Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет имени Евфросинии Полоцкой»

Факультет информационных технологий

Кафедра технологий программирования

**Отчёт по лабораторной работе № 1 по курсу «Распределенные вычисления»**

«Реализация алгоритма кластеризации методом К-средних на CPU»

ВЫПОЛНИЛ студент группы 21-ИТ-1

Чиникайло А.П.

ПРОВЕРИЛ преподаватель

Адамовский Е.Р.

Полоцк, 2024 г.

Цель: Ознакомиться с алгоритмом кластеризации методом К-средних и реализовать данный алгоритм на CPU.

**ОПИСАНИЕ ПРОДЕЛАННОЙ РАБОТЫ**

Задание:

1. Реализовать консольное приложение (на языке программирования C++) для выполнения алгоритма K-средних согласно представленным вариантам в таблице.
2. Запустить реализованное консольное приложение, измерить встроенными средствами время выполнения алгоритма и его отдельных итераций на CPU.
3. Представить результаты измерений в графическом виде.

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Количество элементов и атрибутов элемента** | **Количество кластеров** |
| 12 | 120 000 × 5 | 6 |

Листинг 1 – функция main

int main() {

vector<vector<double>> points(amountOfElements, vector<double>(numberOfElementAttributes)); // объявляем точки

vector<vector<double>> centroids(numberOfClusters, vector<double>(numberOfElementAttributes)); // объявляем центроиды

vector<double> iterationTimes;

for (int i = 0; i < amountOfElements; i++) { // присваиваем точкам рандомные значения

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

points[i][j] = 1 + rand() % 100;

}

}

for (int i = 0; i < numberOfClusters; i++) { // присваиваем центроидам рандомные значения

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

centroids[i][j] = 1 + rand() % 100;

}

}

double s = clock();// присваиваем время старта

kMeans(points, centroids, iterationTimes);

double e = clock();// присваиваем время конца

cout << "K-Means Algorithm Execution Time: " << (e - s) / CLOCKS\_PER\_SEC << " s" << endl;// выводим время выполнения в секундах

ofstream outFile("../../IterationTimes.txt");//открываем файл

if (outFile.is\_open()) {

for (int i = 0; i < iterationTimes.size(); i++) {//записываем в него время выполнения каждой итерации

outFile << iterationTimes[i] << " ";

}

outFile.close();//закрываем файл

}

else { // если не получилось открыть файл

cout << "Error, could not open file.";

}

return 0;

}

Листинг 2 – функция kMeans

// функция алгоритма k-средних

void kMeans(vector<vector<double>>& points, vector<vector<double>>& centroids, vector<double>& iterationTimes) {

vector<int> OldclusterAssignments(amountOfElements);// объявляем vector для сравнение, при поиске, когда алгоритм завершен

int count = 0;

while (true) {

double s = clock(); // присваиваем время старта

vector<int> clusterAssignments(amountOfElements); // объявляем vector для каждой точки, принадлежащей к кластеру

assignToClusters(points, centroids, clusterAssignments);// присваиваем точки кластеру

updateCentroids(points, centroids, clusterAssignments);// обновляем центроиды

double e = clock();// присваиваем время конца

iterationTimes.push\_back((e - s) / CLOCKS\_PER\_SEC); // запоминаем время выполнения итерации в секундах

if (clusterAssignments == OldclusterAssignments) { // если точки не меняют кластеров

cout << "True" << count << endl;

ofstream outFile("../../CountIteration.txt");//открываем файл

if (outFile.is\_open()) {

outFile << count;// записываем количество итераций

outFile.close();//закрываем файл

}

else {// елси файл не открылся

cout << "Error, could not open file.";

}

break;

}

OldclusterAssignments = clusterAssignments; //запоминаем какие точки к какому кластеру принадлежат

count++;// считаем количество итериций

}

}

Листинг 3 – функция updateCentroids

// обновляем координаты центроидов

void updateCentroids(vector<vector<double>>& points, vector<vector<double>>& centroids, vector<int>& clusterAssignments) {

for (int i = 0; i < numberOfClusters; i++) { // цикл по всем кластерам

vector<double> newCentroid(numberOfElementAttributes);// обьявляем новый

int count = 0;// количество точек в кластере

for (int j = 0; j < amountOfElements; j++) {//проходим по всем точкам

if (clusterAssignments[j] == i) {// если точка принадлежит кластеру

for (int l = 0; l < numberOfElementAttributes; l++) {

newCentroid[l] += points[j][l];// суммируем атрибуты точек

}

count++;// считаем количество точек в кластере

}

}

if (count > 0) { // если есть какие либо точки в кластере

for (int l = 0; l < numberOfElementAttributes; l++) {

newCentroid[l] /= count; // делим на количество точек в кластере каждый атрибут

}

centroids[i] = newCentroid;// запоминаем новый центроид

}

}

}

Листинг 4 – функция assignToClusters

// присваиваем точки к кластеру

void assignToClusters(vector<vector<double>>& points, vector<vector<double>>& centroids, vector<int>& clusterAssignments) {

for (int i = 0; i < amountOfElements; i++) {// цикл на все точки проекта

double minDistance = numeric\_limits<double>::max(); // берем наибольшее возможное значения, что было с чем сравнивать

int closestCentroid = -1;

for (int j = 0; j < numberOfClusters; j++) {// цикл по кластерам

double distance = calculateDistance(points[i], centroids[j]);// вычисляем с помощью функуции описаной выше

if (distance < minDistance) { // если расстояние меньше, то запомнаем

minDistance = distance;

closestCentroid = j;

}

}

clusterAssignments[i] = closestCentroid;// присваиваем ближайший центроид для точки

}

}

Листинг 5 – функция calculateDistance

// подсчет растояние между точками

double calculateDistance(vector<double>& point1, vector<double>& point2) {

double distance = 0.0; //обьявляем переменную

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

distance += pow((point2[i] - point1[i]), 2); // проходимся по всем элементам vector и находим сумму квадраов разности

}

return sqrt(distance);// возвращаем корень расстояния

}

**Результаты выполнения лабораторной работы**

Для демонстрации работы программы были изменены входные данные, а именно количество элементов равное 30, количество атрибутов элементов равное 2, количество кластеров равное 3.

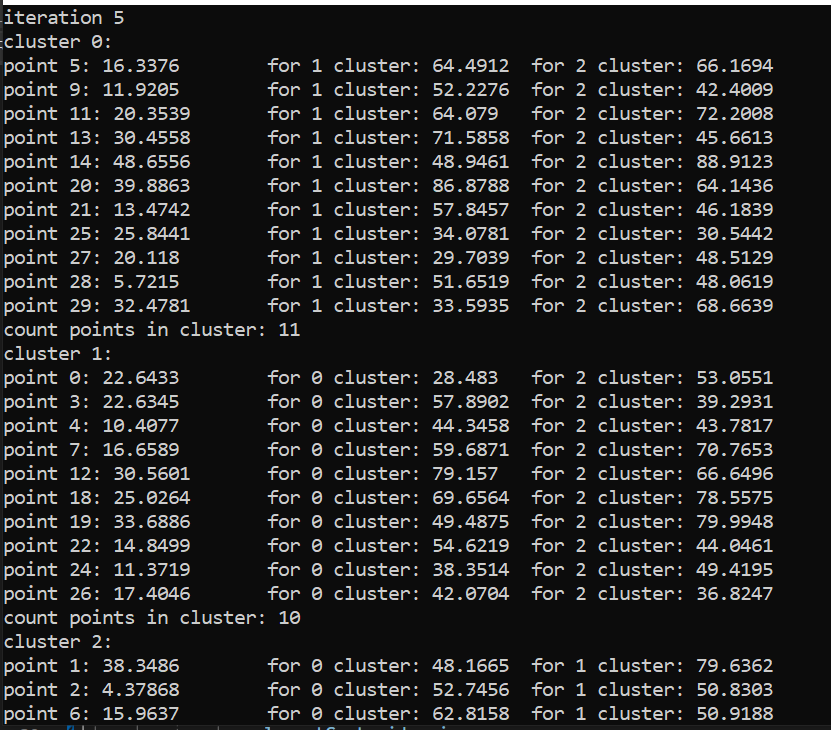


Рисунок 1 – расстояние до принадлежащего кластера и для всех остальных кластеров

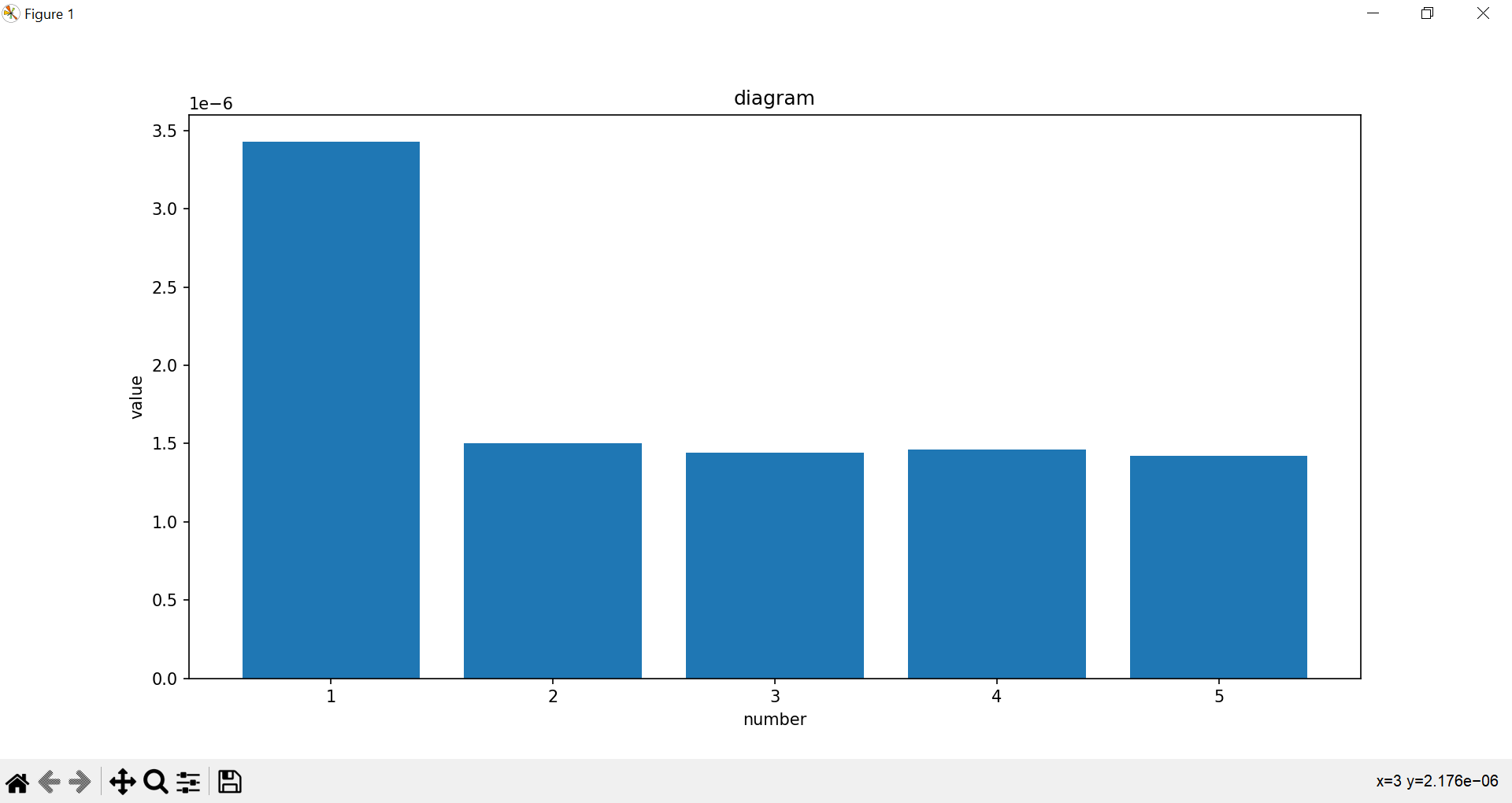


Рисунок 2 – Время выполнения итераций алгоритма

**ВЫВОД**

В результате выполнения лабораторной работы я написал алгоритм кластеризации методом K-средних на C++. По выполнения алгоритма, можно увидеть, что каждая точка принадлежит самому ближайшему центроиду. Это означает, что алгоритм работает верно. Так же при повторном запуске проекта с одними и теми же входными данными, выходные данные остаются такими же.