Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского" (ННГУ)

Институт информационных технологий, математики и механики

Отчет по учебной практике

**Прототип файлового менеджера**

Выполнил: студент гр. 381806-1

Пронькин Д.А.

Проверил: Доцент кафедры МОСТ ИТММ

Кустикова В.Д.

Нижний Новгород

2018 г.

**Содержание**

[Введение 2](#_Toc533343447)

[1. Постановка задачи 3](#_Toc533343448)

[2. Руководство пользователя 4](#_Toc533343449)

[3. Руководство программиста 5](#_Toc533343450)

[3.1. Структура программы 5](#_Toc533343451)

[3.2. Описание алгоритмов 5](#_Toc533343452)

[3.3. Описание функций 8](#_Toc533343453)

[Заключение 12](#_Toc533343454)

# Введение

Каждый человек старается сделать свое рабочее место наиболее удобным и функциональным. Если его основное рабочее место-экран компьютера, то ему приходиться работать с электронными документами, которые соответственно надо копировать, перемещать, переименовывать. Чтобы повысить комфортность работы с ними или просто, так сказать, «сменить привычную обстановку» используются разнообразные файловые менеджеры.

Оглянитесь вокруг: повсюду есть менеджеры. В магазине вас встречает продавец-менеджер, на вопросы отвечает менеджер-консультант, офисом управляет менеджер.

Менеджер – руководитель или управляющий, отвечающий за определенные действия или направления. Тот, кто имеет права, использует возможности и несет ответственность: менеджер по персоналу, инвестиционный менеджер, менеджер по связям с общественностью, финансовый менеджер, банкетный менеджер, риск-менеджер, бренд-менеджер, экаунт-менеджер, менеджер по продажам. Что собой, в таком случае, представляет файловый менеджер? Файловый менеджер – программа, с которой вы работаете больше и чаще всего. Актуальность данной темы объясняется широким использованием файловых менеджеров в повседневной жизни пользователя ПК.

# Постановка задачи

В данной практической работе необходимо реализовать прототип файлового менеджера, который сортирует файлы относительно их размеров в заданном каталоге при помощи сортировок: «пузырьковая», выбором, вставками, подсчетом, быстрая сортировка и сортировка слиянием. Данная программа должна выводить список файлов, их размер и время затраченное на сортировку.

На вход поступает путь до того директории, в которой необходимо выполнить сортировку файлов, вид сортировки в виде порядкового номера.

На выходе пользователь получает отсортированный список файлов с их размерами и время за которое была выполнена сортировка.

# Руководство пользователя

* 1. Пользователь вводит название директории
  2. Из появившегося меню пользователь выбирает пункт с необходимой ему сортировкой
  3. После выполнения сортировки пользователь вводит либо 1 для продолжения исполнения программы и повторного выбора метода сортировки, либо 0 для выхода из программы.

# Руководство программиста

## Структура программы

Программа состоит из одного файла main.c.

## Описание алгоритмов

Алгоритмы сортировок:

1. «Пузырьковая» сортировка

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N – 1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде. Отсюда и название алгоритма).

Реализация

void bubblesort(namesize a[], int n)

{

int i, j;

namesize tmp;

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

{

for (j = 1; j < n - i; j++)

{

if (a[j - 1].Size > a[j].Size)

{

tmp = a[j];

a[j] = a[j - 1];

a[j - 1] = tmp;

}

}

}

}

2. Сортировка вставками

На вход алгоритма подаётся последовательность n {\displaystyle n} чисел:{\displaystyle a\_{1},a\_{2},...,a\_{n}}. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с n {\displaystyle n}элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности {\displaystyle a\_{1}^{'},a\_{2}^{'},...,a\_{n}^{'}}, чтобы выполнялось следующее соотношение

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

Реализация

void insert(namesize a[], int n)

{

int i, j, tmp;

namesize stmp;

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

{

j = i - 1;

tmp = a[i].Size;

stmp = a[i];

while ((j >= 0) && (a[j].Size > tmp))

{

a[j + 1] = a[j];

a[j] = stmp;

j--;

}

}

}

3. Сортировка выбором

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

Для реализации устойчивости алгоритма необходимо в пункте 2 минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов.

Реализация

void choose(namesize a[], int n)

{

int i, j, minidx;

namesize stmp;

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

{

minidx = i;

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

if (a[j].Size < a[minidx].Size)

{

minidx = j;

}

stmp = a[i];

a[i] = a[minidx];

a[minidx] = stmp;

}

}

4. Сортировка подсчетом

Этот вариант используется, когда на вход подается массив структур данных, который следует отсортировать по ключам (key). Нужно создать вспомогательный массив C[0..k - 1], каждый C[i] в дальнейшем будет содержать список элементов из входного массива. Затем последовательно прочитать элементы входного массива A, каждый A[i]добавить в список C[A[i].key]. В заключении пройти по массиву C, для каждого  {\displaystyle j\in \{0,...,k-1\}} в массив A последовательно записывать элементы списка C[j].

Реализация

void count(namesize a[], int n)

{

int i, b = 0;

int j;

namesize\* c = (namesize\*)malloc(K \* sizeof(namesize));

for (i = 0; i < n; i++) c[a[i].Size] = a[i];

for (j = 0; j < K; j++)

if (a[j].Size == 0)

{

a[b] = c[j];

b++;

}

}

5. Быстрая сортировка (Сортировка Хоара)

Быстрая сортировка относится к алгоритмам «разделяй и властвуй».

Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. Разбиение: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

Реализация

void quicksort(namesize a[], int n1, int n2)

{

if (n1 < n2)

{

int left = n1, right = n2;

ULONGLONG middle = a[(left + right) / 2].Size;

do

{

while (a[left].Size < middle) left++;

while (a[right].Size > middle) right--;

if (left <= right)

{

namesize tmp = a[left];

a[left] = a[right];

a[right] = tmp;

left++;

right--;

}

} while (left <= right);

quicksort(a, n1, right);

quicksort(a, left, n2);

}

}

6. Сортировка слиянием

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по неубыванию подмассива.

Тогда:

3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.  
3.3. «Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Реализация

void merge(namesize arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

namesize\* L = (namesize\*)malloc(n1 \* sizeof(namesize));

namesize\* R = (namesize\*)malloc(n2 \* sizeof(namesize));

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i].Size <= R[j].Size)

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergesort(namesize arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergesort(arr, l, m);

mergesort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

## Описание функций

void insert(namesize a[], int n)

**Назначение:** сортировка вставками

**Входные параметры:** namesize a[] - массив типа namesize, int n – длинна массива

**Выходные параметры:** отсутствуют

void bubblesort(namesize a[], int n)

**Назначение:** «пузырьковая сортировка»

**Входные параметры:** namesize a[] - массив типа namesize, int n – длинна массива

**Выходные параметры:** отсутствуют

void choose(namesize a[], int n)

**Назначение:** сортировка выбором

**Входные параметры:** namesize a[] - массив типа namesize, int n – длинна массива

**Выходные параметры:** отсутствуют

void count(namesize a[], int n)

**Назначение:** сортировка подсчетом

**Входные параметры:** namesize a[] - массив типа namesize, int n – длинна массива

**Выходные параметры:** отсутствуют

void quicksort(namesize a[], int n1, int n2)

**Назначение:** быстрая сортировка

**Входные параметры:** namesize a[] - массив типа namesize, int n1 – начало сортируемого фрагмента, int n2 – конец сортируемого фрагмента

**Выходные параметры:** отсутствуют

void merge(namesize arr[], int l, int m, int r)

**Назначение:** слияние двух отсортированных массивов

**Входные параметры:** namesize arr[] - массив типа namesize, int l– левый индекс входящего массива, int m– центральный индекс входящего массива, int r– правый индекс входящего массива

**Выходные параметры:** отсутствуют

void mergesort(namesize arr[], int l, int r)

**Назначение:** сортировка слиянием

**Входные параметры:** namesize arr[] - массив типа namesize, int l – начало сортируемого фрагмента, int r – конец сортируемого фрагмента

**Выходные параметры:** отсутствуют

int NumberOfFiles(const wchar\_t \*sDir)

**Назначение:** нахождение количества файлов в директории

**Входные параметры:** const wchar\_t \*sDir– путь к директории

**Выходные параметры:** количество файлов в директории

int ListDirectoryContents(const wchar\_t \*sDir, namesize a[], int n)

**Назначение:** формирование массива типа namesize и заполнение данного массива

**Входные параметры:** const wchar\_t \*sDir– путь к директории**,** namesize a[] – массив который необходимо заполнить, int n **-** длинна массива

**Выходные параметры:** 0 – если директория найдена, 1 – если директория не найдена

void main()

**Назначение:** функция вызова программы

**Входные параметры:** отсутствуют

**Выходные параметры:** отсутствуют

# Заключение

В данной практической работе был реализован прототип файлового менеджера, который может отсортировать файлы в заданной директории шестью разными сортировками:

1. «Пузырьковая» сортировка

2. Сортировками вставками

3. Сортировка выбором

4. Сортировка подсчетом

5. Быстрая сортировка

6. Сортировка слиянием

И вывести время затраченное на сортировку.

В процессе тестирования была построена следующая таблица (Таблица 1).

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов в каталоге | 6 | 145 | 351 |
| «Пузырьковая» сортировка | 0 | 0,001 | 0,009 |
| Сортировка вставками | 0 | 0,001 | 0,005 |
| Сортировка выбором | 0 | 0 | 0,001 |
| Сортировка подсчетом | 0,5 | - | - |
| Быстрая сортировка | 0 | 0 | 0,001 |
| Сортировка слиянием | 0 | 0,001 | 0,008 |

Таблица 1