Aufgabe 1: Störung (lib/task1.dart)

Problem: Sätze aus "Alice im Wunderland" bestehen teilweise aus Lücken. Diese Lücken müssen durch ein Computerprogramm ergänzt werden. Zuerst habe ich folgendes festgelegt: Die Lücken entsprechen Wörtern aus dem Text "Alice im Wunderland". Diese Wörter werden ohne ihre Satzzeichen ersetzt. Die Wörter müssen also von ihren umschließenden Satzzeichen getrennt in die Lücken eingefügt werden. Im Beispiel: [das _ mir _ _ _ vor -> das kommt mir gar nicht richtig vor] werden schließlich auch die Satzzeichen weggelassen.
Lösungsidee: Für jedes feststehende Wort werden die möglichen Indexe des Wortes innerhalb des gesamten Textes festgelegt. Die möglichen Indexe der jeweiligen Wörter werden verlichen und es wird mit Berücksichtigung der Lücken überprüft, ob Indexe aufeinander folgen. Wenn diese Reihe / Kette auf jeweils einen der Indexe der festehenden Wörter zutrifft. So wurde der Satz im Text gefunden und kann nun ergänzt werden.

Um unabhängig von Satzzeichen und Formatierung zu sein, wird der Gesamttext sowie der Lückentext mittelt der Methode `textToWords()` in eine Liste aus Strings unterteilt. Diese Methode

```
List<String> textToWords(String text) {
    final lines = text.split("\n");
    List<String> words = [];
    for (var element in lines) {
      words.addAll(element.split(" ").map((e) => e.toLowerCase().replaceAll(RegExp(r"
[^a-z0-9üäöß_\n]"), "")).toList());
    }
    return words;
}
```

teilt den Text zunächst (druch `split("\n")`) in Zeilen ein. Anschließend wird jede Zeile durch `split(" ")` in Wörter aufgeteilt. Diese Wörter enthalten allerdings noch Satzzeichen (etc.). Um die Satzzeichen zu entfernen, werden, nachdem das Wort in kleinbuchstaben umtransformiert wurde, alle Characters des Strings, welche die Regular Expression `RegExp(r"[^a-z0-9üäöß_\n]"` inkludiert, gelöscht, sodass ein Wort ohne Satzzeichen entsteht. Die RegExp trifft auf alle Characters zu, die nicht im Bereich a-z, 0-9 liegen oder nicht den Zeichen üäöß_\n entsprechen.

Das Entscheidende an diesem Lösungsweg ist die `List<int> alleIndexe` Liste (l. 204-295). Als nächstes iteriert das Programm durch die Liste des Wörter des Lösungssatzes. Falls es sich um ein Wort und nicht um eine Lücke handelt, so wird von der alleIndexe Liste Gebrauch gemacht.
Wenn es sich um das erste feststehende Wort handelt, so ist die Liste leer. Es werden alle Indexe also alle Stellen, wo das festehende Wort in der Liste der Wörter des Gesamttextes vorkommt, ermittelt (l. 219-245). Da die Liste leer ist, werden nun alle Indexe der Liste hinzugefügt.
Bei der nächsten Iteration und dem nächsten feststehenden Wort werden wieder die Indexe des aktuellen Wortes bestimmt. Diese werden allerdings nun mit den Werten aus der Liste (alleIndexe) verglichen. Falls ein Index 1 größer ist als der alte Index, so wird der alte Index durch den neue Index in der Liste ersetzt. Wenn dies nicht der Fall ist, wird der Index in der Liste mit -1 ersetzt.
In dem Fall, das es sich bei dem aktuellen Wort um eine Lücke handelt, werden alle Indexe (von alleIndexe) um 1 erhöht.

Beispiel:
Eingabe: das _ mir _ _ _ vor Der String "das" kommt an vielen verschiedenen Position in aliceWoerter vor.
Alle Positionen bzw. Indexe werden nun in alleIndexe gelistet. Z.B. an Index 100 und Index 500
Als nächstes kommt die Lücke _, hier werden alle vorherigen Indexe um 1 erhöht. -> Index 101 und Index 501. Der String "mir" wird nun gesucht. Er hat bspw. die Indexe 102 und 300 in aliceWoerter.
Vergleicht man nun die Liste (alleIndexe) mit den Indexen von "mir" so wird deutlich, dass der Index 102 auf den Index 101 (aus alleIndexe) folgt. Demnacht wird der Index 101 mit 102 aktualisiert, während der Index 501 verworfen wird.
Bei den nöchsten 3 Lücken erhöhen sich alleIndexe wieder um 3, d.h. Index 102 -> Index 105. Der String "vor" wird nun gesucht, falls dieser einen Index von 106 in aliceWoerter aufweist, so ist die Reihe komplett und die Endposition der Eingabe steht fest.

```
    List<int> alleIndexe = [];
    /// Mit einem For loop wird durch die woerter Liste iteriert.
    for (var i = 0; i < woerter.length; i++) {
      String eingabeWort = woerter[i];
      /// aliceWoerter wird nach eingabeWort durchsucht, falls das eingabeWort keine
Lücke ist
      if(eingabeWort != "_") {
        int indexWortInAlice = aliceWoerter.indexOf(eingabeWort);
        if(indexWortInAlice == -1){
          /// wenn indexWortInAlice == -1, so befindet sich das eingabeWort nicht in
dem Text
          /// demnach ist die Eingabe nicht korrekt, da sich jedes eingabeWort im Text
befinden muss.
          logResult("Eingabe ist nicht korrekt, da nicht alle sichtbaren Wörter sich
im Text befinden.");
          return;
        }
        /// ueberpruefe ob es das erste sichtbare Wort ist, also alleIndexe leer
ist...
        if(alleIndexe.isEmpty) {
          /// da aliceWoerter mehrmals das eingabeWort beinhaltet kann, und bei
indexOf nur der erste Wert
          /// als Index in der Liste ausgegeben wird, müssen alle anderen Indexe auch
gefunden bzw. überprüft
          /// werden.
          alleIndexe.add(indexWortInAlice);
          /// currentIndex ist der akutelle Wert des Indexes von eingabeWort in
aliceWoerter
          int currentIndex = indexWortInAlice;
          /// in dieser while-Scheleife werden alle Indexe von eingabeWort in
aliceWoerter festgestellt,
          /// wenn ein Index gefunden wurde, anfangs indexWortInAlice, so wird dieser
verwendet um den nächsten
          /// Index des eingabeWorts zu bestimmen. Der zweite Parameter von indexOf
gibt an, ab welchem Index die Suche
```

```
          /// nach dem nächsten Index startet. Der Start-Index liegt hier bei
currentIndex + 1, sodass nur ein neuer
          /// Index gefunden werden kann. Ist dieser neue Index -1, so wurden alle
Vorkommnisse von eingabeWort in
          /// aliceWoerter gefunden, und in alleIndexe aufgelistet.
          while(true) {
            int naechsterIndexWortInAlice = aliceWoerter.indexOf(eingabeWort,
currentIndex + 1);
            if(naechsterIndexWortInAlice == -1) {
              break;
            }
            alleIndexe.add(naechsterIndexWortInAlice);
            currentIndex = naechsterIndexWortInAlice;
          }
        }else{
          List<int> neueAlleIndexeProWort = [];
          neueAlleIndexeProWort.add(indexWortInAlice);
          /// copy (siehe oben)
          int currentIndex = indexWortInAlice;
          while(true) {
            int naechsterIndexWortInAlice = aliceWoerter.indexOf(eingabeWort,
currentIndex + 1);
            if(naechsterIndexWortInAlice == -1) {
              break;
            }
            neueAlleIndexeProWort.add(naechsterIndexWortInAlice);
            currentIndex = naechsterIndexWortInAlice;
          }
          /// nun wird überprüft, welche Indexe aus neueAlleIndexeProWort 1 größer als
ein Wert von alleIndexe sind, denn eine Wort folgt schließlich auf das nächste
(index+1)
          for (var i = 0; i < alleIndexe.length; i++) {
            if(alleIndexe[i] >= 0){
              bool neuerWert = false;
              for (var j = 0; j < neueAlleIndexeProWort.length; j++) {
                if(neueAlleIndexeProWort[j] == alleIndexe[i] + 1){
                  /// wenn der neue Index 1 größer ist als der alte Index (Beispiel
oben, 101 auf 102)
                  /// -> alleIndexe wird bei i, der neue Index zugeschrieben
                  alleIndexe[i] = neueAlleIndexeProWort[j];
                  neuerWert = true;
                  break;
                }
              }
              /// wenn nichts verändert wurde, ist dieser Index falsch und wird mit -1
ersetzt.
              if(!neuerWert){
                alleIndexe[i] = -1;
              }
            }
          }
        }
```

```
      }else{
        /// wenn Indexe bereits in alleIndexe eingetragen wurden, dann wird jedem
Index ein Wert von 1
        /// hinzugefügt, entsprechend der einzelnen Lücke.
        if(alleIndexe.isNotEmpty){
          for (var i = 0; i < alleIndexe.length; i++) {
            /// jedem Index in alleIndexe wird die Lücke (als +1) hinzugefügt.
            /// außer wenn der zugeordnete Index zu große für aliceWoerter ist oder
der Index mit -1 markiert wurde (also falsch ist)
            if(alleIndexe[i] + 1 >= aliceWoerter.length || alleIndexe[i] == -1){
              continue;
            }
            alleIndexe[i] = alleIndexe[i] + 1;
          }
        }
      }
    }
```

Nun werden alle Werte mit -1 aus der Liste gelöscht (l.301-306).

Die Liste der Wörter des Lösungssatzes wird von hinten nach vorne iteriert. Dabei wird für jeden ermittelten Endindex des Satzes innerhalb der Liste der Wörter des Gesamttextes ein Satz erstellt. Die Lösungssätze befinden sich schließlich in `List<String> valideLoesungen`:

```
    List<String> valideLoesungen = List.filled(valideIndexe.length, "");
    for (var i = woerter.length - 1; i >= 0; i--) {
      if(woerter[i] == "_"){
        for (var j = 0; j < valideIndexe.length; j++) {
          /// String der jeweiligen Lücke in woerter wird bestimmt, indem man
rückwärts durch die Liste der Eingabewörter iteriert.
          /// Der Index für die jeweilige Lücke beträgt nun valideIndexe[j] -
(woerter.length - i - 1)
          String? lueckeX = aliceWoerter[valideIndexe[j] - (woerter.length - i - 1)];
          /// Es werden gleichzeitig mehrere Lösungen bereitsgestellt, da es mehrere
korrekte Endindexe geben kann.
          /// trim() wird gecallt, um die Lösung ins optimale Format ohne überflüssige
Leerzeichen zu bringen.
          valideLoesungen[j] = "$lueckeX ${valideLoesungen[j].trim()}";
        }
      }else{
        for (var j = 0; j < valideIndexe.length; j++) {
          /// hier muss die Lücke nicht aus aliceWoerter erschlossen werden, da sich
das Wort bereits in der Eingabe befindet.
          valideLoesungen[j] = "${woerter[i]} ${valideLoesungen[j]}";
        }
      }
    }
```

Am Ende werden die validen Lösungen durch `join(",")` zu einem String verbunden und an die UI als `result` weitergegeben.

Beispiel (stoerung0.txt):

- Als Eingabe wird: "das _ mir _ _ _ vor" verwendet.
- Die Eingabe wird nun in eine Liste aus Strings aufgeteilt: ["das", "", "mir", "", ", "", "vor"]
- Nach der 1. Iteration (durch die Liste an Wörtern) ist folgende Liste enstatnden: [86, 291, 339, 478, 683, 720, 839, 841, 929, 970, 1044, 1307, 1355, 1410, 1438, 1453, 1482, 1730, 1783, 1908, 1936, 2043, 2125, 2462, 2507, 2534, 2564, 2826, 2879, 2882, 2988, 3118, 3230, 3233, 3289, 3307, 3423, 3425, 3549, 3580, 3657, 3705, 3756, 3800, 3880, 3948, 3989, 4005, 4173, 4177, 4271, 4445, 4555, 4589, 4840, 5172, 5492, 5554, 5635, 5656, 5796, 6000, 6034, 6045, 6075, 7634, 7912, 7953, 7981, 8064, 8334, 8337, 8449, 8736, 8746, 8823, 8859, 8989, 9162, 9281, 9284, 9294, 9311, 9326, 9375, 9389, 9424, 9486, 9507, 9543, 9664, 9765, 9792, 10016, ...]
- Nach der 2. Iteration wurde jeder Wert in der Liste um 1 erhöht, da es sich um eine Lücke handelt "_".
- Nach der 3. Iteration ("mir") sieht die Liste wie folgt aus: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 3707, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 10207, -1, ...]
- Nach der letzten Iteration ("vor") bleiben in der Liste nur -1 und 3711 übrig.
- Der Index von dem Wort "vor" im Gesamttext ist also 3711.
- D.h. der Lösungssatz besteht aus den Indexen: [3705, 3706, 3707 ,3708 ,3709, 3710, 3711].
- Der Satz wird mit den Indexen ergänzt. Damit entsteht der Satz: "das kommt mir gar nicht bekannt vor" Das Programm kann genutzt werden um eigenen Lückensätze oder alle anderen Beispielsätze zu testen.