Aufgabe 1 - Stoerung

Team-ID: 00166
Team-Name: ez

Bearbeitet von Florian Bange

21. November 2022

Inhaltsverzeichnis

1 Definition des Problems

Sei ein Wort w definiert als Kette von beliebigen Zeichen:

$$w = a_1 \dots a_n$$

mit $|w| = |a_1 \dots a_n| = n$.

Sei ein Satz

$$s = (s_1,\ \dots,\ s_m)$$

ein Tupel aus m Worten.
Wobei der Satz s genau dann in einem Wort (Text) enthalten ist, wenn die Zeichenkette

$$f_1 s_1 f_2 s_2 f_3 \dots f_m s_m f_{m+1}$$

in dem Wort enthalten ist (als Teilwort).
Wobei f_i mit $|f_i| \geq 1$ ein Wort ausschlieszlich aus Leerzeichen ist.

Ein Lueckensatz

$$l = (l_1,\ \dots,\ l_m)$$

ist ein Satz mit Variablen:
Jedes l_i ist

1. ein festes Wort a, oder
2. eine Variable „_“.

Ein Satz $s = (s_1,\ \dots,\ s_m)$ erfuellt einen Lueckensatz $l = (l_1,\ \dots,\ l_m)$, wenn fuer jedes i Folgendes gilt:

1. Falls $l_i =$ „_“: s_i ist irgendein Wort mit mindestens einem Buchstaben / einer Ziffer.
2. Falls $l_i = a$: $s_i = f_1 a f_2$ fuer Worte $f_1,\ f_2$, welche keine Buchstaben oder Ziffern enthalten (Sonderzeichen)

Diese Definition ermoeglicht es, dass Satzzeichen vor und nach festen Worten sein duerfen und dass Variablen (mit mind. Laenge 1) sowohl Ziffern als auch Worte sein koenen und ebenfalls Sonderzeichen enthalten koenen.

Durch diese Definition sind fuer den Lueckensatz l = (Hallo, _, _) alle der Folgenden Saetze erfuellend:

1. (Hallo, mein, Freund)
2. („Hallo„ mein, Freund!“)
3. („Hallo!, die, U-Bahn!“)
4. („Hallo!“, zum, 69420sten)

2 Loesungsidee / Loesung

Das soeben definierte Problem wird wie folgt geloest:

Sei T der gegebene Text und l der Lueckensatz.
Fuer den gegebenen Text T wird eine Liste L erstellt, welche alle Worte von T enthaelt, welche durch Leerzeichen getrennt werden.
Beispielsweise wird aus $T =$ „Hallo, wie geht’s euch?“ $L =$ [„Hallo,“, „wie“, „geht’s“, „euch?“].

Fuer diese Liste L kann nun jede zusammenhaengende Teilliste der Laenge $|l|$ darauf ueberprueft werden, ob sie den Lueckensatz l erfuellt. Denn diese Teillisten sind als Saetze in T enthalten.

Algorithm 1 Pseudocode zum ueberpruefen aller Teillisten korrekter Laenge

```
1: procedure CHECKALL(L, l)
2:     matches ← leere Liste
3:     for i = 1, ..., |L| − |l| + 1 do
4:         s ← (L[i], ..., L[i + |l| − 1])
5:         if matches(s, l) then
6:             matches.add(s)
7:         end if
8:     end for
9:     return matches
10: end procedure
```

2.1 Ueberpruefen, ob ein Satz einen Lueckensatz erfuellt

Um zu ueberpruefen, ob ein Satz

$$s = (s_1, \ \ldots, \ s_m)$$

einen Lueckensatz

$$l = (l_1, \ \ldots, \ l_m)$$

erfuellt, werden fuer jedes i mit $1 \leq i \leq m$ die Eigenschaften aus [1] ueberprueft:

1. Wenn l_i eine Variable ist ($l_i =$ „_“), enthaelt s_i mindestens einen Buchstaben oder eine Ziffer
2. Wenn l_i keine Variable ist ($l_i = a$), ist $s_i = f_1 a f_2$ fuer $f_1, \ f_2$ Woerter ohne Ziffern und Buchstaben.

Algorithm 2 Pseudocode zum ueberpruefen, ob ein Satz s einen Lueckensatz l erfuellt

```
1: procedure MATCHES(s, l)
2:     if |s| ≠ |l| then
3:         return FALSE
4:     end if
5:
6:     for i = 1, ..., |l| do
7:         if l_i = „_“ then
8:             if !containsDigitOrLetter(s_i) then
9:                 return FALSE
10:            end if
11:        else
12:            b ←s_i ohne das erste l_i
13:            if containsDigitOrLetter(b) then
14:                return FALSE
15:            end if
16:        end if
17:    end for
18:
19:    return TRUE
20: end procedure
```

Die Funktion containsDigitOrLetter ist umsetzbar mit Hilfe einer simplen for-schleife ueber die Zeichen eines Wortes, wobei fuer jedes Wort ueberprueft wird, ob es eine Ziffer bzw. ein Buchstabe ist.

3 Implementierung / Umsetzung

Die soeben beschriebene Loesungsidee wurde in Java 8 implementiert.

Um den Text in eine Liste von Strings umzuwandeln wurde die Methode String#split mit dem regulaeren Ausdruck „\s+“ angewendet.

Fuer die Darstellung der Worte wurden die standard Strings benutzt, welche Methoden bieten, um auf Zeichen des Wortes zu zugreifen, oder Teilworte im Wort zu ersetzen.
Um zu bestimmen, ob ein Wort einen Buchstaben oder eine Ziffer enthaelt, wurde die Funktion Character#isLetterOrDigit benutzt.

4 Laufzeitkomplexitaet

Die Laufzeitkomplexitaet dieses Algorithmus liegt in $\mathcal{O}(n * m)$, wobei n die Anzahl der Woerter im Text ist und m die Laenge des Lueckensatzes darstellt.
Diese ergibt sich aus dem durchgehen aller zusammenhaengender Teillisten der Worte (O (n) viele) und dem ueberpruefen, ob die Teilliste (als Satz) den Lueckensatz erfuellt in O (m).

5 Beispiele

5.1 Beispieleingabe 1 - stoerung0

Eingabe: das _ mir _ _ _ vor

Ergebnisse:

1. das kommt mir gar nicht richtig vor,

5.2 Beispieleingabe 2 - stoerung1

Eingabe: ich muß _ clara _

Ergebnisse:

1. ich muß in clara verwandelt
2. ich muß doch clara sein,

5.3 Beispieleingabe 3 - stoerung2

Eingabe: fressen _ gern _

Ergebnisse:

1. fressen katzen gern spatzen?
2. fressen katzen gern spatzen?
3. fressen spatzen gern katzen?

5.4 Beispieleingabe 4 - stoerung3

Eingabe: das _ fing _

Ergebnisse:

1. das spiel fing an.
2. 'das publikum fing an,

5.5 Beispieleingabe 5 - stoerung4

Eingabe: ein _ _ tag

Ergebnisse:

1. ein sehr schöner tag!

5.6 Beispieleingabe 6 - stoerung5

Eingabe: wollen _ so _ sein

Ergebnisse:

1. wollen sie so gut sein,

5.7 Eigenes Beispiel 1

Eingabe: der _ fing _

Ergebnisse:

1. der fisch-lackei fing damit

Variable mit Bindestrich wird gefunden.

5.8 Eigenes Beispiel 2

Eingabe: _ ist ja _ _

Ergebnisse:

1. es ist ja schon ein
2. es ist ja kaum genug
3. es ist ja nur ein
4. flamingo ist ja ein vogel
5. 's ist ja 'n witz!

's und 'n werden als Variablen erkannt.

5.9 Eigenes Beispiel 3

Eingabe: _ published _ _
Ergebnisse:

1. originally published in 1869.

Zahl gefolgt von Satzzeichen wird als Variale erkannt.

5.10 Eigenes Beispiel 4

Eingabe: wäre _ _ morgens
Ergebnisse:

1. wäre 9 uhr morgens,

Ziffer als Wort wird erkannt.

6 Quellcode

6.1 Funktion, welche alle passenden Saetze fuer den Lueckensatz (format) findet

```
/**
 * Function for getting all sentences (list of words joined by spaces)
 * (of a list of words that match a given sentence format,
 * where any character that can occur in a word is defined by a given Set.)
 *
 * @param format          The (sentence) format
 * @param words           The list of words
 * @param wordCharacters A set of characters allowed in a word
 * @return All subsets of the list of words as sentence
 */
private static List<String> getAllMatches(String format, List<String> words,
                                          Set<Character> wordCharacters) {
    List<String> matches = new ArrayList<>();
    String[] formatArray = format.split("␣");

    // Go through the list of words and check if the current words
    // and the next words match the sentence format
    for (int i = 0; i < words.size() - formatArray.length + 1; i++) {
        // Keep track of the current word in the list with j
        int j = i;
        boolean match = true;

        // Go through the "words" of the sentence format
        for (String formatWord : formatArray) {
            String word = words.get(j);

            // Check if the current word (list at index j) matches the current format-word.
            // This is if
            // 1) The current format word is a variable ("_")
            // 2) The current word is of the form (not-a-word + current format word + no-a-word)
            if (formatWord.equals(VARIABLE) && containsCharacters(word, wordCharacters)) { // 1)
                j++;
            } else if (matches(word, formatWord, wordCharacters)) { // 2)
                j++;
            } else { // Otherwise, it's not a match
                match = false;
                break;
            }
        }

        // Add the current sublist as a sentence (joined by spaces) to all matches
        if (match) {
            matches.add(String.join("␣", words.subList(i, j)));
        }
    }

    return matches;
}
```

6.2 Funktion, welche bestimmt, ob ein bestimmtes Wort ein Wort im Lueckensatz „erfuellt“

```
/**
 * Function for checking if a string (s) matches another one (c).
 * That is the case if s is of the form
 * a_1ca_2 where a_1 and a_2 are words that only contain characters
 * that are NOT in a given set of characters
 *
 * @param string     The string s
 * @param contained  The string c
 * @param characters The set of permitted characters
 * @return If s is of the correct form.
 */
private static boolean matches(String string, String contained, Set<Character> characters) {
    // Return false if c isn't even contained in s
```

```
14          if (!string.contains(contained)) return false;
16          // Get s without c and check if the leftover characters are not contained in the given set
            string = string.replaceFirst(contained, "");
18
            for (char ch : string.toCharArray()) {
20              if (characters.contains(ch)) return false;
            }
22
            return true;
24      }
```