-7

Специальность: 09.02.07 Информационные системы и программирование

Выполнил:

Студент группы Y2333

Сааль С. Ф.

ОТЧЕТ

о лабораторной работе № 4

по теме: Алгоритмы сортировки

по дисциплине: Разработка программных модулей

Санкт-Петербург 2020

Проверил:

Антонов. М.Б.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

УНИВЕРСИТЕТ ИТМО

Факультет среднего профессионального образования

1. Цель и задачи

Цель: Реализовать и замерить скорость работы алгоритмов сортировки.

Задачи:

* изучить несколько алгоритмов сортировки;
* научиться оценивать сложность программы;
* изучить работу с динамическими структурами данных;
* научиться устанавливать метрики на программное обеспечение.

1. Задание

Общее ограничение на лабораторную работу:

* использовать разбитие на файлы основной программы (в main.cpp только функция main.cpp);
* документировать весь код используя аннотации Doxygen.

Требования к исходной программе:

1. Пользователь вводит размер массива в integer.
2. Массив для сортировки генерируется динамически случайно.
3. Массив сортируется используя 2 разных алгоритма сортировки.
4. Массив до сортировки и после сортировки каждым методом выводится в отдельные текстовые файлы (что бы можно было сравнить и проконтролировать работу алгоритмов).
5. Вывод программы: алгоритм сортировки и сколько времени заняла сортировка в секундах.

Требования к лабораторной:

1. Выбрать 1 квадратичный алгоритм сортировки и любой другой из алгоритмов и реализовать его в исходной программе.
2. На массиве из 10 элементов доказывается что программа работает корректно (с выводом до и после сортировки).
3. Замеряется время сортировки массивов размерами: 100.000, 200.000, 300.000, 400.000 элементов.
4. Замерять только время выполнения сортировки, а не само время выполнения программы.
5. Проводится оценка сложности реализованных алгоритмов сортировки.

Усложнение задания (\*):

* Визуализация работы алгоритма.
* Чтение исходного массива из файла.

Заполняется таблица результатов лабораторной работы:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100.000 | 200.000 | 300.000 | 400.000 | 500.000 | Тип зависимости | Сложность O(f) |
| Алгоритм 1 |  |  |  |  |  |  |  |
| Алгоритм 2 |  |  |  |  |  |  |  |

1. КОд программы

Файл main.cpp:

/\*\*

\* @file main.cpp

\* @author Сааль Степан

\* @brief Инициализация рандомного массива, запуск сортировки, замер времени

\*/

#include <iostream>

#include <iomanip>

#include "heapsort.h"

#include "bubblesort.h"

#include "array.h"

#include <ctime>

#include <fstream>

using namespace std;

/\*\*

\* @brief Сортировка пузырьком и вывод информации

\* @param A Массив для сортировки

\* @param size Размер массива

\* @param visualize Требуется ли визуализайия

\*/

void bubbleSortTest(int\* A, int size, bool visualize)

{

time\_t start, end;

ofstream sortedFile;

BubbleSort bubbleSort(visualize);

time(&start);

cout << endl << "BubbleSort:" << endl << endl;

int \*sortedArrayA = bubbleSort.makeSort(A, size);

time(&end);

cout << endl << "Execution time BubbleSort: " << difftime(end, start) << " seconds";

sortedFile.open("../BUBBLE\_SORTED.txt");

arrayToFile(sortedArrayA, size, sortedFile);

}

/\*\*

\* @brief Сортировка кучей и вывод информации

\* @param A Массив для сортировки

\* @param size Размер массива

\* @param visualize Требуется ли визуализайия

\*/

void heapSortTest(int\* B, int size, bool visualize)

{

time\_t start, end;

ofstream sortedFile;

HeapSort heapSort(visualize);

time(&start);

cout << endl << "HeapSort:" << endl << endl;

int \*sortedArrayB = heapSort.makeSort(B, size);

time(&end);

cout << endl << "Execution time HeapSort: " << difftime(end, start) << " seconds";

sortedFile.open("../HEAP\_SORTED.txt");

arrayToFile(sortedArrayB, size, sortedFile);

}

/\*\*

\* @brief Инициализация массива, запуск сортировки, замер времени

\* @return 0

\*/

int main()

{

int size, \*A = nullptr, \*B = nullptr;

bool visualize, fromFile;

ifstream input;

ofstream unsorted;

cout << "Input size of array:" << endl;

cin >> size;

cout << "Do you need visualization? 1-yes 0-no" << endl;

cin >> visualize;

cout << "How to create array? 0-random 1-from 'INPUT.txt'" << endl;

cin >> fromFile;

if (! fromFile) {

A = fillRandomArray(A, size);

B = copyArray(A, size);

} else {

input.open("../INPUT.txt");

A = arrayFromFile(A, size, input);

B = copyArray(A, size);

}

unsorted.open("../UNSORTED.txt");

arrayToFile(A, size, unsorted);

bubbleSortTest(A, size, visualize);

heapSortTest(B, size, visualize);

return 0;

}

Файл array.h:

/\*\*

\* @file array.h

\* @author Сааль Степан

\* @brief Функции для работы с массивами

\*/

#ifndef LAB4\_ARRAY\_H

#define LAB4\_ARRAY\_H

/\*\*

\* Заполнить рандомно массив

\* @param A Массив

\* @param size Размер массива

\* @return Указатель на заполненный массив

\*/

int\* fillRandomArray(int\* A, int size);

/\*\*

\* Скопировать массив

\* @param A Массив

\* @param size Размер массива

\* @return Указатель на копию

\*/

int\* copyArray(const int\* A, int size);

/\*\*

\* Вывести массив в файл

\* @param A Массив

\* @param size Размер массива

\* @param file Файл для сохранения

\*/

void arrayToFile(int\* A, int size, ofstream &file);

/\*\*

\* Ввести массив из файла

\* @param A Массив

\* @param size Размер массива

\* @param file Файл

\*/

int\* arrayFromFile(int\* A, int size, ifstream &file);

#endif //LAB4\_ARRAY\_H

Файл array.cpp:

/\*\*

\* @file array.cpp

\* @author Сааль Степан

\* @brief Реализация функций для работы с массивами

\*/

#include <random>

#include <iostream>

#include <fstream>

using namespace std;

int\* fillRandomArray(int\* A, int size)

{

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dis(0, size \* 2);

A = new int[size];

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

A[i] = dis(gen);

}

return A;

}

int\* copyArray(const int\* A, int size)

{

int \*B = new int [size];

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

B[i] = A[i];

}

return B;

}

void arrayToFile(int \*A, int size, ofstream &file) {

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

file << A[i] << " ";

}

file << endl << endl;

}

int\* arrayFromFile(int \*A, int size, ifstream &file) {

A = new int[size];

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

file >> A[i];

}

return A;

}

Файл bubblesort.h:

/\*\*

\* @file bubblesort.h

\* @author Сааль Степан

\* @brief Сортировка пузырьком

\*/

#ifndef LAB4\_BUBBLESORT\_H

#define LAB4\_BUBBLESORT\_H

/\*\*

\* @brief Сортировка пузырьком

\*/

class BubbleSort {

private:

/\*\* Массив для сортировки \*/

int \*A;

/\*\* Размер массива \*/

int size;

/\*\* Требуется ли визуализация работы массива \*/

bool visualize;

/\*\*

\* Функция для смены элементов массива местами

\* @param f Первый элемент

\* @param s Второй элемент

\*/

void swap(int \*f, int \*s)

{

int temp = \*f;

\*f = \*s;

\*s = temp;

}

public:

/\*\*

\* @param visualize Визуализировать ли сортировку

\*/

BubbleSort(bool visualize = false) : visualize(visualize) {}

/\*\*

\* Произвести сортировку

\* @param Array Массив для сортировки

\* @param arraySize Размер массива

\* @return указатель на отсортированный массив

\*/

int\* makeSort(int \*a, int arraySize)

{

A = a;

size = arraySize;

for (int i = 0; i < size - 1; i++) {

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

if (A[j] > A[j + 1]) {

if (visualize) {

visualizeSort(j, j + 1);

}

swap(&A[j], &A[j + 1]);

}

}

}

return A;

}

private:

/\*\*

\* Визуализация сортировки

\* @param first Первый элемент для свапа

\* @param second Второй элемент для свапа

\*/

void visualizeSort(int first, int second)

{

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

if (i == first || i == second) {

cout << "\\" << left << setw(2) << A[i] << "/ ";

} else {

cout << " " << left << setw(2) << A[i] << " ";

}

}

cout << endl;

}

};

#endif //LAB4\_BUBBLESORT\_H

Файл heapsort.h:

/\*\*

\* @file heapsort.h

\* @author Сааль Степан

\* @brief Сортировка кучей

\*/

#ifndef LAB4\_HEAPSORT\_H

#define LAB4\_HEAPSORT\_H

#include <iostream>

using namespace std;

/\*\*

\* @brief Сортировка кучей

\*/

class HeapSort

{

private:

/\*\*

\* Массив для сортировки

\*/

int \*A;

/\*\*

\* Размер массива

\*/

int size;

/\*\*

\* Кол-во неотсортированных элементов

\*/

int notSortingSize;

/\*\*

\* Визуализировать ли сортировку

\*/

bool visualize;

public:

/\*\*

\* @param visualize Визуализировать ли сортировку

\*/

HeapSort(bool visualize = false) : visualize(visualize) {}

/\*\*

\* Произвести сортировку

\* @param Array Массив для сортировки

\* @param sizeArray Размер массива

\* @return указатель на отсортированный массив

\*/

int\* makeSort(int \*Array, int sizeArray)

{

A = Array;

size = sizeArray;

notSortingSize = sizeArray;

sortArray();

return getSortedArray();

}

private:

/\*\*

\* Индекс левого элемента

\* @param parent Индекс родительского элемента в дереве

\* @return индекс

\*/

int leftEl(int parent)

{

return parent \* 2 + 1;

}

/\*\*

\* Индекс правого элемента

\* @param parent Индекс родительского элемента в дереве

\* @return индекс

\*/

int rightEl(int parent)

{

return parent \* 2 + 2;

}

/\*\*

\* Определение индекса верхушки

\* @return

\*/

int parentEl()

{

return ((notSortingSize - 1) / 2) + (notSortingSize - 1) % 2 - 1;

}

/\*\*

\* Сортировка массива

\*/

void sortArray()

{

int parent = parentEl();

while(parent >= 0){

sortElements(parent);

parent--;

}

}

/\*\*

\* Сортировка одного разветвления дерева

\* @param parent Индекс родительского элемента разветвления

\*/

void sortElements(int parent)

{

int startParent = parent;

int left = leftEl(parent);

int right = rightEl(parent);

if(left < notSortingSize && A[parent] < A[left]){

parent = left;

}

if(right < notSortingSize && A[parent] < A[right]){

parent = right;

}

if(parent != startParent){

if (visualize) {

visualizeSort(startParent, parent);

}

swap(A[startParent], A[parent]);

sortElements(parent);

}

}

/\*\*

\* Получение отсортированного массива

\* @return указатель на массив

\*/

int\* getSortedArray()

{

int parent;

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

if (visualize) {

visualizeSort(0, i);

}

swap(A[0], A[i]);

notSortingSize--;

sortArray();

}

return A;

}

/\*\*

\* Визуализация сортировки

\* @param first Первый элемент для свапа

\* @param second Второй элемент для свапа

\*/

void visualizeSort(int first, int second)

{

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

if (i == first || i == second) {

cout << "\\" << left << setw(2) << A[i] << "/ ";

} else {

cout << " " << left << setw(2) << A[i] << " ";

}

}

cout << endl;

}

};

#endif //LAB4\_HEAPSORT\_H

В качестве результата работы программы представлены результаты замеров выполнения алгоритмов:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100.000 | 200.000 | 300.000 | 400.000 | 500.000 | Тип зависимости | Сложность O(f) |
| BubbleSort | 64 | 241 | 549 | 1161 | 1550 | Квадратичная | O(n^2) |
| HeapSort | 56 | 207 | 502 | 931 | 1529 | Логарифмическая | O(n\*log(n)) |

1. Диаграммы классов

Диаграммы классов представлены на рисунках 1, 2.

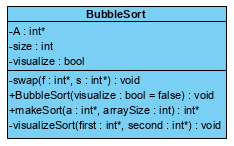


Рисунок 1 – диаграмма класса BubbleSort

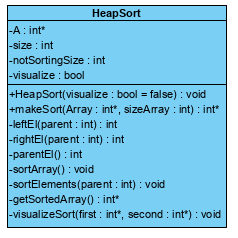


Рисунок 2 – диаграмма класса HeapSort

# ВЫВОД

В процессе выполнения лабораторной работы были изучены сортировка пузырьком и сортировка кучей. Были замерено время сортировки крупных массивов.