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

Адриановская Александрв

Академическая группа C3100

Принято

Магистр практики Вершинин Владислав

Санкт-Петербург

2024

Введение

Цель лабораторной работы – решение NP полной задачи по кластеризации массива посредством полного перебора всех комбинаций значений этого массива. В процессе проведения данной работы требуется на практике убедиться в ресурсоемкости сложных алгоритмов. Необходимо оценить полученную сложность алгоритма и сопоставить время исполнения алгоритма с расчетным временем исполнения алгоритма в зависимости от количества элементов в исходном массиве.

1. Теоретическая подготовка

Для выполнения работы необходимо обладать теоретической подготовкой в части понимания принципов кластеризации массива, вычисления среднего значения элементов массива, вычисления расстояния элементов массива до среднего значения, фиксации лучшего результата. Также потребуется понимание метода преобразования числа десятичной системы счисления в другую систему (двоичную, троичную, и.т.д). Необходимо знать принцип оценки времени исполнения программы и расчет оценки сложности алгоритма.

1. Реализация

В процессе реализации были использованы стандартная библиотека iostream для вывода результата, chrono для фиксации времени работы, cmath для вычисления степени числа, vector для работы со структурой данных типа вектор.

Задача решается в несколько этапов. Основой алгоритма является цикл с количеством итераций 2^N - 1.

for (int number = 1; number < pow(num\_clusters, N) - 1; number++){}

В цикле производится инкрементация значения number от 1 до 2^N – 1. Далее каждое значение преобразуется в функции get\_mask в массив чисел, соответствующих разрядам преобразованного числа в другую систему счисления.

vector<int> get\_mask(int size, int number, int digits) {

vector<int> translated(size);

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

translated[i] = number % digits;

number /= digits;

}

return translated;

}  
Таким образом мы получаем очередную маску current\_mask, согласно которой мы далее будем распределять исходный массив на несколько кластеров.   
 for (int i = 0; i < N; i++) {

if (current\_mask[i] == 0) {

cluster1.push\_back(arr[i]);

} else {

Cluster2.push\_back(arr[i]);

}

}

Для каждого кластера вычисляется среднее значение его элементов.

double mean\_val(vector<double> arr) {

int N = arr.size();

double sum\_ = 0;

for (int i = 0; i < N; i++) { // O(N)

sum\_ = sum\_ + arr[i];

}

return sum\_ / N;

}

Затем вычисляется суммарное расстояние между элементами и средним значением кластера.

double score\_cluster(vector<double> arr, double cluster\_center) {

int N = arr.size();

double score = 0;

for (int i = 0; i < N; i++) { // O(N)

score += abs((arr[i] - cluster\_center));

}

return score;

}

Если сумма полученных значений из предыдущего этапа для всех кластеров оказывается лучше зафиксированного ранее значения, результат фиксируется как текущий лучший. Осуществляется промежуточный вывод результата в консоль.

if (score < best\_score) {

best\_score = score;

best\_cluster1 = cluster1;

best\_cluster2 = cluster2;

print\_clusters(cluster1, cluster2, score); // O(N)

}

После прохождения всех итераций цикла производится вывод конечного результата в консоль.

print\_clusters(best\_cluster1, best\_cluster2, best\_score);

1. Экспериментальная часть

Подсчёт по памяти.

В рамках реализации алгоритма были задействованы следующие структуры данных: исходный массив значений типа double (8\*N байт)

Два вектора для хранения лучших кластеров (2 \* (24 + 8 \* N/2)) байт,

Вектор текущей маски распределения массива по кластерам (24 + 4 \* N байта),

Два вектора текущих кластеров (2 \* (24 + 8 \* N/2)) байт

Все остальные потребители памяти – переменные типа int и double, используемые в функциях подсчета среднего значения, расстояния и преобразования числа в маску.

Подсчёт асимптотики.

В основном цикле происходит итерация по N^2 значениям, в теле цикла происходит несколько последовательных циклов перебора элементов массива размерности N. Итоговая сложность алгоритма O(2^N) \* O (N)

График зависимости времени от числа элементов.

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(2^N) и более. Для расчета возьмем длительность одной операции = 2 \* 10 ^ -7 с. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 2 | 5 | 10 | 15 | 20 | 23 | 25 |
| Время выполнения программы, с | 0 | 0,0002 | 0,006 | 0,164 | 4,9 | 41 | 175 |
| O(2^N)\*O(N), с | 0 | 0,00001 | 0,001 | 0,05 | 2 | 19 | 84 |
| O(2^N)\*O(3N) , с | 0 | 0,00003 | 0,003 | 0,15 | 6 | 57 | 252 |

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

Изображение №1 - График работы алгоритма

Время исполнения реализованного алгоритма находится между расчетными значениями для O(2^N)\*O(N) и O(2^N)\*O(3N) и совпадает с ними по профилю, что отражает сложность O(2^N) \* O(N)

Заключение

В ходе выполнения работы мною был реализован алгоритм кластеризации массива посредством полного перебора всех комбинаций значений этого массива. Цель работы была достигнута путём тестирования на массивах с различным числом и комбинациями входных значений. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма. В целях работы не стояло задачи по оптимизации алгоритма. Наоборот, стояла цель убедиться на практике, насколько ресурсоемкими могут быть решения алгоритмических задач алгоритмами повышенной сложности.

ПРИЛОЖЕНИЕ А

Листинг кода файла lab4.cpp

#include <iostream>

#include <vector>

#include <chrono>

#include <cmath>

using namespace std;

vector<int> get\_mask(int size, int number, int digits) {

//Преобразрвание числа в необходимую систему счисления

//и возврат результата в виде массива размера size

vector<int> translated(size); //O(n)

for (int i = size - 1; i >= 0; i--) { // O(N)

// Сохраняем остаток от деления на digits

translated[i] = number % digits;

// Делим число на digits

number /= digits;

}

return translated;

}

double mean\_val(vector<double> arr) {

// Подчет центра кластера (среднего значения)

int N = arr.size();

double sum\_ = 0;

for (int i = 0; i < N; i++) { // O(N)

sum\_ = sum\_ + arr[i];

}

return sum\_ / N;

}

double score\_cluster(vector<double> arr, double cluster\_center) {

// Подчет скора - сумма разностей каждого элемента и центра кластера (среднего значения)

int N = arr.size();

double score = 0;

for (int i = 0; i < N; i++) { // O(N)

score += abs((arr[i] - cluster\_center));

}

return score;

}

void print\_clusters(vector<double> cluster1, vector<double> cluster2, double score) {

// Тут просто выводим скор и кластера

cout << "Score: " << score << ". Cluster 1:";

for (int i = 0; i < cluster1.size(); i++) { // O(N/2)

cout << " " << cluster1[i];

}

cout << ". Cluster 2:";

for (int i = 0; i < cluster2.size(); i++) { // O(N/2)

cout << " " << cluster2[i];

}

cout << endl;

}

int main() {

//array of int. change to double

// double arr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 100, 101, 5};

double arr[] = {0, 1, 2, 3, 4, 5, 6.9 }; //N \* 8 bytes

int N = sizeof(arr) / sizeof(arr[0]);

int num\_clusters = 2;

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

// Создаем глобальные переменные для хранения лучшего скора и кластеров

double best\_score = numeric\_limits<double>::infinity();

vector<double> best\_cluster1; //24 bytes

// printf("Size of vector: %ld bytes \n", sizeof(best\_cluster1)); //24 bytes

vector<double> best\_cluster2; //24 bytes

//Проходим все значения от 1 до 2^N - 1 - 1 преобразуем их в маску и работаем с ней

for (int number = 1; number < pow(num\_clusters, N) - 1; number++) { // int = 4bytes, O(2^N)

vector<int> current\_mask = get\_mask(N, number, num\_clusters); //24 bytes

// for (int j = 0; j < current\_mask.size(); j++) {

// cout << " " << current\_mask[j];

// }

// cout << endl;

std::vector<double> cluster1; //24 bytes

std::vector<double> cluster2; //24 bytes

// Делаем переспределение значений в кластера согласно новой маски

for (int i = 0; i < N; i++) { // int = 4bytes, O(N)

if (current\_mask[i] == 0) {

cluster1.push\_back(arr[i]); // O(N / 2) = O(N)

} else {

cluster2.push\_back(arr[i]); // O(N / 2) = O(N)

}

}

// Считаем центры кластеров

double cluster1\_center = mean\_val(cluster1); //O(N)

double cluster2\_center = mean\_val(cluster2); //O(N)

// Считаем скор для каждого кластера, как сумма разностей элементов и центра

double score1 = score\_cluster(cluster1, cluster1\_center); // O(N)

double score2 = score\_cluster(cluster2, cluster2\_center); // O(N)

double score = score1 + score2;

// Чем меньше скор, тем лучше получились кластера (минимизируем)

if (score < best\_score) {

// Сохраняем лучший скор

best\_score = score;

// Сохраняем новые полученные кластера

best\_cluster1 = cluster1;

best\_cluster2 = cluster2;

// Отладочная информация

print\_clusters(cluster1, cluster2, score); // O(N)

}

}

auto end = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::milliseconds>(end - start);

cout << endl;

cout << "Время выполнения: " << duration.count() << " милисекунд" << endl;

print\_clusters(best\_cluster1, best\_cluster2, best\_score);

return 0;

}