Junioraufgabe 1 - Reimerei

Team-ID: 00166
Team-Name: ez

Bearbeitet von Florian Bange

21. November 2022

Inhaltsverzeichnis

1 Definition des Problems

Um das Problem aus der Aufgabe zu loesen, wird es nun zunaechst formell defniniert.

1.1 Formelle Definition von Worten

Sei A eine endliche Menge an Zeichen (oder Buchstaben oder Symbole) (bzw. ein Alphabet).
Ein Wort w ist eine Zeichenkette der Zeichen des Alphabets:

$$w = a_1 a_2 \dots a_n$$

mit $n \in \mathbb{N}_0$ und $a_i \in A$ $(1 \leq i \leq n)$.
Die Laenge eines Wortes w ist $|w| = |a_1 a_2 \dots a_m| = m$. Wobei das Wort mit Laenge 0 das leere Wort ϵ mit

$$|\epsilon| = 0$$

ist.
Desweiteren sei

$$A^* := \{w \mid w = a_1 a_2 ... a_k : a_i \in A; k \in \mathbb{N}_0\}$$

die Menge aller Woerter ueber A, inklusive ϵ.

1.2 Die maszgebliche Vokalgruppe

Um zu defnieren, wann sich zwei Worte reimen, wird zu naechst die maszgebliche Vokalgruppe eines Wortes w definiert:
Fuer Worte $w \in A^*$ und $V \subseteq A$, kann jedes w wie folgt geschrieben werden:

$$w = q_1 v_1 q_2 v_2 \dots q_{n-1} v_{n-1} q_n v_n$$

mit $n \in \mathbb{N}$ und $q_i \in A^*$ und $v_i \in V^*$ fuer $1 \leq i \leq n$.
Nun ist (sofern vorhanden) v_n die letzte und v_{n-1} die vorletzte Vokalgruppe. Dabei ist es moeglich, dass ein Wort keine Vokalgruppe oder nur eine enthaelt. Wobei die nicht leeren Vokalgruppen betrachtet werden.

Fuer die nicht leeren Vokalgruppen ($v_i \neq \epsilon$) $v_1,\ v_2,\ \dots,\ v_n$ eines Wortes ist die maszgebliche Vokalgruppe

1. fuer $n = 0$ nicht existent,
2. fuer $n = 1$ die eine existierende Vokalgruppe: v_1,
3. fuer $n > 1$ die vorletzte Vokalgruppe v_{n-1}.

1.3 Definition von Worten, die sich reimen

Zwei Worte w_1 und w_2 reimen sich nun gdw. folgende Regeln gelten:

1. beide Worte besitzen die selbe maszgebliche Vokalgruppe v gefolgt von den selben restlichen Zeichen r rechts von v mit der selben Reihenfolge: $w_1 = avr$ und $w_2 = bvr$ fuer $a,\ b \in A^*$,
2. fuer die Worte gilt $|vr| \geq \frac{|w_1|}{2}$ und $|vr| \geq \frac{|w_2|}{2}$ bzw. die maszgebliche Vokalgruppe und was ihr folgt macht mind. die Haelfte der Buchstaben der beiden Worte aus und
3. keines der beiden Worte endet mit dem je anderen Wort, bzw. $w_1 \neq xw_2$ und $w_2 \neq yw_1$ fuer $x,\ y \in A^*$. Da $x = \epsilon$ oder $y = \epsilon$ moeglich ist, ist $w_1 = w_2$ ebenfalls nicht erlaubt.

Aus dieser Definition geht hervor, dass sich ein Wort w mit keinem anderen Wort reimen kann, wenn $vr < \frac{|w|}{2}$ ist (Regel 2), oder das Wort keine maszgebliche Vokalgruppe (also garkeine Vokalgruppe) enthaelt (Regel 1). Desweiteren muessen zwei Worte, die sich reimen, beide mit dem selben Wort vr enden (Regel 1).

1.4 Formelle Definition des Problems

Gegeben ist eine Menge L an n Worten

$$L = \{l_1, l_2, \ldots, l_n\}$$

ueber dem Alphabet A.

Nun sind alle Paare an Worten $l_1,\ l_2 \in L$ gesucht, wobei sich l_1 und l_2 nach 1.3 reimen. Da die Reihenfolge dabei egal ist, wird nach allen Mengen $N = \{l_1,\ l_2\}$ der Groesze zwei gesucht, sodass l_1 und l_2 sich reimen.

2 Loesungsidee

2.1 Bestimmen der maszgeblichen Vokalgruppe

Um das zuvor beschriebene Problem zu loesen, wird zuerst fuer jedes Wort $w \in M$ die maszgebliche Vokalgruppe $v \in V^*$ bestimmt.

Algorithm 1 Pseudocode zum erhalten v. Start- und Endindex der maszgeblichen Vokalgruppe eines Wortes w

```
1:  procedure getMainVowelGroup(w)
2:      start ← |w| + 1                          ▷ Die 1. Vokalgruppe liegt links von |w| + 1
3:      (i_1, j_1) ← getNext(w, start)           ▷ Die 1. Vokalgruppe geht von i_1 bis j_1
4:
5:      if i_1 = −1 then
6:          return NULL      ▷ Die 1. Vokalgruppe existiert nicht. Somit existiert keine maszgebliche.
7:      end if
8:
9:      (i_2, j_2) ← getNext(w, i_1)      ▷ Die 2. Vokalgruppe (i_2 bis j_2) liegt links vom Anfang der 1.
10:
11:     if i_2 = −1 then
12:         return (i_1, j_1)          ▷ Die 2. Vokalgruppe existiert nicht u. die 1. ist die maszgebliche
13:     else
14:         return (i_2, j_2)                  ▷ Die 2. Vokalgruppe existiert u. ist somit maszgebliche
15:     end if
16: end procedure
```

Algorithm 2 Pseudocode zum erhalten v. Start- und Endindex der naechsten Vokalgruppe eines Wortes $w = a_1 a_2 \ldots a_n$, links von einem Index i:

```
1:  procedure getNext(w, i)
2:      k ← i − 1
3:      while k ≥ 1 ∧ a_k ∉ V do          ▷ Dekrementiere k, solange wie a_k kein Vokal ist.
4:          k ← k − 1
5:      end while
6:      if k = 0 then
7:          return (−1, −1)                ▷ Fuer K = 0, existiert kein Vokal links von i.
8:      end if
9:      j ← k                              ▷ Speichere den Endindex der Vokalgruppe
10:     while k ≥ 1 ∧ a_k ∈ V do           ▷ Dekrementiere k, solange wie a_k ein Vokal ist.
11:         k ← k − 1
12:     end while
13:     l ← k + 1                          ▷ Speichere den Startindex der Vokalgruppe
14:     return (l, j)
15: end procedure
```

Algorithmus 1 Um die maszgebliche Vokalgruppe zu bestimmen, wird von rechts nach links zunaechst

die erste Vokalgruppe und dann die zweite Vokalgruppe erkannt. Fuer den Fall, dass die erste Vokalgruppe nicht existiert, existiert somit auch keine maszgebliche Vokalgruppe.
Fuer den Fall, dass die zweite Vokalgruppe (von rechts nach links) nicht existiert, ist die erste Vokalgruppe die maszgebliche, andernfalls ist die zweite Vokalgruppe die maszgebliche.

Algorithmus 2 Um die naechste Vokalgruppe (von rechts) in einem Wort $w = a_1a_2 \dots a_n$ ab einem Index i, exklusive (also $i = n + 1$ fuer die erste Vokalgruppe) zu finden, wird zunaechst der Index des ersten Vokals links von i gesucht.
Dazu wird ab dem Index $k = i - 1$ ueberprueft, ob a_k ein Vokal ist. Solange dies nicht der Fall ist und $k \geq 1$ gilt, wird k dekrementiert und der Vorgang wiederholt. Anschlieszend gilt fuer den Fall $k = 0$, dass kein Vokal gefunden wurde. Andernfalls wird der Index $j = k$ des ersten Vokals links von i gespeichert.
Nun wird wird $l = k$ solange dekrementiert, bis $l < 1$ oder a_k kein Vokal ist.
Die naechste Vokalgruppe links von i ist nun von $l + 1$ bis j - je inklusive.

2.2 Bestimmen, der sich reimenden Wortpaare

Um nun alle Wortpaare zubestimmen, bei welchen sich die beiden Worte reimen, wird Folgendes fuer alle Worte $w \in L$ durchgefuehrt:
Sofern eine maszgebliche Vokalgruppe $v = a_i \dots a_j$ des Wortes w existiert, kann ebenfalls ermittelt werden, was der maszgeblichen Vokalgruppe folgt. Sei dieses Teilwort $r = a_{j+1} \dots a_n$ fuer $n = |w|$.
Existiert die maszgebliche Vokalgruppe nicht, so kann das Wort ignoriert werden, da es sich mit keinem anderen Wort reimen kann.
Aufgrund desses, dass ein Wort w sich sowieso nur mit einem anderen reimen kann, wenn $|vr| \geq \frac{|w|}{2}$ gilt (Regel 2), wird ueberpueft, ob dies der Fall ist.
Ist dem nicht so, wird das Wort irgnoriert. Andernfalls wird das Wort in einer Tabelle mit zwei Spalten gespeichert. Diese beinhaltet auf der linken Seite alle bisher gefundenen vr ein Mal und auf der rechten Seite eine Liste aller Worte, welche mit dem jeweiligen vr enden.
Es wird also fuer jedes nicht zuvor gesehende vr eine neue Zeile angelegt. Anschlieszend wird das aktuelle Wort zu der, zu dem aktuellen vr gehoerenden, Liste hinzugefuegt.

Nachdem jedes Wort durchgegangen wurde, befinden sich in jeder Zeile der Tabelle eine Endung vr (erste Spalte) und alle Worte, welche diese Endung haben in der zweiten Spalte, sofern die Endung mindestens die Haelfte des Wortes ausmacht.
Alle Worte in den Listen der zweiten Spalte beinhalten Worte, die sich potenziell reimen, denn diese erfuellen (paarweise) Regel 1 und Regel 2.

Nun muss also fuer jedes Paar (ohne doppelte Paare) der Listen in der rechten Spalte der Tabelle ueberprueft werden, ob es Regel 3 erfuellen.
Dazu wird fuer jede Zeile mit vr in der ersten und einer Liste L in der zweiten Spalte jedes Paar (ohne Beachtung der Reihenfolge) durchgegangen.
Fuer ein Paar $\{a,\ b\}$ muss nun also nur Regel drei ueberprueft werden, bzw. ob a mit b endet oder b mit a endet.
Um bei zwei Worten $a = w_1w_2 \dots w_l,\ b = m_1m_2 \dots m_r$ mit $|a| \geq |b|$ zu ueberpruefen, ob a mit b endet, also $a = xb$ wobei x eine beliebiges Wort ist, wird ueberprueft ob die Zeichenkette $w_{l-r+1}w_{l-r+2} \dots w_l = b$ ist.
Endet keines der beiden Worte, mit dem je anderen, so wurde ein Wortpaar gefunden, bei welchem sich die beiden Worte reimen.

2.3 Korrektheit

Die Korrektheit dieser Loesungsidee ist einfach zu begruenden.
Es werden alle moeglichen Paar (ohne beachtung der Reihenfolge) von Worten, die sich reimen gefunden, weil zunaechst nur diejenigen aussortiert werden, welche sich nicht reimen koennen und anschlieszend alle Worte mit gleicher maszgeblichen Vokalgruppe + Rest (Regel 1 und Regeln 2) in einer Liste gespeichert werden. Fuer diese Liste werden dann alle moeglichen Paare ausprobiert und auf Regel 3 ueberprueft.

Widerspruchsansatz: Sei $(a,\ b)$ ein Paar, welches sich reimt und nicht gefunden wurde, so enden beide Worte mit vr (maszgebnliche Vokalgruppe + Rest) und befaenden sich in der selben Liste der

Tabelle. Wodurch sie uebeprueft worden waeren und, sofern keines der Worte mit dem je anderen endet, als Loesung gefunden worden waeren.

3 Implementierung / Umsetzung

3.1 Alphabet A und Vokale V

Fuer die Loesungsidee sind die Mengen A und V nicht von Bedeutung. Damit allerdings die Loesungen fuer die (Beispiel) Eingaben korrekt sind, wurde fuer das Alphabet A jeder Buchstabe der Eingabe benutzt und das Alphabet V definiert als Menge

$$\{a, e, i, o, u, \ddot{a}, \acute{e}, \ddot{o}, \ddot{u}, y\}.$$

Dabei wurden ausschlieszlich Kleinbuchstaben verwendet, um die technische Implementierung zu vereinfachen (s.u.).

3.2 Technische Implementierung

Die soeben beschriebene Loesungsidee wurde in Java 8 umgesetzt.

Um Worte zu representieren wurden Strings benutzt, welche ua. die Moeglichkeit bieten, auf die Symbole und die Laenge der Worte zu zugreifen.
Um die Handhabung zu vereinfachen, wurden diese dabei einheitlich in Kleinbuchstaben transformiert. Dadurch ist beispielsweise die maszgebliche Vokalgrupe „Au“ aequivalent zur mazsgeblichen Vokalgruppe „au“.

Fuer die Menge an Vokalen wurde ein HashSet mit den Symbolen zuvor genannten Symbolen benutzt, dadurch kann in konstanter Zeit ueberprueft werden, ob ein Character ein Vokal ist. Das Alphabet A besteht aus allen moeglichen (ASCII) Zeichen.
Die beschriebene Tabelle wurde als HashMap mit Strings als Keys und ArrayListen an Strings als Values implementiert. In diese wurde, wie beschrieben, die Worte eingetragen.
Um fuer eine Liste aus der HashMap alle zweier Kombinationen (ohne doppellte Paare) zu erhalten wurde eine doppelte for-Schleife benutzt, welche in der aeuszeren Schleife mit der Laufvariable i von 0 bis zur Laenge der Liste minus 1 geht und in der inneren Schleife von $i+1$ bis zur Laenge der Liste minus eins.

Um zu ueberpruefen, ob eines der Worte mit dem je anderen endet, wurde die Java-Methode String#endsWith benutzt.

4 Laufzeitkomplexitaet

Die Laufzeit des beschriebenen Algorithmus, als Funktion der Eingabe Laenge $n :=$ die Anzahl der Worte besteht aus folgenden Teilen:

1. Bestimmen von vr fuer jedes Wort w.
   Dazu wird jedes Wort linear oft durchgegangen.

2. Das Einfuegen der Worte in jewelige Listen benoetigt ebenfalls linear viel Zeit (in n), da maximal jedes Wort eingefuegt werden muss und eine HashMap einen Zugriff auf die benoetigte Liste in (durchschnittlich) konstanter Zeit bietet.

3. Um fuer jede Liste der HashMap alle Paare durchzugehen wird quadratisch viel Zeit benoetigt, da im schlimmsten Fall eine Liste mit n Elementen vorhanden ist, welche $(n-1)+(n-2)+\cdots+2+1 = \frac{1}{2}(n-1)(n) \in \mathcal{O}(n^2)$ Paare erzeugt.

Somit ist die Laufzeit des gesamten Programmes $\mathcal{O}(n^2)$.
Dies ist ebenfalls die best moegliche (worst-case) Laufzeit, da sich im „schlimmsten“ Fall alle Worte reimen und somit $\mathcal{O}(n^2)$ viele Paare entstehen.

5 Beispiele

5.1 Erlaeuterung der Loesungsidee an remerei0.txt

Erlaeuterung der Loesungsidee an der ersten Beispieleingabe der BwInf Website:
bemühen, Biene, breitschlagen, glühen, hersagen, Hygiene, Knecht, Recht, Schiene, schlank, Schwank
bzw. mit Kleinbuchstaben:
bemühen, biene, breitschlagen, glühen, hersagen, hygiene, knecht, recht, schiene, schlank, schwank.

1. Die maszgebliche Vokalgruppe der Woerter und ob sie mind. die Haelfte des Wortes ausmacht:

Wort	bemühen	biene	breitschlagen	glühen	hersagen
Maszgebliche Vokalgruppe	uehen	iene	agen	uehen	agen
Maszgebliche Vokalgruppe gros genugz	ja	ja	nein	ja	ja

Wort	hygiene	knecht	recht	schiene	schlank	schwank
Maszgebliche Vokalgruppe	iene	echt	echt	iene	ank	ank
Maszgebliche Vokalgruppe grosz genug	ja	ja	ja	ja	nein	nein

2. Die Tabelle:

ühen	[bemühen, glühen]
iene	[biene, hygiene, schiene]
agen	[hersagen]
echt	[knecht, recht]

3. Alle Paare aller Listen der Tabelle und ob sie Regel 3 einhalten:

(bemühen, glühen)	ja
(biene, hygiene)	ja
(biene, schiene)	ja
(hygiene, schiene)	ja
(knecht, recht)	ja

5.2 Beispieleingabe 1 - remerei0.txt

Alle 5 Paare, die sich reimen:

1. (bemühen, glühen)
2. (biene, hygiene)
3. (biene, schiene)
4. (hygiene, schiene)
5. (knecht, recht)

5.3 Beispieleingabe 2 - remerei1.txt

Alle 2 Paare, die sich reimen:

1. (brote, note)
2. (bildnis, wildnis)

5.4 Beispieleingabe 3 - remerei2.txt

Ein Paar, das sich reimt:

1. (epsilon, ypsilon)

5.5 Beispieleingabe 4 - remerei3.txt

Alle 61 Paare, die sich reimen:

1. (gas, glas)
2. (bein, wein)
3. (gleis, kreis)
4. (gleis, preis)
5. (gleis, reis)
6. (kreis, preis)
7. (see, tee)
8. (sache, sprache)
9. (hund, mund)
10. (kunde, stunde)
11. (sekunde, stunde)
12. (bitte, mitte)
13. (ansage, frage)
14. (ansage, garage)
15. (frage, garage)
16. (rock, stock)
17. (bahn, zahn)
18. (s-bahn, zahn)
19. (feuer, steuer)
20. (fest, test)
21. (durst, wurst)
22. (lohn, sohn)
23. (dame, name)
24. (keller, teller)
25. (hand, land)
26. (hand, strand)
27. (land, strand)
28. (bank, dank)
29. (gruppe, suppe)
30. (kasse, klasse)
31. (kasse, tasse)
32. (klasse, tasse)
33. (glück, stück)
34. (kanne, panne)

35. (flasche, tasche)
36. (kopf, topf)
37. (platz, satz)
38. (baum, raum)
39. (bier, tier)
40. (ermäßigung, kündigung)
41. (ermäßigung, reinigung)
42. (kündigung, reinigung)
43. (hose, rose)
44. (absender, kalender)
45. (fuß, gruß)
46. (dezember, november)
47. (dezember, september)
48. (november, september)
49. (fisch, tisch)
50. (kassette, kette)
51. (kassette, toilette)
52. (kette, toilette)
53. (magen, wagen)
54. (nachmittag, vormittag)
55. (butter, großmutter)
56. (butter, mutter)
57. (drucker, zucker)
58. (bild, schild)
59. (kind, rind)
60. (kind, wind)
61. (rind, wind)

5.6 Eigenes Beispiel 0

Eingabe, bei welcher sich kein Wortpaar reimt:

1. Ausgang
2. Beruf
3. Blatt
4. Datum
5. Ehefrau
6. Ehemann

7. Einzelzimmer
8. Fahrer
9. Fernsehgerät
10. Frage

Ausgabe:
Keine Wortpaare, die sich reimen.

5.7 Eigenes Beispiel 1

Eingabe, bei welcher sich alle Wortpaare der 10 Worte reimen:

1. fasse
2. rasse
3. passe
4. kasse
5. masse
6. nasse
7. gasse
8. tasse
9. lasse
10. hasse

Ausgabe:
45 Wortpaare, die sich reimen:

1. (fasse, rasse)
2. (fasse, passe)
3. (fasse, kasse)
4. (fasse, masse)
5. (fasse, nasse)
6. (fasse, gasse)
7. (fasse, tasse)
8. (fasse, lasse)
9. (fasse, hasse)
10. (rasse, passe)
11. (rasse, kasse)
12. (rasse, masse)
13. (rasse, nasse)
14. (rasse, gasse)
15. (rasse, tasse)
16. (rasse, lasse)

17. (rasse, hasse)
18. (passe, kasse)
19. (passe, masse)
20. (passe, nasse)
21. (passe, gasse)
22. (passe, tasse)
23. (passe, lasse)
24. (passe, hasse)
25. (kasse, masse)
26. (kasse, nasse)
27. (kasse, gasse)
28. (kasse, tasse)
29. (kasse, lasse)
30. (kasse, hasse)
31. (masse, nasse)
32. (masse, gasse)
33. (masse, tasse)
34. (masse, lasse)
35. (masse, hasse)
36. (nasse, gasse)
37. (nasse, tasse)
38. (nasse, lasse)
39. (nasse, hasse)
40. (gasse, tasse)
41. (gasse, lasse)
42. (gasse, hasse)
43. (tasse, lasse)
44. (tasse, hasse)
45. (lasse, hasse)

45 Paare sind korrekt da aus 10 Worten $9 + 8 + \cdots + 2 + 1 = \frac{1}{2} * (10)(9) = 5 * 9 = 45$ verschiedene Wortpaare entstehen.

6 Quellcode

6.1 Funktion zum erhalten der naechsten Vokalgruppe links von einem gegebenen Index

```java
/**
 * Function to obtain the next vowel group in a word to the left of a given
 * index (going from right to left)
 *
 * @param string The string to find the next vowel group in
 * @param index  The start index (plus one)
 * @return A pair, containing the start index (incl.) and the end index (excl.)
 * of the main vowel group.
 * If there are no vowels to the left of the index, null is returned.
 */
private static Pair<Integer, Integer> getNextVowelGroupFromRight(String string, int index) {
    // Starting at index-1!
    index--;

    // Go to the left starting at (i-1) until finding a vowel or being out of range (< 0)
    while (index >= 0 && !isVowel(string.charAt(index))) {
        index--;
    }

    // Special case: Check if the start of string was reached: there are no vowels
    // in the word to the left of i.
    if (index == -1) return null;

    // Now, string[index] contains a vowel and (index+1) is the excl. end,
    int endEx = index + 1;

    // Find the start of the vowel group by going through the vowel group until
    // there are no vowels anymore or 0 is reached.
    while (index >= 0 && isVowel(string.charAt(index))) {
        index--;
    }

    // Now, A[index+1] contains a vowel (since A[index] is not a vowel)
    // and hence (index+1) is the incl. end of the vowel group
    int startIn = index + 1;
    return Pair.of(startIn, endEx);
}
```

6.2 Funktion zum erhalten der maszgeblichen Vokalgruppe

```java
/**
 * Function to get a word's main vowel group given as a pair of
 * the first index (incl.) and the second index (excl.)
 *
 * @param word The word
 * @return An ordered pair containing the start index (incl.) and the end index (excl.)
 * of the main vowel group. If there are no vowels, null is returned.
 */
private static Pair<Integer, Integer> getMainVowelGroup(String word) {
    int length = word.length();

    // Try to get the first vowel group from the right. If there is none,
    // there cannot be a main vowel group (and null is returned)
    Pair<Integer, Integer> firstVowelGroup = getNextVowelGroupFromRight(word, length);
    if (firstVowelGroup == null) return null;

    // Try to get the second vowel group from the right (to the left of the first one).
    // If it exists, that is the main vowel group. Otherwise, the first one is.
    Pair<Integer, Integer> secondVowelGroup =
          getNextVowelGroupFromRight(word, firstVowelGroup.getFirst());
    if (secondVowelGroup == null) return firstVowelGroup;
    return secondVowelGroup;
```

```
}
```

6.3 Funktion zum erhalten aller Wortpaare, welche sich in einer Liste an Worten reimen

```
/**
 * Get all pairs of words from a list of words that rhyme with each other
 *
 * @param words The list of words
 * @return The list of pairs
 */
public static List<Pair<String, String>> getPairs(List<String> words) {
    // Create and fill HashMap mapping from the ending of a word to a list of words
    // having that ending and can rhyme with any other word
    // (= contain a main vowel group that is together with the rest of
    // the word at least half of it)
    Map<String, List<String>> map = new HashMap<>();

    for (String word : words) {
        // Get the vowel group for the current word (as first index incl, last index excl.)
        Pair<Integer, Integer> mainVowelGroup = getMainVowelGroup(word);

        // If there is no main vowel group, the word cannot rhyme at all.
        // Hence, is not added to the HashMap
        if (mainVowelGroup == null) continue;

        // Check weather the amount of symbols in main vowel group +
        // the rest (to the right) is at least half of the word's length
        // If this isn't true, don't add the word to the HashMap since it
        // cannot rhyme with any word
        int size = word.length() - mainVowelGroup.getFirst();
        boolean canBeUsed = size >= word.length() / 2D;
        if (!canBeUsed) continue;

        // Substring containing main vowel group + rest of the current word
        String ending = word.substring(mainVowelGroup.getFirst());

        // Put the word in its list in the HashMap, create the list if it doesn't exist
        if (!map.containsKey(ending)) map.put(ending, new ArrayList<>());
        map.get(ending).add(word);
    }

    // Create and fill list of all rhyming pairs of words
    List<Pair<String, String>> pairs = new ArrayList<>();

    // Go through all endings (main vowel group + rest) and their corresponding lists
    for (String ending : map.keySet()) {
        List<String> list = map.get(ending);

        // Go through all pairs of words in the current list
        for (int i = 0; i < list.size(); i++) {
            String wordA = list.get(i);

            for (int j = i + 1; j < list.size(); j++) {
                // Check if one of the words ends with the other one.
                // If that isn't the case, add the current pair to pairs.
                String wordB = list.get(j);

                if (wordA.endsWith(wordB) || wordB.endsWith(wordA)) continue;
                pairs.add(Pair.of(wordA, wordB));
            }
        }
    }

    return pairs;
}
```