39. Bundeswettbewerb Informatik

Runde 1

Aufgabe 1: Wörter aufräumen

07. November 2020

Inhaltsverzeichnis

Lösungsidee

Zuerst wird für jeden Lückentext geprüft, welche gegebenen Wörter dieser Lücke passen könnten. Nehmen wir das in der Aufgabe gegebene Beispiel:

```
_h __, _a_ __r ___e __b___!
arbeit eine für je oh was
```

Für den ersten Lückentext *„_h“* ist ersichtlich, dass das Wort nur aus zwei Buchstaben besteht und der zweite Buchstabe *„h“* sein muss. In diesem Fall gibt es nur ein passendes Wort, nämlich *„oh“*. Allerdings gibt es nicht immer nur eine Möglichkeit für einen Lückentext. Bspw. hat der zweite Lückentext *„__“* mehrere mögliche Wörter: *„je”* und *„oh“*.

Nachdem für jeden Lückentext dessen zugehörigen möglichen Wörter gespeichert wurden, wird nun überprüft, welches Wort aus den allen gespeicherten möglichen Wörtern das Richtige ist. Dies erfolgt wie folgt:

1. Zuerst werden allen Lückentexten, für die nur ein einziges mögliches Wort vorhanden ist, dieses Wort zugeordnet. Falls es für den Lückentext mehrere mögliche Wörter gibt, die aber gleich sind, werden diese Wörter auch als ein einziges Wort angesehen. Alle dadurch zugeordneten Wörter werden nicht mehr für andere Lückentexte verfügbar sein. Dabei ist aber zu beachten, dass wenn es zwei identische Wörter gibt und eins davon einem Lückentext zugeordnet wird, wird auch nur dieses nicht mehr zur Verfügung gestellt. Das andere gilt immer noch als eine Möglichkeit für andere Lückentexte.

2. Wiederholt man die 1. Schritt solange, bis jedem Lückentext ein Wort zugeordnet wird.

Demonstriert man das mit dem obigen Beispiel:

Lückentext	Mögliche Wörter	Lösung
_h	oh	
__	je,oh	
_a_	was	
__r	für	
___e	eine	
__b___	arbeit	

Schritt 0: alle möglichen Worte speichern

Nach der ersten Wiederholung:

Lückentext	Mögliche Wörter	Lösung
_h	~~oh~~	oh
__	je,~~oh~~	
_a_	~~was~~	was
__r	~~für~~	für
___e	~~eine~~	eine
__b___	~~arbeit~~	arbeit

Schritt 1: Allen Lückentexten, für die nur ein einziges Wort möglich ist, wird das Wort direkt zugeordnet.

Nach der zweiten Wiederholung:

Lückentext	Mögliche Wörter	Lösung
_h	~~oh~~	oh
__	~~je,oh~~	je
_a_	~~was~~	was
__r	~~für~~	für
___e	~~eine~~	eine
__b___	~~arbeit~~	arbeit

Schritt 1 (1. Mal wiederholt)

Somit wird die richtige Lösung gefunden. Für die komplizierteren Beispiele werden nur noch weitere Wiederholungen benötigt, aber am Ende wird es immer eine Lösung geben. D. h. egal es wie viele mögliche Wörter am Anfang für die Lückentexten gibt, wird nach Wiederholungen des Schritts 1 nur noch für jeden Lückentext ein einziges übrigbleiben.

Die Satzzeichen bleiben auf ihren Stellen unberührt. Nur die Lückentexte werden mit der Lösung ersetzt.

Umsetzung

Der Name der Test-Datei muss beim Öffnen des Programms eingegeben werden. Diese Datei wird dann mithilfe von *StreamReader* eingelesen.

Die Lückentexte und Wörter werden zunächst eingelesen und in zwei Arrays *raetsel* und *woerter* gespeichert.

```
          string[] raetsel = (sr.ReadLine()).Split(' ');
          string[] woerter = (sr.ReadLine()).Split(' ');
```

Danach wird eine Zuordnungstabelle, *raetselZuWoertern*, instanziiert, die für jeden Lückentext dessen mögliche Wörter speichern soll. Hier werden die Indexe der obigen Arrays gespeichert. Der Schlüssel hier ist der Index des Rätsels (Lückentextes) und die zugeordneten Werte sind die Indexe der für diesen Lückentext als möglichen gesehenen Wörter (gespeichert als Array).

```
Dictionary<int, int[]> raetselZuWoertern =
                              new Dictionary<int, int[]>();
```

Dann wird diese Zuordnungstabelle gefüllt. Man prüft also für jeden Lückentext, welche Wörter dazu passen können.

Zuerst wird eine Schleife angelegt, sodass jeder Lückentext geprüft wird.

```
// Man geht die Raetsel durch
for (int rPos = 0; rPos < raetsel.Length; rPos++)
```

Innerhalb dieser Schleife wird eine temporale Liste *possibleWPosForR* erstellt, die speichert, welche Wörter zu dem jetzigen Lückentext passen.

```
   List<int> possibleWPosForR = new List<int>();
```

Für jedes Wort prüft man, ob es diesem Lückentext passt. Wenn ja, dann wird dieses Wort in die Liste eingefügt. Das Wort kann dem Lückentext nur passen, wenn sie gleich lang sind. Deswegen vergleicht man zuerst die Länge des Lückentextes und des Wortes.

```
for (int wPos = 0; wPos < woerter.Length; wPos++)
    {
      // Nur wenn der Raetsel und das Wort gleich lang sind
      // ist es moeglich, dass dieses Wort auch dem Lueckentext passt
      if (raetsel[rPos].GetRealStringLength() == woerter[wPos].Length)
    ..........
```

Sind sie gleich lang, wird dann jeder Buchstabe des Lückentextes und des Wortes vergleicht.

```
// In diesem Fall wird jeder Buchstabe des Wortes geprueft
for (int i = 0; i < woerter[wPos].Length; i++)
```

Wenn es ein Buchstabe sowohl in dem Lückentext als auch in dem Wort stimmt, wird dieses Wort direkt als eine Möglichkeit für den Lückentext angesehen und in die Liste *possibleWPosForR* eingefügt.

```
// Falls ein Buchstabe sowohl im Lueckentext als auch im Wort stimmt
if (raetsel[rPos][i].CompareTo(woerter[wPos][i]) == 0)
{
    // Wird dieses Wort als eine Moeglichkeit
    //fuer diesen Lueckentext angesehen
    possibleWPosForR.Add(wPos);
    break;
}
```

Falls es irgendeinen Buchstaben gibt, der in dem Lückentext vorgegeben ist (also ist der Buchstabe in dem Lückentext an dieser Stelle nicht „_“), aber nicht in dem Wort an dieser Stelle zu finden ist, dann kann das Wort auch keine mögliche Lösung für den Lückentext sein. Mithilfe der 1. If-Block wurde schon geprüft, dass der Buchstabe in dem Lückentext anders (aber möglicherweise gleich ‘_’) ist als in dem Wort.

```
    // Falls ein Buchstabe im Lueckentext diesem Wort widerspricht
    //ex. Lueckentext: __f__, Wort: Birne
    // der 3. Buchstabe f != r
    // kann man das Wort ausschliessen
    else if (raetsel[rPos][i].CompareTo('_') != 0) break;
```

Und wenn der Lückentext nur aus „_“ besteht, wird die Wörter, die gleich lang wie diesen Lückentext sind, auch als eine Möglichkeit in die Liste eingefügt. Es wurde vorher geprüft, dass der Lückentext und das Wort gleich lang sind (vgl. S. 4 Unten).

```
    // Falls dieser Lueckentext nur aus _ besteht
    // Dann wird diese auch als eine Moeglichkeit angesehen
    if (i == woerter[wPos].Length - 1) possibleWPosForR.Add(wPos);
```

Nach der Schleife für die Wörter *(for (int wPos…))* werden dann alle Möglichkeiten für diesen Lückentext, die vorher in der temporalen Liste *possibleWPosForR* gespeichert wurden, an die Zuordnungstabelle weitergegeben.

```
    // alle Moeglichkeiten werden zu diesem Lueckentext zugeordnet
    raetselZuWoertern.Add(rPos, possibleWPosForR.ToArray());
```

Wiederholt man das mit der äußeren Schleife *(for (int rPos…))*, bekommt man alle möglichen Wörter für jeden Lückentext. Nun ist der 0. Schritt der Lösungsidee fertig.

Aus dem gegebenen Beispiel (vgl. Lösungsidee) kann Teil dieser Tabelle nachher so aussehen:

```
    // Dictionary<int, int[]> raetselZuWoertern
    0 -> [4]    // _h -> oh
    1 -> [3,4]  // __ -> je,oh
    3 -> [5]  // _a_ -> was
    ……
```

Danach wird ein Array, *answer*, erzeugt, der die Wörter in der richtigen Reihfolge speichern soll.

```
            // die Array answer speichert die richtige Antwort
            string[] answer = new string[raetsel.Length];
```

Außerdem wird noch eine Liste, *usedWoerter*, instanziiert, die die Indexe der genutzten Wörter speichert.

```
            // speichert die Indexe der genutzten Woerter
            List<int> usedWoerter = new List<int>();
```

Solange nicht für jeden Lückentext das richtige Wort gefunden wird, wird das Verfahren, das in der Lösungsidee als Schritt 1 erwähnt wurde, wiederholt. Hier wird eine While-Schleife gebraucht.

```
// Waehrend nicht alle Woerter genutzt wurden
// bzw. noch weitere Woerter nicht in dem Lueckentext zugeordnet wurden
while (usedWoerter.Count < woerter.Length)
```

Darüber hinaus benötigt man eine For-Schleife, damit man jeden Lückentext überprüfen kann.

```
    // Man geht der Lueckentext nochmal durch
    for (int i = 0; i < raetsel.Length; i++)
```

Bevor man prüft, ob für den Lückentext nur ein mögliches Wort gibt, wird innerhalb dieser Schleife jedes Mal zuerst getestet, ob die zu diesem Lückentext zugeordneten Wörter bereits von den anderen Lückentexten gebraucht wurden. Man updatet also die zugeordnet Wörter.

```
int[] alteMoeglichkeiten = raetselZuWoertern[i];

// Speichert, wie viele Moeglichkeit noch uebrig bleiben
List<int> restlicheWPos = new List<int>();

// Die Code zum Überprüfen wird hier ausgelassen
// Vgl. Quellecode für die vollständige Code

// Update die Moeglichkeiten fuer den Lueckentext
raetselZuWoertern[i] = restlicheWPos.ToArray();
```

Danach prüft man endlich, ob es für diesen Lückentext nur ein mögliches Wort gibt.

```
// Falls es nur eine Moeglichkeit uebrigbleibt
// oder alle restliche moegliche Woerter gleich sind
bool singleOrAllSameWord = true;
for (int j = 0; j < restlicheWPos.Count - 1; j++)
{
    if (woerter[restlicheWPos[j]].CompareTo(woerter[restliche-
WPos[j + 1]]) != 0)
  singleOrAllSameWord = false;
}
```

Ist es der Fall, wird dieses Wort in das Array *answer* gespeichert, und zwar an den Index des Lückentextes. Auch wird der Index des genutzten Wortes in die Liste *usedWoerter* eingefügt, sodass das Wort später für andere Lückentexte nicht mehr verfügbar wird.

```
// Falls es fuer diese Lueckentext noch moegliche Woerter gibt
// und es nur eine Moeglichkeit bleibt (singleOrAllSameWord)
if (restlicheWPos.Count > 0 && singleOrAllSameWord)
{
    // Falls es Satzzeichen gibt, fuegt es noch auch in die Ant-
wort hinzu
    if (raetsel[i].HasPunctuation()) answer[i] = woerter[restliche-
WPos[0]] + raetsel[i][^1];
    // Ansonsten nur das Wort
    else answer[i] = woerter[restlicheWPos[0]];
    // Makiert das Wort als genutzt
    usedWoerter.Add(restlicheWPos[0]);
}
```

Ist hingegen nicht der Fall, dann nimmt man für diesen Lückentext nichts vor (außer dem Update). Der Zeiger der For-Schleife erhöht sich um eins. Wenn die For-Schleife fertig ist

und nicht allen Lückentexten ein Wort zugeordnet wird, wird die For-Schleife wiederholt, bis die Lösung gefunden wird. Die Wiederholung der For-Schleife erfolgt durch die While-Schleife, die oben erwähnt wurde.

Am Ende wird die Liste *answer* in ein *string* umgewandelt und in der Console gezeigt.

```
            // die Array answer wird zu ein string umgewandelt und
            // in Console gezeigt
            Console.WriteLine(String.Join(" ", answer));
```

Beispiele

Erwartete Formen von Daten

Angabe für das Programm:

Name der Test-Datei sollte nach Aufforderung des Programms eingegeben wird.

Nach der Ausgabe des Ergebnisses wird das Programm den Nutzer auffordern, eine beliebige Taste zu drücken, um das Programm zu schließen.

Die Bindestriche zwischen Zeilen sind nicht von dem Programm erzeugt. Die sind nur da, weil Microsoft Word das automatisch eingefügt hat.

Beispiel 1, vorgegeben, raetsel0.txt

Ausgabe:

```
oh je, was für eine arbeit!
```

Die Funktionsweise des Programms wurde bereits mithilfe dieses Beispiels in der Lösungsidee erläutertet.

Beispiel 2, vorgegeben, raetsel1.txt

Ausgabe:

```
Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr w
ütend und wurde allenthalben als Schritt in die falsche Richtung angese
hen.
```

Beispiel 3, vorgegeben, raetsel2.txt

Ausgabe:

```
Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er
sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.
```

Beispiel 4, vorgegeben, raetsel3.txt

Ausgabe:

```
Informatik ist die Wissenschaft von der systematischen Darstellung, Spe
icherung, Verarbeitung und Übertragung von Informationen, besonders der
 automatischen Verarbeitung mit Digitalrechnern.
```

Beispiel 5, vorgegeben, raetsel4.txt

Ausgabe:

```
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 scho
n vorgegeben.
```

Quellcode

```
1.  using System;
2.  using System.IO;
3.  using System.Collections.Generic;
4.
5.  namespace Aufgabe1
6.  {
7.    public static class WoerterAufraeumer
8.    {
9.      public static void Main(string[] args)
10.     {
11.       // Setzt die Encoding zu UTF8 fuer deutsche Buchstaben
12.       Console.OutputEncoding = System.Text.Encoding.UTF8;
13.
14.       Console.WriteLine("Bitte Name der Test-Datei eingeben...");
15.       string testFilePath = Console.ReadLine();
16.
17.       if (File.Exists(testFilePath))
18.       {
19.         using (StreamReader sr = File.OpenText(testFilePath))
20.         {
21.           string[] raetsel = (sr.ReadLine()).Split(' ');
22.           string[] woerter = (sr.ReadLine()).Split(' ');
23.
24.           // speichert, welche moegliche Woerter zu
25.           // dem ggb. Lueckentext passen koennen
26.           // Index des Lueckentextes --
    -> alle moegliche passende ggb. Woerter
27.           // ex. 3 ---> [1, 3, 4]
28.           Dictionary<int, int[]> raetselZuWoertern = new Dictionary<int, int
    []>();
29.
30.           // Man geht die Raetsel durch
31.           for (int rPos = 0; rPos < raetsel.Length; rPos++)
32.           {
33.             // temperale Liste, die speichert
34.             // welche Woerter zu diesem Lueckentext passen
35.             List<int> possibleWPosForR = new List<int>();
36.             // Man geht dann die Woerter durch
37.             for (int wPos = 0; wPos < woerter.Length; wPos++)
38.             {
39.               // Nur wenn der Raetsel und das Wort gleich lang sind
40.               // ist es moeglich, dass dieses Wort auch dem Lueckentext pass
    t
41.               if (raetsel[rPos].GetRealStringLength() == woerter[wPos].Lengt
    h)
42.               {
43.                 // In diesem Fall wird jeder Buchstabe des Wortes geprueft
44.                 for (int i = 0; i < woerter[wPos].Length; i++)
45.                 {
46.                   // Falls ein Buchstabe sowohl in dem Lueckentext als auch
    im Wort stimmt
47.                   if (raetsel[rPos][i].CompareTo(woerter[wPos][i]) == 0)
48.                   {
49.                     // Wird dieses Wort als eine Moeglichkeit fuer diesen Lu
    eckentext angesehen
50.                     possibleWPosForR.Add(wPos);
51.                     break;
52.                   }
53.                   // Falls ein Buchstabe im Lueckentext diesem Wort widerspr
    icht
54.                   // ex. Lueckentext: __f__, Wort: Birne
55.                   // der 3. Buchstabe f != r
56.                   // kann man das Wort ausschliessen
```

```
57.                   else if (raetsel[rPos][i].CompareTo('_') != 0) break;
58.                   // Falls dieser Lueckentext nur aus _ besteht
59.                   // Dann wird diese auch als eine Moeglichkeit angesehen
60.                   if (i == woerter[wPos].Length - 1) possibleWPosForR.Add(wP
    os);
61.                 }
62.               }
63.             }
64.             // alle Moeglichkeiten werden zu diesem Lueckentext zugeordnet
65.             raetselZuWoertern.Add(rPos, possibleWPosForR.ToArray());
66.           }
67.
68.           // die Array answer speichert die richtige Antwort als string
69.           string[] answer = new string[raetsel.Length];
70.           // speichert die Indexe der genutzten Woerter
71.           List<int> usedWoerter = new List<int>();
72.           // Waehrend nicht alle Woerter genutzt wurden
73.           // bzw. noch weitere Woerter nicht in dem Lueckentext zugeordnet w
    urden
74.           while (usedWoerter.Count < woerter.Length)
75.           {
76.             // Man geht der Lueckentext nochmal durch
77.             for (int i = 0; i < raetsel.Length; i++)
78.             {
79.               int[] alteMoeglichkeiten = raetselZuWoertern[i];
80.
81.               // Speichert, wie viele Moeglichkeit noch uebrig bleiben
82.               List<int> restlicheWPos = new List<int>();
83.               // Fuer alle Moeglichkeiten
84.               for (int j = 0; j < alteMoeglichkeiten.Length; j++)
85.               {
86.                 // Falls diese Index nicht genutzt wurde
87.                 if (!usedWoerter.Contains(alteMoeglichkeiten[j]))
88.                 {
89.                   // bleibt dieses immer als eine Moeglichkeit
90.                   restlicheWPos.Add(alteMoeglichkeiten[j]);
91.                 }
92.               }
93.               // Update die Moeglichkeiten fuer den Lueckentext
94.               raetselZuWoertern[i] = restlicheWPos.ToArray();
95.
96.
97.               // Falls es nur eine Moeglichkeit uebrigbleibt
98.               // oder alle restliche moegliche Woerter gleich sind
99.               bool singleOrAllSameWord = true;
100.                  for (int j = 0; j < restlicheWPos.Count - 1; j++)
101.                  {
102.                    if (woerter[restlicheWPos[j]].CompareTo(woerter[restl
    icheWPos[j + 1]]) != 0) singleOrAllSameWord = false;
103.                  }
104.
105.                  // Falls es fuer diese Lueckentext noch moegliche Woert
    er gibt
106.                  // und es nur eine Moeglichkeit bleibt (singleOrAllSame
    Word)
107.                  if (restlicheWPos.Count > 0 && singleOrAllSameWord)
108.                  {
109.                    // Falls es Satzzeichen gibt, fuegt es noch auch in d
    ie Antwort hinzu
110.                    if (raetsel[i].HasPunctuation()) answer[i] = woerter[
    restlicheWPos[0]] + raetsel[i][^1];
111.                    // Ansonsten nur das Wort
112.                    else answer[i] = woerter[restlicheWPos[0]];
113.                    // Makiert das Wort als genutzt
114.                    usedWoerter.Add(restlicheWPos[0]);
115.                  }
```

```
                    }
                }

                // die Array answer wird zu ein string umgewandelt und
                // in Console gezeigt
                Console.WriteLine(String.Join(" ", answer));
            }
          }
          else
          {
            Console.WriteLine("Die gegebene Datei-
Namen ist nicht gueltig. Das Programm und die Datei muessen in demselben Ord
ner stehen. ");
          }

          Console.WriteLine("\r\nDrueck eine beliebige Taste zu schliesse
n...");
          Console.ReadKey();
        }


        private static int GetRealStringLength(this string s)
        {
          int length = 0;
          foreach (char c in s.ToCharArray())
          {
            // Nur Buchstaben oder _ zaehlt als eine Laenge
            // Also sind Satzzeichen kein richtige Laenge
            if (Char.IsLetter(c) || '_'.CompareTo(c) == 0) length++;
          }
          return length;
        }


        private static bool HasPunctuation(this string s)
        {
          foreach (char c in s.ToCharArray())
          {
            if (!Char.IsLetter(c) && '_'.CompareTo(c) != 0) return true;

          }
          return false;
        }
      }
    }
```