ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Алгоритмы cортировки»

Выполнил работу

Быстриков Дмитрий

Академическая группа №J3111

Принято

Должность, звание Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Цель работы: создать программные решения, реализующие 3 различных алгоритма сортировки.

Задачи:

1. Найти алгоритмы сортировки, удовлетворяющие заданным условиям
2. Реализовать алгоритмы на языке C++
3. Протестировать код
4. Подготовить отчёт

Реализация

В качестве алгоритма с временной сложностью до О(N^2) и пространственной сложностью О(1) была выбрана bubble sort. Она рассматривает все возможные пары элементов массива, сравнивая их и меняя местами если правый элемент пары больше левого.

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Автоматически созданное описание

Изображение №1 – Полный алгоритм пузырьковой сортировки

В качестве алгоритма с временной сложностью до О(N \* log N) и пространственной сложностью О(1) был выбрал Heap sort. Он работает с неотсортированной и отсортированной частями массива, представленных в виде дерева и меняет местами первый и последний элементы в неотсортированной части.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение №2 – возвращение нового наибольшего элемента в положение корня

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение №3 – изначальное упорядочивание и уменьшение неотсортированной части

В качестве алгоритма с временной сложностью до О((N+b) \* k) и пространственной сложностью О(N + b) была выбрана radix sort. Она рассматривает элементы порязрядно, перемещая элементы массива согласно их рассматриваемому разряду, сохраняя изначальный порядок.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение №5 – алгоритм перемещения элементы согласно рассматриваемому разряду

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение №6 – вызываем алгоритм сортировки для разных разрядов;

1. Экспериментальная часть

Для проведения экспериментов была создана программа, псевдослучайно генерирующая массив заданного размера.

На первом этапе тестирования было замерено время сортировки псевдослучайных массивов размером от 1000 до 10^6 элементов всеми тремя алгоритмами. На основе полученных данных был построен график.

Изображение №7 – График зависимости времени сортировки от размера массива

Затем было измерено время сортировки всеми алгоритмами 50 различных псевдослучайных массивов размером 10^4. На основе собранных данных был построен графики типа box plot.

Изображение выглядит как текст, снимок экрана, Прямоугольник, диаграмма

Автоматически созданное описание

Изображение №8 – График для 10^4 с Bubble Sort

Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Автоматически созданное описание

Изображение №9 – График для 10^4 без Bubble Sort – для лучшей наглядности

1. Заключение

В ходе выполнения работы мною были реализованы 3 алгоритма сортировки. Для каждого из них были соблюдены требования по асимптотике и памяти. Были собраны экспериментальные данные, на основе которых построены графики.

Cocktail shake sort лучше всего подходит для случаев экстремальных ограничений по памяти и небольших массивов. In-place merge sort немного более требовательна к памяти за счёт большего количества вспомогательных переменных, однако показывает себя сильно лучше на более больших массивах. Наконец, bucket sort – самая быстрая, но весьма требовательная к памяти.

Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла bubble\_sort.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <chrono>

using namespace std;

void read\_csv(const std::string& filename, std::vector<int>& intArray) { // Функция чтения чисел из csv в массив

    std::ifstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file: " << filename << std::endl;

        return;

    }

    std::string line;

    while (std::getline(file, line)) {

        std::stringstream ss(line);

        std::string value;

        while (std::getline(ss, value, ',')) {

            try {

                int number = std::stoi(value);

                intArray.push\_back(number);

            } catch (const std::invalid\_argument& e) {

                std::cerr << "Invalid number: " << value << std::endl;

            } catch (const std::out\_of\_range& e) {

                std::cerr << "Number out of range: " << value << std::endl;

            }

        }

    }

    file.close();

}

void bubble\_sort(std::vector<int>& arr, int n) {

    for (int i = 0; i < n; i++) {

        bool flag = false;

        for (int j = 0; j < n-1; j++) {

        if (arr[j] > arr[j + 1]) {

            int temp = arr[j]; // создали дополнительную переменную

            arr[j] = arr[j + 1]; // меняем местами

            arr[j + 1] = temp; // значения элементов

            flag = true;

        }

        }

        if (!flag){

            break;

        }

    }

    }

int main() {

//    std::vector<std::string> data\_samples = {"1000", "2000", "3000", "4000", "5000", "6000", "7000", "8000", "9000",

//                                           "10000", "20000", "30000", "40000", "50000", "60000", "70000", "80000", "90000",

//                                           "100000", "200000", "300000", "400000", "500000", "600000", "700000", "800000", "900000",

//                                            "1000000"};

    std::vector<std::string> data\_samples = {"10000"};

    std::vector<float> corr\_median\_time;

    std::cout << data\_samples.size() << std::endl;

    for(int i=0; i < data\_samples.size(); i++){

        float sum\_time = 0;

        for (int j = 1; j <= 50; j ++) {

        std::vector<int> arr;

        std::string tmp\_file = data\_samples[i] + "\_" + std::to\_string(j) + ".csv";

        read\_csv(tmp\_file, arr);

        int n = arr.size();

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        bubble\_sort(arr, n);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

        std::chrono::duration<double> duration = end - start;

        std::cout << duration.count() << std::endl; // Выводим время работы

        sum\_time += duration.count();

    }

    corr\_median\_time.push\_back(sum\_time/2);

    }

    for (auto now : corr\_median\_time) {

        cout << "," << now << " ";

    }

  return 0;

    }

ПРИЛОЖЕНИЕ B

Листинг кода файла heap\_sort.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <chrono>

using namespace std;

//сложность выполняемого алгоритма - О(n\*log n)

void read\_csv(const std::string& filename, std::vector<int>& intArray) { // Функция чтения чисел из csv в массив

    std::ifstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file: " << filename << std::endl;

        return;

    }

    std::string line;

    while (std::getline(file, line)) {

        std::stringstream ss(line);

        std::string value;

        while (std::getline(ss, value, ',')) {

            try {

                int number = std::stoi(value);

                intArray.push\_back(number);

            } catch (const std::invalid\_argument& e) {

                std::cerr << "Invalid number: " << value << std::endl;

            } catch (const std::out\_of\_range& e) {

                std::cerr << "Number out of range: " << value << std::endl;

            }

        }

    }

    file.close();

}

void heapify(std::vector<int>& arr, int n, int i) {

    // находим наибольший элемент среди корня, левого потомка и правого

    int largest = i;

    int l = 2 \* i + 1; //левый потомок

    int r = 2 \* i + 2; //правый потомок

    if (l < n && arr[l] > arr[largest]) largest = l;

    if (r < n && arr[r] > arr[largest]) largest = r;

    // если корень не является наибольшим, меняем элементы и продолжаем упорядочивание

    if (largest != i) {

        swap(arr[i], arr[largest]);

        heapify(arr, n, largest);

    }

}

// основная функция для выполнения Heap Sort

void heapSort(std::vector<int>& arr, int n) {

    // упорядочиваем массив

    for (int i = (n / 2) - 1; i >= 0; i--) heapify(arr, n, i);

    // по одному извлекаем элементы из кучи

    for (int i = n - 1; i >= 0; i--) {

        swap(arr[0],arr[i]);  // текущий корень перемещён в конец

        heapify(arr, i, 0);  // снова вызываем функцию heapify для уменьшенной неотсортированной части

    }

}

int main() {

//    std::vector<std::string> data\_samples = {"1000", "2000", "3000", "4000", "5000", "6000", "7000", "8000", "9000",

//                                            "10000", "20000", "30000", "40000", "50000", "60000", "70000", "80000", "90000",

//                                           "100000", "200000", "300000", "400000", "500000", "600000", "700000", "800000", "900000",

//                                            "1000000"};

    std::vector<std::string> data\_samples = {"10000"};

    std::vector<float> corr\_median\_time;

    std::cout << data\_samples.size() << std::endl;

    for(int i=0; i < data\_samples.size(); i++){

        float sum\_time = 0;

        for (int j = 1; j <= 50; j ++) {

        std::vector<int> arr;

        std::string tmp\_file = data\_samples[i] + "\_" + std::to\_string(j) + ".csv";

        read\_csv(tmp\_file, arr);

        int n = arr.size();

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        heapSort(arr, n);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

        std::chrono::duration<double> duration = end - start;

        std::cout << duration.count() << std::endl; // Выводим время работы

        sum\_time += duration.count();

    }

    corr\_median\_time.push\_back(sum\_time/10);

    }

    for (auto now : corr\_median\_time) {

        cout << now << " ";

    }

  return 0;

    }

ПРИЛОЖЕНИЕ C

Листинг кода файла radix\_sort.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <chrono>

using namespace std;

void read\_csv(const std::string& filename, std::vector<int>& intArray) { // Функция чтения чисел из csv в массив

    std::ifstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file: " << filename << std::endl;

        return;

    }

    std::string line;

    while (std::getline(file, line)) {

        std::stringstream ss(line);

        std::string value;

        while (std::getline(ss, value, ',')) {

            try {

                int number = std::stoi(value);

                intArray.push\_back(number);

            } catch (const std::invalid\_argument& e) {

                std::cerr << "Invalid number: " << value << std::endl;

            } catch (const std::out\_of\_range& e) {

                std::cerr << "Number out of range: " << value << std::endl;

            }

        }

    }

    file.close();

}

void CountSort(std::vector<int>& a, int n, int curr\_digit)

{

    int i = {0};

    std::vector<int> count = {0,0,0,0,0,0,0,0,0,0};

    std::vector<int> result(n);

    //подсчитываем количество повторений

    for (i =0; i <n; i++){

        count[(a[i] / curr\_digit) % 10]++;

        }

    //создаем позиционную "карту" для введеных чисел на основе рассматриваемого curr\_digit

    for (i = 1; i<10; i++){

        count[i] += count[i-1];

        }

    //результат

    for (i =n-1; i>= 0; i--){

        result[count[(a[i] / curr\_digit) % 10] - 1] = a[i]; //заносим значения во временный массив по индексации полученной из count

        count[(a[i] / curr\_digit) % 10]--;

        }

    for (i =0; i <n; i++){

        a[i] = result[i]; //изменяем изначальный массив. конец цикла сортировки счетами

        }

    }

void radixsort(std::vector<int>& a, int n)

    {

    int curr\_digit;

    for (curr\_digit = 1; n/curr\_digit > 0; curr\_digit \*= 10){

        CountSort(a, n, curr\_digit);}

}

int main() {

//    std::vector<std::string> data\_samples = {"1000", "2000", "3000", "4000", "5000", "6000", "7000", "8000", "9000",

//                                           "10000", "20000", "30000", "40000", "50000", "60000", "70000", "80000", "90000",

//                                          "100000", "200000", "300000", "400000", "500000", "600000", "700000", "800000", "900000",

//                                           "1000000"};

    std::vector<std::string> data\_samples = {"10000"};

    std::vector<float> corr\_median\_time;

    std::cout << data\_samples.size() << std::endl;

    for(int i=0; i < data\_samples.size(); i++){

        float sum\_time = 0;

        for (int j = 1; j <= 50; j ++) {

        std::vector<int> arr;

        std::string tmp\_file = data\_samples[i] + "\_" + std::to\_string(j) + ".csv";

        read\_csv(tmp\_file, arr);

        int n = arr.size();

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        radixsort(arr, n);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

        std::chrono::duration<double> duration = end - start;

        std::cout << duration.count() << std::endl; // Выводим время работы

        sum\_time += duration.count();

    }

    corr\_median\_time.push\_back(sum\_time/10);

    }

    for (auto now : corr\_median\_time) {

        cout << now << " ";

    }

  return 0;

    }

ПРИЛОЖЕНИЕ D

Листинг кода файла array\_generator.py

import csv

import os

import random

def create\_csv\_files(n\_max, m, output\_dir="output"):

#n\_max - максимальный допустимый элемент; m - количество файлов которые нужно создать

    if not os.path.exists(output\_dir):

        os.makedirs(output\_dir)

    for i in range(1, m + 1):

        filename = os.path.join(output\_dir, f"{n\_max}\_{i}.csv")

        with open(filename, 'w', newline='') as csvfile:

            writer = csv.writer(csvfile)

            mas = [x for x in range(1, n\_max + 1)]

            random.shuffle(mas)

            for j in mas:

                writer.writerow([j])

create\_csv\_files(10, 10)