**Практика «Парсер предложений»**

[Скачайте проект TextAnalysis](https://ulearn.me/Exercise/StudentZip?courseId=BasicProgramming&slideId=d41a27ad-a377-4fbd-ba1c-1bd761c69dd1)

В этом задании нужно реализовать метод в классе SentencesParserTask. Метод должен делать следующее:

1. Разделять текст на предложения, а предложения на слова.

a. Считайте, что слова состоят только из букв (используйте метод char.IsLetter) или символа апострофа ' и отделены друг от друга любыми другими символами.

b. Предложения состоят из слов и отделены друг от друга одним из следующих символов .!?;:()

1. Приводить символы каждого слова в нижний регистр.
2. Пропускать предложения, в которых не оказалось слов.

Метод должен возвращать список предложений, где каждое предложение — это список из одного или более слов в нижнем регистре.

// Вставьте сюда финальное содержимое файла SentencesParserTask.cs

**Содержимое файла Program.cs**

using System;

using System.Collections.Generic;

using System.IO;

using NUnitLite;

namespace TextAnalysis

{

internal static class Program

{

public static void Main(string[] args)

{

// Запуск автоматических тестов. Ниже список тестовых наборов, который нужно запустить.

// Закомментируйте тесты на те задачи, к которым ещё не приступали, чтобы они не мешались в консоли.

// Все непрошедшие тесты

var testsToRun = new string[]

{

"TextAnalysis.SentencesParser\_Tests",

"TextAnalysis.FrequencyAnalysis\_Tests",

"TextAnalysis.TextGenerator\_Tests",

};

new AutoRun().Execute(new[]

{

"--stoponerror", // Останавливать после первого же непрошедшего теста. Закомментируйте, чтобы увидеть все падающие тесты

"--noresult",

"--test=" + string.Join(",", testsToRun)

});

var text = File.ReadAllText("HarryPotterText.txt");

var sentences = SentencesParserTask.ParseSentences(text);

var frequency = FrequencyAnalysisTask.GetMostFrequentNextWords(sentences);

//Расскомментируйте этот блок, если хотите выполнить последнюю задачу до первых двух.

/\*

frequency = new Dictionary<string, string>

{

{"harry", "potter"},

{"potter", "boy" },

{"boy", "who" },

{"who", "likes" },

{"boy who", "survived" },

{"survived", "attack" },

{"he", "likes" },

{"likes", "harry" },

{"ron", "likes" },

{"wizard", "harry" },

};

\*/

while (true)

{

Console.Write("Введите первое слово (например, harry): ");

var beginning = Console.ReadLine();

if (string.IsNullOrEmpty(beginning)) return;

var phrase = TextGeneratorTask.ContinuePhrase(frequency, beginning.ToLower(), 10);

Console.WriteLine(phrase);

}

}

}

}

**Содержимое файла FrequencyAnalysis\_Tests.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using NUnit.Framework;

namespace TextAnalysis

{

[TestFixture]

public class FrequencyAnalysis\_Tests

{

[Test]

[Order(00)]

public void ReturnEmptyDictionary\_OnEmptyText()

{

var text = "";

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>();

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(01)]

public void ReturnEmptyDictionary\_OnTextWithOneSentenceWithOneWord()

{

var text = "abc";

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>();

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(02)]

public void ReturnCorrectResult\_OnTextWithOneSentenceWithTwoWords()

{

var text = "x y";

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>

{

{"x", "y"}

};

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(03)]

public void ReturnCorrectResult\_OnTextWithOneSentenceWithMultipleWords()

{

var text = "x y z";

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>

{

{"x", "y"},

{"y", "z"},

{"x y", "z"}

};

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(04)]

public void ReturnCorrectResult\_OnTextWithTwoSentencesWithOneWord()

{

var text = "x.y";

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>();

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(05)]

public void ReturnResult\_WithMostFrequentBigrams([Values("x y. x z. x y.", "x z. x y. x y", "x y. x y.", "x y")]

string text)

{

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>

{

{"x", "y"}

};

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(06)]

public void ReturnResult\_WithLexicographicallyFirstBigram\_IfBigramsHaveSameFrequency(

[Values("x y. x z.", "x z. x y.", "x y. x yy.", "x yy. x y")]

string text)

{

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>

{

{"x", "y"}

};

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(50)]

public void IgnoreSentencesWithSingleWord([Values("x. ax. y. z")] string text)

{

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>();

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

[Test]

[Order(60)]

public void ReturnPairForEveryBigram([Values("x y. y z.", "y z. x y.")] string text)

{

var parsedText = ParseText(text);

var expected = new Dictionary<string, string>

{

{"x", "y"},

{"y", "z"}

};

var actual = FrequencyAnalysisTask.GetMostFrequentNextWords(parsedText);

AssertResult(expected, actual, text);

}

// Упрощённый парсинг текста

public List<List<string>> ParseText(string text)

{

return text.Split('.')

.Select(sentence => sentence.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries).ToList())

.ToList();

}

public static void AssertResult(

Dictionary<string, string> expected,

Dictionary<string, string> actual,

string text)

{

foreach (var key in expected.Keys)

{

if (!actual.ContainsKey(key))

Assert.Fail($"Input text: [{text}]\nMissing expected key [{key}] in dictionary");

Assert.AreEqual(expected[key], actual[key], $"Input text: [{text}]\nWrong value for key [{key}]");

}

foreach (var key in actual.Keys)

if (!expected.ContainsKey(key))

Assert.Fail($"Input text: [{text}]\nKey [{key}] should not be in dictionary");

}

}

}

**Содержимое файла FrequencyAnalysisTask.cs**

using System.Collections.Generic;

namespace TextAnalysis

{

static class FrequencyAnalysisTask

{

public static Dictionary<string, string> GetMostFrequentNextWords(List<List<string>> text)

{

var result = new Dictionary<string, string>();

//...

return result;

}

}

}

**Содержимое файла SentencesParser\_Tests.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

using NUnit.Framework;

namespace TextAnalysis

{

[TestFixture]

public class SentencesParser\_Tests

{

[Test]

[Order(00)]

public void ReturnCorrectResult\_OnTextWithOneSentenceWithOneWord()

{

var text = "abc";

var expected = new List<List<string>> { new List<string> { "abc" } };

var actual = SentencesParserTask.ParseSentences(text);

AssertAllSentencesEqual(expected, actual, text);

}

[Test]

[Order(10)]

public void ReturnCorrectResult\_OnTextWithOneSentenceWithTwoWords()

{

var text = "b, c";

var expected = new List<List<string>> { new List<string> { "b", "c" } };

var actual = SentencesParserTask.ParseSentences(text);

AssertAllSentencesEqual(expected, actual, text);

}

[Test]

[Order(20)]

public void ReturnCorrectResult\_OnTextWithOneSentence\_WithWordContainingApostrophe()

{

var text = "it's";

var expected = new List<List<string>> { new List<string> { "it's" } };

var actual = SentencesParserTask.ParseSentences(text);

AssertAllSentencesEqual(expected, actual, text);

}

[Test]

[Order(30)]

public void CorrectlyParse\_SentenceDelimiters()

{

var text = "a.b!c?d:e;f(g)h;i";

var expected = new List<List<string>>

{

new List<string> {"a"},

new List<string> {"b"},

new List<string> {"c"},

new List<string> {"d"},

new List<string> {"e"},

new List<string> {"f"},

new List<string> {"g"},

new List<string> {"h"},

new List<string> {"i"}

};

var actual = SentencesParserTask.ParseSentences(text);

AssertAllSentencesEqual(expected, actual, text);

}

[Test]

[Order(40)]

public void CorrectlyParse\_SpecialCharacters()

{

var originalText = "b;\tc;\rd;\ne;\r\nf;\r\n\r\ng";

var escapedText = Regex.Escape(originalText);

var expected = new List<List<string>>

{

new List<string> {"b"},

new List<string> {"c"},

new List<string> {"d"},

new List<string> {"e"},

new List<string> {"f"},

new List<string> {"g"}

};

var actual = SentencesParserTask.ParseSentences(originalText);

AssertAllSentencesEqual(expected, actual, escapedText);

}

[Test]

[Order(50)]

public void CorrectlyParse\_OneSentenceWithWordDelimiter(

[Values('^', '#', '$', '-', '+', '1', '=', ' ', '\t', '\n', '\r')]

char delimiter)

{

var text = "x" + delimiter + "y";

var expected = new List<List<string>>

{

new List<string> {"x", "y"}

};

var actual = SentencesParserTask.ParseSentences(text);

AssertAllSentencesEqual(expected, actual, text);

}

[Test]

[Order(60)]

public void ReturnResult\_InLowerCase()

{

var text = "B.C.D";

var expected = new List<List<string>>

{

new List<string> {"b"},

new List<string> {"c"},

new List<string> {"d"}

};

var actual = SentencesParserTask.ParseSentences(text);

AssertAllSentencesEqual(expected, actual, text);

}

[Test]

[Order(80)]

public void NotReturnEmptySentence([Values("..", "...!!?","")]

string text)

{

var expected = new List<List<string>>();

var actual = SentencesParserTask.ParseSentences(text);

AssertAllSentencesEqual(expected, actual, text);

}

protected static void AssertAllSentencesEqual(

List<List<string>> expectedSentences,

List<List<string>> actualSentences,

string text)

{

var actualLines = actualSentences.Select(sentence => string.Join(" ", sentence)).ToArray();

var expectedLines = expectedSentences.Select(sentence => string.Join(" ", sentence)).ToArray();

for (var i = 0; i < Math.Min(expectedSentences.Count, actualSentences.Count); i++)

if (actualLines[i] != expectedLines[i])

AssertSentenceEuqal(expectedSentences[i], actualSentences[i], text, i);

CollectionAssert.AreEqual(expectedSentences, actualSentences,

$"Input text: [{text}].\nWrong number of sentences.");

}

protected static void AssertSentenceEuqal(

List<string> expected,

List<string> actual,

string text,

int sentenceNumber)

{

CollectionAssert.AreEqual(expected, actual, $"Input text: [{text}]\nSentence #{sentenceNumber} is wrong");

}

}

}

**Содержимое файла SentencesParserTask.cs**

using System.Collections.Generic;

namespace TextAnalysis

{

static class SentencesParserTask

{

public static List<List<string>> ParseSentences(string text)

{

var sentencesList = new List<List<string>>();

//...

return sentencesList;

}

}

}

**Содержимое файла TextGenerator\_Tests.cs**

using System.Collections.Generic;

using System.Linq;

using NUnit.Framework;

namespace TextAnalysis

{

[TestFixture]

public class TextGenerator\_Tests

{

[TestCase("x", 10)]

[TestCase("a b c", 1)]

[Order(00)]

public void ContinuePhrase\_DoNothing\_OnEmptyDictionary(string phraseStart, int phraseWordsCount)

{

var actual = TextGeneratorTask.ContinuePhrase(

new Dictionary<string, string>(),

phraseStart,

phraseWordsCount);

Assert.AreEqual(phraseStart, actual);

}

[Test]

[Order(05)]

public void ContinuePhrase\_DoNothing\_WhenWordsCountIsZero()

{

var mostFrequentNextWords = new Dictionary<string, string>

{

{"x", "y"}

};

var actual =

TextGeneratorTask.ContinuePhrase(mostFrequentNextWords, "x", 0);

Assert.AreEqual("x", actual);

}

[TestCase("x", "y z")]

[TestCase("y", "z x")]

[TestCase("y", "z x")]

[TestCase("a", "b")]

[TestCase("z", "x y")]

[TestCase("a x", "y z")]

[TestCase("a b x", "y z")]

[TestCase("y z x", "y z")]

[TestCase("w x", "y z")]

[Order(10)]

public void ContinuePhrase\_WhenNoTrigrams(string phraseBeginning, string expectedNextWord)

{

var mostFrequentNextWords = new Dictionary<string, string>

{

{"x", "y"},

{"y", "z"},

{"z", "x"},

{"a", "b" }

};

var actual =

TextGeneratorTask.ContinuePhrase(mostFrequentNextWords, phraseBeginning, 2);

Assert.AreEqual(phraseBeginning + " " + expectedNextWord, actual);

}

[TestCase("x", 1, "x y")]

[TestCase("x", 2, "x y z")]

[TestCase("x", 3, "x y z")]

[TestCase("x", 100, "x y z")]

[TestCase("x y", 100, "x y z")]

[TestCase("x x", 2, "x x y z")]

[TestCase("y x", 1, "y x y")]

[TestCase("y y", 1, "y y q")]

[TestCase("y z", 1, "y z")]

[TestCase("a b x y", 1, "a b x y z")]

[TestCase("a b y", 1, "a b y q")]

[TestCase("y", 100, "y q")]

[Order(10)]

public void ContinuePhrase(string phraseBeginning, int wordsCount, string expectedResult)

{

var mostFrequentNextWords = new Dictionary<string, string>

{

{"x", "y"},

{"x y", "z"},

{"y", "q"}

};

var actual =

TextGeneratorTask.ContinuePhrase(mostFrequentNextWords, phraseBeginning, wordsCount);

Assert.AreEqual(expectedResult, actual);

}

[TestCase("x y", "z")]

[TestCase("x y z", "w")]

[TestCase("y z", "w")]

[TestCase("x y z w", "v")]

[TestCase("y z w", "v")]

[TestCase("z w", "v")]

[Order(15)]

public void ContinuePhraseWithTrigrams(string phraseBeginning, string expectedNextWord)

{

var mostFrequentNextWords = new Dictionary<string, string>

{

{"x", "y"},

{"x y", "z"},

{"y z", "w"},

{"z w", "v"},

{"y", "a"},

{"z", "b"}

};

var actual =

TextGeneratorTask.ContinuePhrase(mostFrequentNextWords, phraseBeginning, 1);

Assert.AreEqual(phraseBeginning + " " + expectedNextWord, actual);

}

[Test]

[Order(10)]

public void ContinuePhrase\_StopWhenNoBigrammsAndTrigramms()

{

var mostFrequentNextWords = new Dictionary<string, string>

{

{"x", "y"},

{"x y", "z"},

{"y", "q"}

};

var actual =

TextGeneratorTask.ContinuePhrase(mostFrequentNextWords, "x", 4);

Assert.AreEqual("x y z", actual);

}

[TestCase("x", "")]

[TestCase("hello", "everybody")]

[TestCase("hello everybody", "be")]

[TestCase("hello everybody be", "cool")]

[TestCase("everybody be", "cool")]

[TestCase("be", "")]

[TestCase("goodbye", "")]

[TestCase("be cool", "")]

[Order(20)]

public void ContinuePhrase\_WithMultiletterWords(string phraseBeginning, string expectedNextWord)

{

var mostFrequentNextWords = new Dictionary<string, string>

{

{"hello", "everybody"},

{"everybody be", "cool"},

{"everybody", "be"}

};

var actual =

TextGeneratorTask.ContinuePhrase(mostFrequentNextWords, phraseBeginning, 1);

var expected = string.IsNullOrEmpty(expectedNextWord) ? phraseBeginning : phraseBeginning + " " + expectedNextWord;

Assert.AreEqual(expected, actual);

}

[Order(50)]

[TestCase("x", 2, "x x x")]

[TestCase("x", 5, "x x x x x x")]

[TestCase("y", 3, "y x x x")]

[TestCase("z", 3, "z y x x")]

[TestCase("a", 3, "a b c a")]

[TestCase("a", 7, "a b c a b c a b")]

[TestCase("b b", 7, "b b c a b c a b c")]

public void ContinuePhrase\_WhenCycleInDictionary(string phraseBeginning, int wordsCount, string expectedResult)

{

var mostFrequentNextWords = new Dictionary<string, string>

{

{"x", "x"},

{"a", "b"},

{"b", "c"},

{"c", "a"},

{"y", "x"},

{"z", "y"},

};

var actual =

TextGeneratorTask.ContinuePhrase(mostFrequentNextWords, phraseBeginning, wordsCount);

Assert.AreEqual(expectedResult, actual);

}

}

}

**Содержимое файла TextGeneratorTask.cs**

using System.Collections.Generic;

namespace TextAnalysis

{

static class TextGeneratorTask

{

public static string ContinuePhrase(

Dictionary<string, string> nextWords,

string phraseBeginning,

int wordsCount)

{

return phraseBeginning;

}

}

}

**Программа:**

using System;

using System.Collections.Generic;

using System.Text;

namespace TextAnalysis

{

static class SentencesParserTask

{

public static List<List<string>> ParseSentences(string text)

{

var listSentences = new List<List<string>>(); // Возвращаемый список

if (text == null) return null; //проверка на наличие текста

text = text.ToLower();

var sentences = text.Split(".:;?!()".ToCharArray(),//парсеный на предложения текст

StringSplitOptions.RemoveEmptyEntries);

foreach (var sent in sentences)

{

var listWords = new List<string>(); // Список для слов

var builder = new StringBuilder();

foreach (var ch in sent)

{

if (char.IsLetter(ch) || ch == '\'')

builder.Append(ch);

else

AddNotEmptyWord(builder, listWords);

}

AddNotEmptyWord(builder, listWords);

if (listWords.Count > 0)

listSentences.Add(listWords);

}

return listSentences;

}

public static void AddNotEmptyWord(StringBuilder builder, List<string> listWords)

{

if (builder.Length > 0)

{

listWords.Add(builder.ToString());

builder.Clear();

}

}

}

}