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

Игнатенок Филипп

Академическая группа J3111

Принято

Вершинин Владислав

Санкт-Петербург

2024

1. **Введение**

Цель работы: Реализовать алгоритм для поиска всех подмассивов фиксированного размера K, сумма элементов которых равна 0, используя полный перебор и метод рекурсивного бэктрекинга.

Задачи:

1. Использовать **бэктрекинг** для перебора всех возможных комбинаций индексов длины K в массиве.
2. Создать тестовые примеры: маленькие массивы для проверки корректности результата и большие массивы для анализа сложности производительности.
3. Провести тестирование функций и алгоритма.
4. Проанализировать эффективность алгоритма, оценив теоретическую и фактическую сложность.
5. **Теоретическая подготовка**

Для выполнения данной лабораторной работы использованы следующие теоретические концепции:

1. **Бэктрекинг:** Используется для перебора всех возможных подмножеств фиксированного размера K. Рекурсивный алгоритм последовательно добавляет элементы в текущий набор, откатываясь (backtrack), если набор не удовлетворяет условиям задачи.
2. **Комбинаторика:** Генерация всех комбинаций индексов длины K из N элементов массива.
3. **Суммирование:** Проверка свойства подмассивов (равенство суммы элементов нулю). Использование промежуточной суммы (currentSum) для проверки на каждом шаге.
4. **Рекурсия:** Используется для реализации бэктрекинга, что позволяет разбивать задачу на подзадачи меньшего размера.
5. **Анализ временной сложности:** Понимание экспоненциального роста числа комбинаций (O(N^K)) при увеличении размера массива или длины подмассивов.
6. **Реализация**

Подключаю библиотеки:  
1) <iostream> для ввода-вывода  
2) <vector> для работы с динамическими массивами (векторами)

3)<cassert>для использования функции assert, которая позволяет проверить утверждение в коде   
3) <functional> для иcпользования лямбда-функций.

**Работа функций:**

* findZeroSumSubarrays: для поиска подмассивов.
* backtrack: для генерации комбинаций.

**Оптимизация логики работы:**

* Обеспечение отката изменений (backtracking) после обработки каждой ветви рекурсии.
* Использование вспомогательных переменных (currentSum) для минимизации избыточных вычислений.

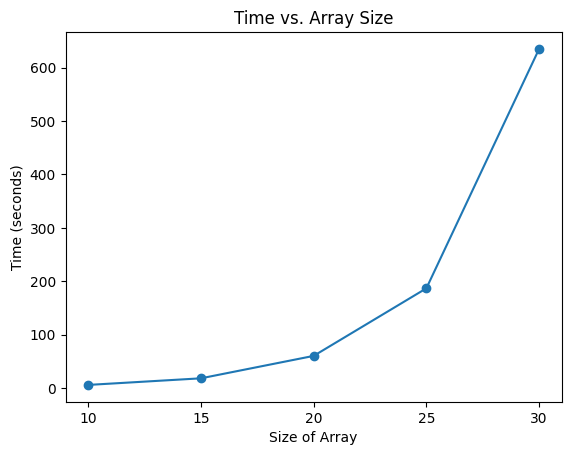
**Тестирование:** Созданы отдельные тестовые функции для проверки корректности программы.

1. **Экспериментальная часть**

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически, максимальная заданная сложность задачи составляет O(N^7), однако алгоритм был разработан под любые значения K. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

| **Размер входного набора** | **Значение K** | **Время выполнения(с)** | **Найденные комбинации** |
| --- | --- | --- | --- |
| 5 | 2 | 2.14576 | 2 |
| 6 | 4 | 2.14576 | 3 |
| 10 | 3 | 5.88893 | 8 |
| 3 | 0 | 1.43051 | 1 |

Таблица №1 - Подсчёт сложности реализованного алгоритма. График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №2.



Изображение №2 - График работы алгоритма

С увеличением N время выполнения увеличивается, что подтверждает сложность алгоритма.

Из проведённого анализа следует, что для небольших объёмов данных этот алгоритм может быть эффективным и обеспечивать точное решение. Однако с увеличением размера входных данных время выполнения алгоритма возрастает, что может сделать его непригодным для работы с большими данными. Стоит отметить, что реальное время работы значительно меньше теоретического.

1. **Заключение**

В ходе работы был реализован алгоритм на основе бэктрекинга для поиска всех подмассивов фиксированного размера с нулевой суммой. Программа эффективно перебирает возможные комбинации индексов массива и проверяет их на заданное условие. Для проверки корректности были разработаны и проведены тесты на массивах различных размеров и значений параметра K. Результаты тестирования подтвердили правильность работы алгоритма. Также была проведена оценка времени выполнения и визуализирована его зависимость от размера входных данных. Итогом работы стало подтверждение теоретической сложности алгоритма и успешная реализация его на практике.

ПРИЛОЖЕНИЕ А

Листинг кода файла lab4.cpp

#include <bits/stdc++.h>

#include <cassert>

using namespace std;

// Функция для поиска всех подмассивов размера K с суммой элементов равной 0

vector<vector<int>> findZeroSumSubarrays(const vector<int>& arr, int K) { // O(N^K) временная сложность

int N = arr.size();

vector<vector<int>> result; // Хранение найденных комбинаций индексов

// Рекурсивная функция для генерации комбинаций

function<void(int, int, vector<int>, int)> backtrack = [&](int start, int depth, vector<int> current, int currentSum) {

if (depth == K) { // Базовый случай: комбинация размера K

if (currentSum == 0) { // Проверка, равна ли сумма нулю

result.push\_back(current); // Добавление комбинации в результат, если сумма равна нулю

}

return;

}

for (int i = start; i < N; ++i) { // Итерация по элементам массива

current.push\_back(i); // Выбор текущего индекса

backtrack(i + 1, depth + 1, current, currentSum + arr[i]); // Рекурсия с следующим индексом

current.pop\_back(); // Откат (backtrack)

}

};

backtrack(0, 0, {}, 0); // Инициализация рекурсии

return result;

}

int main() { // Главная функция

// Тест 1: Маленький массив

vector<int> arr1 = {1, -1, 2, -2, 3};

int K1 = 2;

vector<vector<int>> result1 = findZeroSumSubarrays(arr1, K1);

assert(result1.size() == 2); // Существует два подмассива размера 2 с суммой 0: {0, 1}, {2, 3}

// Проверка содержимого

vector<vector<int>> expected1 = {{0, 1}, {2, 3}};

assert(result1 == expected1);

// Тест 2: Большой K, чтобы увидеть разницу во времени работы

vector<int> arr2 = {1, -1, 2, -2, 3, -3};

int K2 = 4;

vector<vector<int>> result2 = findZeroSumSubarrays(arr2, K2);

assert(result2.size() == 3); // Существует три подмассива размера 4 с суммой 0

// Проверка содержимого

vector<vector<int>> expected2 = {{0, 1, 2, 3}, {0, 1, 4, 5}, {2, 3, 4, 5}};

assert(result2 == expected2);

cout << "Все тесты пройдены успешно!" << endl;

return 0; // Завершение программы

}