ФЕДЕРАЛЬНОЕ Государственное АВТОНОМНОЕ образовательное учреждение

высшего образования

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

Кафедра математического и программного обеспечения информационных систем

**Отчет по лабораторной работе №4**

**по дисциплине: «Структуры и алгоритмы компьютерной обработки данных»**

**Тема работы «Сравнительный анализ алгоритмов поиска»**

студента очного отделения

2 курса 12001801 группы

Скрипниченко Илья Игоревич

Проверил(а):

Курлов Василий Васильевич

Белгород 2019

**Лабораторная работа №4**

**Цель работы:** изучение алгоритмов поиска элемента в массиве и закрепление навыков в проведении сравнительного анализа алгоритмов.

**Листинг программы**

#include "pch.h"

#include <iostream>

#include <cstdlib>

#include <ctime>

/////////////////////////////Алгоритмы поиска в неупорядоченных массивах

unsigned linear\_search(const int\*, const int, const unsigned); //Линейный

unsigned fast\_linear\_search\_for\_disordered(const int\*, const int, const unsigned); //Быстрый линейный (У)

/////////////////////////////Алгоритмы поиска в упорядоченных массивах

unsigned fast\_linear\_search\_for\_ordered(const int\*, const int, const unsigned); //Быстрый линейный (НУ)

unsigned binary\_search(const int\*, const int, const unsigned); //Бинарный

unsigned block\_search(const int\*, const int, const unsigned); //Блочный

/////////////////////////////Заполнение массива с количествами операций сравнения

void fill\_comparisons\_array(unsigned\*, const unsigned, unsigned(\*)(const int\*, const int, const unsigned), const int\*);

/////////////////////////////Нахождения максимального количества операций сравнения

unsigned max\_comparisons\_number(const unsigned\*, const unsigned);

/////////////////////////////Нахождения среднего числа операций сравнения

unsigned comparisons\_average\_number(const unsigned\*, const unsigned);

int main()

{

setlocale(0, "");

const unsigned SIZE = 50;

using namespace std;

int disordered\_array[SIZE];

unsigned i;

int j;

srand(time(NULL));

for (i = 0; i < SIZE; i++)

{

disordered\_array[i] = rand();

disordered\_array[i] %= 999; //генерация случайных чисел в диапазоне до 999

}

/////////////////////////////Создание упорядоченного массива

int ordered\_array[SIZE];

for (i = 0; i < SIZE; i++) ordered\_array[i] = disordered\_array[i];

for (i = 1; i < SIZE; i++)

{

for (j = i; j > 0 && ordered\_array[j - 1] > ordered\_array[j]; j--)

{

int temp = ordered\_array[j];

ordered\_array[j] = ordered\_array[j - 1];

ordered\_array[j - 1] = temp;

}

}

unsigned comparisons\_array[SIZE] = {0};

cout << "1. АЛГОРИТМЫ ПОИСКА В НЕУПОРЯДОЧЕННОМ МАССИВЕ РАЗМЕРА " << SIZE << "\n\n";

cout << "1.1. Линейный поиск\n";

fill\_comparisons\_array(comparisons\_array, SIZE, linear\_search, disordered\_array);

cout << "Максимальное количество операций сравнения = " <<

max\_comparisons\_number(comparisons\_array, SIZE) << endl;

cout << "Среднее количество операций сравнения = " <<

comparisons\_average\_number(comparisons\_array, SIZE) << endl << endl;

cout << "1.2. Быстрый линейный поиск\n";

fill\_comparisons\_array(comparisons\_array, SIZE, fast\_linear\_search\_for\_disordered, disordered\_array);

cout << "Максимальное количество операций сравнения = " <<

max\_comparisons\_number(comparisons\_array, SIZE) << endl;

cout << "Среднее количество операций сравнения = " <<

comparisons\_average\_number(comparisons\_array, SIZE) << endl << endl << endl;

cout << "2. АЛГОРИТМЫ ПОИСКА В УПОРЯДОЧЕННОМ МАССИВЕ РАЗМЕРА " << SIZE << "\n\n";

cout << "2.1. Быстрый линейный поиск\n";

fill\_comparisons\_array(comparisons\_array, SIZE, fast\_linear\_search\_for\_ordered, ordered\_array);

cout << "Максимальное количество операций сравнения = " <<

max\_comparisons\_number(comparisons\_array, SIZE) << endl;

cout << "Среднее количество операций сравнения = " <<

comparisons\_average\_number(comparisons\_array, SIZE) << endl << endl;

cout << "2.2. Бинарный поиск\n";

fill\_comparisons\_array(comparisons\_array, SIZE, binary\_search, ordered\_array);

cout << "Максимальное количество операций сравнения = " <<

max\_comparisons\_number(comparisons\_array, SIZE) << endl;

cout << "Среднее количество операций сравнения = " <<

comparisons\_average\_number(comparisons\_array, SIZE) << endl << endl;

cout << "2.3. Блочный поиск\n";

fill\_comparisons\_array(comparisons\_array, SIZE, block\_search, ordered\_array);

cout << "Максимальное количество операций сравнения = " <<

max\_comparisons\_number(comparisons\_array, SIZE) << endl;

cout << "Среднее количество операций сравнения = " <<

comparisons\_average\_number(comparisons\_array, SIZE) << endl;

return 0;

}

unsigned linear\_search(const int\* arr, const int des, const unsigned sz)

{

unsigned comp\_num = 0;

unsigned i;

for (i = 0; i < sz; i++)

{

comp\_num++;

if(arr[i] == des) break;

}

return comp\_num;

}

unsigned fast\_linear\_search\_for\_disordered(const int\* arr, const int des, const unsigned sz)

{

unsigned comp\_num = 0;

int\* new\_array = new int[sz + 1];

new\_array[sz] = des;

for (unsigned j = 0; j < sz; j++)

{

new\_array[j] = arr[j];

}

unsigned i;

for (i = 0; ; i++)

{

comp\_num++;

if (new\_array[i] == des)

{

if (i == sz) break; //Элемент не найден

else break; //Элемент найден

}

}

delete[] new\_array;

return comp\_num;

}

unsigned fast\_linear\_search\_for\_ordered(const int\* arr, const int des, const unsigned sz)

{

unsigned comp\_num = 0;

int\* new\_array = new int[sz + 1];

new\_array[sz] = des;

for (unsigned j = 0; j < sz; j++)

{

new\_array[j] = arr[j];

}

unsigned i;

for (i = 0; ; i++)

{

if (new\_array[i] > des) break; //В массиве нет элемента

comp\_num++;

if (new\_array[i] == des)

{

if (i == sz) break; //Элемент не найден

else break; //Элемент найден

}

}

delete[] new\_array;

return comp\_num;

}

unsigned binary\_search(const int\* arr, const int des, const unsigned sz)

{

unsigned comp\_num = 0;

unsigned right = sz - 1;

unsigned left = 0;

unsigned middle = 0;

for (;;)

{

middle = (left + right) / 2;

comp\_num++;

if (arr[middle] == des) break;

else

{

comp\_num++;

if (arr[middle] > des) right = middle - 1;

else

{

comp\_num++;

if (arr[middle] < des) left = middle + 1;

}

}

if (left > right) break;

}

return comp\_num;

}

unsigned block\_search(const int\* arr, const int des, const unsigned sz)

{

unsigned comp\_num = 1;

unsigned i;

int j = 0;

unsigned block = sz / 10; //Разбиваем массив на равные блоки, содержащие по 10 элементов

i = block - 1;

while (des > arr[i]) //Просматриваем последний элемент каждого блока

{

comp\_num++;

i += block;

}

for (j = i - block + 1; j < i; j++) //Линейный поиск в нужном блоке

{

comp\_num++;

if (arr[j] == des) break;

}

return comp\_num;

}

void fill\_comparisons\_array(unsigned\* c\_arr, const unsigned sz, unsigned(\*ptr)(const int\*, const int, const unsigned),

const int\* arr)

{

unsigned i;

for (i = 0; i < sz; i++) c\_arr[i] = ptr(arr, arr[i], sz);

}

unsigned max\_comparisons\_number(const unsigned\* arr, const unsigned sz)

{

unsigned i, mc\_index, mc\_num;

for (i = 1, mc\_index = i - 1, mc\_num = arr[mc\_index]; i < sz; i++)

{

if (arr[mc\_index] < arr[i])

{

mc\_index = i;

mc\_num = arr[mc\_index];

}

}

return mc\_num;

}

unsigned comparisons\_average\_number(const unsigned\* arr, const unsigned sz)

{

unsigned i;

unsigned sum = 0;

for (i = 0; i < sz; i++) sum += arr[i];

return sum / sz;

}

**Результаты работы программы**

Таблица 1. Максимальное количество операций сравнения

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Алгоритмы поиска** | **Количество элементов в массиве** | | | | | | | | |
| 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 |
| Линейный | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 |
| Быстрый линейный для неупорядоченного массива | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 |
| Быстрый линейный для упорядоченного массива | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 |
| Бинарный | 16 | 19 | 22 | 22 | 22 | 25 | 24 | 25 | 25 |
| Блочный | 14 | 19 | 24 | 29 | 34 | 39 | 44 | 49 | 54 |

Таблица 1. Среднее количество операций сравнения

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Алгоритмы поиска** | **Количество элементов в массиве** | | | | | | | | |
| 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 |
| Линейный | 25 | 48 | 71 | 92 | 117 | 137 | 158 | 177 | 195 |
| Быстрый линейный для неупорядоченного массива | 25 | 48 | 71 | 92 | 117 | 137 | 158 | 177 | 195 |
| Быстрый линейный для упорядоченного массива | 25 | 50 | 75 | 100 | 125 | 150 | 175 | 200 | 225 |
| Бинарный | 10 | 12 | 14 | 14 | 15 | 16 | 16 | 17 | 17 |
| Блочный | 8 | 11 | 13 | 16 | 18 | 20 | 23 | 25 | 28 |

**Графики зависимостей ФВС**

**Вывод**

Исходя из полученных результатов и графиков зависимостей ФВС, можно сделать следующие вывода по основным алгоритмам поиска.

**1. Алгоритмы линейного поиска**

Порядок функции временной сложности для этих алгоритмом будет равняться **O(N)**, где N – количество элементов в массиве.

Как видно из графика, в среднем для нахождения искомого элемента понадобится **O(N/2)** операций.

**2. Алгоритм бинарного поиска**

На каждом шаге алгоритма область поиска в массиве будет уменьшатся вдвое. Зная этот факт и исходя из графика, можно сделать вывод, что в среднем порядок функции временной сложности будет равен **O(log2N)**, где N – количество элементов.

**3. Алгоритм блочного поиска**

Если массив из N элементов разбивается на M равных блоков, то в худшем случае, если элемент находится в последнем блоке, придется выполнить M операций сравнения, и пройти N/M элементов в блоке. Следовательно максимальное число операций сравнения будет равняться **M + N / M**.

При разбиении массива из N элементов на M равных блоков, порядок ФВС будет в среднем близок к **O(√N)**.