Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы Алгоритмизации и Программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Поиск файлов с искомой строкой

БГУИР КП 1-40 01 01  16  ПЗ

Студент: гр. 381063 Кукареко А.В.

Руководитель: Николаенко Е.В.

Минск 2013

Содержание

[Введение 4](#_Toc375045111)

[1. Анализ предметной области 5](#_Toc375045112)

[1.1 Анализ существующих аналогов 5](#_Toc375045113)

[1.2 Постановка задачи 6](#_Toc375045114)

[2. Разработка программного средства 7](#_Toc375045115)

[2.1 Разработка спецификации требований 7](#_Toc375045116)

[2.2 Разработка архитектуры программного средства 8](#_Toc375045117)

[2.3 Обоснование выбора языка программирования 9](#_Toc375045118)

[2.4 Разработка алгоритма поиска файлов с искомой строкой 11](#_Toc375045119)

[3. Математическая модель 13](#_Toc375045120)

[4. Тестирование программного обеспечения 17](#_Toc375045121)

[5. Методика использования программного обеспечения 19](#_Toc375045122)

[Заключение 21](#_Toc375045123)

[Список используемой литературы 22](#_Toc375045124)

[Приложение А. Исходный код приложения 23](#_Toc375045125)

# Введение

Информация - сведения о чём-либо, независимо от формы их представления.

Слово «информация» происходит от лат.informatio, что в переводе обозначает сведение, разъяснение, ознакомление. Понятие информации рассматривалось ещё античными философами. До начала промышленной революции, определение сути информации оставалось прерогативой преимущественно философов. В XX веке вопросами теории информации стали заниматься кибернетика и информатика.

Хранение информации осуществляется с помощью её переноса на некоторые материальные носители. Семантическая информация, зафиксированная на материальном носителе для хранения, называется документом.

Хранить информацию человечество научилось очень давно. В наиболее древних формах хранения информации использовалось расположение предметов - раковин и камней на песке, узелков на верёвке. Существенным развитием этих способов явилась письменность - графическое изображение символов на камне, глине, папирусе, бумаге. Огромное значение в развитии этого направления имело изобретение книгопечатания. За свою историю человечество накопило огромный объём информации в библиотеках, архивах, периодических изданиях и других письменных документах.

В настоящее время особое значение получило хранение информации в виде последовательностей двоичных символов. Для реализации этих методов используются разнообразные запоминающие устройства. Они являются центральным звеном систем хранения информации. Кроме них в таких системах используются средства поиска информации (поисковая система), средства получения справок (информационно-справочные системы) и средства отображения информации (устройство вывода). Сформированные по назначению информации такие информационные системы образуют базы данных, банки данных и базы знаний.

Поиск информации – одна из основных задач компьютера, и поиск файлов по вхождению строки является одной из самых простейших задач поиска информации. Однако эта задача является чрезвычайно важной и полезной. Данная функция встроена в различные файловые менеджеры, а так же существует ряд утилит которые выполняют эту функцию, что существенно ускоряет работы с файлами и навигацию по документам.

# Анализ предметной области

## 1.1 Анализ существующих аналогов

Для различных операционных систем существуют различные утилиты со своими достоинствами и недостатками, ниже мы рассмотрим некоторые из них:

Total Commander - файловый менеджер с закрытым исходным кодом, работающий на платформе Microsoft Windows. Одной из его основных особенностей является расширенный поиск файлов, включая поиск текста в любых файлах, в том числе на нескольких дисках и FTP; поиск дубликатов файлов; поиск внутри архивов.

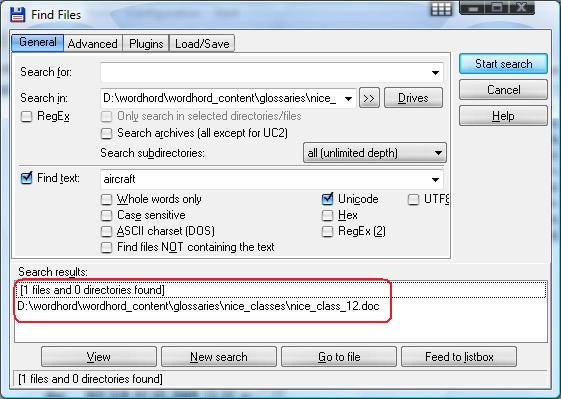


Рисунок 1.1 – Окно поиска файлов программы «Total Commander».

Достоинства:

1. Интуитивно понятный интерфейс
2. Низкая сложность
3. Огромное количество опций(фильтров)
4. Возможность использовать регулярные выражения

Недостатки:

1. Поддерживает только в OS Windows

find + grep:

1. find — утилита поиска файлов по имени, используемая в UNIX‐подобных операционных системах. Может производить поиск в одном или нескольких каталогах с использованием критериев, заданных пользователем. По умолчанию, find возвращает все файлы в рабочей директории. Более того, find позволяет применять пользователю определённые действия ко всем найденным файлам. Также поддерживаются регулярные выражения. Для поиска файлов по содержимому следует пользоваться утилитой grep.
2. grep — утилита командной строки, которая находит на вводе строки, отвечающие заданному регулярному выражению, и выводит их, если вывод не отменён специальным ключом. Название представляет собой акроним английской фразы «search globally for lines matching the regular expression, and print them» — «искать везде строки, соответствующие регулярному выражению, и выводить их».

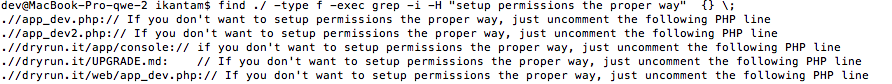


Рисунок 1.2 – Утилиты find и grep.

Достоинства:

1. Огромное количество опций(фильтров)
2. Возможность использовать регулярные выражения

Недостатки:

1. Поддерживает только в nix системами
2. 2 утилиты вместо одной
3. Требует умение работы с командной оболочкой

## 1.2 Постановка задачи

Задача курсового проекта: разработать программное средство с графическим интерфейсом для поиска файлов с искомой строкой, с возможностью выбора директории поиска, с возможность поиска по маске файлов, а также с отображением найденных файлов пользователю.

# 2. Разработка программного средства

## 2.1 Разработка спецификации требований

Программное средство предназначено для поиска файлов, которые содержат искомую строку и их вывода на экран.

Программное средство должно обеспечивать выполнение следующих функций:

1. Рекурсивный поиск файлов по маске
2. Поиск искомого текста в файле

Дополнительные требования для программного средства:

1. Графический интерфейс
2. Поля для ввода данных
3. Поле для отображения результата поиска
4. Панель управления процессом поиска

Консольный интерфейс не предусмотрен.

Входящими данными являются:

1. Корневая папка(папка в которой рекурсивно осуществляется поиск)
2. Маска файлов
3. Искомая строка

В списке результатов должны отображаться файлы которые соответствуют заданной пользователем маске, и в которых была найдена искомая стркоа.

В панель управления входят 3 конпки:

1. Выход – для завершения программы
2. Начать поиск – для того чтобы начать поиск
3. Отмена – кнопка завершает процесс поиска, но не очищает список с результатами.

В списке с результатами должно функционировать popup меню – при клике по файлу в списке результатов должно появляться меню со следующими действиями:

1. Открыть файл
2. Открыть каталог в котором находится файл

## 2.2 Разработка архитектуры программного средства

Проект состоит из нескольких логических компонентов, это графический интерфейс и поиск.

Графический интерфейс будет реализован с помощью стандартных компонентов Delphi.

Программа будет состоять только из одной формы, форма описывается компонентом TForm.

На форме будут размещены элементы для ввода пользовательских данных:

1. Корневая папка
2. Маска файлов
3. Искомая строка

для построения этих элементов будет использоваться компонент TEdit.

Элементов панели управления:

1. Найти
2. Отмена
3. Выход
4. Выбор корневой папки

будут построены с помощью компонента TButton.

Список результатов будет реализован с помощью компонента TListView.

Так же для доступа к “проводнику” – будет использован компонент TSelectDirectoryDialog, а для контекствого меню TPopupMenu.

Остальные элементы, такие как:

1. Текущий сканируемый файл
2. Описание полей

Будут реализованы с помощью компонента TLabel.

Посколько поиск и сканирвоание файлов будут мешать своевременной отрисовке графического интерфейса -во многие места программы будет вставлен код, который принудительно заставляет отрисовать текущее состояние.

Также во многие места программы будет вставлен код проверки прерывания (нажал ли пользователь кнопку отмены), для того что-бы программа своевременно реагировала на действия пользователя.

Класс TForm

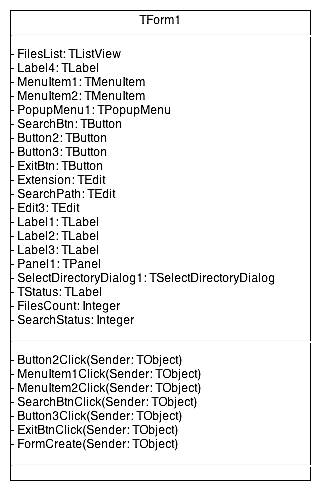


Рисунок 2.1 – Диагармма класса «TForm1».

## 2.3 Обоснование выбора языка программирования

При выборе среды реализации сравнивают программные продукты и пользуются различными средствами разработки приложений. Использование возможностей средств разработки приложений позволяет автоматизировать процесс разработки. Инструментальные средства позволяют:

1. создавать интерфейс, используя стандартные компоненты;
2. передавать управление процессам, в зависимости от состояния системы;
3. создавать оболочки для баз данных, как и сами базы данных;
4. разрабатывать более надежные программы путем обработки исключительных ситуаций возникающих при некорректной работе программы.
5. Современные средства разработки характеризуются параметрами:
6. Поддержка объектно-ориентированного стиля программирования;
7. Возможность использования CASE-технологий, как для проектирования разрабатываемой системы, так и для разработки моделей реляционных баз данных;
8. Использование визуальных компонент для наглядного проектирования интерфейса;
9. Поддержка БД.

Выше перечисленными свойствами обладают языки программирования:

-Visual Fox Pro;

-Delphi;

-Visual C++.

Каждое из этих средств содержит весь спектр современного инструментария, который был перечислен ранее. Главное отличие состоит в области использования рассматриваемых средств.

Для реализации курсового проекта был выбран Delphi. При решении поставленной задачи оптимально использовать для представления информационных материалов язык Delphi, который является языком высокого уровня и позволяет быстро и эффективно создавать приложения.

Delphi – это продукт Borland International для быстрого создания приложений. Высокопроизводительный инструмент визуального построения приложений включает в себя настоящий компилятор кода и предоставляет средства визуального программирования, несколько похожие на те, что можно обнаружить в Microsoft Visual Basic или в других инструментах визуального проектирования. В основе Delphi лежит язык Object Pascal, который является расширением объектно-ориентированного языка Pascal.

Delphi производит небольшие по размерам (до 15-30 Кбайт) высокоэффективные исполняемые модули. С другой стороны небольшие по размерам и быстро исполняемые модули означают, что требования к клиентским рабочим местам существенно снижаются – это имеет немаловажное значение и для конечных пользователей.

Преимущества Delphi по сравнению с аналогичными программными продуктами.

– быстрота разработки приложения;

– высокая производительность разработанного приложения;

– низкие требования разработанного приложения к ресурсам компьютера;

– наращиваемость за счет встраивания новых компонентов и инструментов в среду Delphi;

– возможность разработки новых компонентов и инструментов собственными средствами Delphi (существующие компоненты и инструменты доступны в исходных кодах);

– удачная проработка иерархии объектов.

Система программирования Delphi рассчитана на программирование различных приложений и предоставляет большое количество компонентов для этого.

К тому же работодателей интересует прежде всего скорость и качество создания программ, а эти характеристики может обеспечить только среда визуального проектирования, способная взять на себя значительные объемы рутинной работы по подготовке приложений, а также согласовать деятельность группы постановщиков, кодировщиков, тестеров и технических писателей. Возможности Delphi полностью отвечают подобным требованиям и подходят для создания систем любой сложности.

В качестве среды разработки использовался lazarus - свободная среда разработки программного обеспечения на языке Object Pascal для компилятора Free Pascal (часто используется сокращение FPC — свободно распространяемый компилятор языка программирования Pascal). Интегрированная среда разработки предоставляет возможность кроссплатформенной разработки приложений в Delphi-подобном окружении. На данный момент является единственным инструментом быстрой разработки приложений (RAD), позволяющим Delphi-программистам создавать приложения с графическим интерфейсом для Linux (и других не-Windows) систем.

Позволяет достаточно несложно переносить Delphi-программы с графическим интерфейсом в различные операционные системы: Linux, FreeBSD, Mac OS X, Microsoft Windows, Android[3]. Начиная с Delphi XE2 в самом Delphi имеется возможность компиляции программ для Mac OS X, а с версии XE4 и для iOS, начиная с версии XE5 и для Android.

## 2.4 Разработка алгоритма поиска файлов с искомой строкой

Алгоритм поиска файлов с искомой строкой будет состоять из двух субалгоритмов:

1. Поиск файлов
2. Поиск строки в файле

Алгоритм поиска будет осуществляться с помощью функции FindFirst(SysUtils unit) – которая предназначена для поиска файлов и папок в заданной директории соответствующих заданной маске.

Для того чтобы обеспечить гибкость алгоритму поиска файлов, функцию FindFirst обернем в свою собственную функцию, которая будет состоять из двух этапов:

1. На первом этапе функция-обертка будет искать файлы по заданной пользователем маске в директории которую задал пользователь, и отправлять их(файлы) в функцию которая будет искать вхождение искомой строки.
2. На втором этапе функция-обертка будет искать только директории(папки) в той директории которую задал пользователь, и для каждой найденной папки (за исключением себя) рекурсивно будет вызываться наша же функция-обертка но с измененным корневым путем, который будет состоять из конкатенации пути, который ввел пользователь и пути найденной папки

Таким образом этот алгоритм обеспечит файлам более высокий приоритет, чем каталогам – алгоритм сперва обработает все файлы в одной директории, а затем уже приступит к поиску и обработке файлов в дочерней директории (Рисунок 2.2).

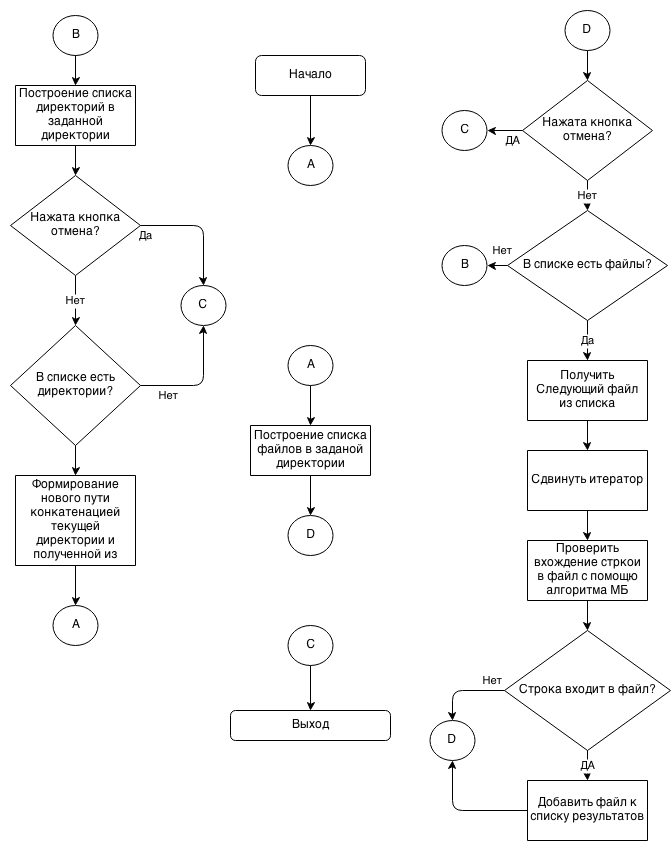


Рисунок 2.2 – схема алгоритма поиска файлов с искомой строкой.

# 3. Математическая модель

Файл можно представить в виде большой строки и тогда задача поиска строки в файле сводится к задаче поиска строки в подстроке. Для решения этой задачи уже существует несколько алгоритмов, которые мы сейчас рассмотрим:

Алгоритм прямого поиска:

Идея алгоритма:

1. I=1,
2. сравнить I-й символ массива T с первым символом массива W,
3. совпадение → сравнить вторые символы и так далее,
4. несовпадение → I:=I+1 и переход на пункт 2,

Условие окончания алгоритма:

1. подряд М сравнений удачны,

2. I+M>N, то есть слово не найдено.

Сложность алгоритма:

Худший случай. Пусть массив T→{AAA….AAAB}, длина │T│=N, образец W→{A….AB}, длина │W│=M. Очевидно, что для обнаружения совпадения в конце строки потребуется произвести порядка N\*M сравнений, то есть O(N\*M).

Недостатки алгоритма:

* + 1. высокая сложность — O(N\*M), в худшем случае – Θ((N-M+1)\*M);
    2. после несовпадения просмотр всегда начинается с первого символа образца и поэтому может включать символы T, которые ранее уже просматривались (если строка читается из вторичной памяти, то такие возвраты занимают много времени);
    3. информация о тексте T, получаемая при проверке данного сдвига S, никак не используется при проверке последующих сдвигов.

Алгоритм Д. Кнута, Д. Мориса и В. Пратта (КМП-поиск):

Алгоритм КМП-поиска фактически требует только порядка N сравнений даже в самом плохом случае.

(Символы, подвергшиеся сравнению, подчеркнуты. Рисунок 3.1)

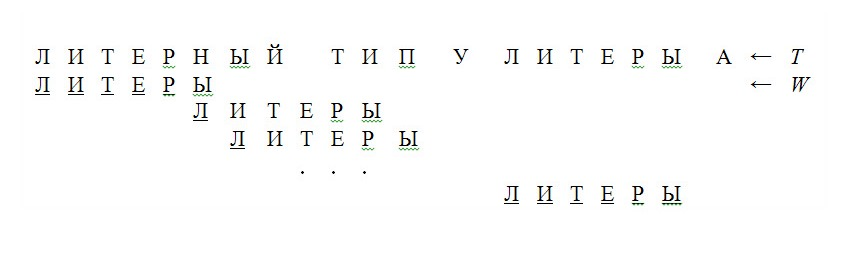


Рисунок 3.1 – Алгоритм КМП-поиск.

После частичного совпадения начальной части образа W с соответствующими символами строки Т мы фактически знаем пройденную часть строки и может «вычислить» некоторые сведения (на основе самого образа W), с помощью которых потом быстро продвинемся по тексту.

Идея КМП-поиска – при каждом несовпадении двух символов текста и образа образ сдвигается на все пройденное расстояние, так как меньшие сдвиги не могут привести к полному совпадению.

Особенности КМП-поиска:

1. требуется порядка (N+M) сравнений символов для получения результата;
2. схема КМП-поиска дает подлинный выигрыш только тогда, когда неудаче предшествовало некоторое число совпадений. Лишь в этом случае образ сдвигается более чем на единицу. К несчастью совпадения встречаются значительно реже чем несовпадения. Поэтому выигрыш от КМП-поиска в большинстве случаев текстов весьма незначителен.

Алгоритм Р. Боуера и Д. Мура (БМ-поиск):

На практике алгоритм БМ-поиска наиболее эффективен, если образец W длинный, а мощность алфавита достаточно велика.

Идея БМ-поиска – сравнение символов начинается с конца образца, а не с начала, то есть сравнение отдельных символов происходит справа налево. Затем с помощью некоторой эвристической процедуры вычисляется величина сдвига вправо s. И снова производится сравнение символов, начиная с конца образца.

Этот метод не только улучшает обработку самого плохого случая, но и даёт выигрыш в промежуточных ситуациях.

Почти всегда, кроме специально построенных примеров, БМ-поиск требует значительно меньше N сравнений. В самых же благоприятных обстоятельствах, когда последний символ образца всегда попадает на несовпадающий символ текста, число сравнений равно (N / M), в худшем же случае – О((N-M+1)\*M+ p), где p – мощность алфавита.

Алгоритм Рабина-Карпа (РК-поиск):

Пусть алфавит D={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, то есть каждый символ в алфавите есть d–ичная цифра, где d=│D│.

Пример. Пусть образец имеет вид W = 3 1 4 1 5

Вычисляем значения чисел из окна длины |W|=5 по mod q, q — простое число.

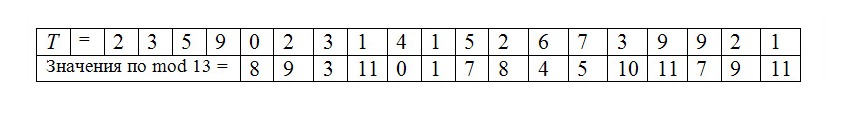


Рисунок.3.2 – Алгоритм РК-поиска.

23590(mod 13)=8, 35902(mod 13)=9, 59023(mod 13)=9, …

k1=314157(mod 13) – вхождение образца,

k2=673997(mod 13) – холостое срабатывание.

Из равенства ki= kj (mod q) не следует, что ki= kj (например, 31415=67399(mod 13), но это не значит, что 31415=67399). Если ki= kj (mod q), то ещё надо проверить, совпадают ли строки W[1…m] и T[s+1…s+m] на самом деле.

Если простое число q достаточно велико, то дополнительные затраты на анализ холостых срабатываний будут невелики.

В худшем случае время работы алгоритма РК — Θ((N-M+1)\*M), в среднем же он работает достаточно быстро – за время О(N+M).

Пример: Сколько холостых срабатываний k сделает алгоритм РК, если

q= 11, 13, 17. Пусть W={2 6}

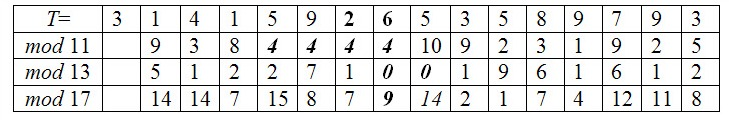


Рисунок 3.2 – Проверка холостых оборотов РК-поиска.

26 mod 11=4 → k =3 холостых срабатывания,

26 mod 13=0 → k =1 холостое срабатывание,

26 mod 17=9 → k =0 холостых срабатываний.

Очевидно, что количество холостых срабатываний k является функцией от величины простого числа q (если функция обработки образца mod q) и, в общем случае, от вида функции для обработки образца W и текста Т.

Алгоритм Боуера и Мура подходит нам больше всего, так как он рассчитан на работу с длинными строками, какими и валяются файлы.

# 4. Тестирование программного обеспечения

Для проведения тестирования разработанного программного средства были написаны тест-кейсы, тестирующие корректность работы графического интерфейса и приложения в целом.

В таблице 4.1 приведен набор тест-кейсов, на основе которых производилось тестирование. Приоритеты распределены в диапазоне от одного до трех.

Таблица 4.1 – Набор тест-кейсов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Идентификатор | Приоритет | Модуль | Условия | Действия | Ожидаемый результат | |
| 1 | 2 | 3 | 4 | 5 | 6 | |
| 1 | 1 | Интерфейс | Программа запущена. | Нажатие кнопки «Выход». | Программа должна завершиться. | |
| 2 | 1 | Интерфейс | Поиск запущен. | Нажатие кнопки «Отмена». | Остановка поиска, вывод в статус надписи – «canceled (by user)». | |
| 3 | 1 | Алгоритм поиска файлов. | В поле «место поиска» - должна быть введена существующая директория с файлами. | Нажатие кнопки «Найти». | В поле с результатами должны отобразится все файлы которые находятся в этой директории и ее поддиректориях. | |
|  | | | | | | |
| Окончание таблицы 4.1 | | | | | | |
| 1 | 2 | 3 | 4 | 5 | | 6 |
| 4 | 2 | Алгоритм поиска файлов. | В поле «место поиска» - должна быть введена существующая директория с файлами с расширением .txt, в поле «Поиск файлов» - должна быть введена маска \*.txt | Нажатие кнопки «Найти». | | В поле с результатами должны отобразиться файлы c расширением .txt. |
| 5 | 2 | Алгоритм поиска подстроки в строке. | Должен быть создан файл содержащий строку «test string», в программе в поле «место поиска» - должна быть введена директория этого файла, а в поле «С текстом» - фраза «test string» | Нажатие кнопки «Найти». | | В поле с результатами должен отобразиться файл который мы только что создали. |

# 5. Методика использования программного обеспечения

Запускаем приложение TextFileFinder.app (Рисунок 4.1).

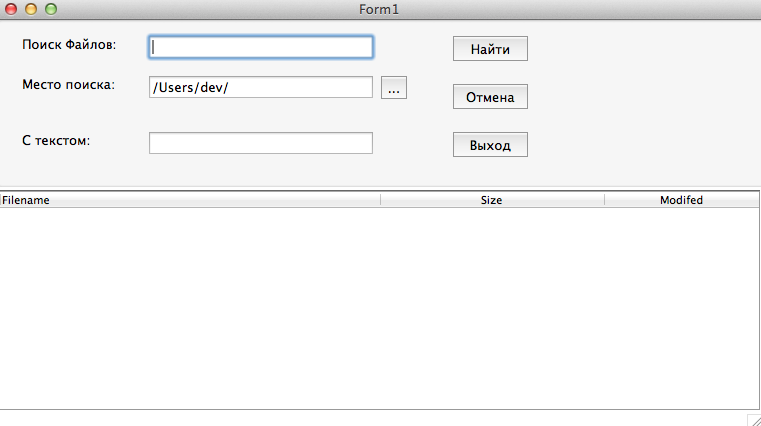


Рисунок 5.1 – Окно программы «TextFileFinder».

Для того чтобы начать поиск, предварительно выберите директорию, это можно сделать либо вписав ее в поле «Место поиска», либо нажать кнопку «...» радом с этим полем, и выбрать нужную директорию (Рисунок 5.2)

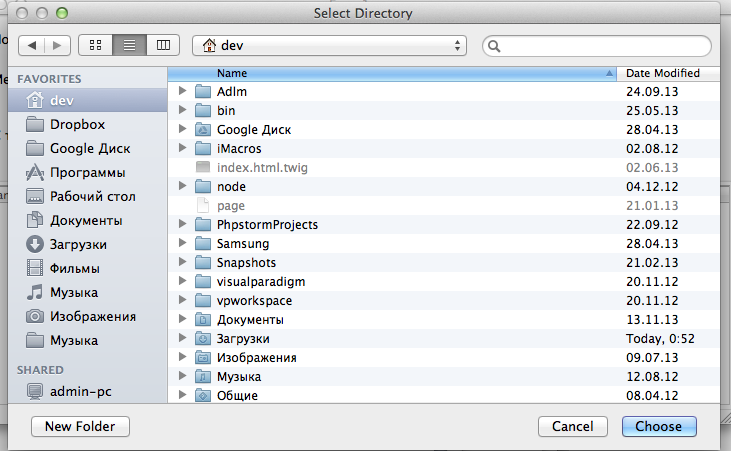


Рисунок 5.2 – Диалоговое окно выбора директории.

Обязательно введите свою искомую строку в поле «С текстом», так же для того чтобы ускорить процесс поиска, можно отсеять файлы при помощи «маски».

Для примера можно попробовать найти все текстовые файлы (\*.txt) со строкой «Заключение» в локальной директории пользователя «Документы» (Рисунок 5.3).

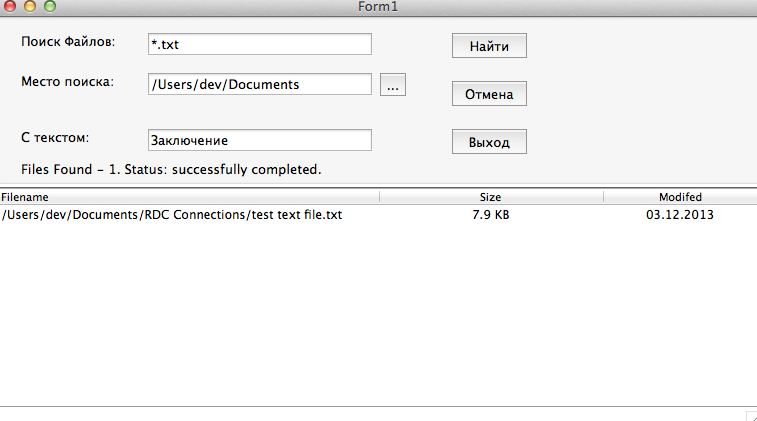


Рисунок 5.3 – Результаты поиска по маске.

Как мы видим из рисунка (Рисунок 5.3) – программа нашла файл с расширением .txt в дирктории «/Users/dev/Documents», так же программа записала в статусной строке о том что нашла 1 файл, и что поиск успешно завершен.

Кликнув правой кнопкой мыши по файлу – откроется popup меню, в котором можно вырать одно из действий (Рисунок 5.4):

1. Открыть файл в редакторе.
2. Открыть папку в которой находится этот файл и сам файл выделить.

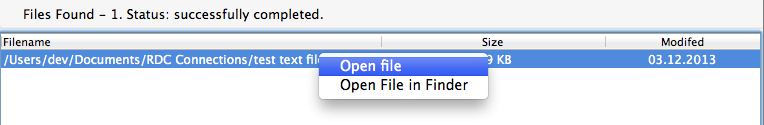


Рисунок 5.4 – Popup меню строки результата.

# Заключение

В данной курсовой работе был проведен анализ существующих аналогов поставленной задаче, были выявлены недостатки и достоинства. Так же был разработан алгоритм рекурсивного поиска файлов на hdd. Был проведен анализ различных алгоритмов поиска, таких как «алгоритм прямого поиска», «алгоритм Д. Кнута, Д. Мориса и В. Пратта», «алгоритм Р. Боуера и Д. Мура», «алгоритм Рабина-Карпа» и выбран подходящий данному контексту алгоритм. Проведенный сбор информации и ее анализ, позволил четко сформулировать задачу и разработать архитектуру приложения, что способствовало быстрому созданию программного средства «поиск файлов с искомой строкой». Разработанные тест-кейсы помогли протестировать программное средство, что непременно отразилось на качестве. В добавок ко всему, мною был еще глубже изучен язык программирования Delphi и среда разработки Lazarus.

При помощи этой программы можно искать файлы на компьютере содержащие «искомую строку», что существенно помогает в работе и улучшает производительность человека.

Достоинства данного программного средства:

1. Маленький размер
2. Не требовательна к системе
3. Кроссплатформенна (поддерживает все платформы которые поддерживает Lazarus)
4. Открытый исходный код.

# Список используемой литературы

1. Культин Н.Б. Delphi в задачах и примерах. – СПб.: БХВ-Петербург, 2003.
2. Галисеев Г.В. Программирование в среде Delphi 7. Самоучитель. – М.: Издательский дом «Вильямс», 2003.
3. Митчелл К. Керман Программирование и отладка в Delphi: Учебный курс: М.; СПб.; Киев, 2003.
4. Архангельский А.Я. Object Pascal в Delphi. – СПб.: Бином, 2002.
5. Фленов, М.Е. Библия Delphi. – Спб.:Питер, 2004. – 880с.;
6. Гофман, В. Э. Delphi 7 / В. Э. Гофман, А. Д. Хомоненко. –СПб. : Санкт-Петербург, 2003.
7. Delphi. Программирование на языке высокого уровня: Учебник/ В.В. Фаронов. - СПб : Питер, 2004 г.
8. Грибанов В.П.,Калмыкова О.В.,Сорока Р.И. «Основы алгоритмизации и программирование», М., ММИЭИФиП.-2003.-151с.
9. Иванова Г.С. Основы программирования: Учебник для вузов. - М.: Изд-во МГТУ им. Н.Э.Баумана,2001.-392с.
10. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М.. МЦНМО, 2004.
11. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование: Учеб для вузов/ под ред. Г.С.Ивановой. -М. :Изд-во МГТУ им.Н.Э.Баумана, 2004.

# Приложение А. Исходный код приложения

// project1.lpr

program project1;

{$mode objfpc}{$H+}

uses

{$IFDEF UNIX}{$IFDEF UseCThreads}

cthreads,

{$ENDIF}{$ENDIF}

Interfaces, // this includes the LCL widgetset

Forms, runtimetypeinfocontrols, Unit1, Unit2

{ you can add units after this };

{$R \*.res}

begin

RequireDerivedFormResource := True;

Application.Initialize;

Application.CreateForm(TForm1, Form1);

Application.Run;

end.

//unit1.pas

unit Unit1;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, FileUtil, RTTIGrids, Forms, Controls, Graphics, Dialogs,

StdCtrls, ExtCtrls, Grids, ComCtrls, ActnList, Menus, process;

type

{ TForm1 }

TForm1 = class(TForm)

FilesList: TListView;

Label4: TLabel;

MenuItem1: TMenuItem;

MenuItem2: TMenuItem;

PopupMenu1: TPopupMenu;

SearchBtn: TButton;

Button2: TButton;

Button3: TButton;

ExitBtn: TButton;

Extension: TEdit;

SearchPath: TEdit;

Edit3: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Panel1: TPanel;

SelectDirectoryDialog1: TSelectDirectoryDialog;

TStatus: TLabel;

procedure Button2Click(Sender: TObject);

procedure MenuItem1Click(Sender: TObject);

procedure MenuItem2Click(Sender: TObject);

procedure SearchBtnClick(Sender: TObject);

procedure Button3Click(Sender: TObject);

procedure ExitBtnClick(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ private declarations }

public

{ public declarations }

FilesCount: Integer;

SearchStatus: Integer;

end;

function NormalizePath(Path: string): string;

function NormalizeExpr(Expr: string): string;

procedure ClearResult();

procedure ListFileDir(Path: string; Expr: string);

procedure AddFileToList(SR: TSearchRec; Path: string);

procedure SetTextStatus(Text: string);

function FormatByteSize(const bytes: Longint): string;

procedure CheckFile(SR: TSearchRec; Path: string);

procedure LockInterface();

procedure UnlockInterface();

procedure StartSearch();

procedure WriteEndSearchStatus();

procedure DoProcess(Command: string);

procedure DoOpenFile(PathToFile: string);

procedure DoOpenFileInFinder(PathToFile: string);

function isFileCintains(const PathToFile: string; const needle: string) : Boolean;

function PosInFile(const PathToFile:string; const needle:string):integer;

var

Form1: TForm1;

implementation

{$R \*.lfm}

{ TForm1 }

procedure TForm1.FormCreate(Sender: TObject);

begin

Form1.SearchPath.Text:= GetUserDir();

end;

procedure TForm1.Button3Click(Sender: TObject);

begin

SelectDirectoryDialog1.InitialDir := Form1.SearchPath.Text;

if SelectDirectoryDialog1.Execute then

begin

Form1.SearchPath.Text := SelectDirectoryDialog1.FileName;

end;

end;

procedure TForm1.SearchBtnClick(Sender: TObject);

begin

StartSearch();

end;

procedure TForm1.Button2Click(Sender: TObject);

begin

Form1.SearchStatus := 2;

end;

procedure TForm1.MenuItem1Click(Sender: TObject);

begin

DoOpenFile(Form1.FilesList.Selected.Caption);

end;

procedure TForm1.MenuItem2Click(Sender: TObject);

begin

DoOpenFileInFinder(Form1.FilesList.Selected.Caption);

end;

procedure TForm1.ExitBtnClick(Sender: TObject);

begin

Form1.Close;

end;

//Add to path

function NormalizePath(Path: string): string;

begin

NormalizePath := IncludeTrailingBackslash(Path);

end;

function NormalizeExpr(Expr: string): string;

begin

if Length(Expr) < 1 then

begin

Expr := '\*.\*';

end;

NormalizeExpr := Expr;

end;

procedure ClearResult();

begin

Form1.TStatus.Caption := '';

Form1.FilesList.Clear();

Form1.FilesCount := 0;

Form1.SearchStatus := 0;

end;

procedure ListFileDir(Path: string; Expr: string);

var

SR: TSearchRec;

begin

Path := NormalizePath(Path);

Expr := NormalizeExpr(Expr);

// find files

if FindFirst(Path + Expr, faAnyFile, SR) = 0 then

begin

repeat

if (Application.Terminated Or (Form1.SearchStatus = 2)) then

Break;

if ( (SR.Attr and faDirectory) <> 0) then Continue;

SetTextStatus('Search: ' + Path + SR.Name);

CheckFile(SR, Path);

until FindNext(SR) <> 0;

FindClose(SR);

end;

// get directories list

if FindFirst(Path + '\*', faDirectory, SR) = 0 then

begin

repeat

if (Application.Terminated Or (Form1.SearchStatus = 2)) then

Break;

if ( ( (SR.Attr and faDirectory) = 0) Or (SR.Name = '.') OR (SR.Name = '..')) then Continue;

ListFileDir(Path + SR.Name, Expr);

until FindNext(SR) <> 0;

FindClose(SR);

end;

end;

procedure AddFileToList(SR: TSearchRec; Path: string);

begin

with Form1.FilesList.Items.Add do begin

Caption:= Path + SR.Name;

SubItems.Add( FormatByteSize(SR.Size) );

SubItems.Add( FormatDateTime('dd.mm.yyyy', FileDateToDateTime(SR.Time) ) );

//SubItems.Add( IntToStr(SR.Time) );

end;

end;

procedure SetTextStatus(Text: string);

begin

Form1.TStatus.Caption := Text;

Application.ProcessMessages;

end;

//Format file byte size

function FormatByteSize(const bytes: Longint): string;

const

B = 1; //byte

KB = 1024 \* B; //kilobyte

MB = 1024 \* KB; //megabyte

GB = 1024 \* MB; //gigabyte

begin

if bytes > GB then

result := FormatFloat('#.## GB', bytes / GB)

else

if bytes > MB then

result := FormatFloat('#.## MB', bytes / MB)

else

if bytes > KB then

result := FormatFloat('#.## KB', bytes / KB)

else

result := FormatFloat('#.## bytes', bytes) ;

end;

procedure CheckFile(SR: TSearchRec; Path: string);

begin

if ( isFileCintains(Path+SR.Name, Form1.Edit3.Text) ) then

begin

Form1.FilesCount := Form1.FilesCount + 1;

AddFileToList(SR, Path);

end;

end;

procedure LockInterface();

begin

Form1.Extension.Enabled := False;

Form1.SearchPath.Enabled:= False;

Form1.Edit3.Enabled:= False;

Form1.Button3.Enabled:= False;

Form1.SearchBtn.Enabled:= False;

Application.ProcessMessages;

end;

procedure UnlockInterface();

begin

Form1.Extension.Enabled := True;

Form1.SearchPath.Enabled:= True;

Form1.Edit3.Enabled:= True;

Form1.Button3.Enabled:= True;

Form1.SearchBtn.Enabled:= True;

Application.ProcessMessages;

end;

procedure WriteEndSearchStatus();

var

Status: string;

begin

Status := 'Files Found - ' + IntToStr(Form1.FilesCount) + '. Status: ';

if (Form1.SearchStatus = 1) then

begin

Status := Status + 'successfully completed.';

end

else

begin

Status := Status + 'canceled (by user).';

end;

SetTextStatus(Status);

end;

procedure StartSearch();

begin

ClearResult();

Form1.SearchStatus := 1;

LockInterface();

ListFileDir(Form1.SearchPath.Text, Form1.Extension.Text);

WriteEndSearchStatus();

UnlockInterface();

end;

procedure DoProcess(Command: string);

Var

Proc : TProcess;

Begin

Proc := TProcess.Create(nil);

try

Proc.CommandLine := Command;

PRoc.Options := Proc.Options + [poWaitOnExit];

PRoc.Execute;

finally

Proc.free;

end;

End;

procedure DoOpenFile(PathToFile: string);

Begin

DoProcess('open -t '+PathToFile);

End;

procedure DoOpenFileInFinder(PathToFile: string);

Begin

DoProcess('open -R '+PathToFile);

End;

function isFileCintains(const PathToFile: string; const needle: string) : Boolean;

begin

Result := False;

if ( (Length(needle) = 0) Or (PosInFile(PathToFile, needle) <> -1) ) then

begin

Result := True;

end;

end;

//http://www.delphifaq.com/faq/delphi/strings/f86.shtml

function PosInFile(const PathToFile:string; const needle:string):integer;

var

Buffer : array [0..1023] of char;

BufPtr,BufEnd:integer;

F:File;

Index : integer;

Increment : integer;

c : char;

function NextChar : char;

begin

if BufPtr>=BufEnd then

begin

BlockRead(F,Buffer,1024,BufEnd);

BufPtr := 0;

Application.ProcessMessages;

end;

Result := Buffer[BufPtr];

Inc(BufPtr);

end;

begin

Result := -1;

AssignFile(F,PathToFile);

Reset(F,1);

BufPtr:=0;

BufEnd:=0;

Index := 0;

Increment := 1;

repeat

c:=NextChar;

if c=needle[Increment] then

Inc(Increment)

else

begin

Inc(Index,Increment);

Increment := 1;

end;

if Increment=(Length(needle)+1) then

begin

Result := Index;

break;

end;

until BufEnd = 0;

CloseFile(F);

end;

end.

//unit1.lfm

object Form1: TForm1

Left = 269

Height = 409

Top = 112

Width = 765

Caption = 'Form1'

ClientHeight = 409

ClientWidth = 765

OnCreate = FormCreate

LCLVersion = '1.0.12.0'

object Panel1: TPanel

Left = 0

Height = 168

Top = 0

Width = 771

Anchors = [akTop, akLeft, akRight]

BidiMode = bdRightToLeft

ClientHeight = 168

ClientWidth = 771

ParentBidiMode = False

PopupMenu = PopupMenu1

TabOrder = 0

object Label1: TLabel

Left = 24

Height = 17

Top = 16

Width = 96

Caption = 'Поиск Файлов:'

ParentColor = False

end

object Extension: TEdit

Left = 152

Height = 22

Top = 16

Width = 224

TabOrder = 0

end

object Label2: TLabel

Left = 24

Height = 17

Top = 56

Width = 94

Caption = 'Место поиска:'

ParentColor = False

end

object SearchPath: TEdit

Left = 152

Height = 22

Top = 56

Width = 224

TabOrder = 1

end

object Label3: TLabel

Left = 24

Height = 17

Top = 112

Width = 70

Caption = 'С текстом:'

ParentColor = False

end

object Edit3: TEdit

Left = 152

Height = 22

Top = 112

Width = 224

TabOrder = 2

end

object SearchBtn: TButton

Left = 456

Height = 25

Top = 16

Width = 75

Cancel = True

Caption = 'Найти'

OnClick = SearchBtnClick

TabOrder = 3

end

object Button2: TButton

Left = 456

Height = 25

Top = 64

Width = 75

Caption = 'Отмена'

OnClick = Button2Click

TabOrder = 4

end

object Button3: TButton

Left = 384

Height = 23

Top = 56

Width = 26

Caption = '...'

OnClick = Button3Click

TabOrder = 5

end

object ExitBtn: TButton

Left = 456

Height = 25

Top = 112

Width = 75

Caption = 'Выход'

OnClick = ExitBtnClick

TabOrder = 6

end

object TStatus: TLabel

Left = 24

Height = 1

Top = 144

Width = 1

ParentColor = False

end

end

object FilesList: TListView

Left = 0

Height = 224

Top = 168

Width = 765

Anchors = [akTop, akLeft, akRight, akBottom]

AutoSort = False

Columns = <

item

Caption = 'Filename'

Width = 380

end

item

Alignment = taCenter

AutoSize = True

Caption = 'Size'

Width = 224

end

item

Alignment = taCenter

Caption = 'Modifed'

Width = 155

end>

ParentShowHint = False

PopupMenu = PopupMenu1

RowSelect = True

ScrollBars = ssAutoBoth

SortType = stBoth

TabOrder = 1

ToolTips = False

ViewStyle = vsReport

end

object SelectDirectoryDialog1: TSelectDirectoryDialog

left = 632

top = 64

end

object PopupMenu1: TPopupMenu

left = 632

top = 8

object MenuItem1: TMenuItem

Caption = 'Open file'

OnClick = MenuItem1Click

end

object MenuItem2: TMenuItem

Caption = 'Open File in Finder'

OnClick = MenuItem2Click

end

end

end