Практическая работа №2. Обработка текстовых файлов с помощью LINQ-запросов

Задачи:

1. нахождение самого длинного слова по множеству файлов;
2. нахождение 10 самых часто встречающихся слов;
3. нахождение статистики по длине слов (получить словарь, в котором ключ – длина слова, а значение – количество слов такой длины; словарь упорядочен по убыванию длины слов);
4. * нахождение файла с наибольшим количеством специфичных слов, которые не встречаются в других файлах.

Необходимо реализовать задачи без LINQ, используя циклы, списки, словари, и с применением технологии LINQ.

Дополнительные задачи:

- один из запросов написать в формате query-syntax(fromfinfiles ..);
- оценить быстродействие алгоритмов с LINQи без LINQ;
- выполнить «автоматическое» распараллеливание LINQ-запросов и оценить быстродействие с последовательными алгоритмами.

Рекомендации

Структурирование кода

Для структурирования кода необходимо, чтобы все вычислительные функции были отделены от взаимодействия с консолью. Для этого расчёт статистики необходимо выделить в отдельные методы, которые возвращают результат. Например, метод для нахождения 10 самых часто встречающихся слов может иметь следующую сигнатуру (модификаторы доступа опущены, так как зависят от контекста метода):

```
List<string> GetTopWords(string[] files, int K = 10);
```

По заданию необходимо реализовать разные способы решения одних их тех же задач (без применения технологии LINQ, с применением LINQ, с внедрением распараллеливания). Для этого рекомендуется задействовать паттерн проектирования «*Стратегия*», основная идея которого заключается в выделении интерфейса (например, `ITextSolver`), который определяет решаемые вычислительные задачи. Реализации этих задач должны быть введены в отдельных классах, например, `SequentialSolver`, `LinqSolver`, `ParallelLinqSolver`. Каждый класс представляет собой реализацию решения задач обработки текстовых файлов по определенной общей *стратегии*. Решатели также могут иметь общий базовый класс, если какая-то функциональность и

какие-то данные должны присутствовать в каждом специализированном решателе (например, массив разделителей).

Код пользовательского интерфейса работает с решателем исключительно через интерфейс.

Чтение и обработка текстовых файлов

Для получения набора файлов, которые содержатся в заданной папке, можно воспользоваться статическими методами класса `Directory` пространства имен `System.IO`. Методы позволяют указать шаблон для файлов, а также настроить обработку вложенных папок при необходимости. В следующей строке получаем массив имен текстовых файлов в указанной папке:

```
string[] files =
    Directory.EnumerateFiles("E:\\Books","*.txt").ToArray();
```

Чтение содержимого файла можно осуществить разными способами. Один из вариантов – применение статических методов класса File. В этом случае вся работа с файловыми потоками проводится внутри метода, мы получаем результат в виде одной строки или набора строк. Методы позволяют настроить кодовую страницу при необходимости. В следующей строке читаем содержимое файла на русском языке, результат возвращается в виде массива строк:

```
string[] lines = File.ReadAllLines(sFile, Encoding.GetPage(1251));
```

Для разбиения строчек на слова можно воспользоваться методом `Split`, указав необходимые разделители («white»-символы, знаки пунктуации):

```
    string[] words = sLine.Split(delimeters);
```

Оценка быстродействия алгоритмов

Для замера времени можно использовать объект `Stopwatch`и его методы `Start`, `Stop`, `Restart`. Например, следующим образом:

```
    Stopwatch sw = new Stopwatch();
    sw.Start();
    var words = solver.GetTopWords(files);
    sw.Stop();
```

Для достоверности результатов рекомендуется проводить несколько экспериментов и усреднять полученное время.