Aufgabe 1: Wörter aufräumen

Team-ID: 00129

Team: PIE_Team

Bearbeiter dieser Aufgabe:
Jonathan Busch

5. November 2020

Inhaltsverzeichnis

Lösungsidee

Ein Worträtsel lässt sich als bipartiter Graph darstellen. Die erste Knotenmenge besteht dabei aus den Wörtern; die andere Menge enthält die zu füllenden Lücken. Eine Kante besteht zwischen zwei Knoten, wenn das Wort in die Lücke geschrieben werden kann, d. h. wenn beide die gleiche Länge haben und wenn vorgeschriebene Buchstaben passen. Das Problem ist nun darauf beschränkt, den Graphen aufzustellen und ein perfektes Matching zu finden.

Aufstellen des Graphen

Der offensichtlichste Ansatz besteht darin, einfach für jedes Wort und jede Lücke zu überprüfen, ob das Wort in die Lücke passt. Dieser Ansatz würde in $O(n^2)$ laufen. Das lässt sich allerdings optimieren, indem man Wörter und Lücken nach Länge sortiert. Dann werden nur noch Wörter und Lücken verglichen, die die gleiche Länge haben. Im Worst-Case (alle Wörter haben die gleiche Länge) braucht diese Optimierung trotzdem noch $O(n^2)$, dieser Fall kommt aber praktisch nicht bzw. mit nur so wenigen Wörtern vor, dass die Laufzeit sowieso vernachlässigbar ist. Im günstigsten Fall (jede Länge kommt maximal ein Mal vor) braucht die Optimierung lineare Laufzeit (O(n log n) allerdings für die Sortierung).

Struktur des Graphen

Der Graph ist bipartit und enthält Knoten mit dem Grad 1, die ohne Risko eines Fehlschlags zugeordnet werden können.

Beweis:

Der Graph ist bipartit, d. h. er enthält nur Zyklen mit gerader Länge. Außerdem existiert genau ein perfektes Matching.

Angenommen, der Graph enthält keinen Knoten mit Grad 1, d. h. jeder Knoten hat mindestens Grad 2. Grad 0 kann nicht vorkommen, denn sonst würde kein perfektes Matching existieren.

Ziel ist nun, einen Zyklus im Graphen zu finden, der abwechselnd Kanten aus dem Matching und Kanten, die nicht im Matching enthalten sind, enthält, zu finden. Daraus kann ein anderes perfektes Matching konstruiert werden, indem man die Matchingkanten im Zyklus durch die Nicht-Matchingkanten im Zyklus ersetzt. Damit wäre dann das perfekte Matching nicht mehr eindeutig und der Graph muss Knoten mit dem Grad 1 enthalten.

So einen Zyklus kann man finden, indem man bei einem beliebigen Knoten startet und abwechselnd Matchingkanten und Nicht-Matchingkanten wählt. Matchingkanten existieren bei jedem Knoten, da das Matching perfekt ist; Nicht-Matchingkanten existieren bei jedem Knoten, da jeder Knoten mindestens Grad 2 hat. Es können also immer weitere Kanten gewählt werden. Da der Graph endlich ist, stößt man zwangsläufig auf einen Knoten, der schon im bisherigen Kantenzug enthalten ist. Dann hat man einen Zyklus gefunden, der eine gerade Länge hat (da der Graph bipartit ist) und abwechselnd Matchingkanten und Nicht-Matchingkanten enthält, da man die Kanten entsprechend gewählt hat.

Das garantiert die Existenz eines weiteren perfekten Matchings, weshalb die Annahme, der Graph hätte keine Knoten mit Grad 1, falsch sein muss.

Finden des Matchings

Dem Beweis zufolge existieren im Graphen Knoten mit dem Grad 1, d. h. Knoten, deren einzige Kante zum perfekten Matching gehören muss. Die Kanten solcher Knoten können sofort zugeordnet werden. Werden beide Knoten nun aus dem Graphen entfernt, entsteht ein kleinerer Graph mit einem ebenfalls eindeutigen perfekten Matching. Die Bedingung, dass der Graph Knoten mit Grad 1 enthält, gilt also immernoch, also gibt es immer Knoten, die eindeutig zugeordnet werden können.

Ein Problem dieses Ansatzes liegt in mehrfach vorkommenden Wörtern. Diese widerlegen (theoretisch) zwangsläufig die Annahme, es gebe nur ein perfektes Matching. Das Problem kann aber gelöst werden, indem man festlegt, dass sich immer nur ein gleiches Wort gleichzeitig im Graphen befindet; sobald dieses Wort zugeordnet wird, wird ein anderes (gleiches) Wort eingefügt.

Umsetzung

Das Programm wird in Java implementiert.

Datentypen

Ein Wort lässt sich am einfachsten als `String` darstellen. Für die Umsetzung der Lösung des Problems mit mehreren Wörten ist es allerdings hilfreich, für jedes Wort zu speichern, wie oft es vorkommt, statt es mehrfach in einer Liste zu speichern. Ein Wort hat als Eigenschaften also einen `String` und einen `int`.

Eine Lücke hat als Eigenschaften die Länge und einige vorgegebene Buchstaben. Die Länge wird als 32-Bit-Ganzzahl gespeichert. Die vorgegebenen Buchstaben werden vom Programm als `char`-Array dargestellt; die Positionen, an denen die Buchstaben vorgegeben sind, werden parallel in einem `int`-Array gespeichert.

Repräsentation des Graphen

Der Graph wird als Adjazenzliste dargestellt. Diese wird mittels eines zweidimensionalen `int`-Arrays realisiert. Der Algorithmus löscht häufig Kanten aus dem Graphen, weshalb in einem zweiten zweidimensionalen `int`-Array zu jeder Kante der Index der Kante beim anderen Knoten gespeichert ist. So kann eine Kante beim Löschen aus dem Graphen bei beiden Knoten zuerst an das Ende der Liste getauscht werden, weshalb das Löschen nur konstante Laufzeit benötigt.

Liste mit Blättern (eindeutigen Zuordnungen)

Es wird keine Liste oder Schlange verwendet, sondern ein Stack. Es spielt keine Rolle, in welcher Reihenfolge die Blätter zugeordnet werden. Ein Stack ist effizienter (konstante Laufzeit für Push und Pop, außerdem speichereffizient) und deshalb wird ein Stack verwendet.

Beispiele

```
$ java A1Main
_h __, _a_ __r ___e __b___!
Arbeit eine für je Oh was
__ matches je
__ matches Oh
_h matches Oh
__r matches für
_a_ matches was
___e matches eine
__b___ matches Arbeit
Matching __b___ and Arbeit
Matching ___e and eine
Matching _a_ and was
Matching __r and für
Matching _h and Oh
Matching __ and je
```

```
Oh je, was für eine Arbeit!

$ java A1Main
_m ___a__ ____e __s ____e____ __________. D__ _a____ _i___ __u__ ____ ___e__ _n_ _u___
___________ _l_ __h____ __ ___ _____h_ ________ _____e___.
Am in als das Das die und sehr Leute viele wurde wurde Anfang machte wütend falsche
Schritt Richtung angesehen Universum erschaffen allenthalben
__ matches Am
_m matches Am
__ matches in
___ matches als
__s matches als
_l_ matches als
___ matches das
__s matches das
___ matches Das
__s matches Das
D__ matches Das
___ matches die
___ matches und
_n_ matches und
____ matches sehr
____e matches Leute
__u__ matches Leute
____e matches viele
_i___ matches viele
____e matches wurde
_u___ matches wurde
___a__ matches Anfang
_a____ matches machte
___e__ matches wütend
_____h_ matches falsche
__h____ matches Schritt
________ matches Richtung
_____e___ matches angesehen
____e____ matches Universum
__________ matches erschaffen
___________ matches allenthalben
Matching ___________ and allenthalben
Matching __________ and erschaffen
Matching ____e____ and Universum
Matching _____e___ and angesehen
Matching ________ and Richtung
Matching __h____ and Schritt
Matching _____h_ and falsche
Matching _a____ and machte
Matching ___e__ and wütend
Matching ___a__ and Anfang
Matching _u___ and wurde
Matching _i___ and viele
Matching __u__ and Leute
Matching ____e and wurde
Matching ____ and sehr
Matching D__ and Das
Matching _n_ and und
Matching _l_ and als
Matching __s and das
```

```
Matching ___ and die
Matching _m and Am
Matching __ and in
Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde
allenthalben als Schritt in die falsche Richtung angesehen.

$ java A1Main
__s __e___ _a___ e____ _____n_ _u_ _________ ____m__ ________, _a__ _r s___ _n _____m
___t _u ____m __________ ______e___ _______e__.
er in zu Als aus Bett fand sich einem eines Samsa Gregor seinem Morgens Träumen
erwachte unruhigen Ungeziefer verwandelt ungeheueren
_r matches er
_n matches in
_u matches zu
__s matches Als
_u_ matches aus
___t matches Bett
_a__ matches fand
s___ matches sich
____m matches einem
e____ matches einem
e____ matches eines
_a___ matches Samsa
__e___ matches Gregor
_____m matches seinem
_____n_ matches Morgens
____m__ matches Träumen
________ matches erwachte
_________ matches unruhigen
______e___ matches Ungeziefer
_______e__ matches verwandelt
__________ matches ungeheueren
Matching __________ and ungeheueren
Matching ______e___ and Ungeziefer
Matching _______e__ and verwandelt
Matching _________ and unruhigen
Matching ________ and erwachte
Matching ____m__ and Träumen
Matching _____n_ and Morgens
Matching __e___ and Gregor
Matching _____m and seinem
Matching _a___ and Samsa
Matching ____m and einem
Matching e____ and eines
Matching s___ and sich
Matching _a__ and fand
Matching ___t and Bett
Matching _u_ and aus
Matching __s and Als
Matching _u and zu
Matching _r and er
Matching _n and in
Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem
Bett zu einem ungeheueren Ungeziefer verwandelt.

$ java A1Main
```

```
__________ __t d__ ____e_______ __n _e_ _______________ __________g, __e________,
___a_________ _n_ ___r_______ __n ___o_________, _________ _e_ ______t______
___a_________ _i_ _______________.
der der die ist mit und von von besonders Informatik Darstellung Speicherung
Übertragung Verarbeitung Verarbeitung Wissenschaft Informationen automatischen
systematischen Digitalrechnern
_e_ matches der
_e_ matches der
d__ matches der
_i_ matches die
d__ matches die
__t matches ist
__t matches mit
_i_ matches mit
_n_ matches und
__n matches von
__n matches von
_________ matches besonders
__________ matches Informatik
__________g matches Darstellung
__________g matches Speicherung
__e_________ matches Speicherung
__________g matches Übertragung
___r_______ matches Übertragung
__e_________ matches Übertragung
___a_________ matches Verarbeitung
___a_________ matches Verarbeitung
____e_______ matches Wissenschaft
______t______ matches automatischen
___o_________ matches automatischen
___o_________ matches Informationen
_______________ matches systematischen
_______________ matches Digitalrechnern
Matching _______________ and Digitalrechnern
Matching _______________ and systematischen
Matching ______t______ and automatischen
Matching ___o_________ and Informationen
Matching ___a_________ and Verarbeitung
Matching ___a_________ and Verarbeitung
Matching ____e_______ and Wissenschaft
Matching ___r_______ and Übertragung
Matching __e_________ and Speicherung
Matching __________g and Darstellung
Matching __________ and Informatik
Matching _________ and besonders
Matching _n_ and und
Matching _e_ and der
Matching _e_ and der
Matching d__ and die
Matching _i_ and mit
Matching __t and ist
Matching __n and von
Matching __n and von
Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung,
Verarbeitung und Übertragung von Informationen, besonders der automatischen
Verarbeitung mit Digitalrechnern.
```

```
$ java A1Main
__a _____n _______t _n _i___ _e_________ __s d__ _____e__ __d _i____ e__ __t___. _s _s_
__n_ ____e __n ______n ___e___, _i_ i_ _i_ ___h____ __________e __b_____ ___d__ s_____,
_o __s_ s__ _i__ ___t___ _e________ _r_____. __________n _n_ _a_________ s____ e_____
_______b__ ___d ____n ______e___.
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
_s matches Es
_n matches in
i_ matches in
_o matches so
__s matches aus
d__ matches der
_i_ matches die
_i_ matches die
d__ matches die
__n matches ein
_i_ matches ein
_i_ matches ein
e__ matches ein
_s_ matches ist
__a matches Opa
_i_ matches sie
_i_ matches sie
s__ matches sie
__d matches und
_n_ matches und
__n matches von
__s_ matches dass
__n_ matches eine
_i__ matches eine
___d matches sind
__n_ matches sind
_i__ matches sind
_i___ matches einer
____e matches Liste
_i___ matches Liste
____n matches schon
s____ matches schon
____e matches sowie
s____ matches sowie
_i____ matches einige
e_____ matches einige
___d__ matches findet
_i____ matches findet
_____n matches Jürgen
__t___ matches Rätsel
_____n matches sollen
s_____ matches sollen
_____n matches werden
___d__ matches werden
______n matches ergeben
___e___ matches ergeben
_r_____ matches ergeben
______n matches gegeben
```

```
___e___ matches gegeben
___t___ matches lustige
______n matches Wörtern
___t___ matches Wörtern
_____e__ matches Apotheke
_______t matches blättert
_____e__ matches blättert
_______t matches gebracht
__b_____ matches gebracht
___h____ matches richtige
_______b__ matches Buchstaben
_e_______ matches Geschichte
_______b__ matches vorgegeben
______e___ matches vorgegeben
__________n matches Leerzeichen
_e_________ matches Leerzeichen
__________e matches Reihenfolge
_e_________ matches Reihenfolge
__________n matches Satzzeichen
_a_________ matches Satzzeichen
_e_________ matches Zeitschrift
Matching _a_________ and Satzzeichen
Matching __________n and Leerzeichen
Matching __________e and Reihenfolge
Matching _e_________ and Zeitschrift
Matching _e________ and Geschichte
Matching ______e___ and vorgegeben
Matching _______b__ and Buchstaben
Matching __b_____ and gebracht
Matching _______t and blättert
Matching _____e__ and Apotheke
Matching ___h____ and richtige
Matching _r_____ and ergeben
Matching ___e___ and gegeben
Matching ______n and Wörtern
Matching ___t___ and lustige
Matching s_____ and sollen
Matching e_____ and einige
Matching _i____ and findet
Matching ___d__ and werden
Matching _____n and Jürgen
Matching __t___ and Rätsel
Matching ____n and schon
Matching s____ and sowie
Matching ____e and Liste
Matching _i___ and einer
Matching __s_ and dass
Matching ___d and sind
Matching __n_ and eine
Matching _i__ and eine
Matching s__ and sie
Matching e__ and ein
Matching _i_ and die
Matching _i_ and die
Matching d__ and der
Matching __n and von
Matching _s_ and ist
```

```
Matching _n_ and und
Matching __s and aus
Matching __d and und
Matching __a and Opa
Matching i_ and in
Matching _s and Es
Matching _o and so
Matching _n and in
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.
```

Quellcode

```
...
public final class A1Main {
  public static void main(String[] args) {
    List<Gap> sentenceOrderGapList = new ArrayList<>();
    List<Gap> gapList = new ArrayList<>();
    List<String> wordList = new ArrayList<>();
    {
      // Einleseprozedur
      ...
    }
    // Satzordnung der Lücken merken
    sentenceOrderGapList.addAll(gapList);
    // Wörter und Lücken nach Länge sortieren
    gapList.sort((a, b) -> {
      int lc = Integer.compare(a.length, b.length);
      return lc == 0 ? a.gapString.compareToIgnoreCase(b.gapString) : lc;
    });
    wordList.sort((a, b) -> {
      int lc = Integer.compare(a.length(), b.length());
      return lc == 0 ? a.compareToIgnoreCase(b) : lc;
    });
    // Mehrfachvorkommen von Wörtern anders speichern
    List<Word> words = new ArrayList<Word>();
    for (int i = 0; i < wordList.size(); ++i) {
      if (!words.isEmpty() && wordList.get(i).contentEquals(words.get(words.size() - 1).word)) {
        ++words.get(words.size() - 1).count; // jedes Wort nur einmal, dafür zählen
      } else {
        words.add(new Word(wordList.get(i), 1));
      }
    }
    // bipartiter Graph
    int gapIndexOffset = words.size(); // Startindex der Lücken im Array
    int[][] graph = new int[gapList.size() + words.size()][1];
    int[][] otherEdgeIndex = new int[graph.length][1]; // Index der Kante beim anderen Knoten
    int[] graphSize = new int[graph.length]; // einfache ArrayList-Struktur implementieren
    {
      int length;
      int startPosWords = 0;
      int endPosWords;
      int startPosGaps = 0;
      int endPosGaps;
      // Wörter und Lücken nach Längen sortiert durchgehen
      while (startPosWords < words.size()) {
        length = words.get(startPosWords).word.length();
        // Endindex zu dieser Länge ermitteln
```

```
      for (endPosWords = startPosWords;
          endPosWords < words.size() && length == words.get(endPosWords).word.length(); +
                        +endPosWords)
        ;
      for (endPosGaps = startPosGaps;
          endPosGaps < gapList.size() && length == gapList.get(endPosGaps).length; +
                        +endPosGaps)
        ;
      // alle Lücken und Wörter auf Matches überprüfen
      for (int i = startPosWords; i < endPosWords; ++i) for (int j = startPosGaps; j <
                        endPosGaps; ++j) {
        // wenn Lücke und Wort passen:
        if (gapList.get(j).matches(words.get(i).word)) {
          System.out.println(gapList.get(j).gapString + " matches " + words.get(i).word);
          // Grapharrayindex der Lücke
          int k = j + gapIndexOffset;
          // Arrays ggf. erweitern
          if (graphSize[i] >= graph[i].length) {
            graph[i] = Arrays.copyOf(graph[i], graphSize[i] << 1);
            otherEdgeIndex[i] = Arrays.copyOf(otherEdgeIndex[i], graphSize[i] << 1);
          }
          if (graphSize[k] >= graph[k].length) {
            graph[k] = Arrays.copyOf(graph[k], graphSize[k] << 1);
            otherEdgeIndex[k] = Arrays.copyOf(otherEdgeIndex[k], graphSize[k] << 1);
          }
          graph[i][graphSize[i]] = k; // neue Kante eintragen
          graph[k][graphSize[k]] = i;
          otherEdgeIndex[i][graphSize[i]] = graphSize[k]; // Index beim anderen Knoten
                        eintragen
          otherEdgeIndex[k][graphSize[k]] = graphSize[i];
          ++graphSize[i]; // Größe aktualisieren
          ++graphSize[k];
        }
      }
      startPosWords = endPosWords;
      startPosGaps = endPosGaps;
    }
  }
  printGraph(graph, graphSize, gapList, words);
  // Stack mit Blättern
  int[] leafs = new int[1];
  int stackSize = 0;
  for (int i = 0; i < graph.length; ++i) {
    if (graphSize[i] == 1) {
      // ggf. Array erweitern
      if (leafs.length <= stackSize) leafs = Arrays.copyOf(leafs, leafs.length << 1);
      // Blatt eintragen
      leafs[stackSize] = i;
      ++stackSize;
    }
  }
  boolean[] matched = new boolean[graph.length];
  while (stackSize > 0) {
    // Blatt aus Stack extrahieren
    --stackSize;
    int leafElement = leafs[stackSize];
    // schon gematcht?
    if (matched[leafElement]) continue;
    // zugehöriges Element ermitteln
    int matchedElement = graph[leafElement][0];
    // in Wort und Lücke teilen
    int gap = Math.max(leafElement, matchedElement);
    int gapIndex = gap - gapIndexOffset;
```

```
      int wordIndex = Math.min(leafElement, matchedElement);
      // matchen: Wort in die Lücke eintragen
      gapList.get(gapIndex).matchedWord = words.get(wordIndex).word;
      // Wortdaten anpassen
      --words.get(wordIndex).count;
      matched[gap] = true; // matched-Array aktualisieren
      matched[wordIndex] = words.get(wordIndex).count == 0;
      System.out.println("Matching " + gapList.get(gapIndex).gapString + " and " +
                              words.get(wordIndex).word);
      // Blatt aus Graph entfernen:
      int ni = otherEdgeIndex[leafElement][0];
      --graphSize[matchedElement];
      // Blatt ans Ende tauschen (Ende zum Blatt kopieren, da das Blatt sowieso nicht mehr
                              gebraucht wird)
      graph[matchedElement][ni] = graph[matchedElement][graphSize[matchedElement]];
      otherEdgeIndex[matchedElement][ni] = otherEdgeIndex[matchedElement]
                              [graphSize[matchedElement]];
      // Indexliste auf der anderen Seite aktualisieren
      otherEdgeIndex[graph[matchedElement][ni]][otherEdgeIndex[matchedElement][ni]] = ni;
      --graphSize[leafElement]; // = 0
      // wenn entweder Lücke? oder (Wort? && Wortvorkommen == 0) -> anderen Knoten auch entfernen
      // bedeutet: das Wort muss dann übrigbleiben, wenn es noch woanders vorkommt
      if (matchedElement >= gapIndexOffset || words.get(wordIndex).count == 0) {
        // alle Kanten entfernen
        for (int i = 0; i < graphSize[matchedElement]; ++i) {
          int other = graph[matchedElement][i];
          int j = otherEdgeIndex[matchedElement][i];
          --graphSize[other];
          // ans Ende tauschen
          graph[other][j] = graph[other][graphSize[other]];
          otherEdgeIndex[other][j] = otherEdgeIndex[other][graphSize[other]];
          // Indexliste auf der anderen Seite aktualisieren
          otherEdgeIndex[graph[other][j]][otherEdgeIndex[other][j]] = j;
          // neues Blatt entstanden?
          if (graphSize[other] == 1) {
            // ggf. Array erweitern
            if (leafs.length <= stackSize) leafs = Arrays.copyOf(leafs, leafs.length << 1);
            // und neues Blatt hinzufügen
            leafs[stackSize] = other;
            ++stackSize;
          }
        }
        graphSize[matchedElement] = 0;
      }
    }
    // Ergebnis ausgeben!
    for (int i = 0; i < sentenceOrderGapList.size(); ++i) {
      System.out.print(sentenceOrderGapList.get(i).sentencePart());
    }
    System.out.println();
  }

  // Zuordnungsgraph visuell ausgeben (mit GraphViz - nur wenn installiert)
  private static void printGraph(int[][] graph, int[] size, List<Gap> gaps, List<Word> words)
                              {...}

  // Lücken-Datentyp
  public static class Gap {
    public final boolean firstGap;     // Satzanfang?
    public final int     length;       // Länge
    public final int[]   positions;    // Positionen der Buchstaben
    public final char[]  chars;        // Buchstaben
    public final String  followingSeq; // folgende Satzzeichen
```

```
    private final String gapString;    // String-Repräsentation der Lücke (Debugging)

    public String matchedWord; // zugeordnetes Wort

    public Gap(boolean fg, int l, int[] p, char[] c, String f) {...}

    public boolean matches(String str) { // Überprüfung von Lücke-Wort-Matches
      boolean matches = str.length() == length && (!firstGap ||
                            Character.isUpperCase(str.charAt(0)));
      for (int i = 0; i < positions.length && matches; ++i) matches = str.charAt(positions[i]) ==
                            chars[i];
      return matches;
    }

    public String sentencePart() { // Satzteil für die Ausgabe am Ende
      return (matchedWord == null ? gapString : matchedWord) + followingSeq;
    }

    public String toString() { // Informationen über die Lücke
      return length + ":" + Arrays.toString(positions) + ":" + Arrays.toString(chars) + ":" +
                            followingSeq
          + ":" + firstGap;
    }
  }

  // Wort-Datentyp
  public static class Word {
    public final String word;  // Wort
    public int          count; // Häufigkeit

    public Word(String w, int c) {...}

    public String toString() { // Informationen über das Wort
      return word + ": " + count;
    }
  }
}
```