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»

Выполнила работу

Крапивина Яна

Академическая группа №J3112

Принято

Должность, звание Фамилия Имя преподавателя

Максим Дунаев, практик

Санкт-Петербург

2024

1. **Введение**

**Цель работы:** разработка и реализация алгоритма кластеризации одномерного массива чисел посредством полного перебора всех возможных комбинаций разбиения массива на заданное количество кластеров.

**Задачи:**

* Кластеризовать таким образом, чтобы минимизировать расстояния между каждым элементом подмассива и средним значением в подмассиве;
* Учесть сохранение порядка;
* Каждый подмассив на выходе должен содержать как минимум один элемент;
* Алгоритмическая сложность не должна быть меньше чем O(2^N);
* Алгоритм не должен работать больше часа.

1. **Теоретическая подготовка**

**Tеоретические аспекты, которые необходимы для выполнения работы:**

* Векторы и вложенные векторы для представления кластеров, где каждый внутренний вектор хранит элементы одного кластера;
* Принципы перебора и генерации всех возможных вариантов;
* Вычисление среднего и абсолютного значений;
* Работа с функциями и библиотеками.

**Используемые типы данных:** double(элементы массива, метрики и среднее значение), int(элементы массива, индексы, количество кластеров, размер массива), bool(указывает является ли кластеризация допустимой).

**Алгоритмы:**

1. Перебор (Brute-force);
2. Вычисление среднего значения (Mean);
3. Вычисление суммы абсолютных отклонений;
4. Поиск минимума;
5. Проверка на пустые кластеры.
6. **Реализация**

**Этапы:**

1. **Инициализация данных**

* массив arr (вектор vector<double>) с числами, которые необходимо кластеризовать;
* переменные для хранения лучшей кластеризации (best\_clustering), минимальной метрики (min\_metric) и других вспомогательных переменных.

1. **Перебор всех возможных комбинаций кластеризации:**

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

vector<vector<double>> clustering(k);

int temp = i;

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

clustering[temp % k].push\_back(arr[j]);

temp /= k;

}

1. **Проверка на валидность кластеризации:**

bool valid\_clustering = true;

for (const auto& cluster : clustering) {

if (cluster.empty()) {

valid\_clustering = false;

break;

}

}

1. **Вычисление метрики для каждой валидной кластеризации:**

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

if (cluster.empty()) {

return 0.0;

}

double mean = accumulate(cluster.begin(), cluster.end(), 0.0) / cluster.size();

double metric = 0.0;

for (double value : cluster) {

metric += abs(value - mean);

}

return metric;

}

1. **Вывод результата.**

**Библиотеки:**

• <iostream>: Для ввода/вывода данных.

• <vector>: Для работы с динамическими массивами.

• <numeric>: Для использования функции accumulate (для вычисления суммы).

• <algorithm>: Для использования стандартных алгоритмов (не используется в данном коде).

• <limits>: Для получения максимального значения типа double (numeric\_limits<double>::max()).

• <math.h>: Для использования функции pow (для возведения в степень).

1. **Экспериментальная часть**

В этом разделе вам необходимо привести результаты работы вашего алгоритма, с таблицами и графиками, демонстрирующими выполнения алгоритма с различными условиями и наборами данных. Оценивается производительность и сравниваются результаты с теоретическими оценками.

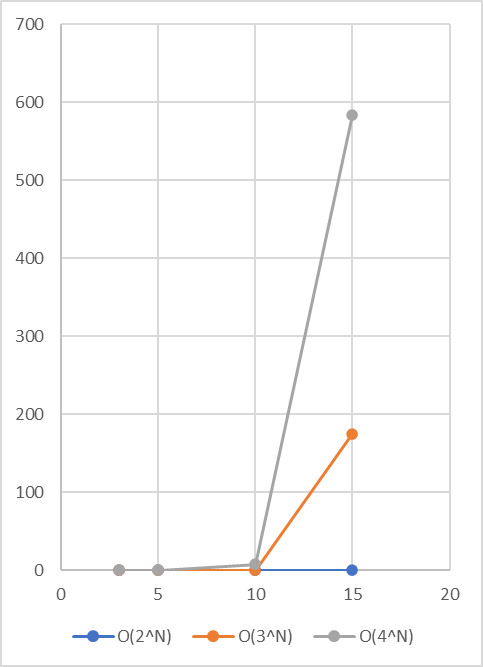
Подсчёт по памяти: 216 байт

Подсчёт асимптотики: O(3\*n^2), где 3 – кол-во кластеров, n – кол-во элементов в массиве.

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 | 3 | 5 | 10 | 15 |
| O(2^N), с | 0,001 | 0,001 | 0,01 | 0,3 |
| Время выполнения программы (O(3^N)), с | 0,002 | 0,002 | 0,4 | 175,302 |
| O(4^N), с | 0 | 0,007 | 7,642 | 584,015 |



Изображение №1 - График работы алгоритма

1. **Заключение**

В ходе выполнения работы был реализован алгоритм кластеризации данных, основанный на переборе всех возможных комбинаций кластеров и минимизации суммы абсолютных отклонений значений в каждом кластере от их среднего значения.

Цель работы была достигнута путем реализации алгоритма, который находит оптимальное разбиение заданного набора данных на заданное количество кластеров (k). Алгоритм перебирает все возможные варианты разбиения и выбирает тот, который минимизирует заданную метрику (сумму абсолютных отклонений).

Недостатком данного алгоритма является его высокая вычислительная сложность, так как он перебирает все возможные комбинации кластеризации. Это делает его непригодным для работы с большими наборами данных и большим количеством кластеров.

В качестве дальнейших исследований можно предложить:

• Разработку более эффективных алгоритмов кластеризации, таких как k-means или иерархическая кластеризация, которые имеют меньшую вычислительную сложность.

• Разработку методов предварительной обработки данных для улучшения качества кластеризации.

1. **Приложения**

#include <iostream>

#include <vector>

#include <numeric>

#include <algorithm>

#include <limits>

#include <math.h>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Функция для расчета метрики (суммы абсолютных отклонений)

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

if (cluster.empty()) {

return 0.0;

}

double mean = accumulate(cluster.begin(), cluster.end(), 0.0) / cluster.size();

double metric = 0.0;

for (double value : cluster) {

metric += abs(value - mean);

}

return metric;

}

int main() {

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

vector<double> arr = { -2.38, 754.0, 3.0,864,345, 670, 765.0854, -268.89, 86, 4.88, 55, 88, 23};

int k = 4;

int n = arr.size();

vector<vector<double>> best\_clustering;

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

// Перебор всех возможных комбинаций кластеров

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

vector<vector<double>> clustering(k);

int temp = i;

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

clustering[temp % k].push\_back(arr[j]);

temp /= k;

}

// Проверка на пустые кластеры

bool valid\_clustering = true;

for (const auto& cluster : clustering) {

if (cluster.empty()) {

valid\_clustering = false;

break;

}

}

if (valid\_clustering) {

double current\_metric = 0.0;

for (const auto& cluster : clustering) {

current\_metric += calculate\_metric(cluster);

}

if (current\_metric < min\_metric) {

min\_metric = current\_metric;

best\_clustering = clustering;

}

}

}

// Вывод результата

std::cout << "[";

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

std::cout << "[";

for (size\_t j = 0; j < best\_clustering[i].size(); ++j) {

std::cout << best\_clustering[i][j] << (j == best\_clustering[i].size() - 1 ? "" : ", ");

}

std::cout << "]" << (i == best\_clustering.size() - 1 ? "" : ", ");

}

std::cout << "]" << endl;

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(stop - start);

cout << duration.count() << endl;

return 0;

}