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

Смирнов Олег

Академическая группа J3110

Принято

Вершинин Владислав

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель: дан на вход массив, необходимо его разбить на K кластеров таким образом, чтобы минимизировать расстояния между каждым элементом подмассива и средним значением в подмассиве.

Задачи: реализовать функцию для перебора всевозможных кластеров, создать функцию для вычисления метрики, написать тесты.

1. Вещественные и целочисленные типы данных, алгоритм перебора всевозможных кластеров, алгоритм для вычисления метрики.
2. Реализация

В коде перебрал всевозможные числа 9-ричной системы счисления меньше n^9, цифра числа – индекс кластера, в который попадёт соответствующий элемент массива.

Написал функцию для вычисления метрики: ищем среднее арифметическое и суммируем разность между ней и элементами массива

1. Экспериментальная часть

Изображение №1 - График работы алгоритма

Далее необходимо привести анализ графика и таблицы.

1. Заключение

В ходе работы мною был реализован алгоритм для кластеризации путём полного перебора комбинаций. Полученные результаты совпадают с оценкой времени работы алгоритма.

Для улучшения временных затрат можно использовать можно использовать следующий алгоритм: отсортировать числа по возрастанию, менять только границы между кластерами. Асимптотика n^8 (всего C из n по 8 способов переставить границы – метод «шаров и перегородок»)

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла lab-4.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

#include <limits>

using namespace std;

// Функция для вычисления метрики

double calculate\_cluster\_distance(const vector<double>& cluster) {

    double sum = 0; //8 bytes

    double mean = 0; //8 bytes

    //O(cluster\_size)

    for (double value : cluster) {

        mean += value;

    }

    mean /= cluster.size();

    //O(cluster\_szie)

    for (double value : cluster) {

        sum += abs(value - mean);

    }

    return sum;

}

// Функция для полного перебора всех комбинаций кластеров

vector<vector<double>> clusterize(const vector<double>& arr, int k) {

    int n = arr.size();

    // если количество кластеров больше, чем количество элементов

    if (k > n){

     return {};

    }

    vector<vector<double>> best\_clusters(k); //O(k) 8n bytes

    double min\_distance = numeric\_limits<double>::max();

    // Перебираем все возможные комбинации кластеров O(9^n)

    for (int i = 0; i < pow(k, n); ++i) {

        vector<vector<double>> clusters(k); //O(k) 8n bytes

        vector<int> number(n); //O(n) 8n bytes

        // Генерация индексов кластеров O(n)

        for (int j = 0; j < n; ++j) {

            number[j] = (i / static\_cast<int>(pow(k, j))) % k; // Находим кластер для элемента j

        }

        // Разбиваем на кластеры O(n)

        for (int j = 0; j < n; ++j) {

            clusters[number[j]].push\_back(arr[j]);

        }

        // O(k)

        bool valid\_clusters = all\_of(clusters.begin(), clusters.end(), [](const vector<double>& cluster) {

            return !cluster.empty();

        });

        // Вычисляем общую сумму расстояний для всех кластеров

        double total\_distance = 0;

        //O(n)

        if (valid\_clusters) {

            for (const auto& cluster : clusters) {

                total\_distance += calculate\_cluster\_distance(cluster);

            }

            // Обновляем лучшие кластеры

            if (total\_distance < min\_distance) {

                min\_distance = total\_distance;

                best\_clusters = clusters;

            }

        }

    }

    return best\_clusters;

}

#include <gtest/gtest.h>

#include <gtest/gtest.h>

// Тестирование на случай, когда k больше n

TEST(Tests, NumberOfClustersGreaterThanElements) {

    vector<double> arr = {1, 2, 3, 4, 5, 6, 7, 8};

    auto clusters = clusterize(arr, 9);

    ASSERT\_TRUE(clusters.empty());

}

// Тестирование на случай, когда k равно n (каждый элемент в своем кластере)

TEST(Tests, NumberOfClustersEqualToElements) {

    vector<double> arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};

    auto clusters = clusterize(arr, 9);

    // Ожидаем, что каждый элемент будет в своем кластере

    vector<vector<double>> expected\_clusters = {{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}};

    ASSERT\_EQ(clusters, expected\_clusters);

}

// Тестирование на случай, когда есть повторяющиеся элементы и k меньше количества уникальных элементов

TEST(Tests, ClustersWithRepeatedElements) {

    vector<double> arr = {1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    auto clusters = clusterize(arr, 9);

    // Ожидаем, что 2 будет в одном кластере

    vector<vector<double>> expected\_clusters = {{1}, {2, 2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}};  // Можно разное распределение

    ASSERT\_EQ(clusters.size(), expected\_clusters.size());

    // Проверка содержимого

    for (size\_t i = 0; i < clusters.size(); ++i) {

        std::sort(clusters[i].begin(), clusters[i].end());

        std::sort(expected\_clusters[i].begin(), expected\_clusters[i].end());

        ASSERT\_EQ(clusters[i], expected\_clusters[i]);

    }

}

// Основная функция для запуска тестов и основной логики

int main(int argc, char \*\*argv) {

    ::testing::InitGoogleTest(&argc, argv);

    vector<double> arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // Данные

    int k = 9; // Количество кластеров

    vector<vector<double>> clusters = clusterize(arr, k);

    for (int i = 0; i < k; ++i) {

        cout << "Cluster " << i + 1 << ": ";

        for (double value : clusters[i]) {

            cout << value << " ";

        }

        cout << endl;

    }

    return RUN\_ALL\_TESTS();

}