поиска подмассивов, сумма которых равна 0.

Измерить время работы программы при различных входных наборах.

Оценить сложность алгоритма, заполнить таблицу результатами измерений и график зависимости времени работы алгоритма от числа элементов в входном наборе.

1. **Теоретическая подготовка**

При выполнении работы были использованы:

Типы данных:

int – целочисленный тип данных, используется для хранения целых чисел (размеры входных наборов (векторов), индексы элементов и сами значения наборов);

vector<int> - динамический массив (вектор) целых чисел; используется для хранения входных данных и для хранения комбинаций индексов;

vector<vector<int>> - используется для хранения всех найденных комбинаций индексов (каждый внутренний вектор представляет собой отдельную комбинацию);

bool – логический тип данных (принимает значения true или false); используется в функции GetNextCombination и указывает была ли найдена новая комбинация;

auto – ключевое слово, которое позволяет компилятору автоматически выводить тип переменной на основе ее инициализации; используется для объявления переменных, что позволяет сократить запись кода.

Алгоритм:

В данной работе используется алгоритм поиска всех комбинаций индексов фиксированной длины (K) из вектора целых чисел так, чтобы сумма элементов, соответствующих этим индексам, была равна нулю. Алгоритм основан на комбинаторном генерировании (генерирует все возможные комбинации) и проверке условий (для каждой сгенерированной комбинации индексов алгоритм проверяет, равна ли сумма элементов массива нулю).

Измерение времени:

Для измерения времени выполнения программы используется стандартная библиотека <chrono>, в которой есть несколько классов. В данной работе использовался класс chrono::high\_resolution\_clock, который предоставляет высокую точность измерения времени алгоритма.

1. **Реализация**

Подключение библиотек:

<iostream> - используется для ввода и вывода.

<vector> - используется для работы с динамическими массивами (векторами).

<cassert> - используется для тестирования, позволяет проверять условия во время выполнения программы.

<chrono> - предоставляет возможности для работы с временем. Используется для измерения производительности программы.

Объявление функций:

В начале программы объявляются использованные в алгоритме функции.

Реализация функций:

Функция SumVector:

Эта функция принимает вектор arr и вектор indexes. Она возвращает сумму элементов в векторе arr, соответствующих индексам в векторе indexes. Сумма вычисляется с помощью цикла for.

Функция GetFirstCombination:

Эта функция создает вектор длиной k, который заполняется числами от 0 до (K-1). Этот вектор соответствует первой комбинации индексов.

Функция GetNextCombination:

Эта функция генерирует следующие комбинации индексов, увеличивая текущий индекс и обновляя последующие, если это возможно. Возвращает true, если была сгенерирована новая комбинация, в противном случае – false.

Функция Split:

Эта функция принимает массив arr, элементы которого целые числа, и натуральное число k, и возвращает вектор комбинаций индексов, соответствующие элементам массива arr, сумма которых равна 0.

Сначала происходит необходимая проверка на длину исходного массива (она должна быть больше k)

Далее в цикле do-while проверяется каждая комбинация на сумму, начиная с первой (полученной в GetFirstCombination). Если сумма равна 0, комбинация добавляется в результирующий вектор result. После чего генерируется следующая комбинация индексов.

Измерение времени выполнения программы:

Функция PrintTime:

Эта функция рассчитывает и выводит время выполнения в наносекундах для наибольшей точности, используя функцию duration\_cast для преобразования разницы замеренного начального и конечного времени (stop - start) и перевод его в необходимый формат (наносекунды).

Функция Time:

Функция создает набор массивов и измеряет время выполнения функции Split в зависимости от размера конкретного набора.

Засекаем время до начала и после завершения выполнения алгоритма с помощью high\_resolution\_clock::now() (позволяет получить текущее время). После чего выводим результат с помощью функции PrintTime.

Тестирование, функция Test:

Эта функция проверяет корректность работы функции Split, сравнивая результат работы функции ожидаемым с помощью функции assert.

Основная функция:

В функции main вызывается функция Test, проверяя правильность выполнения программы.

1. **Экспериментальная часть**

В этом разделе вам необходимо привести результаты работы вашего алгоритма, с таблицами и графиками, демонстрирующими выполнения алгоритма с различными условиями и наборами данных. Оценивается производительность и сравниваются результаты с теоретическими оценками.

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

Паямть вектора sub\_arr\_inx можем посчитать аналогичным образом:

24 байта + K\*4 = 24+16 = 40 байт

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

Таким образом, общая память составляет:

40 + 202 400+40+4 = 202 484 байта ≈ 0.192 МБ

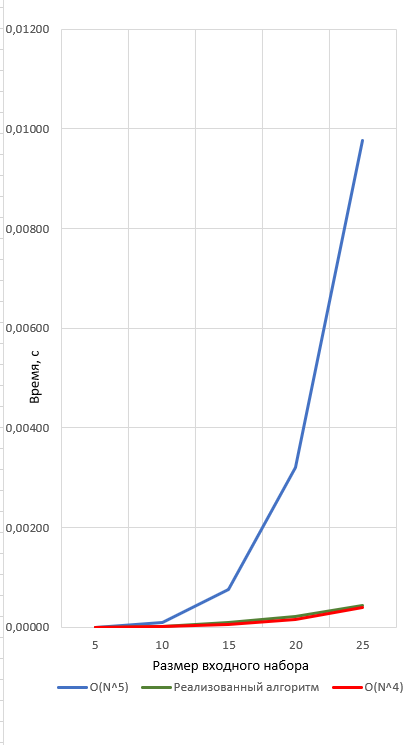
Подсчёт асимптотики:

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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер входного набора | 5 | 10 | 15 | 20 | 25 |
| Время выполнения программы, с | 0,0000011 | 0,0000127 | 0,000101 | 0,0002112 | 0,0004313 |
| O(N^K), K=4, с | 0,000000625 | 0,00001 | 0,000050625 | 0,00016 | 0,000390625 |
| O(N^K), K=5, с (доверительная асимптотика) | 0,000000125 | 0,000001 | 0,000003375 | 0,000008 | 0,000015625 |

Таблица №1 - Подсчёт сложности реализованного алгоритма

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №3.



Изображение №3 - 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.

На данном графике представлены разные возможные сложности алгоритмов, проанализировав графики, можно заметить, что график реализованного алгоритма значительно ниже графика O(N^5), но практически повторяет график O(N^4), то есть сложность данного алгоритма равна O(N^4), что удовлетворяет условию задачи.

Заключение

В ходе выполнения работы был реализован алгоритм поиска подмассивов, которые включают в себя индексы элементов исходного набора с суммой ноль. Данный алгоритм использует комбинаторный код для генерации всех возможных комбинаций индексов. Цель работы была достигнута путём реализации эффективного алгоритма поиска с подсчётом потребляемой памяти и последующим измерением времени выполнения. Полученные результаты подтверждают теоретические ожидания по использованию памяти и времени в зависимости от количества элементов в массиве и размера подмассива.

Данный алгоритм можно оптимизировать для уменьшения затрат памяти. Например, вместо создания множества векторов для хранения всех комбинаций, можно использовать один вектор и изменять его содержимое по мере необходимости. Это позволит избежать дополнительных затрат на память при создании временных векторов.

ПРИЛОЖЕНИЕ А

Полный код программы.

