# Лабораторная работа №3: индексированный поиск

## Цели

1. Отработать на практике реализацию эффективного алгоритма и структур данных.

## Задание

Написать программу, выполняющую поиск текста в текстовых файлах. Основное требование: поиск не должен выполняться полным перебором. Необходимо реализовать так называемый «индекс» (структуру данных для быстрого поиска) и искать по индексу. Формат индекса и способ его хранения студент выбирает самостоятельно (ниже есть описание распространённого индекса – так называемый «инвертированный индекс»). Допускается хранить индекс в оперативной памяти, создавая его каждый раз при запуске программы. Не допускается использование любых готовых программных библиотек по реализации индексированного поиска.

Программа должна быть реализована в виде консольного приложения, которое выполняет следующие действия:

1. При запуске выполняет индексацию файлов в некотором каталоге.
2. Запрашивает у пользователя строку поиска.
3. Выводит результаты поиска по индексу.

В рамках данной работы предполагается, что индексируемые файлы это простые текстовые файлы в кодировке, определяемой студентом (ASCII, UTF8, UTF16 или любом другом). Строка поиска, вводимая пользователем, представляет собой набор слов, разделённых символами, не образующими слово (например, пробелами). Считать «словом» в строке запроса и в файле непрерывную последовательность букв или чисел. Программа должна выводить время, затраченное на индексацию и поиск.

Пример работы программы:

1. При запуске программа индексирует содержимое файлов в некотором каталоге (можно вводить с клавиатуры или прописать путь в программе). Выводится время индексации.
2. Пользователь вводит: « иван васильевич »
3. Программа осуществляет поиск по индексу и выводит список файлов, в которых найдены указанные слова:

32 ms

book1.txt

ivan\_vas\_men\_professiy.txt

Поиск должен быть реализован в виде отдельного класса, который принимает в качестве параметра путь к индексируемым файлам, а также содержит методы для индексации и поиска. Пример класса:

class FileSearch {

public FileSearch ( string folder ) { … }

public void BuildIndex ( ) { … }

public IEnumerable<SearchResult> Find ( string phrase ) { … }

}

Выполнение перечисленных выше требований оценивается как «удовлетворительно». Для получения более высоких оценок необходимо дополнительно:

* Для получения оценки «Хорошо» выполнить все следующие требования:
  + Программа должна занимать менее 400 строк кода. Считаются только строки с кодом, но не с комментариями или пустые. Вычислять размер кода можно встроенным средством Visual Studio, если оно доступно.
* Для получения оценки «Отлично» необходимо к требованиям оценки «хорошо» выполнить следующее требование:
  + Программа должна выполнять нечёткий поиск. То есть должна находить слова, которые похожи на то, что ввёл пользователь, но не являются точно такими же. Как минимум, для этого необходимо искать без учёта регистра символов и трактовать похожие буквы как одинаковые. А именно, одинаковыми могут считаться: русская «е» и латинская «e», русская «о» и латинская «o», прочие похожие буквы разных алфавитов и цифры (полный набор заменяемых букв остаётся на усмотрение студента).
  + Выводить фрагмент текста, в котором найдены искомые слова. Достаточно показать фрагмент текста, в котором есть хотя бы одно из искомых слов.
  + Выполнять ранжирование результатов поиска: сортировать результаты так, чтобы первыми отображались наиболее близкие к тому, что искал пользователь. Конкретный алгоритм ранжирование остаётся на усмотрение студента.

Пример работы программы, выполненной на отлично:

* Пользователь вводит « иван васильевич »

Система выводит:

32 ms

ivan\_vas\_men.txt: …мнату вошёл Иван Васильевич, сел и за…

book1.txt: … лежал ивАн на печи 3 года, да так ду …

## Демонстрация работы

1. Рассказать коротко используемые алгоритмы индексирования, поиска и ранжирования (если реализовано).
2. Продемонстрировать поиск по файлам.
3. Увеличить количество файлов в папке в три раза, продемонстрировать, что время поиска достаточно невысокое (не дольше секунды на десять мегабайт текста), и при этом осталось гораздо меньше времени индексации.

## Вспомогательные материалы

### Индексы

Основным индексом, используемым для поиска текста, является так называемый «инвертированный индекс». Он представляет собой структуру данных, в которой каждому слову ставится в соответствие список файлов, в которых упоминается это слово. Построение индекса заключается в выделении слов в файле и построении списка вида (файл,позиция в файле) для каждого из слов. Пример индекса:

иван -> (файл1,534), (файл1,987), (файл2,7656)

васильевич -> (файл1,541), (файл2,4567)

Поиск по индексу сводится к получению всех файлов, в которых упоминаются слова из запроса, их объединению (в простейшем случае – пересечению) и ранжированию. Для реализации нечёткого поиска в простейшем случае можно «нормализовать» индексируемые и искомые слова – привести их к одному регистру, заменить символы и т.д.

### Ранжирование результатов поиска

Реализовать ранжирование в общем случае сложно, так как нельзя заранее сказать, что интересно пользователю. В простейшем случае ранжировать можно по расстоянию (в символах, байтах, словах – в чём угодно) между найденными в тексте словами. Таким образом, результаты, в которых искомые слова находятся рядом, будут показаны раньше тех, в которых искомые строки далеко.

### Полезные методы и структуры .NET

* Проверка, что символ является буквой или цифрой: сhar.IsLetterOrDigit
* Каталог, из которого запущена программа: System.IO.Path.GetDirectoryName ( System.Reflection.Assembly.GetExecutingAssembly ( ).Location )
* Словарь, отображающий строки на списки структур Position (можно использовать в качестве индекса): System.Collections.Generic. Dictionary<string, List<Position>>
* Полный список файлов в папке: System.IO.Directory.GetFiles ( folder )
* Класс для манипуляции строками (можно использовать как буфер для добавления символов с целью выделения строк): System.Text.StringBuilder
* Чтение текстовых файлов (кодировка в данном случае определяется автоматически, но можно указать её явно):

using ( var sr = new StreamReader ( file, true ) ) {

int charCount = sr.ReadBlock ( buf, 0, buf.Length );

* Замена символа в строке: string.Replace