Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Моделирование работы сортировщика файлов ОС»**

**Выполнил**:

студент группы 381803-1

Журилова А.С

**Проверил**:

доцент кафедры МОСТ, к.т.н.,

Сысоев А.В.

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc529541653)

[1. Постановка задачи 4](#_Toc529541654)

[2. Руководство пользователя 5](#_Toc529541655)

[3.1. Описание структуры программы 7](#_Toc529541656)

[3.2. Описание алгоритмов 8](#_Toc529541657)

[4. Результаты экспериментов 11](#_Toc529541658)

[Заключение 12](#_Toc529541659)

[Литература 13](#_Toc529541660)

[Приложение 14](#_Toc529541661)

# Введение

В современном мире с постоянным повышением уровня сложности задач, которые решаются посредствам программирования, растёт в первую очередь необходимость работы с большим количеством информации. Для того чтобы иметь быстрый доступ к необходимой информации существует операция под названием сортировка массивов. Эта операция является наиболее распространённой среди пользователей, так как упорядочивание информации (в виде массивов и файлов) в нужном пользователю порядке значительно уменьшает время, затраченное на поиск необходимых сведений. Существует много различных критериев сортировки массивов: по возрастанию или убыванию, по алфавиту, по дате и времени, по размеру файлов и т.д.

# Постановка задачи

Разработать прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по возрастанию/убыванию размера.

Входные данные:

* Путь до директории, в которой необходимо отсортировать содержимое.
* Метод сортировки.

Выходные данные:

* Отсортированный список имен файлов с указанием размера.
* Время сортировки.

Программа должна предоставлять пользователю возможность сменить метод сортировки и повторно формировать выходные данные.

Программа должна реализовывать диалог с пользователем посредством интерфейса, который включает:

* возможность ввода пути до заданного каталога;
* возможность выбора метода сортировки;
* возможность просмотра отсортированного списка файлов с указанием размера.

# Руководство пользователя

# Первый шаг в использовании программы – ввод пути до директории, в которой необходимо отсортировать содержимое. Важно! Вводимый пользователем путь должен выглядеть так, как показано в программе, в противном случае, исход программы неизвестен (скорее всего, программа перестанет работать).

C:\Users\Анастасия\Desktop\ЯиМП\task5\рпол01.PNG

Рис. 1. Начало работы

После ввода пути программа выведет список хранящихся в этом месте файлов(по алфавиту), в конце будет выведено количество файлов в директории.



Рис. 2. Начало списка файлов

Следующее, что следует сделать – ввести номер сортировки, которую нужно провести. В случае если номер будет введен неверно, следует ввести номер ещё раз.

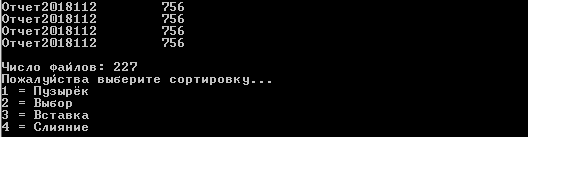


Рис. 3. Окончание списка, выбор сортировки

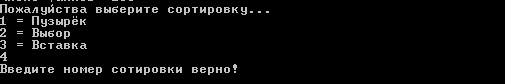


Рис. 4. Контроль номера сортировки

Следующий шаг состоит в выборе способа сортировке – по убыванию или возрастанию. В случае неправильно введенного значения, программа попросит ввести его ещё раз.



Рис. 5. Выбор способа сортировки по расположению элементов

Далее, после получения программой всех необходимых данных, будет выведен упорядоченный список файлов, согласно выбранной сортировки. После списка выведено время, затраченное на выполнение сортировки (в секундах). Следующее, что необходимо ввести – решение о продолжении работы или её окончании, в случае продолжения работы программа спросит о смене сортировки. Если значение будет введено неверно, будет предложена повторная попытка.

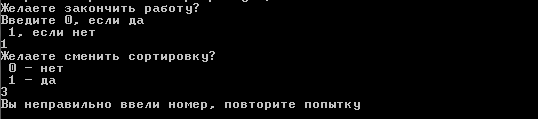


Рис. 6. Продолжение работы

В случае если вы не желаете менять сортировку, будет предложено ввести другой директорий (см. рис. 1).

Если выбран вариант со сменой сортировки, появится выбор нового способа упорядочивания.



Рис. 7. Повторный выбор сортировки

Программа заканчивает свою работу, когда в условии выхода будет выбран 0(см.рис.6).

1. **Руководство программиста**

## Описание структуры программы

Программа реализована на языке программирования С. Также создан дополнительный тип данных (структура) хранящий в себе информацию о файлах (их размер; имя, не превышающее 50 символов).

Для удобства расположения кода, были созданы несколько функций, отвечающих за сортировки файлов.

1. Функция «**compare**».

Отвечает за сравнение двух элементов.

Входные данные: Два элемента типа dlfail(структура) f и l; переменная целого типа s(отвечает за сравнения, в зависимости от выбора сортировки по убыванию или по возрастанию. Имеет возвращаемый тип.

Выходные данные:

1. в случае если (s=1 и f>l) или (s=0 и f<l)
2. в случае если (s=1 и f<l) или (s=0 и f>l)
3. Функция «**puz\_sort**»

Реализует сортировку пузырьком.

Входные данные: массив типа dlfail f[], переменная типа long size (количество элементов в массиве), переменная целого типа s (упорядочить по возрастанию или убыванию.

Выходные данные: массив является упорядоченным.

1. Функция «**vibor**»

Реализует сортировку выбором.

Входные данные: массив типа dlfail f[], переменная типа long size (количество элементов в массиве), переменная целого типа s (упорядочить по возрастанию или убыванию.

Выходные данные: массив является упорядоченным.

1. Функция «**vstavka**»

Реализует сортировку вставками.

Входные данные: массив типа dlfail f[], переменная типа long size (количество элементов в массиве), переменная целого типа s (упорядочить по возрастанию или убыванию.

Выходные данные: массив является упорядоченным.

## Описание алгоритмов

Использовались следующие виды сортировок:

* Пузырьком;
* Выбором;
* Вставками.

**Пузырёк.**

Идея метода состоит в следующем: мы проходим снизу вверх по массиву. При этом просматриваются пары соседних элементов. Если же элементы некоторой пары находятся в неправильном порядке, то меняем их местами. После нулевого прохода по массиву "вверху" оказывается самый "легкий" элемент. Далее проходы делаются до последующего сверху элемента, таким образом, каждый последующий по величине элемент поднимается на правильную позицию.

**Код сортировки пузырька.**

**Алг**. **Пузырёк** (арг. структура f(массив), длинное вещ. Size, цел. S)

{

**Начало**

длинное вещ. i, j;

структура с;

**Цикл** (для i=0; пока i<size; i++(увеличение на 1))

{

**Цикл** (для j=(size-1); пока j>i; j- - (уменьшение на 1))

{

Если ( алг. Сравнение (f(j), f(j-1), s)==1)

{

C=f(j);

f[j] = f[j-1];

f[j-1] = c;

}

}

**Конец.**

}

**Выбор.**

Метод состоит в том, чтобы «создавать» упорядоченную последовательность, присоединяя к ней один элемент за другим в правильном порядке. Таким образом строиться готовая последовательность, начиная с левого конца массива. Алгоритм состоит из n последовательных шагов, начиная от нулевого и заканчивая (n-1)-м. На i-м шаге выбираем наименьший из элементов a[i] ... a[n] и меняем его местами с a[i].

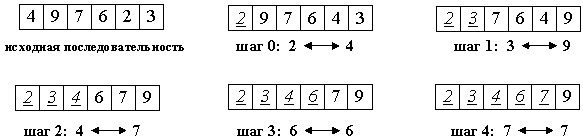


Рис. 8. Выбор при n=5

Вне зависимости от номера текущего шага i, последовательность a[0]...a[i] уже будет упорядоченной. То есть, на (n-1)-м шаге вся последовательность, кроме a[n], разумеется, оказывается отсортированной.

**Код выбора.**

**Алг**. **Выбор** (арг. структура f(массив), длинное вещ. Size, цел. S)

{

**Начало**

длинное вещ. i, j, index;

структура m;

**Цикл** (для i=0; пока i<size; i++(увеличение на 1))

{

index = i;

m = f[i];

**Цикл** (для j=(i+1); пока j<size; j++ (увеличение на 1))

{

Если ( алг. Сравнение (f(j), m, s)==1)

{

index = j;

m = f[j];

}

}

f[index] = f[i];

f[i] = m;

}

**Конец.**

}

**Вставка.**

Смысл сортировки состоит в следующем: производятся проходы по части массива, в результате чего в его начале "вырастает" отсортированная последовательность.

Однако здесь мы не можем утверждать, что в начале массива всегда последовательность a[0]...a[i] упорядочена. Поэтому по ходу алгоритма в нее будут вставляться все новые элементы.

Будем разбирать алгоритм, рассматривая его действия на i-м шаге. Последовательность к этому моменту разделена на две части: готовую a[0]...a[i] и неупорядоченную a[i+1]...a[n].

Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним.  
В зависимости от исхода сравнения, выбранный элемент либо остается на своём первоначальном месте, либо они меняются местами и процесс повторяется.

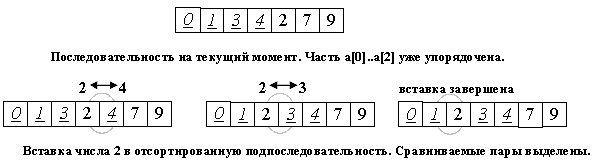


Рис. 9. Сортировка вставками

Таким образом, в процессе вставки мы "просеиваем" элемент x к началу массива, останавливаясь в случае, когда:

1. Мы нашли элемент, меньший x;
2. Достигнуто начало последовательности;

**Код сортировки вставками.**

**Алг**. **Встака** (арг. структура f(массив), длинное вещ. Size, цел. S)

{

**Начало**

длинное вещ. i, j;

структура c;

**Цикл** (для i=0; пока i<size; i++(увеличение на 1))

{

с = f[i];

**Цикл** (для j = (i-1) ; (пока j >= 0) и (( алг. Сравнение (c, f(j), s)==1); j- -(уменьшение на 1))

{

f[j + 1] = f[j];

}

f[j + 1] = c;

}

**Конец.**

}

# Результаты экспериментов

Для того чтобы выявить, какая сортировка является самой быстрой, необходимо протестировать программу на разных директориях, с различным количеством файлов. Ниже приведена таблица зависимости времени проведения сортировок, от количества сортируемых элементов (взяты средние значения).

|  |  |  |  |
| --- | --- | --- | --- |
| Количество  Элементов(шт.) | Сортировка  **Пузырьком (сек)** | Сортировка  **Выбором (сек)** | Сортировка  **Вставками (сек)** |
| 356 | 0,030000 | 0,024000 | 0,022000 |
| 530 | 0,054000 | 0,041000 | 0,035000 |
| 1 002 | 0,144000 | 0,086000 | 0,074000 |
| 2 100 | 0,537000 | 0,354000 | 0,318000 |

Таблица. 1. Результаты экспериментов

Для того чтобы результат выглядел более наглядным, итги экспериментов представлены на графике.

Диаграмма. 1. Результаты экспериментов

# Заключение

Проанализировав полученные результаты экспериментов, можно сделать вывод, что сортировка вставками работает быстрее всех из представленных трёх. Сортировка выбором немного уступает по скорости вставке. Однако разница во времени между сортировкой по убыванию и по возрастанию составляет, в большинстве случаев 0,002 секунды, для каждого из видов сортировки. К тому же время, затраченное на выполнение работы, сильно зависит от входных данных. Но, несмотря на все эти параметры, сортировка пузырьком неизбежно уступает в скорости другим двум, что делает её наименее эффективной в работе.

Таким образом, на основе проведённых экспериментов, можно выстроить порядок сортировок в порядке возрастания в зависимости от скорости:

* Пузырёк;
* Выбор;
* Вставка.

# Литература

1. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). – 2-е изд. – Москва: Вильямс, 2007. – Т. 3. – 832 с.
2. Сайт Algolist. Сортировка выбором – http://algolist.manual.ru/sort/select\_sort.php
3. Сайт Algolist. Сортировка пузырьком – http://algolist.manual.ru/sort/bubble\_sort.php
4. Сайт Algolist. Сортировка вставками - http://algolist.manual.ru/sort/insert\_sort.php

# Приложение

Код программы.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <locale.h>

#include <stdlib.h>

#include<string.h>

#include<stdio.h>

#include<time.h>

#include<io.h>

//структура для файлов

struct dlfail

{

unsigned long size;

char name[50];

};

int compare(struct dlfail f, struct dlfail l, int s)//сравнение двух элементов по возрастанию или убыванию(в зависимости от выбора в программе)

{

if (s == 1)//по возрастанию

{

if (f.size > l.size)

{

return 1;

}

else

return 0;

}

if (s == 0)//по убыванию

{

if (f.size < l.size)

{

return 1;

}

else

return 0;

}

}

//сортировка пузырьком

void puz\_sort(struct dlfail f[], long size, int s)

{

long i, j;

struct dlfail c;

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

{

for (j = (size - 1); j > i; j--)

{

if (compare(f[j], f[j - 1], s) == 1)//Если сравнение прошло успешно и требуются действия с массивом

{

c = f[j];

f[j] = f[j-1];

f[j-1] = c;

}

}

}

}

//сортировка выбором

void vibor(struct dlfail f[], long size, int s)

{

long i, j, index;

struct dlfail m;//min или max

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

{

index = i;

m = f[i];

for (j = (i + 1); j < size; j++)

{

if (compare(f[j], m, s) == 1)

{

index = j;

m = f[j];

}

}

f[index] = f[i];

f[i] = m;

}

}

//сортировка вставка

void vstavka(struct dlfail f[], long size, int s)

{

long i, j;

struct dlfail c;

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

{

c = f[i];

for (j =( i - 1 ); (j >= 0) && (compare(c, f[j], s)==1); j--)

{

f[j + 1] = f[j];

}

f[j + 1] = c;

}

}

void main()

{

setlocale(LC\_ALL, "Rus");

struct \_finddata\_t c\_file;

intptr\_t hFile;

char path[200];

struct dlfail f[5000];//Массив для файлов

long kol=0;//Число файлов в наличии

clock\_t t1, t2;

double tt;

int sort=0, prov=0, vid=-1, exit=0;

long i;

while (1!= 0)

{

if (exit == 0) {

printf("WELCOME!!!\nВведите путь до необходимого вам директория:\nпожалуйства вводите путь в таком виде: C:\\Zhurilova.a\\mp1-2018-2\\Zhurilova.a\\\* \n");

gets\_s(path, 200);

}

if ((hFile = \_findfirst(path, &c\_file)) == -1L)

printf("Здесь совсем нет файлов :((\n");

else

{

//Для начала, посчитать количество файлов директории

kol = 0;

printf("FILE SIZE\n", ' ');

printf("---- ----\n", ' ');

do {

strncpy(f[kol].name, c\_file.name, 50);

f[kol].size = c\_file.size;

printf("%-12.12s %10u\n", f[kol].name, f[kol].size);

kol++;

} while (\_findnext(hFile, &c\_file) == 0);

\_findclose(hFile);

printf("\nЧисло файлов: %d\n", kol);

printf("Пожалуйства выберите сортировку...\n1 = Пузырёк\n2 = Выбор\n3 = Вставка\n");

prov = 1;

do {

if (prov == 1)

{

scanf("%d", &sort);

prov = 0;

}

else

{

printf("Введите номер сотировки верно!\n");

prov = 1;

}

} while (sort < 1 || sort > 3);

printf("Сортировка по возрастанию = 0\nСортировка по убыванию = 1\n");

prov = 1;

do {

if (prov == 1)

{

scanf("%d", &vid);

prov = 0;

}

else

{

prov = 1;

printf("Введите способ сортировки правильно!\n");

}

} while (vid > 1 || vid < 0);

switch (sort)

{

case 1:

t1 = clock();

puz\_sort(f, kol, vid);

t2 = clock();

break;

case 2:

t1 = clock();

vibor(f, kol, vid);

t2 = clock();

break;

case 3:

t1 = clock();

vstavka(f, kol, vid);

t2 = clock();

break;

}

tt = double(t2 - t1) / CLOCKS\_PER\_SEC;

printf("FILE SIZE\n", ' ');

printf("---- ----\n", ' ');

for (i = 0; i < kol; i++) {

printf("%-12.12s %10u\n", f[i].name, f[i].size);

}

printf("Затрачено времени на сортировку %f", tt);

}

printf("\nЖелаете закончить работу?\nВведите 0, если да\n 1, если нет\n");

scanf("%d", &exit);

getchar();

if (exit == 0)

break;

printf("Желаете сменить сортировку?\n 0 - нет\n 1 - да\n");

prov = 0;

do

{

if (prov == 1)

printf("Вы неправильно ввели номер, повторите попытку\n");

else

prov = 1;

scanf\_s("%d", &exit);

} while (exit < 0 || exit > 1);

getchar();

}

}