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

Кащеев Максим

Академическая группа: J3113

Принято

Иван Владимирович Ходненко

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель - понять, насколько время-затратными могут быть NP-задачи  
Задача – написать алгоритм кластеризации массива посредством  
полного перебора всех комбинаций значений этого массива,  
минимизировав при этом расстояние между каждым элементом  
подмассива и средним значением в подмассиве

1. Теоретическая подготовка

Для выполнения работы необходимы следующие аспекты: типы  
данных, такие как: vector, const, int, double, auto, chrono. Также необходимо  
знание как пользоваться О-нотацией, знание методов для соответствующих типов данных, умение считать время выполнения алгоритмов и знание как  
вызвать константу для типа данных double, минимальное понимание  
ссылок и работу алгоритма k mean.

1. Реализация

В ходе выполнения работы я сделал два файла: main, в котором запускается алгоритм, выполняется проверка тестов и подсчитывается время работы алгоритма и файл lab4, в котором описана работа алгоритма. В файле lab4 я задаю работу таких функций, как: distance, которая считает метрику для кластеров; score\_cluster, которая высчитывает сумму расстояний для кластера; функция setPartitionAsBest, которая с помощью ссылок уменьшает затраты памяти для функции генерации партиций; функция generationPartitions в котором рекурсивно идёт проверка всех существующих разбиений массива на k кластеров и поиск минимального по расстоянию разбиения между каждым элементом подмассива и средним значением соответственно; функция generateAndFindBestPartition, которая объявляет начальные переменные для рекурсии, и стартует рекурсию.

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 – В коде используются несколько векторов, partition и bestPartition, каждый из которых не длиннее N по сумме всех подэлементов, таким образом память внутри каждой итерации можно аппроксимировать как O(2N) -> O(N). Рекурсия имеет глубину не более N, таким образом стек вызовов будет расти не быстрее чем O(n). Поскольку используемые вектора передаются по ссылке, значения не перемножаются, а складываются, таким образом получаем общее значение по памяти O(2N) -> O(N).

Подсчёт асимптотики – для проверки всех возможных кластеров нам надо составить все варианты разбиений, в худшем случае количество разбиений, которые надо обойти это число Стирлинга второго рода

Аппроксимируем примерно как O((K^N)\*N) для худшего случая и O(K+N) - O(N) для лучшего - как минимум мы пройдем каждый элемент в отдельную группу, и пройдемся по группам

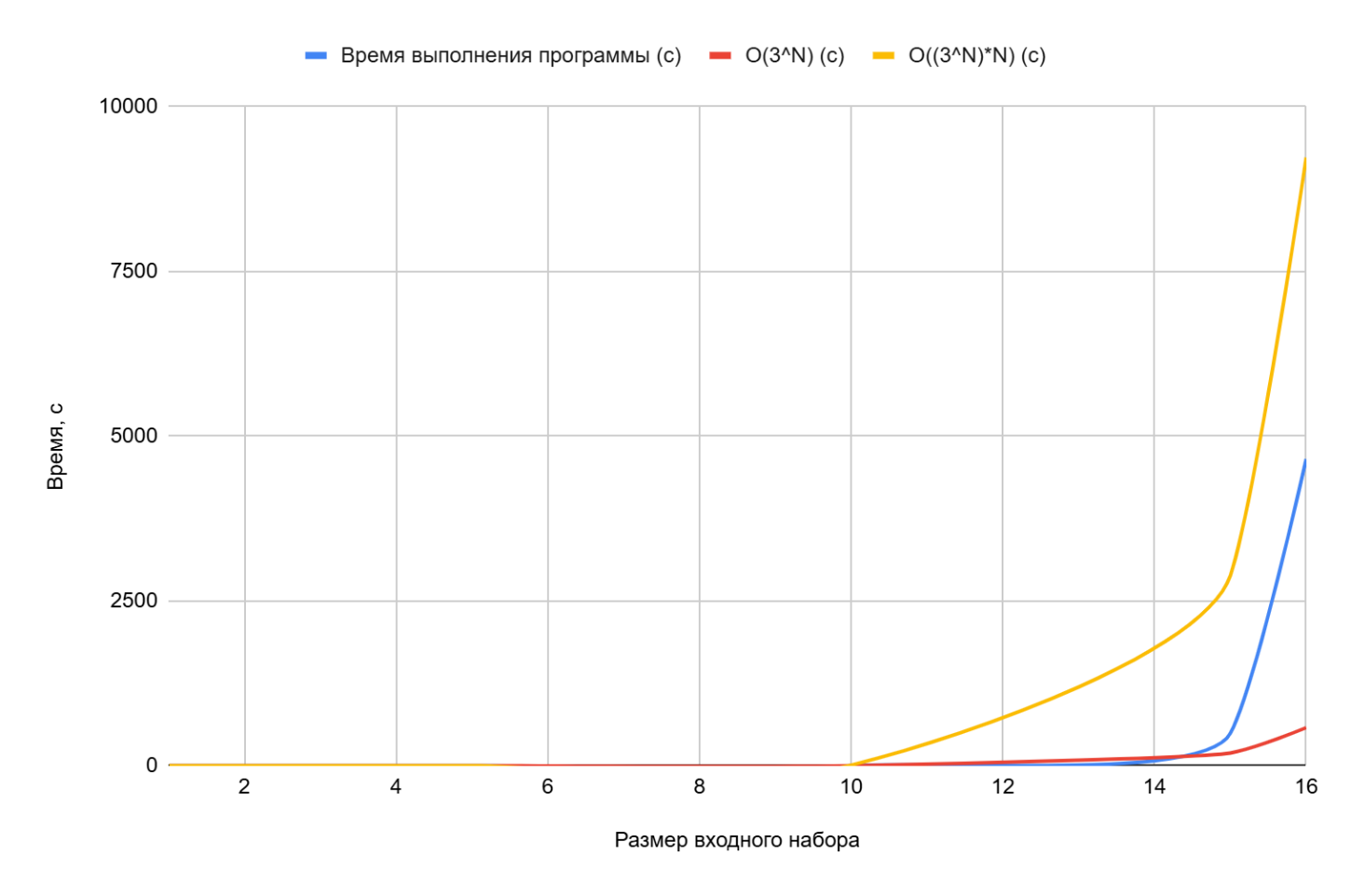
generatePartitions(arr, partition, 0, K, bestPartition, distanceInitial); // Генерация разбиений на K подмножеств.

Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер входного набора | 1 | 5 | 10 | 15 | 16 |
| Время выполнения программы, с | 0.0000134 | 0.0000361 | 0.0077 | 490.646 | 4654.06 |
| O(3^N) (c) | 0.0000402 | 0.0032562 | 0.7912566 | 192.2753538 | 576.8260614 |
| O((3^N)\*N) (c) | 0.0000402 | 0.016281 | 7.912566 | 2884.130307 | 9229.216982 |
| O((7^N)\*N) (c) | 0.0000938 | 1.126069 | 37851.68337 | 954259863.5 | 7125140314 |

Графики представляющие визуально удобный формат данных из таблицы №1 представлен на изображениях №1, №2.



Изображение №1 – Сравнительный график работы алгоритма для O(3^N) (c), O((3^N)\*N) (с) и времени выполнения программы

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

Изображение №2 – График теоретического худшего случая выполнения программы O((7^N)\*N) (c)

Далее необходимо привести анализ графика и таблицы.

1. Заключение

В ходе выполнения работы мною был реализован алгоритм, который делит данный массив на K кластеров посредством полного перебора всех возможных комбинаций значений этого массива (при условии, что количество элементов массива больше, чем требуемое количество K, на которые его нужно поделить). Все расстояния между элементами соответствующих подмассивов и средними значениями в этих подмассивах минимизированы. Цель работы была выполнена путём написания тестов и таблиц для данной задачи, из которых видно резкое возрастание затрат времени на выполнения программы в зависимости от количества элементов в исходном массиве.

Задача была выполнена при помощи рекурсивного алгоритма, который проходит по всем возможным сочетанием K кластеров и находит самоё оптимальное решение, удовлетворяющее условиям задачи.

Функция generatePartitions, написанная для выполнения поставленной задачи была оптимизирована при помощи ссылок, благодаря которым снижаются затраты на память при выполнении рекурсии.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат памяти и времени, например, используя алгоритм k-средних, заранее генерируя центральные точки каждого кластера.

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла lab4.cpp

#include <iostream>

#include <vector>

#include <limits>

double distance(double a, double b) {

    return std::abs(a - b);

}

double score\_cluster(const std::vector<double>& cluster) {

    double mean = 0.0;

    for (auto& item : cluster) {

        mean += item;

    }

    mean /= cluster.size();

    double metric = 0.0;

    for (auto& item : cluster) {

        metric += distance(item, mean);

    }

    return metric;

}

void setPartitionAsBest(std::vector<std::vector<double>>& bestPartition, std::vector<std::vector<double>> partition) {

    bestPartition = partition;

}

// Функция для генерации всех возможных разбиений на K подмножеств

void generatePartitions(

    const std::vector<double>& arr,

    std::vector<std::vector<double>>& partition,

    int start,

    int k,

    std::vector<std::vector<double>>& bestPartition,

    double& currentMinDistance) {

    if (start == arr.size()) {

        // Если количество подмножеств соответствует K, сохраняем разбиение в общий массив

        if (partition.size() == k) {

            double scoresTotal = 0;

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

                scoresTotal += score\_cluster(partition[i]);

            }

            double partitionTotalDistance = scoresTotal / k;

            if (currentMinDistance > partitionTotalDistance) {

                setPartitionAsBest(bestPartition, partition);  // Берем разбиение

                currentMinDistance = partitionTotalDistance;

            }

        }

        return;

    }

    int remainingLength = arr.size() - start;

    if ((partition.size() + remainingLength) < k) {

        return; // Если элементов не хватит - не продолжаем итерацию

    }

    else {

        // Либо создаём новое подмножество для текущего элемента, если их меньше чем требуемое количество

        partition.push\_back({arr[start]});

        generatePartitions(arr, partition, start + 1, k, bestPartition, currentMinDistance);

        partition.pop\_back();  // Убираем последнее подмножество

    }

    if ((partition.size() + remainingLength) <= k) {

        return; // Если элементов не хватит - не продолжаем итерацию

    }

    for (int i = partition.size() - 1; i >= 0; i--) {

        partition[i].push\_back(arr[start]);

        generatePartitions(arr, partition, start + 1, k, bestPartition, currentMinDistance);  // Рекурсивно обрабатываем следующий элемент

        partition[i].pop\_back();  // Убираем элемент из текущего подмножества

    }

}

std::vector<std::vector<double>> generateAndFindBestPartition(std::vector<double>& arr, int k) {

    std::vector<std::vector<double>> bestPartition;

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

    std::vector<std::vector<double>> partition;  // Вектор для хранения текущих подмножеств

    generatePartitions(arr, partition, 0, k, bestPartition, distanceInitial);  // Генерация разбиений на K подмножеств

    return bestPartition;

}

ПРИЛОЖЕНИЕ B

Листинг кода файла main.cpp

#include <iostream>

#include <vector>

#include <chrono>

#include <cassert>

#include "lab4.cpp"

void runTests() {

    std::vector<double> arr = {99.5, -1.1, 101};

    int k = 2;

    std::vector<std::vector<double>> ans\_ex1 = { {99.5, 101}, {-1.1} };

    std::vector<std::vector<double>> partition = generateAndFindBestPartition(arr, k);

    assert(partition == ans\_ex1);

    std::cout << "Test {99.5, -1.1, 101} passed"<< std::endl;

    arr = {1, 10, 9, 2, 3, 8};

    k = 2;

    std::vector<std::vector<double>> ans\_ex2 = { {1, 2, 3}, {10, 9, 8} };

    partition = generateAndFindBestPartition(arr, k);

    assert(partition == ans\_ex2);

    std::cout << "Test {1, 10, 9, 2, 3, 8} passed"<< std::endl;

    arr = {1, 2, 3, 4, 5, 6};

    k = 2;

    std::vector<std::vector<double>> ans\_ex3 = { {1, 2, 3}, {4, 5, 6} };

    partition = generateAndFindBestPartition(arr, k);

    assert(partition == ans\_ex3);

    std::cout << "Test {1, 10, 9, 2, 3, 8} passed"<< std::endl;

    arr = {1, 2, 3, 4, 5, 6, 7};

    k = 7;

    std::vector<std::vector<double>> ans\_1 = { {1}, {2}, {3}, {4}, {5}, {6}, {7} };

    partition = generateAndFindBestPartition(arr, k);

    assert(partition == ans\_1);

    std::cout << "Test {1, 2, 3, 4, 5, 6, 7} passed"<< std::endl;

    arr = {99.5, -1.1, 101, 1, 10, 9, 2, 3, 8, 12};

    k = 7;

    std::vector<std::vector<double>> ans\_2 = { {99.5}, {-1.1}, {101}, {1, 2, 3}, {10}, {9, 8}, {12} };

    partition = generateAndFindBestPartition(arr, k);

    assert(partition == ans\_2);

    std::cout << "Test {99.5, -1.1, 101, 1, 10, 9, 2, 3, 8, 12} passed"<< std::endl;

}

int main() {

    runTests();

    using std::chrono::high\_resolution\_clock;

    using std::chrono::duration;

    using std::chrono::milliseconds;

    std::vector<double> arr = {99.5, -1.1, 101};

    int numSubsets = 2;

    auto t1 = high\_resolution\_clock::now();

    std::vector<std::vector<double>> partition = generateAndFindBestPartition(arr, numSubsets);

    auto t2 = high\_resolution\_clock::now();

    duration<double, std::milli> ms\_double = t2 - t1;

    for (const auto& subset : partition) {

        std::cout << "[";

        bool first = true;

        for (const double& val : subset) {

            if (first) {

                first = false;

            } else {

                std::cout << ", ";

            }

            std::cout << val;

        }

        std::cout << "] ";

    }

    std::cout << "#" << arr.size() << " - ";

    std::cout << ms\_double.count() << "ms";

    return 0;

}