|  |  |
| --- | --- |
| Министерство науки и высшего образования Российской Федерации  Федеральное государственное автономное образовательное  учреждение высшего образования  «Южно-Уральский государственный университет  (национальный исследовательский университет)»  Институт естественных и точных наук  Факультет математики, механики и компьютерных технологий  Кафедра прикладной математики и программирования | |
| Сравнение быстродействия методов сортировки, быстрой сортировки Хоара и метода пузырька | |
| Пояснительная записка к курсовой работе  по дисциплине «Алгоритмы и структуры данных»  ЮУрГУ–010302.2020.153.ПЗ КР | |
|  | Автор работы,  студент группы ЕТ-212  \_\_\_\_\_\_\_\_\_\_ / Т.С. Шерстобитов  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г. |
|  | Руководитель работы,  старший преподаватель  \_\_\_\_\_\_\_\_\_\_ / А.С. Шелудько  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г. |
|  | Работа защищена с оценкой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г. |
| Челябинск 2020 | |

Аннотация

|  |  |
| --- | --- |
|  | Шерстобитов Т.С. Сравнение быстродействия методов сортировки, быстрой сортировки Хоара и метода пузырька. – Челябинск: ЮУрГУ,  ЕТ-212, 2020. – 32 с., 17 ил, библиогр. список – 6 наим., 3 прил. |

Оглавление

Введение 4

1 Постановка задачи 5

2 Алгоритм решения 6

3 Описание программы 10

Заключение 13

Библиографический список 14

Приложение 1 Текст программы 15

Приложение 2 Руководство пользователя 28

Приложение 3 Результат выполнения программы 32

Введение

Актуальность темы – правильный выбор алгоритма сортировки позволяет значительно оптимизировать время работы программ.

Цель работы – разработать программу, выполняющую сравнение быстродействия двух методов сортировки, быстрой сортировки Хоара и метода пузырька.

Объект работы– программа выполняющая сравнение алгоритмов сортировки.

Написать задание.авбфбыдпджывьбвыжмьджыьавь

1 Постановка задачи

Необходимо разработать программу, выполняющую сравнение быстродействия двух методов сортировки:

* метод пузырька;
* быстрая сортировка Хоара.

Схема сравнения времени сортировки целого массива из N чисел.

1. Генерировать случайный целочисленный массив A. Числа в нём ограниченны диапазоном от 0 до 109-1.

2. Создать массив B копию массива A.

3. Выполнить сортировку массива A методом пузырька и вычислить потраченное время.

4. Выполнить сортировку массива B с помощью быстрой сортировки Хоара и вычислить потраченное время

5. Повторить пункты 1-4 L раз.

6. Найти среднее время сортировки для каждого из методов.

7. Перейти к следующему значению N.

С помощью разработанной программы получить достаточное количество точек для построения графика зависимости времени сортировки от размера массива.

Значения L,N и другие исходные данные вводятся на старте программы из входного файла.

Напечатать таблицы зависимостей и построить совмещенные графики зависимостей времени сортировки от размера массива в координатах: по оси х – log10 (N), по оси у – log10(время(ms)).

Для построения графиков подберите в интернете подходящую программу, например, http://soft.mydiv.net/win/download-Graph.html.

2 Алгоритм решения

Программа выполняет сравнение двух алгоритмов сортировки.

* метод пузырька;
* быстрая сортировка Хоара.

Ниже на рисунке 2.1 и рисунке 2.2 представлены блок схемы данных алгоритмов.



Рисунок 2.1 – Алгоритм сортировки метода пузырька



Рисунок 2.2 – Алгоритм быстрой сортировки Хоара

Алгоритм выполнения тестирования алгоритмов сортировки приведен в рисунке 2.3.



Рисунок 2.3 – Алгоритм выполнения сравнения алгоритмов сортировки

3 Описание программы

Для выполнения сравнения методов сортировки необходимо разработать модуль выполняющий сравнение и содержащий данные о сравнении. Так как время выполнения быстрой сортировки Хоара в большинстве опытных случаев ровнялось 0 миллисекунд, было принято решение в качестве единицы измерения брать не миллисекунд, а время процессора (тики). Основываясь на алгоритме выполнения тестирования алгоритмов сортировки (см. рисунок 2.3), разработана приведённая в Приложении 1.

Для получения инструкций по использованию разрабатываемой программы см. Приложение 2.

Пример результатов выполнения программы см. Приложение 3.

Заключение

В результате работы была разработана программа выполняющая сравнение быстродействия двух методов сортировки:

метод пузырька;

быстрая сортировка Хоара.

Библиографический список

1. Ахо, А.В. Структуры данных и алгоритмы / А.В. Ахо, Д.Э. Хопкрофт, Д.Д. Ульман. – М.: Вильямс, 2000. – 382 с.

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – М.: Мир, 1989. – 360 c.

3. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2001. – 955 с.

4. Кнут, Д. Искусство программирования. Том 3. Сортировка и поиск / Д. Кнут. – М.: Вильямс, 2000. – 822 с.

5. Подбельский, В.В. Курс программирования на языке Си / В.В. Подбельский, С.С. Фомин. – М.: ДМК Пресс, 2012. – 384 с.

6. Хаггарти, Р. Дискретная математика для программистов / Р. Хаггарти. – М.: Техносфера, 2012. – 399 с.

Приложение 1

Текст программы

### П1.1 Sort.hpp

#ifndef SORT\_HPP

#define SORT\_HPP

void bubbleSort(int\* array, int n);

void quickSort(int\* array,int first, int last);

#endif //MEASUREMENT\_HPP

### П1.2 Sort.cpp

#include "Sort.hpp"

void bubbleSort(int\* array, int n){

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

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

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

int tmp = array[j];

array[j] = array[j + 1];

array[j + 1] = tmp;

}

}

}

}

void quickSort(int\* array,int first, int last){

if (first < last){

int left = first, right = last, middle = array[(left + right) / 2];

do{

while (array[left] < middle){

left++;

}

while (array[right] > middle){

right--;

}

if (left <= right){

int tmp = array[left];

array[left] = array[right];

array[right] = tmp;

left++;

right--;

}

} while (left <= right);

quickSort(array, first, right);

quickSort(array, left, last);

}

}

### П1.3 Analysis.hpp

#include "Measurement.hpp"

#ifndef ANALYSIS\_HPP

#define ANALYSIS\_HPP

class Analysis{

public:

/\*\*

\* Конструктор.

\* Число N - Размер массива для анализа

\*/

Analysis(unsigned int N);

virtual ~Analysis();

/\*\*

\* Возвращает данные о работе алгоритма сравнения метода пузырька

\*/

Measurement \*getBubbleMeasurement() const;

/\*\*

\* Возвращает данные о работе алгоритма сравнения быстрой сортировки Хоара

\*/

Measurement \*getQuickMeasurement() const;

private:

Measurement \*bubble, \*quick;

unsigned int N;

};

#endif //ANALYSIS\_HPP

### П1.4 Measurement.hpp

#include <ctime>

#ifndef MEASUREMENT\_HPP

#define MEASUREMENT\_HPP

/\*\*

\* Даные о работе алгоритма

\*/

class Measurement{

public:

/\*\*

\* Создает новый объект класса данных о замерах,

\* с случайно сгенерированым массивом размера size.

\*/

Measurement(unsigned int size);

Measurement(const Measurement &orig);

virtual ~Measurement();

/\*\*

\* Установка точки начала замера

\*/

void begin();

/\*\*

\* Установка точки окончания замера

\*/

void end();

/\*\*

\* Возращает начальный массив;

\*/

int\* getRowArray() const;

/\*\*

\* Возращает массив для сортировки

\*/

int\* getArray() const;

/\*\*

\* Возращает размер массивов

\*/

unsigned int getArraySize() const;

/\*\*

\* Возращает время начала замера

\*/

clock\_t getBeginTime() const;

/\*\*

\* Возращает время окончания замера

\*/

clock\_t getEndTime() const;

/\*\*

\* Возращает время замера

\*/

clock\_t getDeltaTime() const;

private:

int\* rowArray;//Копия массива до сортировки

int\* array;//Массив для сортировки

unsigned int size;//Размер массива

clock\_t beginTime = 0, endTime = 0;//Время начала/конца замера

};

#endif //MEASUREMENT\_HPP

### П1.5 Analysis.cpp

#include "Analysis.hpp"

#include "Sort.hpp"

Analysis::Analysis(unsigned int N){

bubble = new Measurement(N);

quick = new Measurement(\*bubble);

bubble->begin();

bubbleSort(bubble->getArray(), N);

bubble->end();

quick->begin();

quickSort(quick->getArray(), 0, N - 1);

quick->end();

}

Analysis::~Analysis(){

delete bubble;

delete quick;

}

Measurement \*Analysis::getBubbleMeasurement() const{

return bubble;

}

Measurement \*Analysis::getQuickMeasurement() const{

return quick;

}

### П1.6 Measurement.cpp

#include "Measurement.hpp"

#include <random>

//Максимально возможное сгенерированное число

#define MAX 999999999

Measurement::Measurement(unsigned int size) : size(size) {

static std::default\_random\_engine dre(time(0));

array = new int[size];

rowArray = new int[size];

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

array[i] = rowArray[i] = dre() % MAX;

}

}

Measurement::Measurement(const Measurement &orig){

size = orig.size;

array = new int[size];

rowArray = new int[size];

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

array[i] = orig.array[i];

rowArray[i] = orig.array[i];

}

}

Measurement::~Measurement() {

delete[] array;

delete[] rowArray;

}

void Measurement::begin() {

beginTime = clock();

}

void Measurement::end() {

endTime = clock();

}

int \*Measurement::getRowArray() const {

return rowArray;

}

int\* Measurement::getArray() const{

return array;

}

unsigned int Measurement::getArraySize() const {

return size;

}

clock\_t Measurement::getBeginTime() const {

return beginTime;

}

clock\_t Measurement::getEndTime() const {

return endTime;

}

clock\_t Measurement::getDeltaTime() const {

return endTime - beginTime;

}

### П1.7 Main.cpp

#include <iostream>

#include <conio.h>

using namespace std;

/\*\*

\* Старт программы выполняющей анализ алгоритмов сортировки

\*/

int main(int argc, char \*\*argv) {

unsigned int N,L;

cin >> N >> L;

ofstream out;

out.open(to\_string(N) + "\_" + to\_string(L) + ".txt");

if (!out.is\_open())

return EXIT\_FAILURE;

out << N << " " << L << "\n";

for (unsigned int i = 0; i < L; i++) {

Analysis \*a = new Analysis(N);

out << a->getBubbleMeasurement()->getDeltaTime() << " "

<< a->getQuickMeasurement()->getDeltaTime() << "\n";

delete a;

}

out.close();

return 0;

}

Приложение 2

Руководство пользователя

1. Запустите выполнение исполняемого файла программы.
2. Введите начальный и конечный размер массива, шаг размера, количество тестов на каждый размер.

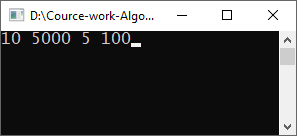


Рисунок П.2.1 – Ввод данных

1. Дождитесь завершения работы программы.

Приложение 3

Результат выполнения программы

Время выполнения представлено в тиках (процессорное время). Название файла строка содержит информацию о начальном и конечном размере массивов и количестве тестов. Строки содержат размера массива и среднее время выполнения алгоритмов, первым идет время выполнения метода пузырька, следующим время выполнения быстрой сортировки Хоара.