ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Кластеризация массива»

Выполнил работу

Смирнов Александр

Академическая группа №J3111

Принято

Ментор Вершинин Владислав

Санкт-Петербург

2024

Структура отчёта:

1. Введение

Цель: попрактиковаться в плюсах и динамическом программировании, решить поставленную задачу.

Задачи:

- Продумать, как применить идеи из ДП к решению задачи
- Реализовать алгоритм
- Протестировать алгоритм

2. Теоретическая подготовка

Использовал идеи из ДП: декомпозировал задачу выделения кластеров по метрике до трех более простых задач: сортировка входного массива, вычисление метрики, перебор всех возможных расстановок разделителей на кластеры. Одна из важных идей – сортировка входного массива, она значительно ускоряет алгоритм, поскольку в отсортированном массиве не может быть пересекающихся кластеров при заданной метрике. Используемые типы данных: векторы, целые числа и числа с плавающей точкой.

3. Реализация

Этапы выполнения задачи:

- Проанализировать задачу
- Декомпозировать до нескольких простых подзадач (ДП)
- Решить каждую из задач
- Продебажить
- Написать тесты
- Обрадоваться, что поставили зачет и от души накинули баллов

Использовал библиотеку algorithm для сортировки; limits для того, чтобы инициализировать матрицу, хранящую лучшие метрики для каждого из вариантов разбиений на кластеры, максимально возможными значениями переменной типа double; cassert для тестов; а также vector и iostream.

Реализовал две вспомогательные функции (одна для подсчета метрики, другая для поиска лучшего разбиения на кластеры):

```
double metricCounter(const vector<double>& arr, int start, int end) {
    double sum = 0;
    for (int i = start; i <= end; i++) {
        sum += arr[i];
    }
    double mean = sum / (end - start + 1); // вычисляем среднее значение кластера
    double metric = 0;
    for (int i = start; i <= end; i++) {
        metric += abs(arr[i] - mean); // считаем метрику по формуле из ТЗ
    }
    return metric;
```

Изображение 1 – Функция для подсчета метрики

```
vector<vector<double>> dp_memory_metric(length + 1, vector<double>(K + 1, numeric_limits<double>::max()));
vector<vector<int>> dp_memory_split(length + 1, vector<int>(K + 1, -1));

dp_memory_metric[0][0] = 0; // метрика для одного кластера только с одним элементом равна 0 (следует из фо

for (int k = 1; k <= K; k++) { // пробегаемся по каждому разделителю
    for (int i = 1; i <= length; i++) { // пробегаемся по каждому элементу
        for (int j = 0; j < i; j++) { // пробегаемся по всем подмассивам от начала массива до элемента с и
            double current_metric = metricCounter(arr, j, i - 1); // считаем метрику (ТОЛЬКО в одном этом
            if (dp_memory_metric[j][k - 1] + current_metric < dp_memory_metric[i][k]) { // кладем в матриц
                dp_memory_metric[i][k] = dp_memory_metric[j][k - 1] + current_metric;
                dp_memory_split[i][k] = j;
            }
        }
    }
}
```

Изображение 2 – Фрагмент из функции для перебора разбиения на кластеры (с запоминанием метрики для посчитанных разбиений)

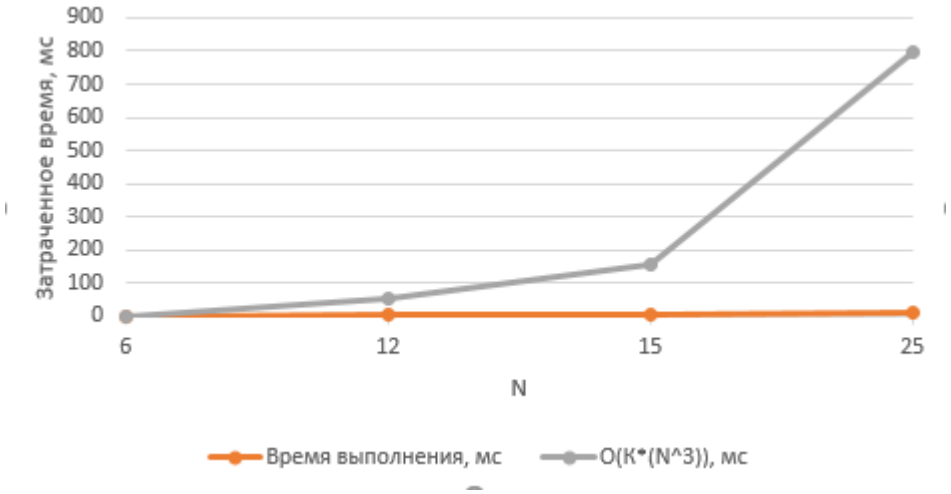
Написал два блока тестов для каждой из этих двух вспомогательных функций, в сумме 7 тестов, проверил крайние случаи.

4. Экспериментальная часть

Таблица №1 - Подсчёт сложности реализованного алгоритма

Размер входного набора (N)	6	12	15	25
Количество кластеров (K)	2	6	8	10
Время выполнения программы, сек	0.001	0.003	0.005	0.011
O(K*(N^3)), сек	0.001	0.052	0.154	0.798

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №3.



Изображение №3 - График работы алгоритма

Сложность алгоритма считал для худшего случая (пояснения в коде), этим объясняется сильная разница с фактической скоростью выполнения. Так же в целом на мой взгляд не очень корректна постановка задачи, в которой нужно сопоставить сложности алгоритма конкретное время выполнения, ведь из-за разных факторов (железо, какое время уходит на выполнение одной элементарной операции и тд, наличие памяти - ДП) нельзя сделать это достаточно точно. Но можно явно понять, что алгоритм работает гораздо быстрее, чем требовалось в ТЗ (2^N).

5. Заключение

В ходе выполнения работы мною был реализован алгоритм кластеризации массива с использованием техник динамического программирования. Цель работы была достигнута путём тестирования на массивах с различным количеством элементов и кластеров. Полученные результаты превосходят теоретическую оценку сложности алгоритма и сложность в ТЗ.

В качестве дальнейших исследований можно предложить усложнение метрики и структуры данных (чтобы учитывалось n-ое количество параметров, как в реальных алгоритмах кластеризации в МО). Так же можно добавить новые функции, например, для вставки новых элементов (метод К-ближайших соседей)

6. Приложения

Полный исходный код программы (без блоков с тестами, чтобы не занимать много места ими)

ПРИЛОЖЕНИЕ А

Листинг кода файла clustering.cpp

```cpp
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>
#include <time.h>
#include <cassert>

using namespace std;

// функция для вычисления метрики по формуле из ТЗ
double metricCounter(const vector<double>& arr, int start, int end) {
    double sum = 0;
    for (int i = start; i <= end; i++) {
        sum += arr[i];
    }
    double mean = sum / (end - start + 1); // вычисляем среднее значение кластера
    double metric = 0;
    for (int i = start; i <= end; i++) {
        metric += abs(arr[i] - mean); // считаем метрику по формуле из ТЗ
    }
    return metric;
}

// основная функция, ищет оптимальное разбиение на кластеры для минимизации метрики
vector<vector<double>> findClusters(vector<double>& arr, int K) {
    // засекаем время, надо для отчета
    clock_t tStart = clock();

    // O(N*log(N))
    sort(arr.begin(), arr.end()); // сортируем, тогда для решения задачи нужно будет в отсортированном массиве только расставить границы кластеров, т.к.

    int length = arr.size();
    // тут идея из ДП: создаем матрицу размера K*длина_входного_вектора, в которой будем хранить значения метрик
    // аналогично создаем матрицу dp_memory_split, в которой будем хранить индексы разделителей, при которых достигается минимальное значение метрики
    // вторая матрица нужна только для вывода оптимальных кластеров, если б надо было выводить только лучшую метрику, можно было бы без этой матрицы обой
    vector<vector<double>> dp_memory_metric(length + 1, vector<double>(K + 1, numeric_limits<double>::max())); // изначально все значения приравниваем к
    vector<vector<int>> dp_memory_split(length + 1, vector<int>(K + 1, -1));

    dp_memory_metric[0][0] = 0; // метрика для одного кластера только с одним элементом равна 0 (следует из формулы метрики)
```

```cpp
    // O(K*(N^3))
    for (int k = 1; k <= K; k++) { // пробегаемся по каждому разделителю
        for (int i = 1; i <= length; i++) { // пробегаемся по каждому элементу
            for (int j = 0; j < i; j++) { // пробегаемся по всем подмассивам от начала массива до элемента с индексом i
                double current_metric = metricCounter(arr, j, i - 1); // считаем метрику (ТОЛЬКО в одном этом подмассиве)
                if (dp_memory_metric[j][k - 1] + current_metric < dp_memory_metric[i][k]) { // кладем в матрицу с метриками минимальную метрику на соот
                    dp_memory_metric[i][k] = dp_memory_metric[j][k - 1] + current_metric;
                    dp_memory_split[i][k] = j;
                }
            }
        }
    }

    // восстановливаем кластеры
    vector<vector<double>> clusters(K); // тут создаем массив массивов (не матрицу), каждой строке соответсвует подмассив из оптимального разбиения
    int idx = length;

    // O(K*N)
    for (int k = K; k > 0; k--) {
        int j = dp_memory_split[idx][k];
        for (int i = j; i < idx; i++) {
            clusters[k - 1].push_back(arr[i]); // заполняем этот массив подмассивами
        }
        idx = j;
    }

    // выводим кластеры
    // O(K*N)
    for (int k = 0; k < K; k++) {
        cout << "Cluster " << k + 1 << ": ";
        for (auto num : clusters[k]) {
            cout << num << " ";
        }
        cout << endl;
    }
    cout << "Time taken: " << (double)(clock() - tStart) / CLOCKS_PER_SEC << " sec" << endl;

    return clusters;
```

```cpp
int main() {
    test_metricCounter();
    test_findClusters();

    int K;
    cout << "Enter the number of clusters (K): ";
    cin >> K;

    cout << "Enter the number of elements in the array (>=K): ";
    int length;
    cin >> length;

    vector<double> arr(length);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < length; i++) {
        cin >> arr[i];
    }

    findClusters(arr, K);

    return 0;
}
```

Спасибо за прочтение, желаю позитивного настроения и поменьше отчетов в жизни <3