Balsemin-Traniello 2A

Clément

- Rapport d’analyse -

16/06/2023

- Sommaire -

Table des matières

- Présentation des outils d'évaluation - 3
Les tests 3
Codacy 5
Temps d'exécution 5
- Evaluation de chaque algorithmes - 6
Efficacité 6
Simplicité 13
Sobriété 18
- Classement des algorithmes - 21
Simplicité 21
Efficacité 21
Sobriété numérique 21

- Présentation des outils d'évaluation -

Les tests

Pour vérifier si les algorithmes fonctionnent correctement j'ai fait une série de test qui permettent de détecter les erreurs, de valider les comportements attendus et s'assurer que le code répond aux exigences spécifiées.

Tout d'abord nous avons un test qui vérifie l'algorithme avec une chaine simple.

```
// Chaine simple avec ordre complet
        assertEquals(List.of("666","the", "the", "number", "of", "beast"),
Exercice.solution("666, the number of the beast",List.of('6', 't', 'n', 'o',
'b')));
```

Ensuite une vérification d'une chaine a 1 mot.

```
// Chaine à 1 mot
        assertEquals("Erreur de chaine à 1 mot", List.of("OK"),
Exercice.solution("OK",List.of('a', 'b', 'c', 'd', 'e')));
```

Ensuite une chaine vide.

```
// Chaine vide
        assertEquals(List.of(), Exercice.solution("",List.of('6', 't', 'n',
'o', 'b')));
```

Ensuite la chaine donnée en exemple.

```
// Chaine donnée en exemple
        assertEquals(List.of("fait", "Il", "aujourd", "aout", "beau", "hui",
"comme", "en"), Exercice.solution("Il fait beau aujourd'hui comme en
aout",List.of('f', 'I', 'z', 'u', 'k', 'a', 'b', 'o')));
```

Ensuite une chaine avec des mots identiques.

```
// Chaine avec des mots identiques
        assertEquals(List.of("demain","aujourd","hui","il", "il", "imagine",
"imagine", "pleut", "pleut"), Exercice.solution("imagine aujourd'hui il pleut,
imagine demain il pleut",List.of('d','a','h','i','p')));
```

Ensuite un Test de sensibilité de casse.

```
// Chaine sensibilité casse
        assertEquals(List.of("c", "Est", "En", "moment", "le", "aller", "A",
"d", "piscine", "La", "somme", "nous", "juin"), Exercice.solution("nous somme
En juin, c'Est le moment d'aller A La piscine",List.of('c', 'E', 'm', 'l',
'a', 'A', 'd', 'p', 'L', 's', 'e', 'n', 'j')));
```

Et pour finir un test avec une chaine avec beaucoup de caractère.

```
// Chaine avec beaucoup de caractere
        assertEquals(List.of("adipisci", "adipisci", "aliquam", "amet",
"animi", "architecto", "architecto", "aspernatur", "aspernatur", "assumenda",
"at", "at", "aut", "aut", "aut", "aut", "aut", "aut", "aut", "aut", "aut",
"autem", "beatae", "beatae", "blanditiis", "consequatur", "consequuntur",
"corrupti", "cum", "delectus", "deleniti", "deleniti", "deleniti", "deleniti",
"deserunt", "dicta", "dicta", "dicta", "distinctio", "distinctio",
"distinctio", "dolor", "dolor", "dolor", "dolor", "dolore", "dolorem",
"doloremque", "dolores", "dolores", "dolorum", "ducimus", "ea", "ea", "ea",
"ea", "ea", "ea", "earum", "earum", "earum", "eius", "eligendi", "eligendi",
"enim", "enim", "eos", "error", "esse", "est", "est", "est", "est", "et",
"et", "et", "et", "et", "et", "et", "et", "et", "et", "et", "et", "eum",
"eum", "eum", "eum", "eveniet", "excepturi", "expedita", "explicabo",
"explicabo", "facere", "facilis", "fuga", "fuga", "galisum", "hic", "id",
"id", "id", "illo", "illo", "illum", "in", "in", "in", "in", "internos",
"ipsam", "ipsum", "iste", "iste", "itaque", "iure", "iusto", "laboriosam",
"laudantium", "laudantium", "lorem", "maiores", "maxime", "minus", "modi",
"modi", "molestiae", "molestiae", "molestiae", "molestias", "molestias",
"nam", "natus", "necessitatibus", "nemo", "nihil", "nihil", "nisi", "non",
"non", "non", "non", "obcaecati", "obcaecati", "odio", "odio", "officiis",
"omnis", "omnis", "omnis", "omnis", "optio", "perferendis", "porro",
"possimus", "praesentium", "quas", "quas", "quasi", "quasi", "qui", "qui",
"qui", "qui", "qui", "qui", "quia", "quia", "quia", "quia", "quibusdam",
"quibusdam", "quidem", "quis", "quis", "quis", "quisquam", "quisquam", "quo",
"quo", "quod", "quos", "ratione", "ratione", "ratione", "ratione", "ratione",
"rem", "rem", "repellat", "repellat", "repellat", "repellendus", "rerum",
"rerum", "sed", "sed", "similique", "similique", "similique", "sint", "sit",
"sit", "sit", "sit", "sit", "sit", "soluta", "sunt", "sunt", "sunt", "sunt",
"suscipit", "temporibus", "tenetur", "totam", "ullam", "unde", "unde", "unde",
"ut", "ut", "ut", "ut", "ut", "vel", "vel", "vel", "vel", "vel", "velit",
"velit", "veritatis", "veritatis", "vero", "voluptas", "voluptas",
"voluptatem", "voluptatem", "voluptatem", "voluptatem"),
Exercice.solution("lorem ipsum dolor sit amet non voluptas expedita rem omnis
earum sit quis enim ut laudantium sunt est ratione iste sed laudantium illo at
dolore distinctio in dolorem ratione sit consequatur velit ut repellendus
dolorum est temporibus odio ea architecto illum aut similique officiis eum
optio quibusdam ea quas facere qui natus porro et quibusdam molestias aut
similique assumenda ea obcaecati quia id nemo totam et molestiae quidem non
error unde et quas soluta vel iusto repellat aut quisquam minus in enim unde
et perferendis beatae ut velit explicabo et voluptatem rerum vel maxime
```

```
facilis aut dolores deserunt qui eius architecto quo deleniti quia at modi
iure qui suscipit nisi in dolor dolores qui deleniti ducimus eum voluptatem
eligendi sit earum quia ea aspernatur molestias ea earum laboriosam id quasi
dolor et omnis dicta est repellat nihil non autem nihil et molestiae voluptas
qui similique internos rem quis blanditiis sit ullam omnis sed sint dicta vel
dicta sunt non beatae quasi aut eveniet illo eum maiores praesentium et rerum
consequuntur id ratione ratione est quis adipisci quo explicabo esse ea
aliquam possimus hic distinctio corrupti qui veritatis necessitatibus et
delectus quia aut tenetur modi cum sunt obcaecati nam omnis ratione aut
deleniti vero et dolor quisquam ut distinctio molestiae sit ipsam unde eum
quod animi aut voluptatem galisum eos sunt excepturi aut quos deleniti ut fuga
doloremque in eligendi adipisci et fuga odio vel veritatis aspernatur et iste
itaque vel repellat voluptatem", List.of('a', 'b', 'c', 'd', 'e', 'f', 'g',
'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z')));
```

Codacy

Le site Codacy permet de mesurer la qualité du code, en donnant des données sur la taille, la structure, la complexité ainsi que la duplication du code.

Temps d'exécution

Le temps d'exécution est primordial, pour le calculer je me sers de :

```
long startTime = System.nanoTime();
// appel de la méthode a tester
long endTime = System.nanoTime();
        long durationInNanos = endTime - startTime;
        double durationInMilliseconds = (double) durationInNanos / 1_000_000;

        System.out.println("Durée en millisecondes : " +
durationInMilliseconds);
```

Et je renvoie le temps d'exécution en milliseconde pour que ce soit plus lisible.

- Evaluation de chaque algorithmes -

Efficacité

Meilleur

Algo n°10

```java
public class Exercice {
    public static List<String> solution(String texte, List<Character> ordre) {
        List<String> motsEnOrdre = new ArrayList<>();
        List<String> motsRestants = new ArrayList<>();

        StringBuilder constructMots = new StringBuilder();
        for (int i = 0; i < texte.length(); i++) {
            char c = texte.charAt(i);
            if (Character.isLetter(c)) {
                constructMots.append(c);
            } else if (constructMots.length() > 0) {
                motsRestants.add(constructMots.toString());
                constructMots.setLength(newLength:0);
            }
        }

        if (constructMots.length() > 0) {
            motsRestants.add(constructMots.toString()); //Ajouter les mots sans lettres dans ordre
        }

        List<String> motsASupprimer = new ArrayList<>();

        for (int i = 0; i < ordre.size(); i++) {
            char ch = ordre.get(i);
            for (int j = 0; j < motsRestants.size(); j++) {
                String mot = motsRestants.get(j);
                if (Character.toLowerCase(mot.charAt(index:0)) == Character.toLowerCase(ch)) {
                    motsEnOrdre.add(mot);
                    motsRestants.remove(j);
                }
            }
        }
        motsEnOrdre.addAll(motsRestants);
        return motsEnOrdre;
    }
}
```

Note = 10

Il ne passe pas le premier test fourni initialement (Chaine simple avec ordre complet), et ne passe pas les tests plus complets.

Codacy à évaluer le code en lui mettant le critère C, et 8 de complexité.

Durée d'exécution moyenne en millisecondes : 1.1833ms

Algo n°43

```java
public class Exercice {
    public static List<String> solution(String str, List<Character> ordre) {
        ArrayList<String> mots = stringToWords(str);
        triFusion(mots, ordre);
        return mots;
    }

    public static void triFusion(ArrayList<String> mots, List<Character> ordre) {
        int taille = mots.size();

        if (taille <= 1) {
            return;
        }

        int mid = taille / 2;
        ArrayList<String> left = new ArrayList<>(mots.subList(fromIndex:0, mid));
        ArrayList<String> right = new ArrayList<>(mots.subList(mid, taille));

        triFusion(left, ordre);
        triFusion(right, ordre);

        fusion(left, right, mots, ordre);
    }

    public static void fusion(ArrayList<String> left, ArrayList<String> right, ArrayList<String> mots, List<Character> ordre) {
        int tailleL = left.size();
        int tailleR = right.size();
        int i = 0, j = 0, k = 0;

        while (i < tailleL && j < tailleR) {
            String leftI = left.get(i);
            String rightJ = right.get(j);

            if (compare(leftI, rightJ, ordre) <= 0) {
                mots.set(k, leftI);
                i++;
            } else {
                mots.set(k, rightJ);
                j++;
            }
            k++;
        }

        while (i < tailleL) {
            mots.set(k, left.get(i));
            i++;
            k++;
        }
```

```
        while (j < tailleR) {
            mots.set(k, right.get(j));
            j++;
            k++;
        }
    }

    public static int compare (String chaine1, String chaine2, List<Character> ordre){
        int position1;
        int position2;

        if (chaine1.equals(chaine2)){
            return 0;
        }
        if (chaine1.equals(anObject:"")){
            return -1;
        }
        if (chaine2.equals(anObject:"")){
            return 1;
        }
        if (chaine1.charAt(index:0)==(chaine2.charAt(index:0))){
            return compare(chaine1.substring(beginIndex:1),chaine2.substring(beginIndex:1), ordre);
        }

        position1 = getPosition(ordre, chaine1.charAt(index:0));
        position2 = getPosition(ordre, chaine2.charAt(index:0));

        if (position1 == -1 && position2 != -1){
            return 1;
        }
        if (position2 == -1){
            return -1;
        }
        if ( position1 > position2 ){
            return 1;
        }
        else {
            return -1;
        }
    }


    public static int getPosition(List<Character> ordre, char lettre) {
        return ordre.indexOf(lettre);
    }
```

```java
    public static int getPosition(List<Character> ordre, char lettre) {
        return ordre.indexOf(lettre);
    }

    public static ArrayList<String> stringToWords(String str) {
        ArrayList<String> sousChaines = new ArrayList<String>();
        int startIndex = -1;

        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);

            if (Character.isLetterOrDigit(c)) {
                if (startIndex == -1) {
                    startIndex = i;
                }
            } else {
                if (startIndex != -1) {
                    String sousChaine = str.substring(startIndex, i);
                    int startIndex - exercice.Exercice.stringToWords(String)
                    startIndex = -1;
                }
            }
        }

        if (startIndex != -1) {
            String sousChaine = str.substring(startIndex);
            sousChaines.add(sousChaine);
        }

        return sousChaines;
    }
}
```

Note = 10

Il ne passe pas les tests supplémentaires.

Codacy à évaluer le code en lui mettant le critère B, et 9 de complexité.

Durée d'exécution moyenne en millisecondes : 0.7431ms

Pire

Algo n°9

```java
public class Exercice {
    public static List<String> solution(String str, List<Character> ordre) {
        // Liste des mots classés dans l'ordre
        List<String> solution = new ArrayList<>();
        // On vérifie si le string rentré est vide, cela permet d'éviter de perdre du temps à parcourir toutes les boucles si le string est vide
        if (!str.isEmpty()) {
            // Liste des mots séparés
            List<String> stringToWordList = new ArrayList<>();
            // Mot que l'on va append
            String current = "";
            // On parcours tout le string renseigné avec une for each
            for (char c : str.toCharArray()) {
                // Si le caractère courant est un espace ou un apostrophe on termine le mot et on l'ajoute à la liste
                if (!isAlphanumeric(c)) {
                    if(!current.isEmpty()){
                        stringToWordList.add(current);
                        current = "";
                    }
                } else {
                    current += c;
                }
            }
            // On ajoute le dernier mot à la liste
            stringToWordList.add(current);
            // On crée une nouvelle liste qui contient en premier lieu tous les mots non classés
            List<String> unsorted = new ArrayList<>();
            unsorted.addAll(stringToWordList);
            System.out.println(stringToWordList);
            // On parcours tous les caractères de l'ordre donné
            for (char c : ordre) {
                // on parcours tous les mots de la liste
                for (String s : stringToWordList) {
                    // Si le premier caractère du mot correspond alors on l'ajoute et on le retire de la liste des non classés pour ne finir qu'avec ceux qui ne le sont pas
                    if (s.charAt(index:0) == c) {
                        solution.add(s);
                        unsorted.remove(s);
                    }
                }
            }
            // On ajoute à la fin tous les mots non classés dans l'ordre dans lequel ils sont arrivés dans la liste de mots initiale
            solution.addAll(unsorted);
```

```java
        }
        // Si le string de base est vide on renvoie simplement un liste vide.
        else{
            solution = List.of();
        }

        return solution;
    }



    // Méthode permettant de vérifier si un caractère est alphanumérique
    private static boolean isAlphanumeric(char c){
        return (Character.isLetter(c) || Character.isDigit(c));
    }
}
```

Note = 10

Ne passe pas les tests supplémentaires mais passe les tests initiaux.

Codacy à évaluer le code en lui mettant le critère B, et 8 de complexité.

Durée d'exécution moyenne en millisecondes : 0.9473ms

Algo n° 22

```java
public class Exercice {

    public static List<String> solution(String texte, List<Character> ordre) {
        Map<Character, Integer> orderMap = new HashMap<>();
        int d = 0;
        for(int e=0; e<10000; e++) {
            d++;
        }
        // Création d'une hashMap pour stocker l'indice de chaque lettre dans l'ordre
        for (int i = 0; i < ordre.size(); i++) {
            orderMap.put(ordre.get(i), i);
        }

        // Liste permettant de stocker les mots ayant leur première lettre dans le dictionnaire
        List<String> words = new ArrayList<>();
        // Liste permettant de stocker les mots n'ayant pas leur première lettre dans le dictionnaire
        List<String> unknownWords = new ArrayList<>();

        StringBuilder currentWord = new StringBuilder();

        // Parcours du texte caractère par caractère
        for (int i = 0; i < texte.length(); i++) {
            char c = texte.charAt(i);

            // Si le caractère est une lettre
            if (Character.isLetter(c) || Character.isDigit(c)) {
                currentWord.append(c);

                // Si le caractère suivant n'est pas une lettre ou si on a atteint la fin du texte
                if (i == texte.length() - 1 || !Character.isLetterOrDigit(texte.charAt(i + 1))) {
                    String word = currentWord.toString();

                    // Si la lettre est dans la hashMap dictionnaire
                    if (orderMap.containsKey(word.charAt(index:0))) {
                        words.add(word);
                    } else {
                        unknownWords.add(word);
                    }

                    currentWord.setLength(newLength:0); // Réinitialisation du mot en cours
                }
            }
        }
```

```
        // Trie des mots en utilisant l'ordre spécifié
        bogoSort(words, orderMap);

        words.addAll(unknownWords);

        return words;
    }


    /*
     * Mélange les mots aléatoirement
     */
    private static void bogoSort(List<String> words, Map<Character, Integer> orderMap) {
        while (!isSorted(words, orderMap)) {
            shuffle(words);
        }
    }

    /*
     * Vérifie si la liste de mots est triée dans l'ordre spécifié
     */
    private static boolean isSorted(List<String> words, Map<Character, Integer> orderMap) {
        for (int i = 0; i < words.size() - 1; i++) {
            String word1 = words.get(i);
            String word2 = words.get(i + 1);

            if (compareWords(word1, word2, orderMap) > 0) {
                return false;
            }
        }
        return true;
    }
```

```
    /*
     * Compare deux mots en utilisant l'ordre du dictionnaire
     */
    private static int compareWords(String word1, String word2, Map<Character, Integer> orderMap) {
        int minLength = Math.min(word1.length(), word2.length());

        for (int i = 0; i < minLength; i++) {
            char char1 = word1.charAt(i);
            char char2 = word2.charAt(i);

            int order1 = orderMap.containsKey(char1) ? orderMap.get(char1) : Integer.MAX_VALUE;
            int order2 = orderMap.containsKey(char2) ? orderMap.get(char2) : Integer.MAX_VALUE;

            if (order1 != order2) {
                return order1 - order2;
            }
        }

        return word1.length() - word2.length();
    }

    /*
     * mélange aléatoirement les mots de la liste
     */
    private static void shuffle(List<String> words) {
        for (int i = words.size() - 1; i > 0; i--) {
            int j = (int) (Math.random() * (i + 1));
            swapWords(words, i, j);
        }
    }

    /*
     *Change les mots de position dans la liste
     */
    private static void swapWords(List<String> words, int i, int j) {
        String temp = words.get(i);
        words.set(i, words.get(j));
        words.set(j, temp);
    }
}
```

Note = 8

Le code boucle sur les tests mais ne boucle pas dans le programme principal

Codacy à évaluer le code en lui mettant le critère C, et 9 de complexité.

Durée d'exécution moyenne en millisecondes : 3.8254ms

Simplicité

Meilleur

Algo n°1

```java
import java.util.*;

public class Exercice {
    public static List<String> solution(String str, List<Character> ordre) {
        List<String> result = new ArrayList<String>();
        String[] mots = str.split(regex:" ");

        // Utilisation d'un comparateur personnalisé pour trier les mots selon l'ordre fixé
        Arrays.sort(mots, new ComparateurOrdreFixe(ordreFixe));

        // Affichage des mots classés
        for (String mot : mots) {
            result.add(mot);
        }
    }
}

class ComparateurOrdreFixe implements Comparator<String> {
    private Map<String, Integer> indexMap = new HashMap<>();

    public ComparateurOrdreFixe(List<String> ordreFixe) {
        for (int i = 0; i < ordreFixe.size(); i++) {
            indexMap.put(ordreFixe.get(i), i);
        }
    }

    @Override
    public int compare(String mot1, String mot2) {
        Integer index1 = indexMap.getOrDefault(mot1, Integer.MAX_VALUE);
        Integer index2 = indexMap.getOrDefault(mot2, Integer.MAX_VALUE);
        return index1.compareTo(index2);
    }
}
```

L'algo ne marche pas, il y a une erreur.

Note 0

Temps = NULL

Le code est cependant lisible, les variables sont bien nommées, la structure est clair, il utilise des méthodes prédéfinies ainsi qu'une classe externe pour le comparateur.

Hors concours.

Algo n°16

```java
public class Exercice {
    public static List<String> solution(String str, List<Character> ordre) {
        List<String> mots = new ArrayList<>();
        String[] motsTexte = texte.split(" ");

        for (char lettre : ordre) {
            for (String mot : motsTexte) {
                if (mot.charAt(index:0) == lettre) {
                    mots.add(mot);
                }
            }
        }

        return mots;
    }
}
```

Je pense que le code a été fait avec chat GPT car il y a une erreur et il suffit seulement de remplacer 'texte' par 'str'.

Note = 8

Le code est cependant lisible du fait de sa petite taille. Le nom des variables est plutot clair. Mais il manque de commentaire ainsi que Le nom 'motsTexte' pourrait être plus explicite, par exemple 'motsDansLeTexte'.

Temps d'exécution : 1.6274ms

Algo n°36

```java
public class Exercice {

    public static List<String> solution(String texte, List<Character> ordre) {
        texte = texte.replace(target:"'", replacement:" ");
        texte = texte.replace(target:",", replacement:"");
        texte = texte.replace(target:".", replacement:"");
        String[] mots = texte.split(regex:" ");
        triParOrdre(mots, ordre);
        return List.of(mots);
    }



    private static void triParOrdre(String[] mots, List<Character> ordre) {
        for (int i = 0; i < mots.length - 1; i++) {
            for (int j = i + 1; j < mots.length; j++) {
                if (compareMots(mots[j], mots[i], ordre)) {
                    String temp = mots[i];
                    mots[i] = mots[j];
                    mots[j] = temp;
                }
            }
        }
    }

    private static boolean compareMots(String mot1, String mot2, List<Character> ordre) {
        int length1 = mot1.length();
        int length2 = mot2.length();
        int minLength = Math.min(length1, length2);

        for (int i = 0; i < minLength; i++) {
            char char1 = mot1.charAt(i);
            char char2 = mot2.charAt(i);
            int index1 = getOrdreIndex(ordre, char1);
            int index2 = getOrdreIndex(ordre, char2);
```

```java
            if (index1 != index2) {
                return index1 < index2;
            }
        }

        return length1 < length2;
    }



    private static int getOrdreIndex(List<Character> ordre, char lettre) {

        for (int i = 0; i < ordre.size(); i++) {
            if (ordre.get(i) == lettre) {
                return i;
            }
        }

        return ordre.size();
    }

}
```

Note = 15

Certes il ne passe pas le test chaine vide, mais il passe tous les tests approfondit d'où la note.

Le nom des variables est clair, la structure est plutôt claire. Il manque cependant un peut de commentaire.

Temps d'exécution 1.1756ms.

Pire

Algo n°3

```
public class Exercice {
    /**
     * Trie les mots d'une phrase selon un ordre spécifié.
     *
     * @param pfPhrase La chaîne de caractères contenant les mots à trier.
     * @param pfOrdre  La liste des caractères spécifiant l'ordre de tri.
     * @return Une liste contenant les mots triés selon l'ordre spécifié.
     */
    public static List<String> solution(String str, List<Character> ordre) {

        List
        <String>
        //////////////////////


        y

        =
        diviserMots(str)
        ;


        y.sort(Comparator.comparingInt(s -> {
            int d = Math.min(s.length(), ordre.size());
            for (int h = 0; h < d; h++) {
                int g =
                        ordre.indexOf(s.charAt(h));
                if (g != -1) {
                    return g;
                }
            }



            ////////////////////////////////////////
            ////////////////////////////////////////
```

```
                return s.length();
            }
)
)
;

        return y;
    }

    /**
     * Divise une chaîne de caractères en mots individuels.
     *
     * @param pfPhrase La chaîne de caractères à diviser en mots.
     * @return Une liste contenant les mots individuels extraits de la phrase.
     */
    public static List<String> diviserMots(String pfPhrase) {
        List<String> p = new ArrayList<>();
        StringBuilder pp = new StringBuilder();
        ////////////////////////////////////////////
        for (
                int
                i = 0; i < pfPhrase.length();


                i++) {
            char ppp = pfPhrase.charAt(i);
            if (Character.isLetter(ppp)) {
                pp.append(ppp);
            } else if (pp.length() > 0) {
                p.add(pp.toString());
                pp.setLength(newLength:0);
            }
        }
        if (
```

```
                pp.length()
                >
        ////////////////////////////////////


        0) {


            p.add(pp.toString());}
        return
                p;
    }
}
```

Il passe seulement le test de la chaine a 1 mot

Note 5

Les noms de variables sont vraiment peut explicite, les indentations sont incohérentes et il manque des séparations logiques.

Temps d'exécution 9.1752ms

Algo n°48

```
public class Exercice {
    public static List<String> solution(String str, List<Character> ordre) {
        // Divise la chaîne de caractères en mots à l'aide de l'espace comme séparateur
        List<String> words = new ArrayList<>(Arrays.asList(str.split(regex:"[ ']")));

        // Pour chaque mot
        for (String word : words) {
            // Pour chaque caractère du mot
            for (char c : word.toCharArray()) {
                // Pour chaque caractère dans la liste d'ordre
                for (Character ordreChar : ordre) {
                    // Si le caractère du mot est dans la liste d'ordre
                    if (c == ordreChar) {
                        // Récupère son indice dans la liste d'ordre
                        int index = ordre.indexOf(ordreChar);
                    }
                }
            }
        }
```

```
        // Trie les mots en utilisant un comparateur personnalisé
        Collections.sort(words, new Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                // Récupère la position de la première lettre de chaque mot dans la liste
                // 'ordre'
                int index1 = ordre.indexOf(o1.charAt(index:0));
                int index2 = ordre.indexOf(o2.charAt(index:0));

                // Si les deux lettres ne figurent pas dans 'ordre', elles sont considérées
                // comme égales
                if (index1 == -1 && index2 == -1)
                    return 0;
                // Si la première lettre figure dans 'ordre' et la deuxième non, la première est
                // placée en premier
                else if (index1 != -1 && index2 == -1)
                    return -1;
                // Si la première lettre ne figure pas dans 'ordre' et la deuxième si, la
                // deuxième est placée en premier
                else if (index1 == -1)
                    return 1;
                // Si les deux lettres figurent dans 'ordre', leur position dans 'ordre'
                // détermine leur ordre
                else
                    return Integer.compare(index1, index2);
            }
        });

        // Retourne la liste de mots triés
        return words;
    }

}
```

Note = 0

Hors concours

Le type de la méthode solution n'est pas le bon, j'ai dû faire les imports car aucun n'était fait. La classe n'avais pas le bon nom, cependant les commentaires sont pertinents et la structure est cohérente.

Sobriété

Meilleur

Algo n°40

```java
public class Exercice {

    public static List<String> solution(String str, List <Character> ordre) {

        List<String> liste = new ArrayList<>();
        StringBuilder motActuel = new StringBuilder();

        //on isole chacun des mots pour pouvoir ensuite les triers
        for (int i =0; i<str.length(); i++) {
            if (Character.isLetterOrDigit(str.charAt(i))) {
                motActuel.append(str.charAt(i));
            } else if (motActuel.length()>0) {
                liste.add(motActuel.toString());
                motActuel.setLength(newLength:0);
            }

            //condition pour ajouter le dernier mot de la chaine str
        }
        if (motActuel.length() > 0) {
            liste.add(motActuel.toString());
        }

        for (int i = 1; i < liste.size(); i++) {
            String mot = liste.get(i);
            int j = i - 1;
            while (j >= 0 && comparerMots(liste.get(j), mot, ordre) > 0) {
                liste.set(j + 1, liste.get(j));
                j--;
            }
            liste.set(j + 1, mot);


        }
        return liste;
    }
```

```java
    public static int comparerMots(String mot1, String mot2, List<Character> ordre) {
        int minLength = Math.min(mot1.length(), mot2.length());
        for (int i = 0; i < minLength; i++) {
            char c1 = Character.toLowerCase(mot1.charAt(i));
            char c2 = Character.toLowerCase(mot2.charAt(i));
            int index1 = ordre.indexOf(c1);
            int index2 = ordre.indexOf(c2);
            if (index1 != -1 && index2 != -1 && index1 != index2) {
                return Integer.compare(index1, index2);
            } else if (index1 == -1 && index2 != -1) {
                return 1; // mot1 commence par une lettre non pr�sente dans l'ordre
            } else if (index1 != -1 && index2 == -1) {
                return -1; // mot2 commence par une lettre non pr�sente dans l'ordre
            }
        }
        return Integer.compare(mot1.length(), mot2.length());
    }

}
```

Note = 10

Il ne passe pas tous les tests d'approfondissement.

Temps d'exécution = 0.8264ms

Algo n°68

```java
public class Exercice {

    public static List<String> solution(String texte, List<String> ordre) {

        // Expression Regex pour séparer les mots à chaque caractère non-alphabétique
        List<String> decoupage = Arrays.asList(texte.split(regex:"[^\\p{L}]"));

        // Tri en fonction de l'ordre
        List<String> resultat = new ArrayList<>();
        Set<String> motsDejaPresent = new HashSet<>();
        for (String motOrdre : ordre) {
            for (String mot : decoupage) {
                if (!motsDejaPresent.contains(mot) && mot.startsWith(motOrdre)) {
                    resultat.add(mot);
                    motsDejaPresent.add(mot);
                }
            }
        }

        // Ajout des mots restants
        for (String mot : decoupage) {
            if (!motsDejaPresent.contains(mot)) {
                resultat.add(mot);
                motsDejaPresent.add(mot);
            }
        }

        return resultat;

    }
```

Note 0

Hors concours

Il ne compile pas, les arguments attendus ne sont pas les même que ceux qu'on veut.

Pire

Algo n°23

```java
public class Exercice {
    public static String trierMots(String phrase, List<Character> alphabet) {
        // Créer une map pour associer chaque lettre à une liste de mots commençant par cette lettre
        Map<Character, List<String>> motsParLettre = new HashMap<>();
        for (char lettre : alphabet) {
            motsParLettre.put(lettre, new ArrayList<>());
        }

        // Diviser la phrase en mots
        String[] mots = phrase.toLowerCase().split(regex:"\\W+");

        // Parcourir chaque mot et l'ajouter à la liste correspondante dans la map
        for (String mot : mots) {
            if (!mot.isEmpty()) {
                char premiereLettre = mot.charAt(index:0);
                if (motsParLettre.containsKey(premiereLettre)) {
                    List<String> motsDeLaLettre = motsParLettre.get(premiereLettre);
                    motsDeLaLettre.add(mot);
                }
            }
        }

        // Créer une liste de mots triés en parcourant l'alphabet et concaténant les listes de mots
        StringBuilder phraseTrie = new StringBuilder();
        for (char lettre : alphabet) {
            List<String> motsDeLaLettre = motsParLettre.get(lettre);
            for (String mot : motsDeLaLettre) {
                phraseTrie.append(mot).append(str:" ");
            }
        }

        return phraseTrie.toString().trim();
    }
}
```

Note 0

Hors concours

Il ne compile pas, les arguments attendus ne sont pas les même que ceux qu'on veut.

Algo n°64

```java
public class Exercice {
    public static List<String> solution(String texte, List<Character> ordre) {
        List<String> motsTries = new ArrayList<>();

        for (int i = 0; i < texte.length(); i++) {
            String mot = extraireMot(texte, i);
            if (!mot.isEmpty()) {
                ajouterMot(motsTries, mot, ordre);
                i += mot.length() - 1;
            }
        }

        return motsTries;
    }

    private static String extraireMot(String texte, int startIndex) {
        StringBuilder mot = new StringBuilder();
        int i = startIndex;

        while (i < texte.length()) {
            char c = texte.charAt(i);

            if (Character.isLetterOrDigit(c)) {
                mot.append(c);
            } else {
                break;
            }

            i++;
        }
```

```java
        return mot.toString();
    }

    private static void ajouterMot(List<String> motsTries, String mot, List<Character> ordre) {
        motsTries.add(mot);

        for (int i = 0; i < motsTries.size() - 1; i++) {
            for (int j = i + 1; j < motsTries.size(); j++) {
                String mot1 = motsTries.get(i);
                String mot2 = motsTries.get(j);
                if (comparerMots(mot1, mot2, ordre) > 0) {
                    // Échanger les mots
                    motsTries.set(i, mot2);
                    motsTries.set(j, mot1);
                }
            }
        }
    }

    private static int comparerMots(String mot1, String mot2, List<Character> ordre) {
        int longueur1 = mot1.length();
        int longueur2 = mot2.length();
        int longueurMin = Math.min(longueur1, longueur2);

        for (int i = 0; i < longueurMin; i++) {
            char char1 = mot1.charAt(i);
            char char2 = mot2.charAt(i);
            int index1 = ordre.indexOf(char1);
            int index2 = ordre.indexOf(char2);

            if (index1 != -1 && index2 != -1) {
                if (index1 != index2) {
                    return index1 - index2;
                }
```

```java
            } else if (index1 == -1 && index2 == -1) {
                continue;
            } else if (index1 == -1) {
                return 1;
            } else if (index2 == -1) {
                return -1;
            }
        }

        return longueur1 - longueur2;
    }
}
```

Note 20

Il passe tous les tests.

Temps d'exécution 0.9356ms

- Classement des algorithmes -

Simplicité

Algo 3 est 3eme

Algo 16 est 2eme

Et l'algo 36 est premier

Efficacité

Algo 22 est 3 eme

Algos 43 et 9 2eme

Algo 10 premier

Sobriété numérique

Premier 64

Deuxième 40