-7

Специальность: 09.02.07 Информационные системы и программирование

Выполнил:

Студент группы Y2433

Тертень Д. В.

ОТЧЕТ

о лабораторной работе № 5

по теме: Алгоритмы поиска

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 поиска на больших структурах данных.

Задачи:

* изучить несколько алгоритмов поиска элементов в массиве;
* научиться оценивать сложность программы;
* 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. Изначально в программе размер массива задан как 1.000.000 (один миллион элементов).
2. Все элементы должны быть от [0, 6.000.000)
3. Массив для поиска генерируется динамически случайно.
4. Генерируется массив из 40 элементов (или предоставить ввод этих элементов пользователем). Элементы должны быть от [0, 6.000.000)
5. Вывод программы: алгоритм поиска и сколько времени занял поиск в секундах.

Требования к лабораторной:

1. Выбрать один алгоритм модификации метода перебора и один алгоритм поиска в сортированном массиве и реализовать его в исходной программе.
2. За один запуск программа должна выполнить поиск всех 40 элементов и вывести результаты поиска в консоль.
3. Замеряется время поиска массиве всех 40 элементов и результат выводится в конце поиска.
4. Замерять только время выполнения поиска, а не само время выполнения программы.
5. Проводится оценка сложности реализованных алгоритмов сортировки.
6. Заполняется таблица результатов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Перебор (модификация) |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

1. КОд программы

Файл main.cpp:

/\*\*

\* @author Тертень Даниил

\* @file main.cpp

\* @brief Точка входа в программу

\*/

#include <iostream>

#include "array.h"

#include <iomanip>

#include "heapsort.h"

using namespace std;

/\*\*

\* @brief Точка входу в программу, получение вводимых данных

\* @return

\*/

int main() {

int \*Array = nullptr, sizeArray = 1000000, \*DesiredArray = nullptr, sizeDesired = 40, maxValue = 6000000;

bool showResult;

cout << "Show results? 0-no 1-yes" << endl;

cin >> showResult;

Array = fillRandomArray(Array, sizeArray, maxValue);

DesiredArray = fillRandomArray(DesiredArray, sizeDesired, maxValue, 10);

transpositionSearchManyElements(Array, sizeArray, DesiredArray, sizeDesired, showResult);

HeapSort sort;

Array = sort.makeSort(Array, sizeArray);

binarySearchManyElements(Array, sizeArray, DesiredArray, sizeDesired, showResult);

return 0;

}

Файл array.h:

/\*\*

\* @file array.h

\* @author Тертень Даниил

\* @brief Функции для работы с массивами

\*/

#ifndef LAB5\_ARRAY\_H

#define LAB5\_ARRAY\_H

/\*\*

\* Заполнить рандомно массив

\* @param A Массив

\* @param size Размер массива

\* @return Указатель на заполненный массив

\*/

int\* fillRandomArray(int\* A, int size, int maxValue, int randomOffset = 0);

/\*\*

\* Вывести результат на экран

\* @param value Значение

\* @param index Индекс значения

\* @param showResult Показывать ли результат или нет

\*/

void printResult(int value, int index, bool showResult);

/\*\*

\* Поиск массива элементов в массиве с помощью transpositionSearch, вывод информации на экран

\* @param Array Массив, в котором искать

\* @param sizeArray

\* @param DesiredArray Массив, элементы которого надо искать

\* @param sizeDesired

\* @param showResults

\*/

void transpositionSearchManyElements(int\* Array, int sizeArray, int\* DesiredArray, int sizeDesired, bool showResults);

/\*\*

\* Поиск массива элементов в массиве с помощью transpositionSearch, вывод информации на экран

\* @param Array Массив, в котором искать

\* @param sizeArray

\* @param DesiredArray Массив, элементы которого надо искать

\* @param sizeDesired

\* @param showResults

\*/

void binarySearchManyElements(int\* Array, int sizeArray, int\* DesiredArray, int sizeDesired, bool showResults);

#endif //LAB5\_ARRAY\_H

Файл array.cpp:

/\*\*

\* @file array.cpp

\* @author Тертень Даниил

\* @brief Реализация функций для работы с массивами

\*/

#include <iostream>

#include <ctime>

#include "array.h"

#include "transpositionsearch.h"

#include "binarysearch.h"

#include <random>

#include <chrono>

#include <zconf.h>

using namespace std;

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

{

std::mt19937 rng(time(0) + randomOffset);

std::uniform\_int\_distribution<int> uni(0, maxValue);

A = new int[size];

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

A[i] = uni(rng);

}

return A;

}

void printResult(int value, int index, bool showResult)

{

if (showResult) {

if (index != -1) {

cout << "Value " << value << ": index " << index << endl;

} else {

cout << "Value " << value << ": not found" << endl;

}

}

}

void transpositionSearchManyElements(int \*Array, int sizeArray, int \*DesiredArray, int sizeDesired, bool showResults) {

int index;

cout << endl << endl << "Transposition search: " << endl << endl;

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

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

index = transpositionSearch(Array, sizeArray, DesiredArray[i]);

printResult(DesiredArray[i], index, showResults);

}

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

cout << endl << "Execution time: " << chrono::duration\_cast<chrono::nanoseconds>(end-begin).count() << "ns";

}

void binarySearchManyElements(int \*Array, int sizeArray, int \*DesiredArray, int sizeDesired, bool showResults) {

int index;

cout << endl << endl << "Binary search: " << endl << endl;

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

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

index = binarySearch(Array, sizeArray, DesiredArray[i]);

printResult(DesiredArray[i], index, showResults);

}

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

cout << endl << "Execution time: " << chrono::duration\_cast<chrono::nanoseconds>(end-begin).count() << "ns";

}

Файл transpositionsearch.h:

/\*\*

\* @file transpositionsearch.h

\* @author Тертень Даниил

\* @brief Определение функции поиска перебором с транспозицией

\*/

#ifndef LAB5\_TRANSPOSITIONSEARCH\_H

#define LAB5\_TRANSPOSITIONSEARCH\_H

/\*\*

\* Поиск методом перебора с транспозицией

\* @param array Массив, в котором производится поиск

\* @param size Размер массива

\* @param desired Искомое значение

\* @return

\*/

int transpositionSearch(int \*array, int size, int desired);

#endif //LAB5\_TRANSPOSITIONSEARCH\_H

Файл transpositionsearch.cpp:

/\*\*

\* @file transpositionsearch.cpp

\* @author Тертень Даниил

\* @brief Реализация функции поиска перебором с транспозицией

\*/

#include "transpositionsearch.h"

/\*\*

\* Смена двух элементов массива

\* @param first

\* @param second

\*/

void swap(int \*first, int \*second)

{

int temp = \*first;

\*first = \*second;

\*second = temp;

}

int transpositionSearch(int \*array, int size, int desired)

{

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

if (array[i] == desired) {

if (i != 0) {

swap(&array[i], &array[i - 1]);

}

return i;

}

}

return -1;

}

Файл binarysearch.h:

/\*\*

\* @file transposition.h

\* @author Тертень Даниил

\* @brief Определение функции поиска перебором с транспозицией

\*/

#ifndef LAB5\_BINARYSEARCH\_H

#define LAB5\_BINARYSEARCH\_H

/\*\*

\* @brief Бинарный поиск

\* @param array

\* @param size

\* @param desired

\* @return

\*/

int binarySearch(const int \*array, int size, int desired);

#endif //LAB5\_BINARYSEARCH\_H

Файл binarysearch.cpp:

//

// Created by miste on 01.10.2020.

//

#include "binarysearch.h"

int binarySearch(const int \*array, int size, int desired) {

int middle, left = 0, right = size - 1;

while (true)

{

middle = (left + right) / 2;

if (desired == array[middle]) return middle;

right = (desired < array[middle]) ? (middle - 1) : (right);

left = (desired > array[middle]) ? (middle + 1) : (left);

if (left > right) return -1;

}

}

Файл heapsort.h (взят из предыдущей лабораторной работы):

/\*\*

\* @file heapsort.h

\* @author Тертень Даниил

\* @brief Сортировка кучей

\*/

#ifndef LAB5\_HEAPSORT\_H

#define LAB5\_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 //LAB5\_HEAPSORT\_H

В качестве результата работы программы представлены результаты замеров времени выполнения алгоритмов поиска в миллесекундах в таблице 1 и скриншот результата выполнения программы на рисунке 1:

Таблица 1 – время выполнения алгоритмов поиска.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Сложность |
| Перебор с транспозицией | 217мс | 328мс | 244мс | 267мс | 229мс | 242мс | 194мс | 319мс | 309мс | 160мс | O(n) |
| Бинарный поиск | 102мс | 22мс | 164мс | 46мс | 26мс | 152мс | 4мс | 105мс | 12мс | 22мс | O(log n) |

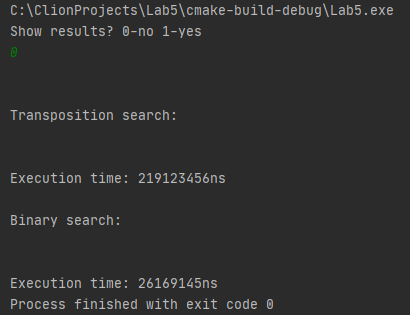


Рисунок 1 – вывод времени выполнения алгоритмов поиска

1. Диаграммы классов

Диаграмма класса представлена на рисунке 2.

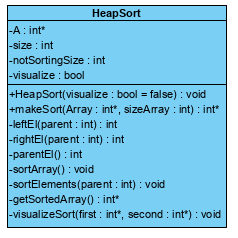


Рисунок 2 – диаграмма класса HeapSort (взята из лабораторной работы №4)

# ВЫВОД

В процессе выполнения лабораторной работы были изучены алгоритм поиска перебором с модификацией транспозиции и алгоритм бинарного поиска. Было замерено время поиска элементов в крупных массивах. По результатам замеров видно, что бинарный поиск гораздо быстрее перебора, но надо учитывать, что в бинарном поиске используется сортированный массив. Также заметно, что у бинарного поиска нестабильное время выполнения.