|  |  |
| --- | --- |
| HÜ 02 | Abgabetermin: 17.10.2025 7:35 |
| Name: | Aufwand in h: |
|  | Punkte: |

## Approximate string matching

Erstelle eine Klasse ApproximateMatchingString um eine einfache Art einer „Rechtschreibkorrektur“ zu implementieren. Dabei sollen die Fälle eines Fehlers mit 1 falschen Buchstaben korrigiert werden.

Implementiere dazu folgenden Konstruktor:

* public ApproximateMatchingString(string word, string pathToWordList)

Im Parameter soll ein Pfad zu einer Datei angegeben werden können, in welcher sich eine Liste von Wörtern (siehe Datei german\_unsorted.txt) befindet. Diese Liste soll im Konstruktor eingelesen werden und anschließend in der Klasse verwendet werden. Achte auch auf mögliche Exceptions beim Dateizugriff.

Implementiere ein Property (Getter)

* public List<string> Suggestions

Dieses Property soll einen Vorschlag für mögliche korrigierte Wörter liefern. Hier sollen natürlich nur Wörter vorgeschlagen werden, die in der zuvor eingelesenen Wörterliste zu finden sind.

Dabei sollen folgende Methoden implementiert und verwendet werden:

* private List<string> Insertion()

Es wird versucht einen Buchstaben an jeder möglichen Stelle einzufügen. Zurückgegeben werden mögliche Wörter, welche in der zuvor eingelesenen Wörterliste zu finden sind.

z.B.: cot 🡪 coat

* private List<string> Deletion()

Es wird versucht einen einzelnen Buchstaben an jeder möglichen Stelle zu löschen. Zurückgegeben werden mögliche Wörter, welche in der zuvor eingelesenen Wörterliste zu finden sind.

z.B.: coat 🡪 cot

* private List<string> Replace()

Es wird versucht einen einzelnen Buchstaben zu ersetzen. Zurückgegeben werden mögliche Wörter, welche in der zuvor eingelesenen Wörterliste zu finden sind.

z.B.: coat 🡪 cost

* private List<string> Transposition()

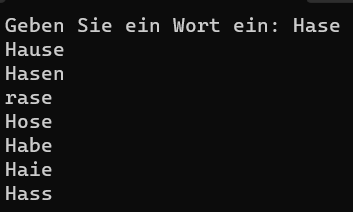
Es wird versucht zwei Nachbarbuchstaben zu vertauschen. Zurückgegeben werden mögliche Wörter, welche in der zuvor eingelesenen Wörterliste zu finden sind.

z.B.: cost 🡪 cots

* Überlege, welche zusätzlichen Operationen noch durchgeführt werden können. Beschreibe zwei weitere Vorschläge und gib jeweils ein Beispiel dazu.

Teste deine Implementierung ausführlich!

Testfällle zum vergleichen:



Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Hue2\_2\_Approxi

{

internal class ApproximateMatchingString

{

private List<char> word;

private string pathToWordList;

public ApproximateMatchingString(List<char> word, string pathToWordList)

{

this.word = word;

this.pathToWordList = pathToWordList;

}

public List<string> Suggestions(List<char> word)

{

List<string> result = new List<string>();

result.AddRange(Insertion(word));

result.AddRange(Deletion(word));

result.AddRange(Replace(word));

result.AddRange(Transposition(word));

return result;

}

private List<string> Search(List<char> word)

{

string[] germanText = File.ReadAllLines(pathToWordList);

List<string> result = new List<string>();

string searchWord = string.Concat(word);

for (int i = 0; i < germanText.Length; i++)

{

if (germanText[i] == searchWord)

{

result.Add(germanText[i]);

}

}

return result;

}

private List<string> Insertion(List<char> word)

{

List<string> result = new List<string>();

for (int i = 0; i <= word.Count; i++)

{

for (char c = 'A'; c <= 'Z'; c++)

{

List<char> editedWord = new List<char>(word);

editedWord.Insert(i, c);

result.AddRange(Search(editedWord));

}

for (char c = 'a'; c <= 'z'; c++)

{

List<char> editedWord = new List<char>(word);

editedWord.Insert(i, c);

result.AddRange(Search(editedWord));

}

}

return result;

}

private List<string> Deletion(List<char> word)

{

List<string> result = new List<string>();

for (int i = 0; i < word.Count; i++)

{

List<char> editedWord = new List<char>(word);

editedWord.RemoveAt(i);

result.AddRange(Search(editedWord));

}

return result;

}

private List<string> Replace(List<char> word)

{

List<string> result = new List<string>();

for (int i = 0; i < word.Count; i++)

{

List<char> editedWord = new List<char>(word);

for (int j = 0; j < 26; j++)

{

if (editedWord[i] <= 'Z' && editedWord[i] >= 'A')

{

editedWord[i] = (char)(editedWord[i] + 1);

}

if (editedWord[i] > 'Z')

{

editedWord[i] = (char)(editedWord[i] - 26);

}

result.AddRange(Search(editedWord));

}

for (int j = 0; j < 26; j++)

{

if (editedWord[i] <= 'z' && editedWord[i] >= 'a')

{

editedWord[i] = (char)(editedWord[i] + 1);

}

if (editedWord[i] > 'z')

{

editedWord[i] = (char)(editedWord[i] - 26);

}

result.AddRange(Search(editedWord));

}

}

return result;

}

private List<string> Transposition(List<char> word)

{

List<string> result = new List<string>();

for (int i = 0; i < word.Count - 1; i++)

{

List<char> editedWord = new List<char>(word);

char temp = editedWord[i];

editedWord[i] = editedWord[i + 1];

editedWord[i + 1] = temp;

result.AddRange(Search(editedWord));

}

return result;

}

}

}