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

Бочаров Евгений

Академическая группа №J3113

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

**Структура отчёта:**

* Введение

Цель - написать программу для кластеризации массива посредством полного перебора всех комбинаций значений этого массива

Задачи:

* Написать функцию для решения задачи
* Проверить правильность значений, которые она выдает
* Написать тесты для функции
* Теоретическая подготовка

Int – целое число

Double – число с плавающей точкой

Vector – вектор

Sort – сортировка вектора

Next\_permutation – алгоритм для нахождения перестановок в векторе

* Реализация

Библиотеки: iostream – для вывода текста

Vector – для использования векторов

Set – для использования множеств

Algorithm – для использования sort и next\_permutation

Этапы работы:

Написание функции mean

double mean(const vector<double>& arr) {

if (arr.size() == 0) return 0;

double sum = 0;

for (int i = 0; i < arr.size(); ++i) {

sum += arr[i];

}

return sum / arr.size();

}

Функция проходит по всем элементам вектора, прибавляя их к сумме, а потом делит значение суммы на длину вектора, возвращает среднее значение элементов вектора.

Написание функции metric

double metric(const vector<vector<double>>& arr) {

double ans = 0;

for (int i = 0; i < arr.size(); i++) {

double sumc = 0;

for (int k = 0; k < arr[i].size(); k++) {

sumc += abs(mean(arr[i]) - arr[i][k]);

}

ans += sumc;

}

return ans;

}

Функция проходит по всем подвекторам(кластерам), проходит по элементам каждого, находя расстояние до среднего в кластере и суммирует все эти расстояния и возращает сумму.

Написание функции find\_cluster\_sizes\_combs

vector<vector<int>> find\_cluster\_sizes\_combs(int n) {

vector<vector<int>> a;

set<vector<int>> unique\_combinations;

for (int i1 = 1; i1 < n - 3; ++i1) {

for (int i2 = i1 + 1; i2 < n - 2; ++i2) {

for (int i3 = i2 + 1; i3 < n - 1; ++i3) {

for (int i4 = i3 + 1; i4 < n; ++i4) {

for (int i5 = i4 + 1; i5 <= n; ++i5) {

if (i1 + (i2 - i1) + (i3 - i2) + (i4 - i3) + (i5 - i4) == n) {

vector<int> sorted\_combination = { i1, i2 - i1, i3 - i2, i4 - i3, i5 - i4 };

sort(sorted\_combination.begin(), sorted\_combination.end());

unique\_combinations.insert(sorted\_combination);

}

}

}

}

}

}

for (const auto& item : unique\_combinations) {

a.push\_back(item);

}

return a;

}

Функция рассматривает все возможные разбиения длины вектора на сумму из пяти натуральных чисел и возвращает вектор с векторами с длинами кластеров, отсортированными в порядке возрастания.

Написание функции cluster5:

vector<vector<double>> cluster5(vector<double>& arr) {

vector<double> arr1 = arr;

sort(arr1.begin(), arr1.end());

vector<vector<int>> csc = find\_cluster\_sizes\_combs(arr.size());

vector<vector<double>> ans = { {1, 999999999}, {1, 999999999}, {1, 999999999},{1, 999999999},{1, 999999999} };

do {

for (int i = 0; i < csc.size(); i++) {

vector<vector<double>> clcomb(5);

int j = 0, k = 0;

while (j < 5 && k < arr1.size()) {

int c = 0;

while (c < csc[i][j] && k < arr1.size()) {

clcomb[j].push\_back(arr1[k]);

k++;

c++;

}

j++;

}

// Проверка на корректность clcomb перед использованием metric

if ((clcomb.size() != 0) && metric(clcomb) < metric(ans)) {

ans = clcomb;

}

}

} while (next\_permutation(arr1.begin(), arr1.end()));

return ans;

}

Это основная функция, которая решает задачу. Получая на вход вектор с числами, она перебирает все перестановки элементов в векторе. Для каждой перестановки она перебирает варианты разбиения на кластеры, заполняет их, проверяет то, что значения сохранили порядок. Затем поэлементно сравнивает метрику полученного разбиения с метрикой ответа, присваивает значение разбиения ответа, если его метрика меньше, тем самым получая наименьший ответ.

* 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 (только для циклов и сложных структур) – 8(n+1) + 8(mn+2) + 8k + 16n + 40. см. приложение 1

Подсчёт асимптотики (только для циклов и сложных структур) – n log n +n!. см. приложение 1

Таблица №1 -

|  |  |  |  |
| --- | --- | --- | --- |
| Размер входного набора | 7 | 8 | 11 |
| Время выполнения программы, с | 0 | 2,7 | 5735 |
| O(N!), с | 0 | 0 | 27 |
| O(N log N + N!), с | 0 | 3,2 | 7584 |

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



рис 1. график зависимости времени работы программы

Далее необходимо привести анализ графика и таблицы.

* Заключение

В ходе выполнения работы мною был реализован алгоритм кластеризации с условием соблюдения минимума по метрике. Цель работы была достигнута путём тестирования на наборах чисел с различным количеством чисел. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти, а также рассмотреть параллельные версии алгоритма для работы с большими наборами чисел.

* Приложения

ПРИЛОЖЕНИЕ 1

#include <iostream>

#include <vector>

#include <algorithm>

#include <set>

#include <cmath>

using namespace std;

// Функция для вычисления среднего значения массива

double mean(const vector<double>& arr) {

// Сложность: O(n) - n - размер массива

// Память: O(1) - используем только несколько переменных для хранения суммы и результата (примерно 8 байт для double)

if (arr.size() == 0) return 0;

double sum = 0; // 8 байт

for (int i = 0; i < arr.size(); ++i) {

sum += arr[i]; // sum, arr[i] - 8 байт для каждого double

}

return sum / arr.size(); // возвращаем среднее

}

// Функция metric, вычисляющая сумму абсолютных отклонений от среднего

double metric(const vector<vector<double>>& arr) {

// Сложность: O(m \* n) - m - количество массивов, n - среднее количество элементов в каждом массиве

// Память: O(1) - используем только переменные для хранения результата и суммы (примерно 16 байт)

double ans = 0; // 8 байт

for (int i = 0; i < arr.size(); i++) {

double sumc = 0; // 8 байт

for (int k = 0; k < arr[i].size(); k++) {

sumc += abs(mean(arr[i]) - arr[i][k]); // вызываем mean, arr[i][k] - 8 байт

}

ans += sumc; // ans - 8 байт

}

return ans; // возвращаем итоговое значение

}

// Функция для нахождения уникальных комбинаций размеров кластеров

vector<vector<int>> find\_cluster\_sizes\_combs(int n) {

// Сложность: O(n^5) - производительность из-за пяти вложенных циклов

// Память: O(k) - где k - количество уникальных комбинаций, хранящихся в векторе

// Каждый вектор<int> занимает 4 байта за элемент, и размер может варьироваться

vector<vector<int>> a; // вектор векторов

set<vector<int>> unique\_combinations; // набор уникальных комбинаций

for (int i1 = 1; i1 < n - 3; ++i1) {

for (int i2 = i1 + 1; i2 < n - 2; ++i2) {

for (int i3 = i2 + 1; i3 < n - 1; ++i3) {

for (int i4 = i3 + 1; i4 < n; ++i4) {

for (int i5 = i4 + 1; i5 <= n; ++i5) {

if (i1 + (i2 - i1) + (i3 - i2) + (i4 - i3) + (i5 - i4) == n) {

vector<int> sorted\_combination = { i1, i2 - i1, i3 - i2, i4 - i3, i5 - i4 }; // 4 байта на int, 5 элементов = 20 байт

sort(sorted\_combination.begin(), sorted\_combination.end()); // сортировка

unique\_combinations.insert(sorted\_combination); // сохраняем в набор

}

}

}

}

}

}

for (const auto& item : unique\_combinations) {

a.push\_back(item); // добавление уникальных комбинаций в вектор

}

return a; // возвращаем уникальные комбинации

}

// Функция для кластеризации массива

vector<vector<double>> cluster5(vector<double>& arr) {

// Сложность: O(n log n + m!) - O(n log n) из-за сортировки и O(m!) из-за перестановок

// Память: O(n + k) - где n - количество входных данных, k - количество кластеров (векторов)

vector<double> arr1 = arr; // O(n) занимает память, n - размер массива, n \* 8 байт

sort(arr1.begin(), arr1.end()); // сортировка

vector<vector<int>> csc = find\_cluster\_sizes\_combs(arr.size()); // результат функции предыдущего определения

vector<vector<double>> ans = { {1, 999999999}, {1, 999999999}, {1, 999999999}, {1, 999999999}, {1, 999999999} }; // 5 векторов с 2 элементами - 80 байт

do {

for (int i = 0; i < csc.size(); i++) {

vector<vector<double>> clcomb(5); // 5 векторов, примерно 0-80 байт в зависимости от их размера

int j = 0, k = 0;

while (j < 5 && k < arr1.size()) {

int c = 0;

while (c < csc[i][j] && k < arr1.size()) {

clcomb[j].push\_back(arr1[k]); // добавление элемента

k++;

c++;

}

j++;

}

// Проверка на корректность clcomb перед использованием metric

if ((clcomb.size() != 0) && metric(clcomb) < metric(ans)) {

ans = clcomb; // обновляем лучший вариант

}

}

} while (next\_permutation(arr1.begin(), arr1.end())); // все перестановки

return ans; // возвращаем результат

}

// Функция для тестирования

bool test(vector<double> arr, vector<vector<double>> solution) {

// Сложность: O(n log n + m!) - аналогично cluster5, из-за вызова функции cluster5

// Память: O(1) - не используем дополнительно память, кроме параметров

return (cluster5(arr) == solution);

}

// Функция для вывода результатов

void hui(const std::vector<std::vector<double>>& vec) {

// Сложность: O(m) - где m - количество подмассивов

// Память: O(1) - не используем дополнительную память

for (const auto& innerVec : vec) {

std::cout << "{ ";

for (int num : innerVec) {

std::cout << num << " "; // вывод каждого числа

}

std::cout << "}" << std::endl;

}

}

// Основная функция

int main() {

// Тестирование функции

cout << test({ 2, 3, 4, 5, 6, 82, 100 }, { {2}, {3}, {82}, {100}, {4, 5, 6} }) << endl;

cout << test({ 1, 3, 4, 5, 6, 82, 1555, -1123123 }, { {-1123123}, {1}, {82}, {1555}, {3, 4, 5, 6} }) << endl;

cout << test({ 1, 3, -3321, 5, 6, 82, 1555, -1123123, 234234, -11111, 223321}, { {-1123123}, {-3321}, {33221}, {234234}, {1, 3, 5, 6, 82, 1555} }) << endl;

return 0; // Успех

}