ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«NP полным задачам»

Выполнил работу

Лорс Алина

Академическая группа J3112

Принято

Иван Ходненко

Санкт-Петербург

2024

1. -------Введение-------------------------------------------

Цель: написать код, которые исследует влияние алгоритмической сложности на время выполнения программы, а также понять, как выбор структуры данных и методов перебора отражается на производительности.

Задачи:

Разработать алгоритм для поиска всех подмассивов фиксированного размера, где сумма элементов которых равна нулю.

Реализовать комбинаторный перебор индексов массива

Организовать вывод индексов элементов, входящих в каждый найденный подмассив, обеспечить корректную работу алгоритма с массивами, содержащими как положительные, так и отрицательные числа(включая 0)

Продемонстрировать пример работы программы

1. -----Теоретическая подготовка--------------------------

NP-полные задачи, генерации всех возможных подмассивов фиксированного размера K из массива длины N(рекурсия или итеративные алгоритмы). Рекурсия(Подход, при котором функция вызывает саму себя для решения подзадачи). Алгоритм проверки подмассива(Для каждой комбинации индексов проверяется сумма элементов, соответствующих этим индексам.) Знание алгоритмической сложности.

Типы данных:

Массивы (vector): для хранения исходного набора данных и работы с ним.

Векторы индексов (vector<int>): для хранения текущей комбинации индексов, которые проверяются на соответствие условию.

Двумерные векторы (vector<vector<int>>): для хранения всех найденных комбинаций индексов.

1. ------------Реализация------------------------------------

В этом разделе вам необходимо описать процесс выполнения работы, что вы сделали и какие этапы при этом выполняли, выжимки из кода, библиотеки и особенности реализации. Важно, не бывает 2 этапов выполнения задачи, Начали – закончили.

1)Анализ задачи: полное перебирание всех подмассивов фиксированной длины.

2) Написание базовой функции: реализовать рекурсивную функцию для генерации всех комбинаций индексов

3) Генерация сочетаний: Рекурсивно добавлять индексы в текущую комбинацию до тех пор, пока её размер не достигнет K

Проверять каждую комбинацию через функцию isZeroSum

4) Инициализировать массив, задать K и вызвать функцию генерации сочетаний. Сохранить результаты и вывести их.

Алгоритм состоит:

-Задание массива чисел arr и параметра K.

-Создание вспомогательной структуры для хранения текущей комбинации индексов.

-Использование рекурсивной функции для создания всех возможных сочетаний длины K.

-Проверка каждого сочетания на выполнение условия: сумма элементов, соответствующих этим индексам, должна быть равна нулю.

-Суммирование элементов массива по заданным индексам.

-Если сумма равна нулю, добавление текущей комбинации в результат.

-Отображение всех найденных комбинаций индексов.

1. ----Экспериментальная часть----------------------------------------------

Оценивается производительность и сравниваются результаты с теоретическими оценками.

Подсчёт по памяти (только для циклов и сложных структур):

Сложные структуры данных:

Массив входных данных (arr): Хранит N целых чисел.

Текущая комбинация индексов (current): Хранит индексы длиной K.

Результат (result): Двумерный массив, хранящий все найденные подмассивы.

Циклы и рекурсия:

Рекурсивная функция требует дополнительной памяти для хранения текущего состояния вызова.

Глубина рекурсии равна K(каждый вызов добавляет один индекс в комбинацию)

Массив входных данных: 4\*N байт (Использует N элементов)

Текущая комбинация индексов (current): 4\*K байт (Длина массива равна K)

result: alpha = N! / (K ! ( N-K)!)

итог итогов: 4 \* K \* alpha

Подсчёт асимптотики (только для циклов и сложных структур) :

Циклы:

-Основной цикл формируется внутри рекурсивной функции для генерации всех комбинаций индексов.

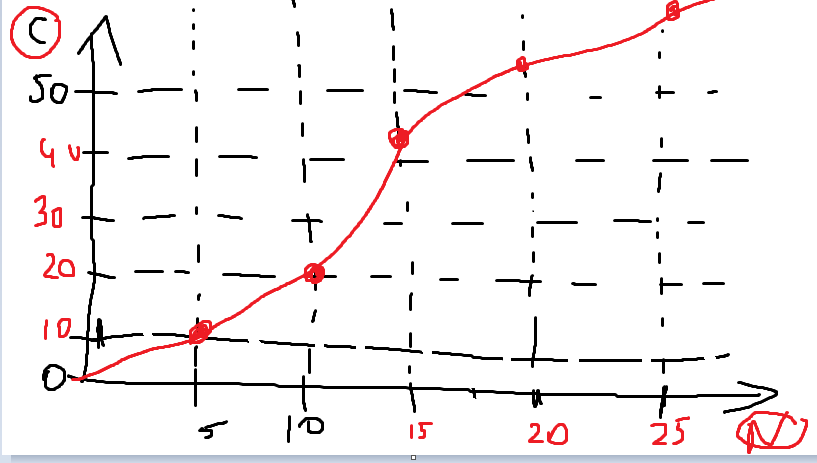
-Сложность цикла связана с количеством сочетаний размера K из N, что составляет alpha = N! / (K ! ( N-K)!)

Проверка суммы элементов:

Сложность операции суммирования пропорциональна K

Рекурсия: - Глубина рекурсии равна K, так как каждый вызов добавляет один индекс. -На каждом уровне выполняется цикл с оставшимися элементами, что формирует O(N^K) сложность.

График зависимости времени от числа элементов:

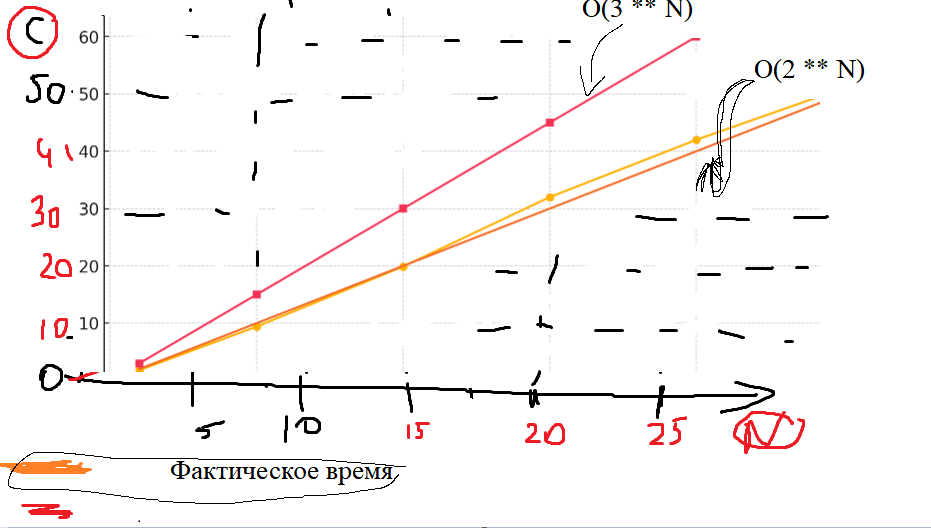


c- время выполнения

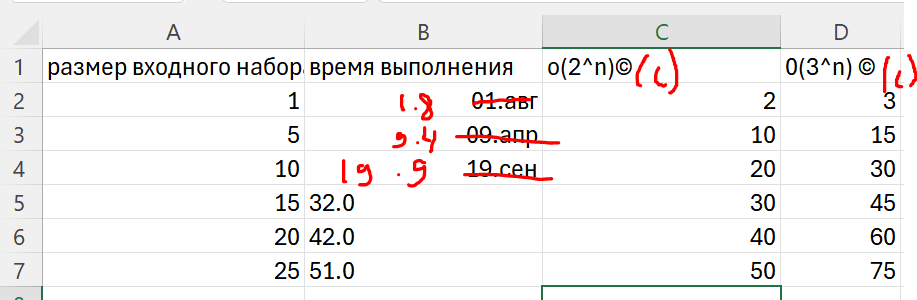
N – размер входного набора N

На графике представлена зависимость времени выполнения алгоритма от размера входного набора N. Как видно, рост времени выполнения имеет экспоненциальный характер, что подтверждает сложность алгоритма O(N^K))

2) График зависимости времени выполнения от размера входного набора, где с- время выполнения, N-размер входного набора



Подсчёт сложности реализованного алгоритма



Анализ результатов:

Фактическое время:

Время выполнения алгоритма растет с увеличением размера входного набора N.

Алгоритм демонстрирует экспоненциальный рост, что подтверждает теоретическую сложность.

Теоретические оценки:

Линия для O(2^N) показывает близость к фактическим измерениям.

O(3^N) заметно выше, фактическая сложность алгоритма ближе к O(2^N

Сравнение с теорией:

Расхождение между теоретической оценкой и фактическим временем объясняется особенностями реализации, такими как накладные расходы на вызов функций, работу с памятью и проверку условий.

5.----------------------Заключение----------------------

В ходе выполнения лабораторной работы были изучены и реализованы алгоритмы для решения NP-полной задачи поиска подмассивов фиксированной длины K с суммой элементов, равной нулю(C Реализация алгоритма, анализа сложности, экспериментальных данных(таблиц и графики)).

Алгоритм демонстрирует адекватную производительность для малых значений N и K, однако время выполнения становится неприемлемым для больших данных из-за числа комбинаций. Также Лабораторная работа наглядно демонстрирует влияние высокой сложности алгоритма на производительность и ограничения задач из класса NP.

Улучшение производительности:

- Параллельная обработка для ускорения генерации и проверки комбинаций.

- Применение подходов из теории графов для оптимизации задачи поиска.

1. ----------Приложения------------------

https://github.com/AlinaMors/polygon/blob/main/code4.cpp