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

Эль-Сибаи Амир

Академическая группа №J3112

Принято

Ассистент (квалификационная категория "ассистент"), Дунаев Максим

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Целью данной работы является изучение метода полного перебора для кластеризации массива данных. Задачи лабораторной работы:

* Реализовать алгоритм кластеризации на языке программирования C++.
* Проанализировать результаты работы алгоритма с использованием различных тестовых данных.
* Оценить эффективность алгоритма по временной сложности и использованию памяти.

1. Теоретическая подготовка

Кластеризация — это задача разбиения множества объектов на кластеры, такие что элементы внутри одного кластера расположены ближе друг к другу по значению, чем элементы из разных кластеров.

Метод полного перебора заключается в исследовании всех возможных комбинаций разбиения для поиска лучшего решения.

Использую:

* Полный перебор всех возможные комбинаций разбиения массива.
* Метрика оценки кластеров: для каждого разбиения вычисляется сумма отклонений элементов каждого кластера от их среднего значения.
* Сложность: Алгоритм обладает сложностью O(2^N), что ограничивает его применимость для больших массивов.

1. Реализация

Для выполнения задачи был разработан код на C++ с использованием следующих библиотек:

* <vector> для работы с массивами.
* <cmath> для математических вычислений.
* <limits> для работы с предельными значениями.
* <tuple> для возвращения нескольких значений из функции.

Основные этапы реализации:

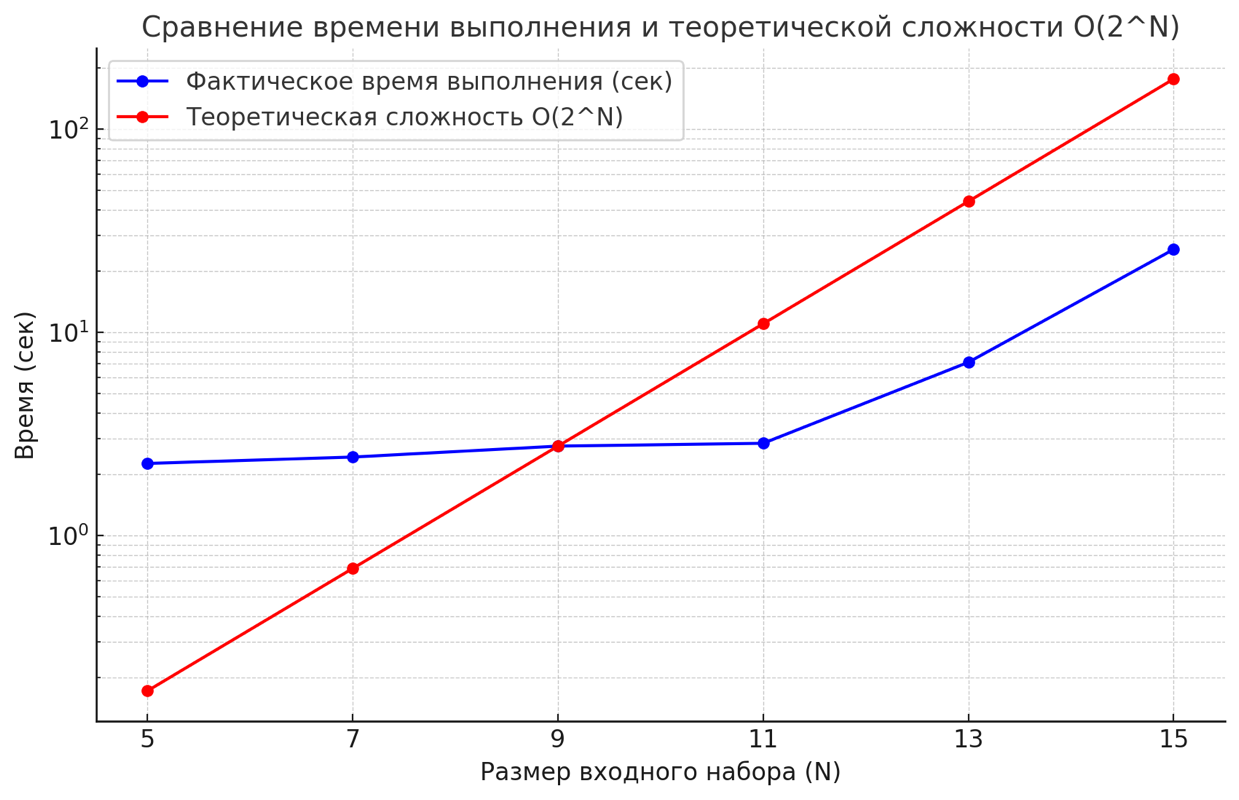
1. Вычисление среднего значения кластера: average\_val возвращает среднее значение элементов вектора.
2. Оценка качества кластера: len\_average вычисляет сумму отклонений элементов от среднего значения.
3. Перебор всех разбиений: klast перебирает все возможные варианты разбиений массива на три кластера с использованием побитовых масок.
4. Вывод кластеров: prClusters форматирует и выводит результат кластеризации.
5. Экспериментальная часть

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов, которые кластеризуются на 3 кластера. Теоретически заданная сложность задачи составляет O(2^N) и более, где N – количество элементов. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 5 | 7 | 9 | 11 | 13 | 15 |
| Время выполнения программы, с | 2,26 | 2,43 | 2,75 | 2,84 | 7,11 | 25,54 |
| O(2^N), с | 0,172 | 0,688 | 2,75 | 11.0 | 44.0 | 176.0 |

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



Изображение №1 - График работы алгоритма

По графикам заметно, что время фактической работы программы близка к ожидаемой. Отличия вероятно происходят из-за неоптимального подсчёта метрики.

1. Заключение

В ходе выполнения работы мною был реализован алгоритм кластеризации исходного массива на 3 кластера. Цель работы была достигнута путём полного перебора всех вариантов кластеров и отбор лучшего путём выбора наименьшей метрики. Полученные результаты близки к теоретическим оценкам сложности алгоритма.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма путём сортировки исходного массива, чтобы сократить время выполнения кода.

1. Приложения

В этом разделе вам необходимо указать всю дополнительную информацию, которая не вошла в основной текст, например полный исходный код программы. Пример:

ПРИЛОЖЕНИЕ А

Листинг кода файла main.cpp

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <limits>  
#include <tuple>  
  
using namespace std;  
  
int average\_val(const vector<int>& cluster) {  
 int sum = 0  
 for (int val: cluster){  
 sum += val;  
 }  
 if (cluster.empty()){  
 return 0;  
 }  
 else {  
 return sum / cluster.size();  
 }  
}  
int len\_average(const vector<int>& cluster) {  
 int mean = average\_val(cluster);   
 int sum = 0;  
 for (int val: cluster){  
 sum += abs(val - mean);  
 }  
 return sum;  
}

tuple<vector<int>, vector<int>, vector<int>> klast(const vector<int>& arr) {  
 int n = arr.size(); // Количество элементов массива  
 int minMetric = numeric\_limits<int>::*max*();  
 tuple<vector<int>, vector<int>, vector<int>> bestClusters;

for (int mask1 = 1; mask1 < (1 << n) - 1; ++mask1) {  
 for (int mask2 = mask1 + 1; mask2 < (1 << n); ++mask2) {  
 if ((mask1 & mask2) == 0) {  
 vector<int> cluster1, cluster2, cluster3;

for (int i = 0; i < n; ++i) {  
 if (mask1 & (1 << i)) {  
 cluster1.push\_back(arr[i]);  
 }  
 else if (mask2 & (1 << i)) {  
 cluster2.push\_back(arr[i]);  
 }  
 else {  
 cluster3.push\_back(arr[i]);  
 }  
 }   
 int totalMetric = len\_average(cluster1) + len\_average(cluster2) + len\_average(cluster3);  
 if (totalMetric < minMetric) {  
 minMetric = totalMetric;  
 bestClusters = make\_tuple(cluster1, cluster2, cluster3);  
 }  
 }  
 }  
 }  
  
 return bestClusters;  
}  
  
void prClusters(const tuple<vector<int>, vector<int>, vector<int>>& clusters) {  
 cout << "[";  
 for (size\_t i = 0; i < get<0>(clusters).size(); ++i) {  
 if (i > 0) {  
 cout << ", ";  
 }  
 cout << get<0>(clusters)[i];  
 }  
 cout << "], [";  
 for (size\_t i = 0; i < get<1>(clusters).size(); ++i) {  
 if (i > 0) {  
 cout << ", ";  
 }  
 cout << get<1>(clusters)[i];  
 }  
 cout << "], [";  
 for (size\_t i = 0; i < get<2>(clusters).size(); ++i) {  
 if (i > 0) {  
 cout << ", ";  
 }  
 cout << get<2>(clusters)[i];  
 }  
 cout << "]" << endl;

}  
int main() {  
 vector<vector<int>> testCases = {  
 {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25},  
 {1, 10, 9, 2, 3, 8, 11, 15, 9, 99, 1, 4, 8, 8, 14, 88, 1488, 1405, 1404, 2202, 2337, 2330, 413, 424, 2201}  
 };  
 for (const auto& testCase: testCases) {  
 cout << "Input: ";  
 for (int val: testCase){  
 cout << val << " ";  
 }

cout << endl;

auto clusters = klast(testCase);  
 cout << "Clusters: ";  
 prClusters(clusters); // Выводим кластеры  
 cout << endl;  
 }  
  
 return 0;  
}