ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Кластеризация массива»

Выполнил работу

Алёшкин Дмитрий

Академическая группа J3113

Принято

Максимом Дунаевым

Санкт-Петербург

2024

Содержание

1. Введение……………………………………………………………………2
2. Теоретическая подготовка…………………………………………………3
3. Реализация………………………………………………………………….4-6
4. Экспериментальная часть…………………………………………………
5. Заключение…………………………………………………………………
6. Приложения…………………………………………………………………
7. Введение

Цель: Кластеризировать массив (на 5 подмассивов) посредством полного перебора всех комбинаций значений этого массива таким образом, чтобы минимизировать расстояния между каждым элементом подмассива и средним значением в подмассиве.

Задачи:

1. Считывание значений из файла и запись их в массив
2. Разбиение массива на 5 подмассивов, где происходит минимизация расстояния между каждым элементом подмассива и средним значением в подмассиве
3. Вывод полученных подмассивов и их проверка
4. Теоретическая подготовка

В данной работе были использованы такие типы данных как: int, string, double, а также vector.

В данной работе были написаны следующие функции:

1. Функция для расчета метрики по массиву
2. Функция для записи значений из файла в динамический массив
3. Генерация всех возможных сочетаний массивов (комбинаций)
4. Преобразование double\*\* в string
5. Функция для проверки ответов

Метрика в данной работе представлена следующим образом:

sum(abs(arr\_cluster[i] - arr\_cluster\_mean))

1. Реализация

Первой реализованной в данной программе функцией расчёт метрики. Для этого был реализован подсчёт суммы, расчёт его среднего значения и сам расчёт метрики. Была подключена библиотека vector, так как в дальнейшем нам предстоит менять значения местами, а изначально нам не известно количество элементов подданых на вход.

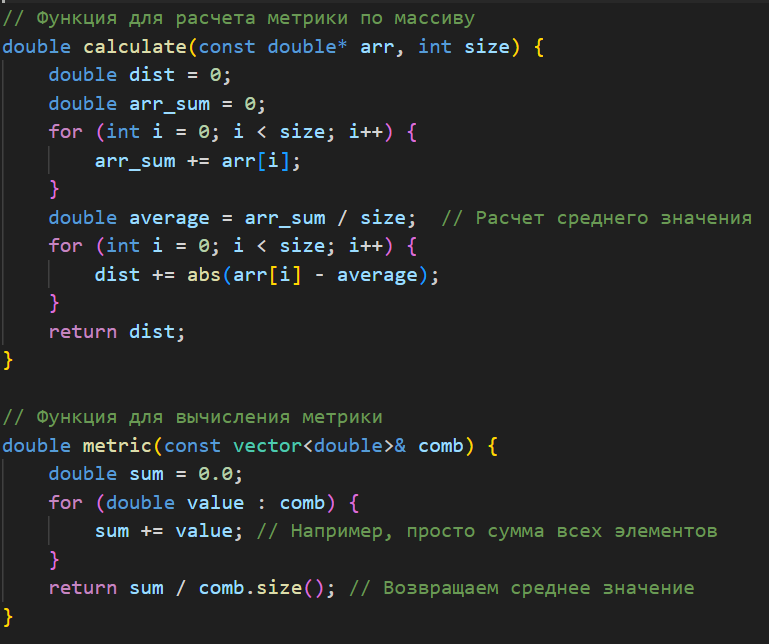


Рис. 1. Код для расчёта метрики

Для проверки данной функции было реализовано чтение данных из файла test.exe. Для этого были подключены библиотеки fstream. Так же сразу сделали проверку ответов. Здесь возникли первые трудности. При чтении из файла мы получаем строку, но на данном этапе программа выдавала числовое значение. Это показало необходимость в реализации следующей функции - преобразование double\*\* в string (преобразовываем именно double\*\* так как в дальнейшем нам необходимо массив разбить на 5 подмассивов). Конечно, можно было ответ преобразовать особым образом в double\*\* (к примеру разбивать подмассивы с помощью “ ; ”), но вариант преобразования подмассивов в троку показался нам легче. Реализация данной функции выглядит следующим образом:

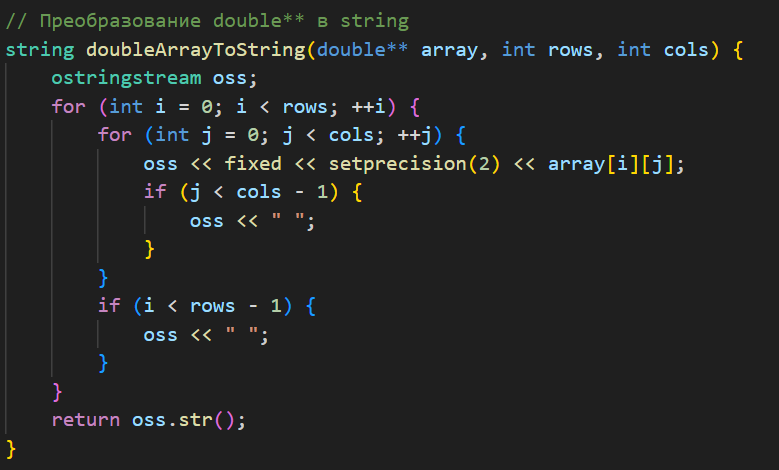


Рис. 2. Код для преобразования double\*\* в string

Так как нам нужен перебор всех возможных значений, то была реализована генерация всех возможных сочетаний массивов (комбинаций). В данной ситуации была использована рекурсия. Пример кода:

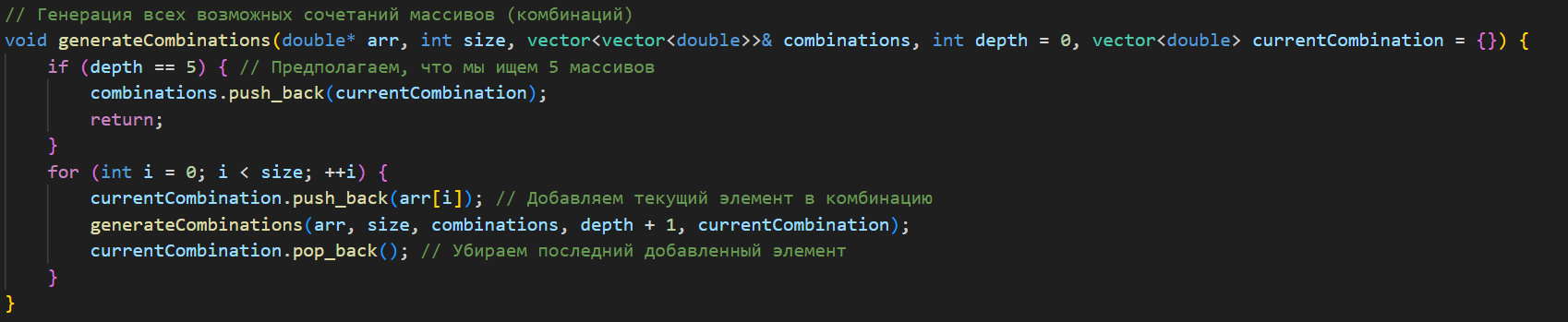


Рис. 3. Код генерации всех комбинаций

Далее была реализована самая основная функция в данной работе – разбитие основного массива на 5 подмассивов, с минимальным средним значением и расстоянием. В данном случае очень помогла метрика. Так же пришлось внимательно следить за памятью, чтобы программа не занимала всё свободное пространство.

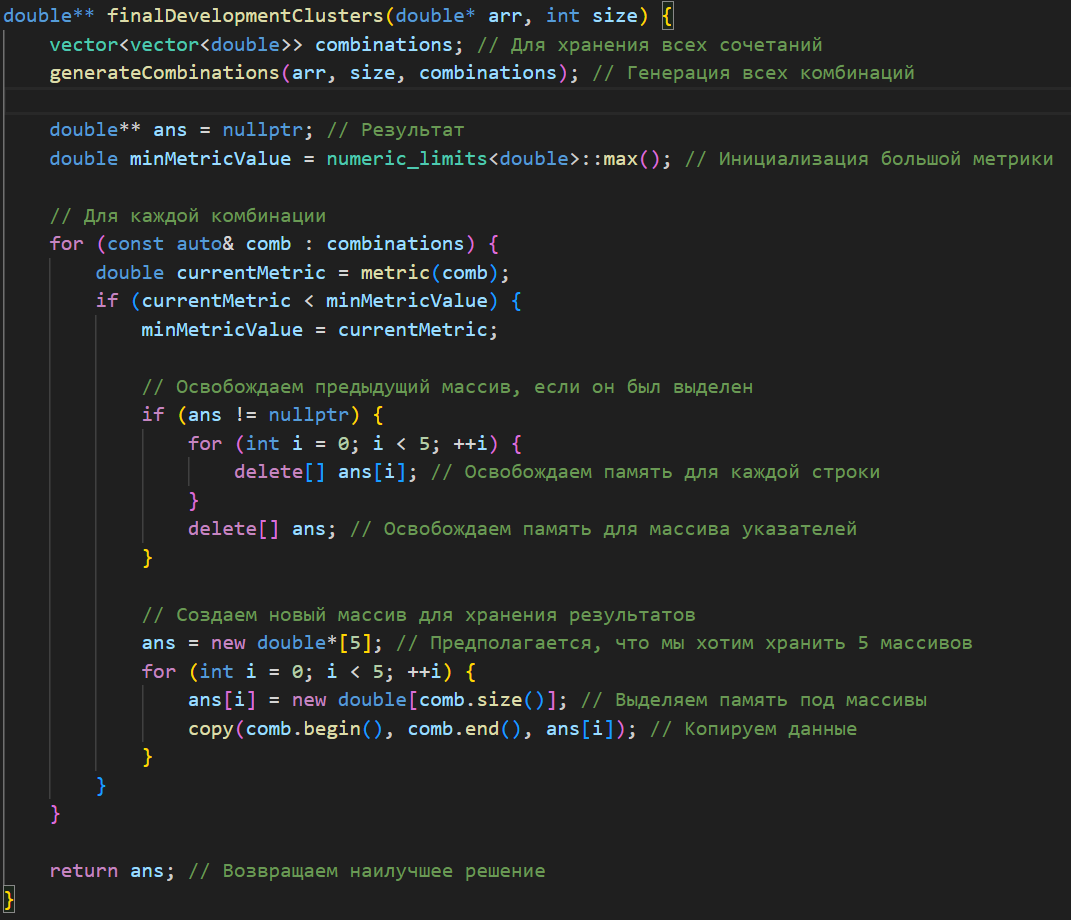


Рис. 4. Код разбиения массива на кластеры

В результате, после написания всех функций мы получили программу, которая минимизирует расстояниу между каждым элементом подмассива и средним значением в подмассиве.

1. Экспериментальная часть

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

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

Проведя анализ данной таблицы и графика, можно утверждать, что:

1. Заключение

В ходе выполнения работы мною был реализован алгоритм кластеризации массива на 5 подмассивов с минимальным средним значением и расстоянием в них. Цель работы была достигнута путём тестирования на различных входных данных, которые представляли из себя набор (от 5 до 25) чисел. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти, а также рассмотреть иной подход для решения данной задачи, ведь полный перебор является слишком затратным.

ПРИЛОЖЕНИЕ А

Листинг кода файла lab-4.cpp

#include <iostream>

#include <fstream>

#include <iomanip>

#include <sstream>

#include <string>

#include <vector>

#include <limits>

using namespace std;

// Функция для расчета метрики по массиву

double calculate(const double\* arr, int size) {

    double dist = 0;

    double arr\_sum = 0;

    for (int i = 0; i < size; i++) {

        arr\_sum += arr[i];

    }

    double average = arr\_sum / size;  // Расчет среднего значения

    for (int i = 0; i < size; i++) {

        dist += abs(arr[i] - average);

    }

    return dist;

}

// Функция для вычисления метрики

double metric(const vector<double>& comb) {

    double sum = 0.0;

    for (double value : comb) {

        sum += value; // Например, просто сумма всех элементов

    }

    return sum / comb.size(); // Возвращаем среднее значение

}

// Функция для записи значений из файла в динамический массив

double\* readArrayFromFileToDynamicArray(const string& filename, int& size) {

    vector<double> dynamicArray;  // Использование vector для динамического массива

    ifstream inputFile(filename);

    double number;

    while (inputFile >> number) {

        dynamicArray.push\_back(number);

    }

    size = dynamicArray.size();

    double\* result = new double[size];

    copy(dynamicArray.begin(), dynamicArray.end(), result); // Копируем данные в массив

    return result;

}

// Генерация всех возможных сочетаний массивов (комбинаций)

void generateCombinations(double\* arr, int size, vector<vector<double>>& combinations, int depth = 0, vector<double> currentCombination = {}) {

    if (depth == 5) { // Предполагаем, что мы ищем 5 массивов

        combinations.push\_back(currentCombination);

        return;

    }

    for (int i = 0; i < size; ++i) {

        currentCombination.push\_back(arr[i]); // Добавляем текущий элемент в комбинацию

        generateCombinations(arr, size, combinations, depth + 1, currentCombination);

        currentCombination.pop\_back(); // Убираем последний добавленный элемент

    }

}

double\*\* finalDevelopmentClusters(double\* arr, int size) {

    vector<vector<double>> combinations; // Для хранения всех сочетаний

    generateCombinations(arr, size, combinations); // Генерация всех комбинаций

    double\*\* ans = nullptr; // Результат

    double minMetricValue = numeric\_limits<double>::max(); // Инициализация большой метрики

    // Для каждой комбинации

    for (const auto& comb : combinations) {

        double currentMetric = metric(comb);

        if (currentMetric < minMetricValue) {

            minMetricValue = currentMetric;

            // Освобождаем предыдущий массив, если он был выделен

            if (ans != nullptr) {

                for (int i = 0; i < 5; ++i) {

                    delete[] ans[i]; // Освобождаем память для каждой строки

                }

                delete[] ans; // Освобождаем память для массива указателей

            }

            // Создаем новый массив для хранения результатов

            ans = new double\*[5]; // Предполагается, что мы хотим хранить 5 массивов

            for (int i = 0; i < 5; ++i) {

                ans[i] = new double[comb.size()]; // Выделяем память под массивы

                copy(comb.begin(), comb.end(), ans[i]); // Копируем данные

            }

        }

    }

    return ans; // Возвращаем наилучшее решение

}

// Преобразование double\*\* в string

string doubleArrayToString(double\*\* array, int rows, int cols) {

    ostringstream oss;

    for (int i = 0; i < rows; ++i) {

        for (int j = 0; j < cols; ++j) {

            oss << fixed << setprecision(2) << array[i][j];

            if (j < cols - 1) {

                oss << " ";

            }

        }

        if (i < rows - 1) {

            oss << " ";

        }

    }

    return oss.str();

}

// Функция для проверки ответов

string result\_check\_test(int number\_line, string current\_line\_test) {

    ifstream stream\_answer("answer.txt");

    string current\_line\_answer;

    for (int i = 0; getline(stream\_answer, current\_line\_answer); i++) {

        if (i == number\_line) {

            break;

        }

    }

    return current\_line\_answer == current\_line\_test ? "True" : "False";

}

// Главная функция

int main() {

    int size = 0;

    ifstream stream\_test("test.txt");

    if (!stream\_test.is\_open()) {

        cout << "Ошибка открытия файла: test.txt" << endl;

        return 1;

    }

    string current\_line\_test;

    int number\_line = 0;

    while (getline(stream\_test, current\_line\_test)) {

        number\_line++;

        double\* arr = readArrayFromFileToDynamicArray("test.txt", size);

        double\*\* result = finalDevelopmentClusters(arr, size);

        cout << result\_check\_test(number\_line, doubleArrayToString(result, 1, 5)) << endl;

        //cout << doubleArrayToString(result, 1, 5);

        delete[] arr; // Освобождение памяти

        // Освобождение памяти результата

        if (result != nullptr) { // Проверяем, был ли результат найден

            for (int i = 0; i < 5; ++i) {

                delete[] result[i]; // Освобождение памяти для каждого массива

            }

            delete[] result; // Освобождение памяти для указателя на массив

        }

    }

    stream\_test.close();

    return 0;

}