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

Отчёт по лабораторной работе № 4

«Кластеризация массива посредством полного перебора всех комбинаций значений этого массива»

Выполнил работу

Бабич Александр

Академическая группа №J3111

Принято

Ментор, Вершинин Владислав

Санкт-Петербург

2024

Структура отчёта:

1. Введение

Цель работы: целью данной работы является разработка алгоритма для кластеризации массива чисел, который разбивает его на K кластеров с минимизацией суммы отклонений значений элементов каждого кластера от их среднего значения. **Задачи:** реализовать алгоритм, позволяющий находить все возможные разбиения массива на заданное количество кластеров. 2)Разработать функцию для вычисления метрики для каждого кластера. 3)Провести тестирование и оценить производительность алгоритма на различных наборах данных.

2. Теоретическая подготовка

Кластеризация - задача группировки объектов (в данном случае чисел) в такие группы (кластеры), что объекты внутри группы более похожи друг на друга, чем объекты из разных групп. Одной из распространенных метрик для измерения "похожести" является сумма абсолютных отклонений от среднего значения кластера.

Типы данных: Числа с плавающей точкой (double), целый числа (int). А также двумерный массив (vector) для хранения элементов массива и кластеров. Это позволяет динамически управлять размером массивов и легко добавлять или удалять элементы.

Алгоритмы: Полный перебор: Реализован рекурсивный алгоритм, который находит все возможные разбиения массива на K кластеров.

Вычисление метрики: для каждого кластера рассчитывается метрика, основанная на сумме абсолютных отклонений от среднего значения кластера.

3. Реализация

Моя реализация началась с прочтения документации, что такое кластеризация. Изучив её, я начал думать как ее реализовать, чтобы она работала примерно за O(2^N). Мне пришел на ум метод полного перебора, то есть рекурсивно. Далее я приступил к реализации моей программы. Я понял, что для реализации мне хватит 4 стандартных библиотек: iostream(для ввода-

вывода), vector(двумерный массив), limits(установка предельного значения), cassert(тестирование). И одного своего заголовочного файла с объявлением функций(рисунок1.1)

```
#ifndef POLYGON_TESTCLUSTER_H
#define POLYGON_TESTCLUSTER_H
#include <vector>

// Функция для вычисления метрики
double calculateMetric(const std::vector<double>& cluster);

// Рекурсивная функция для нахождения всех разбиений
void findClusters(const std::vector<double>& array, int index, int K,
                  std::vector<std::vector<std::vector<double>>>& allClusters,
                  std::vector<std::vector<double>>& currentClusters);

// Основная функция для нахождения кластеров
std::vector<std::vector<double>> findOptimalClusters(const std::vector<double>& array, int K);

// Функция для печати кластеров
void printClusters(const std::vector<std::vector<double>>& clusters);

// Функция для запуска тестов
void runTests();
#endif //POLYGON_TESTCLUSTER_H
```

Рисунок 1.1

После этого я приступил к реализации файла clusteritaion.cpp, где эти фукнции реализованы: Сначала реализовал функцию calculateMetric (рисунок1.2):

```cpp
// Функция для вычисления метрики кластера
double calculateMetric(const std::vector<double>& cluster) {
    double sum = 0;
    // Суммируем все значения в кластере
    for (double value : cluster) {
        sum += value;
    }
    // Вычисляем среднее значение
    double mean = sum / cluster.size();
    double metric = 0;
    // Считаем сумму абсолютных отклонений от среднего
    for (double value : cluster) {
        metric += std::abs( x: value - mean);
    }
    return metric; // Возвращаем вычисленную метрику
}
// Временная сложность: O(m), где m — количество элементов в кластере.
// Память: O(1), так как используем ссылку(передачу по адресу).
```

Рисунок 1.2

calculateMetric: Эта функция принимает кластер (вектор) и вычисляет его метрику на основе суммы абсолютных отклонений от среднего значения.

Дальше реализовал функцию findClusters (рисунок 1.3)

```cpp
// Рекурсивная функция для нахождения всех возможных кластеров
void findClusters(const std::vector<double>& array, int index, int K,
                  std::vector<std::vector<std::vector<double>>>& allClusters,
                  std::vector<std::vector<double>>& currentClusters) {
    // Если достигнут конец массива
    if (index == array.size()) {
        // Если текущий набор кластеров содержит K кластеров, сохраняем его
        if (currentClusters.size() == K) {
            allClusters.push_back(currentClusters);
        }
        return;
    }

    // Рекурсивно добавляем элементы в существующие кластеры
    for (int i = 0; i < currentClusters.size(); ++i) {
        currentClusters[i].push_back(array[index]); // Добавляем элемент
        findClusters(array, index: index + 1, K, &: allClusters, &: currentClusters); // Рекурсивный вызов
        currentClusters[i].pop_back(); // Удаляем элемент для следующего вызова
    }

    // Добавляем новый кластер, если еще не достигли K
    if (currentClusters.size() < K) {
        currentClusters.push_back(std::vector<double>{array[index]}); // Создаем новый кластер
        findClusters(array, index: index + 1, K, &: allClusters, &: currentClusters); // Рекурсивный вызов
        currentClusters.pop_back(); // Удаляем новый кластер для следующего вызова
    }
}
// Временная сложность: O(K^n), где n — количество элементов в массиве и K — количество кластеров.
// Память: O(K^n), так как сохраняем все возможные комбинации кластеров в allClusters.
```

Рисунок 1.3

findClusters: Эта рекурсивная функция генерирует все возможные разбиения массива на K кластеров. Важно помнить, что для каждого элемента массива необходимо решать, добавлять ли его в существующий кластер или создавать новый.

А затем реализовал findOptimalClusters, которая выбирает разбиение с минимальной метрикой (рисунок 1.4)

```cpp
std::vector<std::vector<double>> findOptimalClusters(const std::vector<double>& array, int K) {
    std::vector<std::vector<std::vector<double>>> allClusters;  // Для хранения всех кластеров
    std::vector<std::vector<double>> currentClusters;  // Для текущих кластеров

    findClusters(array, index: 0, K, &: allClusters, &: currentClusters);  // Запуск функции

    // Поиск кластера с минимальной метрикой
    double minMetric = std::numeric_limits<double>::max();  // Инициализируем минимальную метрику
    std::vector<std::vector<double>> bestClusters;  // Для хранения лучших кластеров

    // Проход по всем кластерам, чтобы найти минимальную метрику
    for (const auto& clusters : vector<...> const &  : allClusters) {
        double totalMetric = 0;
        for (const auto& cluster : vector<double> const &  : clusters) {
            totalMetric += calculateMetric(cluster);  // Суммируем метрики всех кластеров
        }
        if (totalMetric < minMetric) {  // Если найден новый минимум
            minMetric = totalMetric;
            bestClusters = clusters;  // Сохраняем лучшие кластеры
        }
    }

    return bestClusters;  // Возвращаем лучшие кластеры
}

// Временная сложность: O(K^n * m), где n — количество элементов в массиве, K — количество кластеров и m — размер
// Память: O(K^n), так как храним все возможные комбинации кластеров в allClusters и лучшие кластеры в bestCluster
```

Рисунок1.4

Реализовал функцию для печати кластеров(Рисунок 1.5)

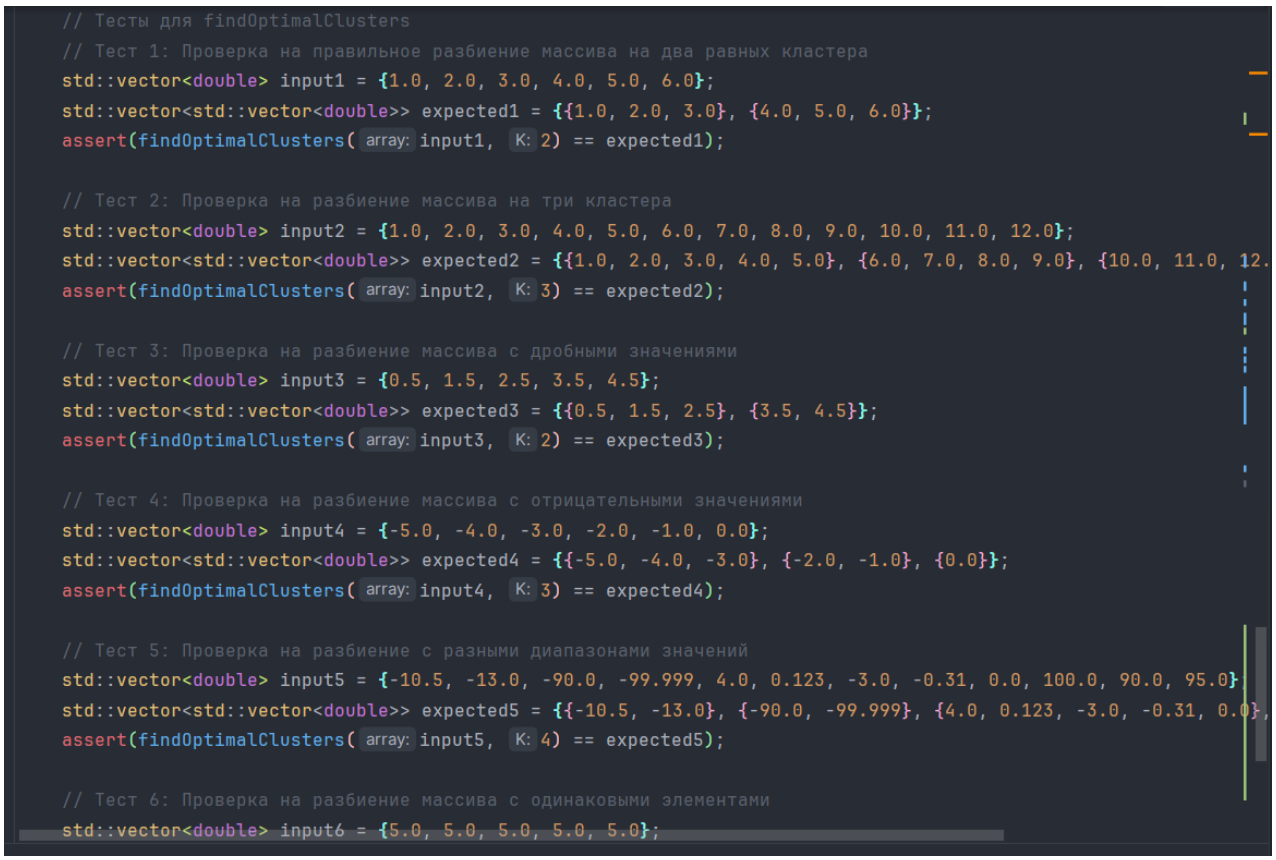
```cpp
void printClusters(const std::vector<std::vector<double>>& clusters) {
    for (const auto& cluster : vector<double> const &  : clusters) {
        std::cout << "[ ";
        for (double element : cluster) {
            std::cout << element << " ";
        }
        std::cout << "] ";
    }
    std::cout << std::endl;
}
```

Рисунок 1.5

И наконец написан функцию тестирования программы runTests, разбитую на 2 части (рисунки 1.6 и 1.7)

```
    // 1. Тест с положительными целыми числами
    std::vector<double> testCluster1 = {1, 2, 3, 4, 5};
    assert(calculateMetric( cluster: testCluster1) == 6.0); // Ожидаемое значение: 6.0 (отклонения от 3)

    // 2. Тест с равными значениями
    std::vector<double> testCluster2 = {5, 5, 5};
    assert(calculateMetric( cluster: testCluster2) == 0.0); // Ожидаемое значение: 0.0 (отклонения от 5)

    // 3. Тест с дробными числами
    std::vector<double> testCluster3 = {0.5, 1.5, 2.5};
    assert(calculateMetric( cluster: testCluster3) == 2.0); // Ожидаемое значение: 2.0 (отклонения от 1.5)

    // 4. Тест с отрицательными числами
    std::vector<double> testCluster4 = {-1, -2, -3};
    assert(calculateMetric( cluster: testCluster4) == 2.0); // Ожидаемое значение: 2.0 (отклонения от -2)

    // 5. Тест с отрицательными и положительными числами
    std::vector<double> testCluster5 = {-5.5, 0.0, 5.5};
    assert(calculateMetric( cluster: testCluster5) == 11.0); // Ожидаемое значение: 11.0 (отклонения от 0)

    // 6. Тест с пустым кластером
    std::vector<double> testCluster6 = {};
    assert(calculateMetric( cluster: testCluster6) == 0.0); // Ожидаемое значение: 0.0 (пустой кластер)

    // 7. Просто тест с большим отклонением
    std::vector<double> testCluster7 = {10, 20, 30, 40, 50};
    assert(calculateMetric( cluster: testCluster7) == 60.0); // Ожидаемое значение: 60.0 (отклонения от 30)
```

Рисунок 1.6



```
// Тесты для findOptimalClusters
// Тест 1: Проверка на правильное разбиение массива на два равных кластера
std::vector<double> input1 = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0};
std::vector<std::vector<double>> expected1 = {{1.0, 2.0, 3.0}, {4.0, 5.0, 6.0}};
assert(findOptimalClusters( array: input1,  K: 2) == expected1);

// Тест 2: Проверка на разбиение массива на три кластера
std::vector<double> input2 = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0};
std::vector<std::vector<double>> expected2 = {{1.0, 2.0, 3.0, 4.0, 5.0}, {6.0, 7.0, 8.0, 9.0}, {10.0, 11.0, 12.
assert(findOptimalClusters( array: input2,  K: 3) == expected2);

// Тест 3: Проверка на разбиение массива с дробными значениями
std::vector<double> input3 = {0.5, 1.5, 2.5, 3.5, 4.5};
std::vector<std::vector<double>> expected3 = {{0.5, 1.5, 2.5}, {3.5, 4.5}};
assert(findOptimalClusters( array: input3,  K: 2) == expected3);

// Тест 4: Проверка на разбиение массива с отрицательными значениями
std::vector<double> input4 = {-5.0, -4.0, -3.0, -2.0, -1.0, 0.0};
std::vector<std::vector<double>> expected4 = {{-5.0, -4.0, -3.0}, {-2.0, -1.0}, {0.0}};
assert(findOptimalClusters( array: input4,  K: 3) == expected4);

// Тест 5: Проверка на разбиение с разными диапазонами значений
std::vector<double> input5 = {-10.5, -13.0, -90.0, -99.999, 4.0, 0.123, -3.0, -0.31, 0.0, 100.0, 90.0, 95.0}
std::vector<std::vector<double>> expected5 = {{-10.5, -13.0}, {-90.0, -99.999}, {4.0, 0.123, -3.0, -0.31, 0.0},
assert(findOptimalClusters( array: input5,  K: 4) == expected5);

// Тест 6: Проверка на разбиение массива с одинаковыми элементами
std::vector<double> input6 = {5.0, 5.0, 5.0, 5.0, 5.0};
```

Рисунок 1.7

Затем был реализован третий (так называемый мейн файл) с запуском тестов и ручным вводом-выводом. (Рисунок 1.8)

```
int main() {
    runTests(); // Запускаем тесты

    int K;
    std::cout << "Enter the number of clusters (K):\n";
    std::cin >> K;

    int n;
    std::cout << "Enter the number of elements in the array (>=K):\n ";
    std::cin >> n;

    std::vector<double> array(n);
    std::cout << "Enter the elements of the array:\n ";
    for (int i = 0; i < n; ++i) {
        std::cin >> array[i];
    }

    findOptimalClusters(array, K);
    printClusters( clusters: findOptimalClusters(array,K));
    return 0;
}
```

Рисунок 1.8

4. Экспериментальная часть

Подсчёт по памяти Общая память:

В худшем случае, общая память будет O(K^n), так как мы храним все возможные комбинации кластеров в allClusters.

Дополнительно, в памяти будет O(n) для хранения входного массива, где n — количество элементов.

Общая память: O (K^n + n).

Подсчёт асимптотики Общая временная сложность алгоритма:

В худшем случае, общая временная сложность будет O (K^n * m), где n — количество элементов в массиве, K — количество кластеров,

и m — размер кластера. Для каждого разбиения (K) мы вычисляем метрику для каждого кластера.

График зависимости времени от числа элементов. Пример выполнения:

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(2^N) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

Размер входного набора	1	5	10	15
Количество кластеров	1	2	3	3
Время выполнения программы, с	0.000	0.001	0.033	3.43
O (K^n * m),с	1	96	236196	717444535

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

Изображение №1 - График работы алгоритма

На графике и таблице видно, как быстро растет скорость выполнения алгоритма при повышении количества элементов и количества кластеров, так как происходит полный перебор. Следовательно это NP полная задача.

5. Заключение

В данной работе была успешно выполнена задача кластеризации массива, сохраняя порядок элементов и минимизируя сумму отклонений значений в каждом кластере от их среднего. Я разработал алгоритм на языке программирования C++, который обеспечивает эффективное разбиение входного массива на заданное количество кластеров, принимая во внимание как положительные, так и отрицательные числа. Также проведенные тесты подтвердили корректность работы алгоритма и его способность обрабатывать различные сценарии. В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти и ускорения его работы по времени

6. Приложения

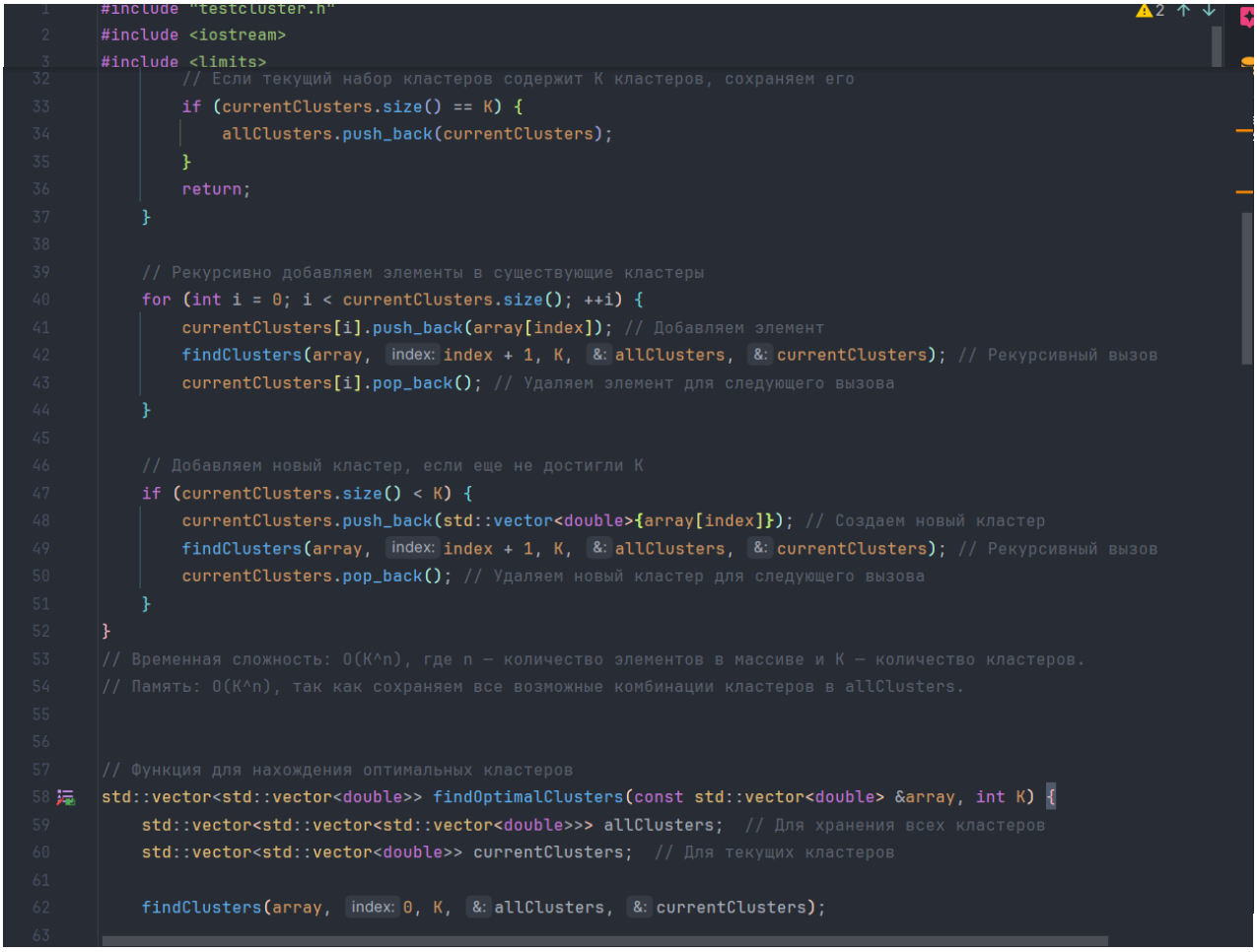
ПРИЛОЖЕНИЕ А

Листинг кода файла testcluster.h

```
#ifndef POLYGON_TESTCLUSTER_H
#define POLYGON_TESTCLUSTER_H

#include <vector>

// Функция для вычисления метрики
double calculateMetric(const std::vector<double> &cluster);

// Рекурсивная функция для нахождения всех разбиений
void findClusters(const std::vector<double> &array, int index, int K,
                  std::vector<std::vector<std::vector<double>>> &allClusters,
                  std::vector<std::vector<double>> &currentClusters);

// Основная функция для нахождения кластеров
std::vector<std::vector<double>> findOptimalClusters(const std::vector<double> &array, int K);

// Функция для печати кластеров
void printClusters(const std::vector<std::vector<double>> &clusters);

// Функция для запуска тестов
void runTests();

#endif //POLYGON_TESTCLUSTER_H
```

ПРИЛОЖЕНИЕ Б

Листинг кода файла clusteritaion.cpp



```
#include "testcluster.h"
#include <iostream>
#include <limits>
        // Если текущий набор кластеров содержит K кластеров, сохраняем его
        if (currentClusters.size() == K) {
            allClusters.push_back(currentClusters);
        }
        return;
    }

    // Рекурсивно добавляем элементы в существующие кластеры
    for (int i = 0; i < currentClusters.size(); ++i) {
        currentClusters[i].push_back(array[index]); // Добавляем элемент
        findClusters(array, index: index + 1, K, &: allClusters, &: currentClusters); // Рекурсивный вызов
        currentClusters[i].pop_back(); // Удаляем элемент для следующего вызова
    }

    // Добавляем новый кластер, если еще не достигли K
    if (currentClusters.size() < K) {
        currentClusters.push_back(std::vector<double>{array[index]}); // Создаем новый кластер
        findClusters(array, index: index + 1, K, &: allClusters, &: currentClusters); // Рекурсивный вызов
        currentClusters.pop_back(); // Удаляем новый кластер для следующего вызова
    }
}
// Временная сложность: O(K^n), где n — количество элементов в массиве и K — количество кластеров.
// Память: O(K^n), так как сохраняем все возможные комбинации кластеров в allClusters.


// Функция для нахождения оптимальных кластеров
std::vector<std::vector<double>> findOptimalClusters(const std::vector<double> &array, int K) {
    std::vector<std::vector<std::vector<double>>> allClusters;  // Для хранения всех кластеров
    std::vector<std::vector<double>> currentClusters;  // Для текущих кластеров

    findClusters(array, index: 0, K, &: allClusters, &: currentClusters);
```

```
    findClusters(array, index: 0, K, &: allClusters, &: currentClusters);

    // Поиск кластера с минимальной метрикой
    double minMetric = std::numeric_limits<double>::max();  // Инициализируем минимальную метрику
    std::vector<std::vector<double>> bestClusters;  // Для хранения лучших кластеров

    // Проход по всем кластерам, чтобы найти минимальную метрику
    for (const auto &clusters : vector<...> const & : allClusters) {
        double totalMetric = 0;
        for (const auto &cluster : vector<double> const & : clusters) {
            totalMetric += calculateMetric(cluster);  // Суммируем метрики всех кластеров
        }
        if (totalMetric < minMetric) {  // Если найден новый минимум
            minMetric = totalMetric;
            bestClusters = clusters;  // Сохраняем лучшие кластеры
        }
    }

    return bestClusters;  // Возвращаем лучшие кластеры
}

// Временная сложность: O(K^n * m), где n — количество элементов в массиве, K — количество кластеров и m — размер к
// Память: O(K^n), так как храним все возможные комбинации кластеров в allClusters и лучшие кластеры в bestClusters

// Функция для печати кластеров
void printClusters(const std::vector<std::vector<double>> &clusters) {
    for (const auto &cluster : vector<double> const & : clusters) {
        std::cout << "[ ";
        for (double element: cluster) {
            std::cout << element << " ";
        }
        std::cout << "] ";
```

f findOptimalClusters

```
    }
    std::cout << std::endl;
}


void runTests() {
    // Тесты для calculateMetric
    // 1. Тест с положительными целыми числами
    std::vector<double> testCluster1 = {1, 2, 3, 4, 5};
    assert(calculateMetric( cluster: testCluster1) == 6.0); // Ожидаемое значение: 6.0 (отклонения от 3)

    // 2. Тест с равными значениями
    std::vector<double> testCluster2 = {5, 5, 5};
    assert(calculateMetric( cluster: testCluster2) == 0.0); // Ожидаемое значение: 0.0 (отклонения от 5)

    // 3. Тест с дробными числами
    std::vector<double> testCluster3 = {0.5, 1.5, 2.5};
    assert(calculateMetric( cluster: testCluster3) == 2.0); // Ожидаемое значение: 2.0 (отклонения от 1.5)

    // 4. Тест с отрицательными числами
    std::vector<double> testCluster4 = {-1, -2, -3};
    assert(calculateMetric( cluster: testCluster4) == 2.0); // Ожидаемое значение: 2.0 (отклонения от -2)

    // 5. Тест с отрицательными и положительными числами
    std::vector<double> testCluster5 = {-5.5, 0.0, 5.5};
    assert(calculateMetric( cluster: testCluster5) == 11.0); // Ожидаемое значение: 11.0 (отклонения от 0)

    // 6. Тест с пустым кластером
    std::vector<double> testCluster6 = {};
    assert(calculateMetric( cluster: testCluster6) == 0.0); // Ожидаемое значение: 0.0 (пустой кластер)
```

```
    // 7. Просто тест с большим отклонением
    std::vector<double> testCluster7 = {10, 20, 30, 40, 50};
    assert(calculateMetric( cluster: testCluster7) == 60.0); // Ожидаемое значение: 60.0 (отклонения от 30)

    std::cout << "All tests for calculateMetric passed!" << std::endl;

    // Тесты для findOptimalClusters
    // Тест 1: Проверка на правильное разбиение массива на два равных кластера
    std::vector<double> input1 = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0};
    std::vector<std::vector<double>> expected1 = {{1.0, 2.0, 3.0},
                                                  {4.0, 5.0, 6.0}};
    assert(findOptimalClusters( array: input1,  K: 2) == expected1);

    // Тест 2: Проверка на разбиение массива на три кластера
    std::vector<double> input2 = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0};
    std::vector<std::vector<double>> expected2 = {{1.0,  2.0,  3.0, 4.0, 5.0},
                                                  {6.0,  7.0,  8.0, 9.0},
                                                  {10.0, 11.0, 12.0}};
    assert(findOptimalClusters( array: input2,  K: 3) == expected2);

    // Тест 3: Проверка на разбиение массива с дробными значениями
    std::vector<double> input3 = {0.5, 1.5, 2.5, 3.5, 4.5};
    std::vector<std::vector<double>> expected3 = {{0.5, 1.5, 2.5},
                                                  {3.5, 4.5}};
    assert(findOptimalClusters( array: input3,  K: 2) == expected3);

    // Тест 4: Проверка на разбиение массива с отрицательными значениями
    std::vector<double> input4 = {-5.0, -4.0, -3.0, -2.0, -1.0, 0.0};
    std::vector<std::vector<double>> expected4 = {{-5.0, -4.0, -3.0},
```

```
    // Тест 4: Проверка на разбиение массива с отрицательными значениями
    std::vector<double> input4 = {-5.0, -4.0, -3.0, -2.0, -1.0, 0.0};
    std::vector<std::vector<double>> expected4 = {{-5.0, -4.0, -3.0},
                                                  {-2.0, -1.0},
                                                  {0.0}};
    assert(findOptimalClusters( array: input4,  K: 3) == expected4);

    // Тест 5: Проверка на разбиение с разными диапазонами значений
    std::vector<double> input5 = {-10.5, -13.0, -90.0, -99.999, 4.0, 0.123, -3.0, -0.31, 0.0, 100.0, 90.0, 95.0};
    std::vector<std::vector<double>> expected5 = {{-10.5, -13.0},
                                                  {-90.0, -99.999},
                                                  {4.0,   0.123, -3.0, -0.31, 0.0},
                                                  {100.0, 90.0,  95.0}};
    assert(findOptimalClusters( array: input5,  K: 4) == expected5);

    // Тест 6: Проверка на разбиение массива с одинаковыми элементами
    std::vector<double> input6 = {5.0, 5.0, 5.0, 5.0, 5.0};
    std::vector<std::vector<double>> expected6 = {{5.0, 5.0, 5.0, 5.0},
                                                  {5.0}};
    assert(findOptimalClusters( array: input6,  K: 2) == expected6);

    // Тест 7: Проверка на разбиение массива с экстремальными значениями
    std::vector<double> input7 = {-1000.0, -500.0, 0.0, 500.0, 1000.0};
    std::vector<std::vector<double>> expected7 = {{-1000.0, -500.0, 0.0},
                                                  {500.0},
                                                  {1000.0}};
    assert(findOptimalClusters( array: input7,  K: 3) == expected7);

    std::cout << "All tests for findOptimalClusters passed!" << std::endl;
}
```

ПРИЛОЖЕНИЕ В

Листинг кода файла maincluster.cpp

```
#include <iostream>
#include <vector>
#include "testcluster.h"

/*
Общая временная сложность алгоритма:
   - В худшем случае, общая временная сложность будет O(K^n * m), где n — количество элементов в массиве, K — количе
     и m — размер кластера. Для каждого разбиения (K) мы вычисляем метрику для каждого кластера.

Общая память:
   - В худшем случае, общая память будет O(K^n), так как мы храним все возможные комбинации кластеров в allClusters.
   - Дополнительно, в памяти будет O(n) для хранения входного массива, где n — количество элементов.
   - Общая память: O(K^n + n).
*/
```

```cpp
int main() {
    runTests(); // Запускаем тесты

    int K;
    std::cout << "Enter the number of clusters (K):\n";
    std::cin >> K;

    int n;
    std::cout << "Enter the number of elements in the array (>=K):\n ";
    std::cin >> n;

    std::vector<double> array(n);
    std::cout << "Enter the elements of the array:\n ";
    for (int i = 0; i < n; ++i) {
        std::cin >> array[i];
    }

    findOptimalClusters(array, K);
    printClusters( clusters: findOptimalClusters(array, K));
    return 0;
}
```