

Aufgabe 1: Wörter aufräumen

Team-ID: 00111

Team-Name: Team Name

Marek Freunscht

8. September 2020

Inhaltsverzeichnis

| | |
|------------------|---|
| Lösungsidee..... | 1 |
| Umsetzung | 1 |
| Beispiele | 2 |
| Quellcode | 4 |

Lösungsidee

Um alle Wörter gut zuzuweisen, muss man sich zuerst eine geeignete Einteilung der Wörter überlegen. Eine solche Einteilung wäre zum Beispiel nach Wortlänge. In der Einteilung nach Wortlänge kann man über diese iterieren und überprüfen welche Wörter in ein unvollständiges Wort passen würden. Wenn dies nur ein Wort ist kann man dieses direkt Abspeichern. Dies würde zum Beispiel in einem Dictionary geschehen, welches unvollständiges Wort zu vollständigem Wort zuordnet. Diese müsste man dann auch aus der aktuellen Liste von vollständigen und unvollständigen Wörtern löschen. Wenn nun aber mehrere Wörter in ein unvollständiges Wort passen kann man dies fürs Erste ignorieren, da durch das Löschen von Wörtern aus der aktuellen Wörterliste sich diese Möglichkeiten letztendlich verringern. Nun muss man so lange über diese aktuelle Längenliste iterieren, bis es keine Wörter mehr zu vergeben gibt. Diese Schritte muss man nun für jede mögliche Wortlänge wiederholen.

Umsetzung

Um die Gruppierung nach Wortlänge überhaupt umzusetzen, muss man zuerst die maximale Wortlänge herausfinden. Dies war schnell durch ein Sortieren der Wortliste nach Wortlänge und dem Auswählen des letzten Wortes in dieser sortierten Liste getan:

```
int maxLength = availableWords.OrderBy(word => word.Length).Last().Length;
```

Nun kann man über eine Schleife bis zu dieser maximalen Länge an jede Wortlänge kommen. In dieser Schleife werden dann 2 Arrays und 2 Listen initialisiert. Die 2 Arrays (hier `currentAvailableWords` und `currentWordsToFind`) stellen die beiden gesamten Wortlisten für diese Wortlänge dar, während die beiden Listen (hier `notUsedWords` und `notUsedUnfinishedWords`) Kopien dieser Arrays sind und den Zweck haben dass aus ihnen entfernt werden kann. Ob für die gesamten Wörterlisten ein Array oder eine Liste benutzt werden sollten ist in diesem Fall recht irrelevant, nur da sich die Länge von ihnen nicht ändert bevorzuge ich

Arrays. Der nächste Teil ist nun der Hauptteil vom Algorithmus. Als äußersten Teil findet man eine While-Schleife:

```
bool finished = false;
while (!finished)
{
    ...
}
```

In dieser While-Schleife wird nun immer wieder über alle Wörter in currentWordsToFind iteriert. Die Zuweisungslogik (im nächsten Codeblock '...') erfolgt aber nur für diejenigen Wörter, die sich auch noch in der notUsedUnfinishedWords Liste befinden, da nurnoch diese zur Verfügung stehen.

```
for (int i = 0; i < currentWordsToFind.Length; i++)
{
    if (notUsedUnfinishedWords.Contains(currentWordsToFind[i]))
    {
        ...
    }

    if (notUsedUnfinishedWords.Count == 0)
    {
        finished = true;
        break;
    }
}
```

Wenn die Listen mit den noch zur Verfügung stehenden Wörtern leer sind, wird die Schleife abgebrochen. Die Zuweisungslogik lässt sich zuerst ausgeben, wie viele Wörter aus den übrigen vollständigen Wörtern in das unvollständige Wort am Index i passen. Sollten dies mehr als 1 Wort sein, wird nichts getan und die Schleife läuft weiter. Passt jedoch nur ein Wort, wird dies, sowohl als auch das unvollständige Wort aus den jeweiligen Listen gelöscht und in ein Dictionary hinzugefügt. (Funktion wordsThatFit [hier](#)):

```
List<string> wordsThatFit = WordsThatFit(currentWordsToFind[i], notUsedWords);
//Wenn nur ein Wort passt kann dieses direkt eingesetzt werden
if (wordsThatFit.Count == 1)
{
    notUsedWords.Remove(wordsThatFit[0]);
    notUsedUnfinishedWords.Remove(currentWordsToFind[i]);
    if (!allWords.ContainsKey(currentWordsToFind[i]))
        allWords.Add(currentWordsToFind[i], wordsThatFit[0]);
}
```

Am Ende muss dann nurnoch der Output zusammengesetzt werden. Dies geschieht indem man über die originale Liste mit unvollständigen Wörtern iteriert und nachschaut, welchen value im Dictionary das momentane unvollständige Wort hat, da diese als keys fungieren. Dies muss man dann noch an den string anhängen.

Beispiele

Bei den Beispieldateien und zwei eigenen Dateien werden folgende Outputs produziert:

Raetsel0.txt

_h __, _a __r __e __b __!

arbeit eine für je oh was

Es in in so aus der die die ein ist Opa sie und und von dass eine eine sind einer Liste schon sowie einige findet
Jürgen Rätsel sollen werden ergeben gegeben lustige Wörtern Apotheke blättert gebracht richtige Buchstaben
Geschichte vorgegeben Leerzeichen Reihenfolge Satzzeichen Zeitschrift

Output:

Opa Jürgen blättert in einer Zeitschrift aus der Apotheke und findet ein Rätsel. Es ist eine Liste von Wörtern
gegeben, die in die richtige Reihenfolge gebracht werden sollen, so dass sie eine lustige Geschichte ergeben.
Leerzeichen und Satzzeichen sowie einige Buchstaben sind schon vorgegeben.

Raetsel5.txt

O_o _ _u _ _u _ _ _m _i _ _ , d _r _c _f _ _n _ _i _we _ _ _ _ _h _ _ _d _r _ü _ _r.

seinem kam war er sich hatte ging zur Otto da an und und Hinweg verlaufen Müde auf er nicht Schule

Output:

Otto ging zur Schule und kam nicht an, da er sich auf seinem Hinweg verlaufen hatte und er Müde war.

Hier hat man den Sonderfall das bei einem unvollständigen Wort zwei statt nur einem Buchstaben gegeben
sind, was den Algorithmus jedoch nicht behindert.

Quellcode

Initialisierung von Hauptvariablen:

```
List<List<string>> temp;  
String[] wordsToFind;  
string[] availableWords;  
Dictionary<string, string> allWords = new Dictionary<string, string>();  
string result = string.Empty;  
string originalString = string.Empty;
```

Hauptteil:

```

//maximale Wortlänge herausfinden
int maxLength = availableWords.OrderBy(word => word.Length).Last().Length;

//über jede mögliche Wortlänge iterieren
for (int wordLength = 1; wordLength <= maxLength; wordLength++)
{
    //Wenn es kein Wort mit der Länge wordLength gibt soll diese nicht beachtet werden
    if (availableWords.Find(word => word.Length == wordLength) != null)
    {
        //Alle Wörter gleicher Länge zusammenfassen
        string[] currentAvailableWords = availableWords.Where(word => word.Length == wordLength).ToArray();
        string[] currentWordsToFind = wordsToFind.Where(word => word.Length == wordLength).ToArray();
        //Kopien der Listen erstellen aus denen man Elemente entfernen kann ohne dass es zu Index Problemen
        kommt
        List<string> notUsedWords = new List<string>();
        List<string> notUsedUnfinishedWords = new List<string>();
        notUsedWords.AddRange(currentAvailableWords);
        notUsedUnfinishedWords.AddRange(currentWordsToFind);

        bool finished = false;
        while (!finished)
        {
            for (int i = 0; i < currentWordsToFind.Length; i++)
            {
                if (notUsedUnfinishedWords.Contains(currentWordsToFind[i]))
                {
                    List<string> wordsThatFit = WordsThatFit(currentWordsToFind[i], notUsedWords);
                    //Wenn nur ein Wort passt kann dieses direkt eingesetzt werden
                    if (wordsThatFit.Count == 1)
                    {
                        notUsedWords.Remove(wordsThatFit[0]);
                        notUsedUnfinishedWords.Remove(currentWordsToFind[i]);
                        if (!allWords.ContainsKey(currentWordsToFind[i]))
                            allWords.Add(currentWordsToFind[i], wordsThatFit[0]);
                    }
                }

                if (notUsedUnfinishedWords.Count == 0)
                {
                    finished = true;
                    break;
                }
            }
        }
    }
}

```

Funktion „WordsThatFit“:

```

static List<string> WordsThatFit(string unfinishedWord, List<string> possibleWords)
{
    List<string> wordsFit = new List<string>();
    foreach (string word in possibleWords)
    {
        if (WordsFit(unfinishedWord, word) && !wordsFit.Contains(word))
        {
            wordsFit.Add(word);
        }
    }
    return wordsFit;
}

```

Funktion „WordsFit“:

```
static bool WordsFit(string unfinishedWord, string finishedWord)
{
    for (int i = 0; i < unfinishedWord.Length; i++)
    {
        if (unfinishedWord[i] != '_' && unfinishedWord[i] != finishedWord[i])
        {
            return false;
        }
    }
    return true;
}
```