ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Кластеризация массива по средствам полного перебора всех наборов кластеров»

Выполнил работу

Никитин Данил Алексеевич

Академическая группа №j3110

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

Структура отчёта:

Введение

Цель: Познакомиться с NP-полными алгоритмами.

Задачи:

1)Получить все возможные разбиения массива на кластеры

2)Посчитать для каждого разбиения сумму метрик ее кластеров

3) Создать словарь ключами которого будут суммы метрик разбиения, а значениями разбиения.

4)Найти наименьший ключ и вывести его значение.

Теоретическая подготовка

Методы библиотек #include <fstream>, #include <sstream> (работа с текстовыми файлами в С++)

Циклы while ,for

Unordered\_map хэш-таблица (хранения неупорядоченных данных по ключу) ,операция добавления элементов в словарь. Методы (обращение к ключу ,значению ) ->first,->second.

Динамический массив vector и его методы вектора, двумерный вектор

Функции ,их создание и вызов

Условные инструкции if

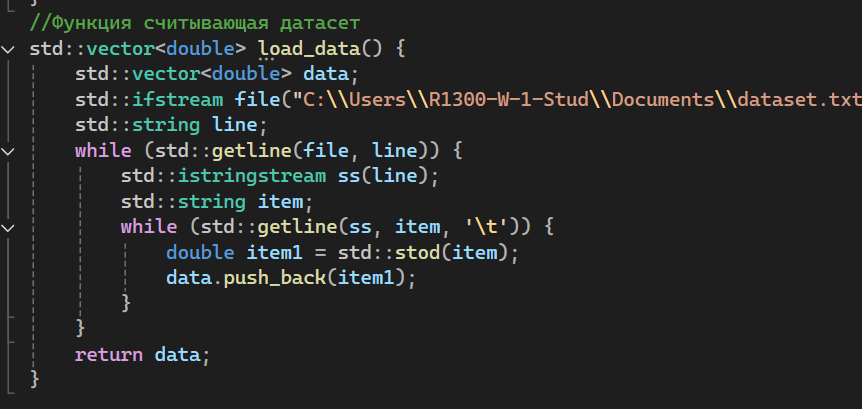
Тип данных bool ,для работы с тестами

Алгоритм поиск минимального обычным сравнением.

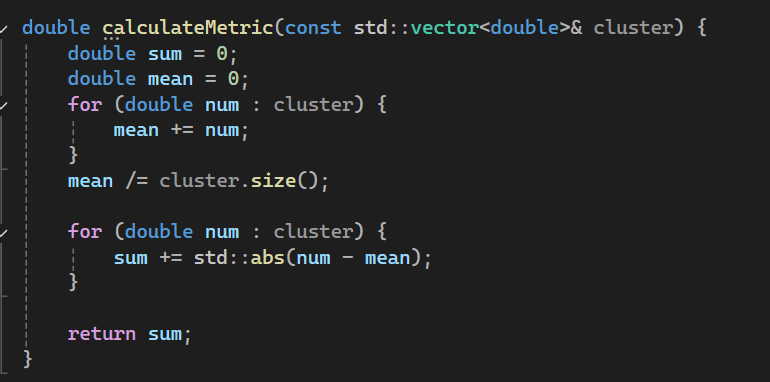
Ссылки на переменные, контейнеры и структуры данных &

Реализация

1)Создал функцию для чтения датасета (load\_data()) ,получил исходный массив

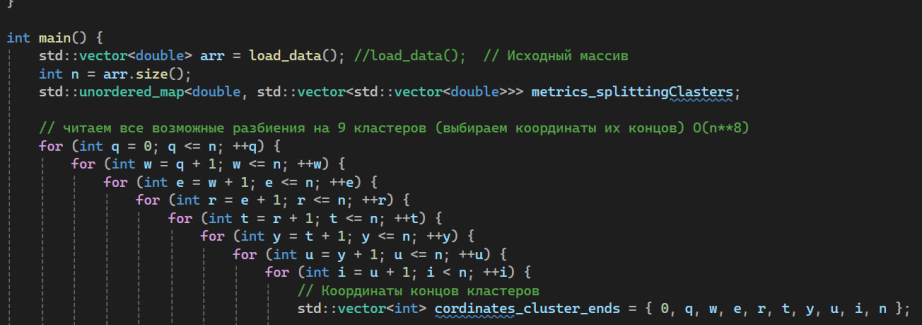


2)Функция находящая суммы метрик для одного конкретного разбиения на кластеры.



2) Создал словарь ключи сумма метрик ,значения конкретные разбиения на кластеры, инициализирую исхдный массив из функции чтения.

Затем перебираю все возможные сочетания координат концов кластеров ,чтоб затем по координатам концов разбить на сами кластеры. Вложенных циклов 8 ,т.к. индекс первой и последней координаты известен.

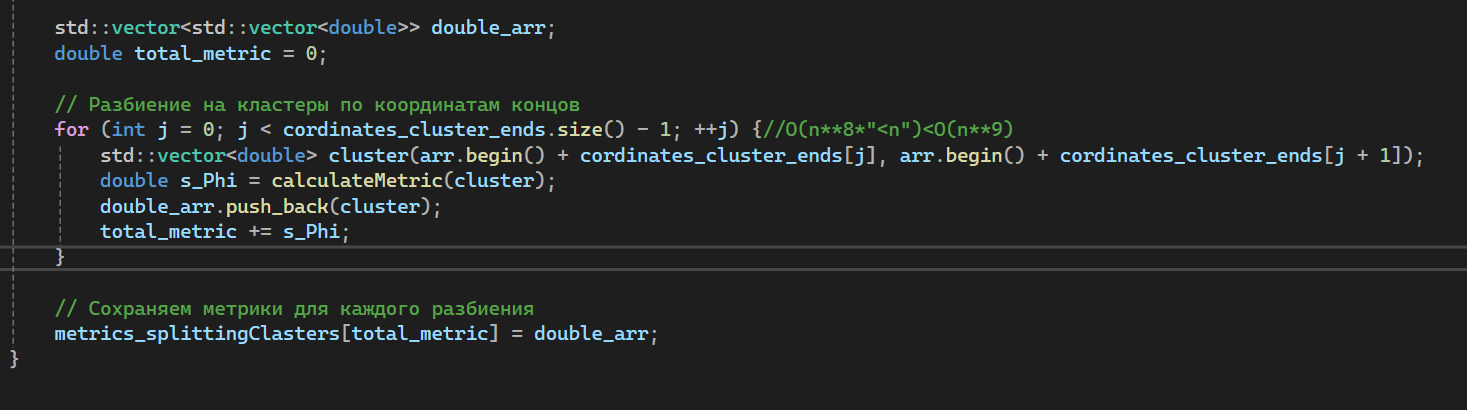


Создаю двумерный массив для хранения кластеров (это конкретное разбиение) ,инициализирую total\_metric есть переменная где будут храниться суммы метрик для конкретного разбиения.

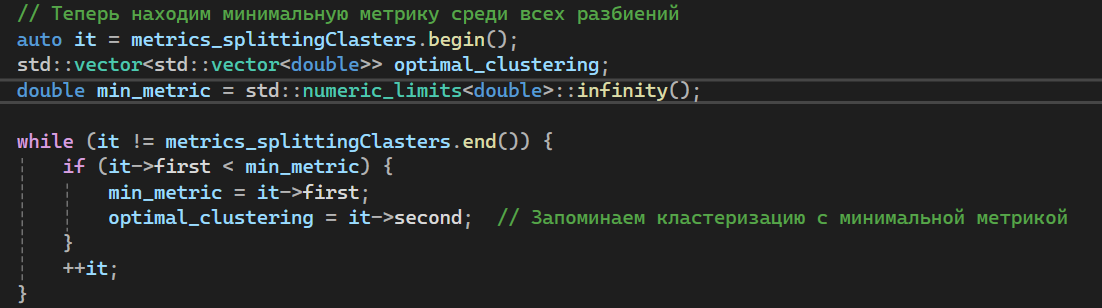
Идем по длине массива координат концов, создаем по одному кластеру ,инициализируем этот кластер элементами из исходного ,имеющими координаты его начала и его конца.

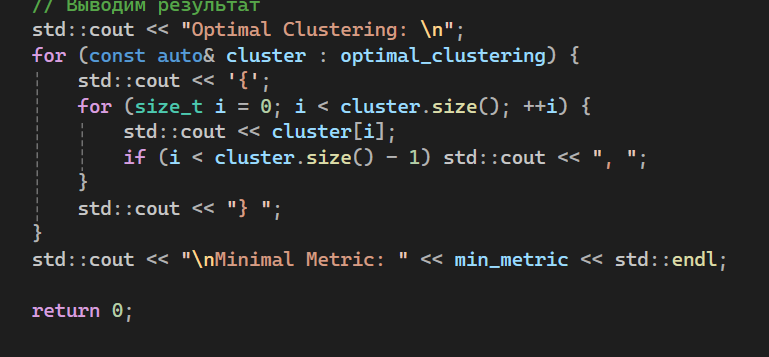
По ходу цикла добавляем каждый кластер в массив , добавляем сумму метрик для каждого элемента кластера (получена из обращения к функции) в общую сумму всех метрик кластеров.

После того как одно разбиение изучено полностью ,Добавляем пару ключ и значение в словарь.



Выбираем минимальный ключ ,сравнивая сначала с бесконечностью ,а потом с предыдущим ,сохраняя меньший из них. Для этого идем пока итератор ,не будет равен концу.

Сохраняем значение минимального в массив.

5) Проход по двумерному массиву с выводом результата.

Экспериментальная часть

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

8байт\*O(n\*\*9) + n\*O(n\*\*9)\*8 байт + 4 байта \* k\*О(1)+8 байт\*O(n)\*3⬄8 байт \* O(n\*\*10)

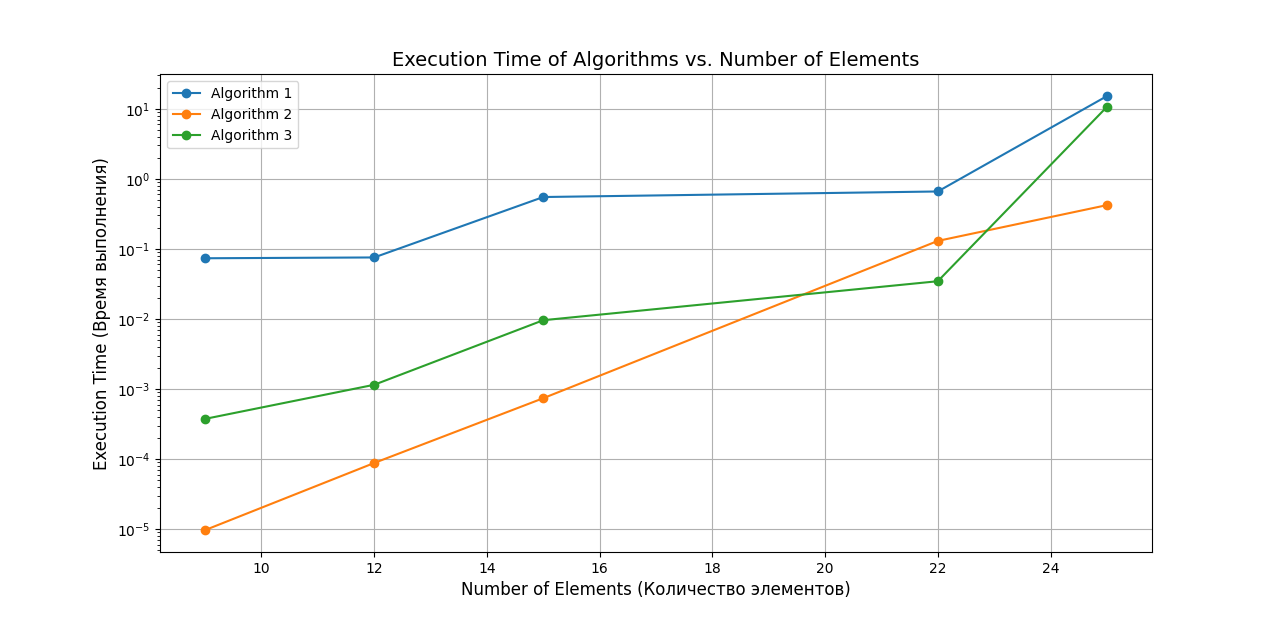
Подсчёт асимптотики (только для циклов и сложных структур) – O(n\*\*9)+O(n)+O(9\*n)+O(2\*”<n”)=O(n\*\*9), учитывая что n маленькие это намного медленнее чем О(2\*\*n)

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. Данные для третей и четвертой строки были созданы домножением количества операций для такой сложности на коэффициент (равный времени выполнения делить на количество итераций)

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 9 | 12 | 15 | 18 | 22 | 25 |
| Время выполнения программы, с | 0.07352 | 0.0755 | 0.55 | 0.66 | 5.01 | 15.16 |
| O(2\*N), с | 0.0000097 | 0.000088 | 0.00074 | 0.0011 | 0.13 | 0.4213 |
| O(3\*N), с | 0.000375 | 0.00115 | 0.0096 | 0.03457 | 3.51 | 10.6396 |

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



Изображение №1 - График работы алгоритма без сравнения

Заметим что время работы алгоритма растет экспоненциально.

Заключение

В ходе работы мною был реализована программа сложность ,которой схожа с NP полной ,был проведен анализ сложности по памяти и скорости ,затем результат был доказан путем тестирования времни работы программы.

Я могу сказать ,что узнал несколько новых приемов в синтаксисе С++ (конкретно это инициализация массива ,содержащего один кластер в пункте 3 реализации) , а также поработал с кластеризацией (почитал об более продвинутой ее версии в МЛ) ,увидел приложения линейной алгебры (на тривиальном уровне – метрика ,расстояние) ,конечно если можно считать кластеры элементами носителя метрического пространства.

Программа не запускалась до оптимизации (поэтому пришлось поработать с памятью – использовать ссылки)

Увидел ,что данная задача намного эффективней решается методом динамического программирования ,что слегка предвосхитило лекции, дало импульс к дальнейшему освоению темы.

ПРИЛОЖЕНИЕ А

Файл NP-coplate\_lab4.cpp

Код

#include <iostream>

#include <vector>

#include <cmath>

#include <limits>

#include <unordered\_map>

#include <fstream>

#include <sstream>

#include <chrono>

//Тесты

int test\_passed = 0;

int test\_failed = 0;

void assertEqual(bool condition, const std::string& testName) {

if (condition) {

std::cout << "[PASSED]" << testName << "\n";

test\_passed++;

}

else {

std::cout << "[NOT PASSED]" << testName << "\n";

test\_failed++;

}

}

void report() {

std::cout << "\nTests passed total: " << test\_passed << "\n";

std::cout << "\nTests not passed total: " << test\_failed << "\n";

}

void test(std::vector<double>& arr) {

assertEqual(arr.empty(), "Test empty array");

assertEqual((arr.size() >= 9), "test input data by condition");

assertEqual((arr.size() <= 25), "test input data by condition");

for (double value : arr) {

assertEqual(std::isfinite(value), "Test if all values are finite (not NaN or infinity)");

}

report();

}

//Функция считывающая датасет

std::vector<double> load\_data() {//О(n) так первый вайл идет по одной строке ,в файле всего одна строка.

std::vector<double> data;

std::ifstream file("C:\\Users\\admin\\Downloads\\dataset.txt");

std::string line;

while (std::getline(file, line)) {

std::istringstream ss(line);

std::string item;

while (std::getline(ss, item, '\t')) {

double item1 = std::stod(item);

data.push\_back(item1);

}

}

return data;

}

double calculateMetric(const std::vector<double>& cluster) {//O("<n"\*2) , тк для отдельного кластера

double sum = 0;

double mean = 0;

for (double num : cluster) {

mean += num;

}

mean /= cluster.size();

for (double num : cluster) {

sum += std::abs(num - mean);

}

return sum;

}

int main() {

auto start = std::chrono::high\_resolution\_clock::now();

std::vector<double> arr = load\_data(); // n\*8 байт

//{ 1,2,3,4,55,11,14,233,-6,11111,-0.00099999,0.999999999,12,34,109,1,1.11,1.9999999,1,2,6,67,21,3.09865734521}; //load\_data();-Масссив по которому считал время

int n = arr.size();

std::unordered\_map<double, std::vector<std::vector<double>>> metrics\_splittingClasters;

// читаем все возможные разбиения на 9 кластеров (выбираем координаты их концов) O(n\*\*8)

for (int q = 0; q <= n; ++q) {

for (int w = q + 1; w <= n; ++w) {

for (int e = w + 1; e <= n; ++e) {

for (int r = e + 1; r <= n; ++r) {

for (int t = r + 1; t <= n; ++t) {

for (int y = t + 1; y <= n; ++y) {

for (int u = y + 1; u <= n; ++u) {

for (int i = u + 1; i < n; ++i) {

// Координаты концов кластеров

std::vector<int> сordinates\_cluster\_ends = { 0, q, w, e, r, t, y, u, i, n };//10\*4байта

std::vector<std::vector<double>> double\_arr;

double total\_metric = 0;

// Разбиение на кластеры по координатам концов

for (int j = 0; j < сordinates\_cluster\_ends.size() - 1; ++j) {//O(n\*\*8\*"<n")<O(n\*\*9)

std::vector<double> cluster(arr.begin() + сordinates\_cluster\_ends[j], arr.begin() + сordinates\_cluster\_ends[j + 1]);//О(k)

double s\_Phi = calculateMetric(cluster);

double\_arr.push\_back(cluster);

total\_metric += s\_Phi;

}

// Сохраняем метрики для каждого разбиения

metrics\_splittingClasters[total\_metric] = double\_arr;// 8байт\*O(n\*\*9) + n\*O(n\*\*9)\*8 байт

}

}

}

}

}

}

}

}

// Теперь находим минимальную метрику среди всех разбиений

auto it = metrics\_splittingClasters.begin();

std::vector<std::vector<double>> optimal\_clustering;

double min\_metric = std::numeric\_limits<double>::infinity();

while (it != metrics\_splittingClasters.end()) {//О(n\*\*9) тк асимптотика то понятное дело берем явно больше

if (it->first < min\_metric) {

min\_metric = it->first;

optimal\_clustering = it->second; // Запоминаем кластеризацию с минимальной метрикой

}

++it;

}

test(arr);

// Выводим результат

std::cout << "Optimal Clustering: \n";

for (const auto& cluster : optimal\_clustering) {//О(9) тк кластеров 9 по условию

std::cout << '{';

for (size\_t i = 0; i < cluster.size(); ++i) {//О(n)

std::cout << cluster[i];

if (i < cluster.size() - 1) std::cout << ", ";

}

std::cout << "} ";

}

std::cout << "\nMinimal Metric: " << min\_metric << std::endl;

//Выводим время выполнения программы

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> diff = end - start;

std::cout << "Time execution: " << diff.count() << " seconds";

}

Код построения графиков.

import matplotlib.pyplot as plt  
  
# Данные  
elements = [9, 12, 15, 22, 25]  
algorithm\_1 = [0.07352, 0.0755, 0.55, 0.66, 15.16]  
algorithm\_2 = [0.0000097, 0.000088, 0.00074, 0.13, 0.4213]  
algorithm\_3 = [0.000375, 0.00115, 0.0096, 0.03457, 10.639]  
  
# Построение графика  
plt.figure(figsize=(10, 6))  
plt.plot(elements, algorithm\_1, marker='o', label='Algorithm 1')  
plt.plot(elements, algorithm\_2, marker='o', label='Algorithm 2')  
plt.plot(elements, algorithm\_3, marker='o', label='Algorithm 3')  
  
# Настройки графика  
plt.xlabel('Number of Elements (Количество элементов)', fontsize=12)  
plt.ylabel('Execution Time (Время выполнения)', fontsize=12)  
plt.title('Execution Time of Algorithms vs. Number of Elements', fontsize=14)  
plt.legend()  
plt.grid(True)  
plt.yscale('log') # Логарифмическая шкала для оси Y  
  
# Показать график  
plt.show()