

SAE 2.02

Exploration algorithmique d'un problème

18 juin 2024



Bouyssou Melvin 3A

Table des matières

[Table des matières](#)

[Introduction](#)

[Présentation des outils utilisés](#)

[Codacy : outil en ligne d'analyse de code dans les langages java, python et C \(et d'autres...\).](#)

[FindBugs : outil de détection de bug dans un code java.](#)

[PMD : outil d'analyse de code similaire à Codacy \(uniquement utilisé pour java\). installation sur la machine.](#)

[Unity \(C\) : outil de test similaire à JUnit pour du C.](#)

[Lien vers les tests java utilisés](#)

[Explication de la notation](#)

[Algorithmes simplicité](#)

[Notation de l'algorithme : 03simplicite.java](#)

[Détails de la notation de 03simplicite.java](#)

[Notation de l'algorithme : 05simplicite.java](#)

[Détails de la notation de 05simplicite.java](#)

[Notation de l'algorithme : 18simplicite.java](#)

[Détails de la notation de 18simplicite.java](#)

[Notation de l'algorithme : 25simplicite.java](#)

[Détails de la notation de 25simplicite.java](#)

[Notation de l'algorithme : 38simplicite.java](#)

[Détails de la notation de 38simplicite.java](#)

[Classement Simplicité](#)

[Algorithmes efficacité](#)

[Notation de l'algorithme : 06efficacite.java](#)

[Détails de la notation de 06efficacite.java](#)

[Notation de l'algorithme : 12efficacite.java](#)

[Détails de la notation de 12efficacite.java](#)

[Notation de l'algorithme : 14efficacite.java](#)

[Détails de la notation de 14efficacite.java](#)

[Notation de l'algorithme : 28efficacite.java](#)

[Détails de la notation de 28efficacite.java](#)

[Notation de l'algorithme : 59efficacite.java](#)

[Détails de la notation de 59efficacite.java](#)

[Classement Efficacité](#)

[Algorithme sobriété](#)

[Notation de l'algorithme : 13sobriete.c](#)

[Détails de la notation de 13sobriete.c](#)

[Notation de l'algorithme : 37sobriete.c](#)

[Détails de la notation de 37sobriete.c](#)

[Notation de l'algorithme : 44sobriete.c](#)

[Détails de la notation de 44sobriete.c](#)

[Notation de l'algorithme : 19sobriete.py](#)

[Détails de la notation de 19sobriete.py](#)

[Classement Sobriété](#)

Introduction

Document de rapport de correction et classement de la SAE 2.02 Exploration algorithmique d'un problème. Ce document à pour but de fournir une évaluation des codes et en détailler les notes. Ce document présente les outils utilisés ainsi que la grille de notation "générale" mise en place. Notez que les coefficients des notes varient en fonction de la catégorie du concours. L'efficacité d'un algorithme aura plus de poids dans le concours efficacité que simplicité.

Présentation des outils utilisés

[Codacy](#) : outil en ligne d'analyse de code dans les langages java, python et C (et d'autres...).

[FindBugs](#) : outil de détection de bug dans un code java.

[PMD](#) : outil d'analyse de code similaire à Codacy (uniquement utilisé pour java), installation sur la machine.

[Unity](#) (C) : outil de test similaire à JUnit pour du C.

Par soucis de signature entre les algorithmes, je ne l'ai utilisé que pour 1 algorithme et j'ai terminé avec des printf pour les autres.

[Lien vers les tests java utilisés](#)

Explication de la notation

Notation de l'algorithme : note finale attribuée		
Sanction	Problème	Note
Hors concours	Ne compile pas	Note divisé /2
/	Non respect de l'anonymat	-1
Hors concours	Non respect de la consigne sur les méthodes de java.util (pour efficacité)	-1
/	Ne fonctionne pas (retour erroné, ou pas du bon type attendu) Ne fonctionne pas (retour eronné, ou pas du bon ty... ▾	0-6
/	Fonctionne mais ne passe pas les tests fournis initialement Fonctionne mais ne passe pas les tests fournis init... ▾	7-14
/	Passe tous les tests fournis initialement Passe tous les tests fournis initialement ▾	15-18
/	Passe des tests supplémentaires plus complets Passe vos tests supplémentaires plus complets ▾	19-20
Notation du programme pour le classement : note servant uniquement au classement		
Notation	Remarque / explication	Note
Lisibilité du code	Notation subjective, facilité à comprendre le code.	0-20
Qualité du code	Utilisation de Codacy, FindBugs ou PMD pour déterminer la qualité du code.	0-20
Efficacité	Efficacité du programme (complexité algorithmique)	0-20
Temps d'exécution	Temps d'exécution du programme pour 50 itérations	0-20

Certaines classes java sont nommées différemment de Algo, dans le [salon discord dédié](#), pour une [question](#) concernant le nom des méthodes modifiées JMB a répondu **malus de -2**. J'ai donc appliqué le même malus pour les classes dont le nom n'est pas "Algo".

Algorithmes simplicité

Notation de l'algorithme : 03simplicite.java

03simplicite.java		
Notation du programme		
Notation	Remarque / explication	Note
<p style="text-align: center;">*</p> <p>Fonctionne mais ne pas... ▾</p>	<p>La classe est nommée "Simplicite" au lieu de "Algo"... Dans cette notation je la renommerais et malus -2 sur la note finale.</p> <p>L'algorithme ne prend pas en compte la consigne de repasser à 9 au lieu de 10 lorsque le compteur dépasse.</p> <p>Attendu : 1a1A9a1a -> Réel : 1a1A10a</p> <p>Voir testRLE</p>	12 - 2







Notation du programme pour le classement Simplicité			
Coefficient	Notation	Remarque / explication	Note
50	Lisibilité du code	Notation subjective, facilité à comprendre le code.	16
40	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	18
5	Efficacité	Méthodes "normal" : RLE : $O(n)$ UnRLE : $O(n)$ Méthodes récursives : RLE : $O(n * k)$ UnRLE : $O(n * k)$	20
5	Temps d'exécution de RLE / unRLE	Impossible de déterminer avec mon test car le comportement de l'algorithme n'est pas complet (test RLE échoué et unRLE utilise RLE...), cependant cette approche algorithmique est normalement très rapide.	10

Note finale : 10

Moyenne pour le classement : **16.7** -> $((16*50)+(18*40)+(20*5)+(10*5))/100$

**Fonctionne mais ne passe pas tout les tests fournis initialement*

Détails de la notation de 03simplicite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecuratif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecuratif	Test le décodage récursif	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecuratifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	/
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	/

Détails notation qualité du code	
Note attribuée	Explication notation qualité de code
18	<div><div><div><div>MINOR</div><div>Code style</div></div><div>The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'</div><div>src/main/java/iut/sae/algo/Algo.java</div><div>67public static String RLE(String in, int iteration) throws AlgoException{</div></div><div><div>MINOR</div><div>Code style</div></div><div>The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'</div><div>src/main/java/iut/sae/algo/Algo.java</div><div>13public static String RLE(String in){</div></div> <div><div>MEDIUM</div><div>Code style</div></div> <div>Avoid reassigning parameters such as 'in'</div> <div>src/main/java/iut/sae/algo/Algo.java</div> <div>84in = unRLE(in);</div>

MEDIUM

Code style

Avoid reassigning parameters such as 'in'

src/main/java/iut/sae/algo/Algo.java

69in = RLE(in);

Extrait Codacy

Ici ces critiques sont "inévitables", le nom de la méthode RLE est imposé par l'exercice et la récursivité impose d'appeler la variable en tant que paramètre

Pas de bugs mentionnables avec FindBugs.

```
C:\Users\firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\ma
\java\iut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text
[WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argumen
-r <file> to output the report to a file instead.
[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.
0/pmd_userdocs_incremental_analysis.html
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:6: UseUtilityClass:
This utility class has a non-private constructor
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:13: MethodNamingCon
entions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:67: MethodNamingCor
entions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'
```

Capture d'écran de PMD.

PMD détecte un constructeur non privé (pas important dans cet exercice) et un problème dans le nom des méthodes comme Codacy, donc pas de problèmes notables.

Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
20	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p> <p>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</p> <p>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</p>

Notation de l'algorithme : 05simplicite.java

05simplicite.java			
Notation du programme			
Notation		Remarque / explication	Note
* Passe tous les tests fou... ▾		Changement de nom de classe nécessaire -2. Ne passe pas mes tests de performances réglés avec 50 itérations dans un temps raisonnable.	18 - 2
Notation du programme pour le classement Simplicité			
Coefficient	Notation	Remarque / explication	Note
50	Lisibilité du code	Notation subjective, facilité à comprendre le code.	19
40	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles, etc...	16
5	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k) Utilisation de String ce qui est lent.	10







5	Temps d'exécution de RLE / unRLE	Temps non raisonnable pour 50 itérations, ce temps concerne donc 20 itérations et la note en est impactée.	10
---	----------------------------------	--	----

Note finale : **16**

Moyenne pour le classement : **16.9** -> $((19*50)+(16*40)+(10*5)+(10*5))/100$

**Passe tous les tests fournis initialement*

Détails de la notation de 05simplicite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecuratif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecuratif	Test le décodage récursif	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecuratifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	/ Pas assez efficace pour 50. Temps trop grand.
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	/ Pas assez efficace pour 50. Temps trop grand.

Détails notation qualité du code	
Note attribuée	Explication notation qualité de code
18	<div><div><div><div>MINORCode style</div><div>The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'</div><div>src/main/java/iut/sae/algo/Algo.java</div><div>67public static String RLE(String in, int iteration) throws AlgoException{</div></div><div><div>MINORCode style</div><div>The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'</div><div>src/main/java/iut/sae/algo/Algo.java</div><div>13public static String RLE(String in){</div></div><div><div>MEDIUMCode style</div><div>Avoid reassigning parameters such as 'in'</div><div>src/main/java/iut/sae/algo/Algo.java</div><div>84in = unRLE(in);</div></div><div><div>MEDIUMCode style</div><div>Avoid reassigning parameters such as 'in'</div><div>src/main/java/iut/sae/algo/Algo.java</div><div>69in = RLE(in);</div></div></div><div><p><i>Extrait Codacy</i></p><p>Ici ces critiques sont "inévitables", le nom de la méthode RLE est imposé par l'exercice et la récursivité impose d'appeler la variable en tant que paramètre...</p><p>Pas de bugs mentionnables avec FindBugs.</p><pre>C:\Users\firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text [WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argument -r <file> to output the report to a file instead. [WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:3: UseUtilityClass: This utility class has a non-private constructor C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:4: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:5: LiteralsFirstInComparisons: Position literals first in String comparisons C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:5: ControlStatementBraces: This statement should have braces C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:23: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'</pre><p><i>Capture d'écran de PMD.</i></p><p>PMD détecte un constructeur non privé (pas important dans cet exercice) et un problème dans le nom des méthodes comme Codacy. Également il détecte des problèmes plus concrets dans la ligne 5 ou un <code>if</code> est construit sans accolades ce qui peut induire en erreur la compréhension du code.</p></div></div>

Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
20	RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée. RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations. unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée. unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.

Notation de l'algorithme : 18simplicite.java

18simplicite.java			
Notation du programme			
Notation		Remarque / explication	Note
<div>* Fonctionne mais ne pas... ▾</div>		La classe est nommée "Simplicite" au lieu de "Algo"... Dans cette notation je la renommerais et malus -2 sur la note finale. Ne passe pas testUnRLERecursif Valeur attendu : abc Valeur recu : (vide)	12 - 2
Notation du programme pour le classement Simplicité			
Coefficient	Notation	Remarque / explication	Note
50	Lisibilité du code	Notation subjective, facilité à comprendre le code.	17
40	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	18
5	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k)	20







18

5	Temps d'exécution de RLE / unRLE.	Exécution rapide pour RLE et plus lente pour unRLE.	16
---	-----------------------------------	---	----

Note finale : 10

Moyenne pour le classement : 17.5 -> $((17*50)+(18*40)+(20*5)+(16*5))/100$

Détails de la notation de 18simplicite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecursif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecursif	Test le décodage récursif	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecursifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	~125ms
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	~4720ms

Détails notation qualité du code	
Note attribuée	Explication notation qualité de code
18	<div data-bbox="483 342 1263 772"> </div> <p><i>Extrait Codacy</i></p> <p>Nom de méthode imposé par l'exercice. Pas de bugs mentionnables avec FindBugs.</p> <pre> C:\Users\firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell11\src\main\java\iut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text [WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argument -r <file> to output the report to a file instead. [WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell11\src\main\java\iut\sae\algo\Algo.java:3: UseUtilityClass: This utility class has a non-private constructor C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell11\src\main\java\iut\sae\algo\Algo.java:5: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell11\src\main\java\iut\sae\algo\Algo.java:24: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell11\src\main\java\iut\sae\algo\Algo.java:57: ControlStatementBraces: This statement should have braces </pre> <p><i>Capture d'écran de PMD.</i> Un <code>if</code> sans accolade ligne 57 qui peut impacter la compréhension du code.</p>
Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
20	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p> <p>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</p> <p>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</p>

Notation de l'algorithme : 25simplicite.java







25simplicite.java			
Notation du programme			
Notation		Remarque / explication	Note
<div>* Fonctionne mais ne pas... ▾</div>		<p>La classe est nommée "Simplicite" au lieu de "Algo"... Dans cette notation je la renommerais et malus -2 sur la note finale.</p> <p>TestUnRLERecursif</p> <p>Valeur attendu : <i>abc</i></p> <p>Valeur recu : <i>1a1b1c</i></p> <p>Peut renvoyer des exceptions non attendus mais il n'est pas précise le comportement attendu donc je n'en tiens pas compte</p>	10 - 2
Notation du programme pour le classement Simplicité			
Coefficient	Notation	Remarque / explication	Note
50	Lisibilité du code	Notation subjective, facilité à comprendre le code.	14
40	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	19
5	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k)	20

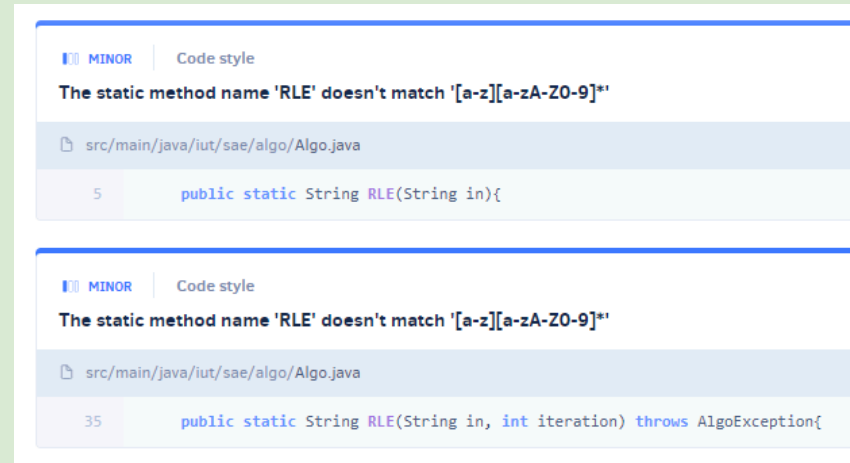
5	Temps d'exécution de RLE / unRLE	L'algorithme ne remplit pas tous les tests de base, il ne peut donc pas être évalué par mes tests de performances (j'utilise un assertTrue en partant du principe que l'algo fonctionne dans toutes circonstances, ici non) + unRLE remplit la mémoire (java.lang.OutOfMemoryError pour 50 itérations...)	0
---	----------------------------------	---	---

Note finale : 8

Moyenne pour le classement : **15.6** -> $((14*50)+(19*40)+(20*5)+(0*5))/100$

Détails de la notation de 25simplicite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecursif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecursif	Test le décodage récursif - bizarre qu'il ne passe pas ici	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecursifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	~115ms
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	/ ne passe pas testUnRLERecursif
Détails notation qualité du code		
Note attribuée	Explication notation qualité de code	
14	Beaucoup trop de commentaires (chaque ligne est commentée et certaines fois même pour des choses très simples), cela complique la lecture. Prend en compte un cas ou un chiffre est plus grand que 9 si j'ai bien compris ? -> normalement impossible donc inutile	



Extrait Codacy

```
C:\Users\firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main
(java\ut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text
[WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argument
-r <file> to output the report to a file instead.
[WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.
9/pmd_userdocs_incremental_analysis.html
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main\java\ut\sae\algo\Algo.java:4: UseUtilityClass:
This utility class has a non-private constructor
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main\java\ut\sae\algo\Algo.java:5: MethodNamingConventions:
The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main\java\ut\sae\algo\Algo.java:35: MethodNamingConv
entions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'
```

Capture d'écran de PMD.

Rien de particulier dans les analyses des différents outils.

Détails détermination de la complexité algorithmique

Note attribuée	Explication complexité
20	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p> <p>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</p> <p>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</p>

Notation de l'algorithme : 38simplicite.java

38simplicite.java			
Notation du programme			
Notation		Remarque / explication	Note
* Passe tous les tests fou... ▾		java.lang.StackOverflowError -> Recursion excessive / infini ? Problème de récursion à partir d'un chiffre plus grand que 3 ou inférieur à 1.	17
Notation du programme pour le classement Simplicité			
Coefficient	Notation	Remarque / explication	Note
50	Lisibilité du code	Notation subjective, facilité à comprendre le code.	16
40	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	13
5	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k)	16 (des problèmes dans unRLE)

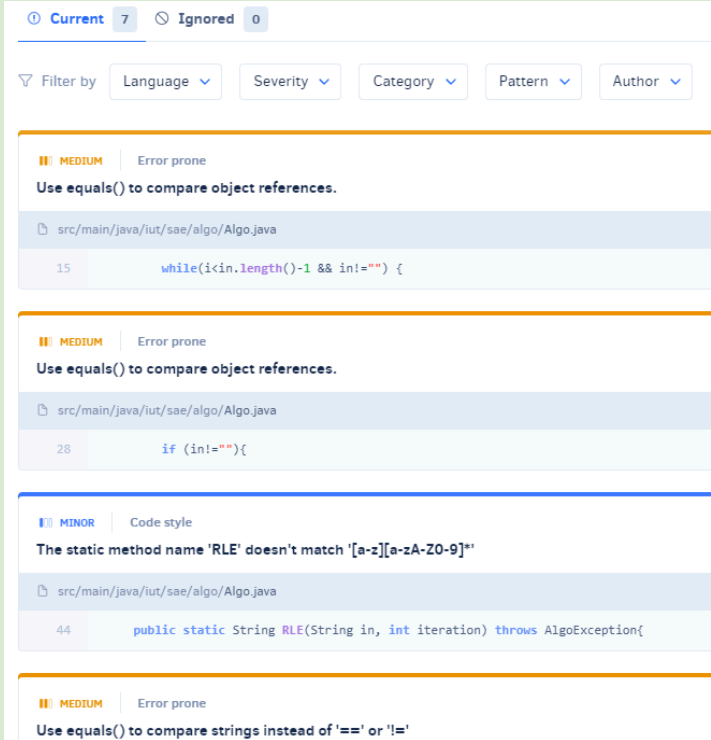
5	Temps d'exécution de RLE / unRLE	Rapide pour RLE, infinie pour unRLE (problème de reccursion). Utilise des String, trop lent pour 50 itérations...	10
---	----------------------------------	---	----

Note finale : 9

Moyenne pour le classement : **14.5** -> $((16*50)+(13*40)+(16*5)+(10*5))/100$

Détails de la notation de 38simplicite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	✓
testRLERecuratif	Test l'encodage récursif	✓
testUnRLE	Test le décodage	✓
testUnRLERecuratif	Test le décodage récursif	✓
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	✓
testRLERecuratifGrand	Test le comportement pour une itération = 5	✓
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	Trop long
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	Trop long
Détails notation qualité du code		

Note attribuée	Explication notation qualité de code
13	 <p>Extrait Codacy Utilisation de "==" pour comparer des String. Problème de récursion détecter par FindBugs</p> <pre> C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:4: UseUtilityClass: This utility class has a non-private constructor C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:11: MethodNamingConv entions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:15: CompareObjectsWi thEquals: Use equals() to compare object references. C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:15: UseEqualsToCompa reStrings: Use equals() to compare strings instead of '==' or '!=' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:28: CompareObjectsWi thEquals: Use equals() to compare object references. C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:28: UseEqualsToCompa reStrings: Use equals() to compare strings instead of '==' or '!=' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:44: MethodNamingConv entions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:62: CompareObjectsWi thEquals: Use equals() to compare object references. C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:62: UseEqualsToCompa reStrings: Use equals() to compare strings instead of '==' or '!=' </pre> <p>Capture d'écran de PMD. Pareil que pour Codacy par PMD.</p>

Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
16	RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée. RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations. unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée. unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.

Classement Simplicité

Position	Algorithme	Note classement
1er 🏆	18simplicite.java	17.5
2ème 🥈	05simplicite.java	16.9
3ème 🥉	03simplicite.java	16.7
4ème 🙌	25simplicite.java	15.6
5ème 🙌	38simplicite.java	14.5

Algorithmes efficacité

Notation de l'algorithme : 06efficacite.java







06efficacite.java			
Notation du programme			
Notation		Remarque / explication	Note
<div>* Passe tous les tests fou... ▾</div>		<div>La classe est nommée "Algo1" au lieu de "Algo"...</div> <div>Dans cette notation je la renommerais et malus -2 sur la note finale.</div> <div>Erreur test10Caractere :</div> <div>Valeur attendu : 1a1A9a</div> <div>Valeur recu : 1a1A9a0a</div>	18 - 2
Notation du programme pour le classement Efficacité			
Coefficient	Notation	Remarque / explication	Note
5	Lisibilité du code	Notation subjective, facilité à comprendre le code.	14 (boucle while dans une boucle for dans RLE)
5	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	18
40	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k)	18 (la boucle while de RLE pourrait rallonger un peu les temps de calcules)

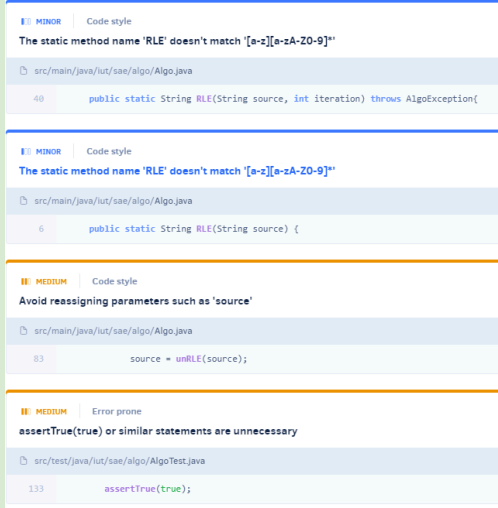
50	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE. L'idée derrière cet algorithme est bien cependant il n'est pas fiable et se trompe dans de trop nombreux cas pour avoir une meilleure note et être mieux placé dans le classement.	20
----	----------------------------------	--	----

Note finale : **16**

Moyenne pour le classement : **18.8** -> $((20*50)+(18*40)+(18*5)+(14*5))/100$

Détails de la notation de 06efficacite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecursif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecursif	Test le décodage récursif	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecursifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	~106ms
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	~205ms

Détails notation qualité du code	
Note attribuée	Explication notation qualité de code
18	 <p>Extrait Codacy</p> <pre>C:\Users\firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main\java\iut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text [WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argument -r <file> to output the report to a file instead. [WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main\java\iut\sae\algo\Algo.java:4: UseUtilityClass: This utility class has a non-private constructor C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main\java\iut\sae\algo\Algo.java:6: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewell\src\main\java\iut\sae\algo\Algo.java:40: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'</pre> <p>Extrait PMD</p> <p>Problèmes classiques détectés, je n'en tiens pas compte.</p>
Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
18	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p> <p>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</p> <p>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</p> <p>Pas 20 car il y a une boucle for qui me semble évitable.</p>

Notation de l'algorithme : 12efficacite.java

12efficacite.java			
Notation du programme			
Notation		Remarque / explication	Note
<div>* Passe vos tests supplé... ▾</div>		<i>Manque l'importation des package (n'impacte pas la note)</i> Pas de surcharge des méthodes donc les méthodes n'ont pas le bon nom, malus -2. Pas de throw dans les méthodes qui en avait. Ajout de throw dans la signature de 2 méthodes par le correcteur pour pouvoir faire fonctionner les tests... -2 malus.	19-4
Notation du programme pour le classement Efficacité			
Coefficient	Notation	Remarque / explication	Note
5	Lisibilité du code	Notation subjective, facilité à comprendre le code.	19
5	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	19
40	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k) ~ O(n/2 * k)	20

50	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE.	20
----	----------------------------------	---	----

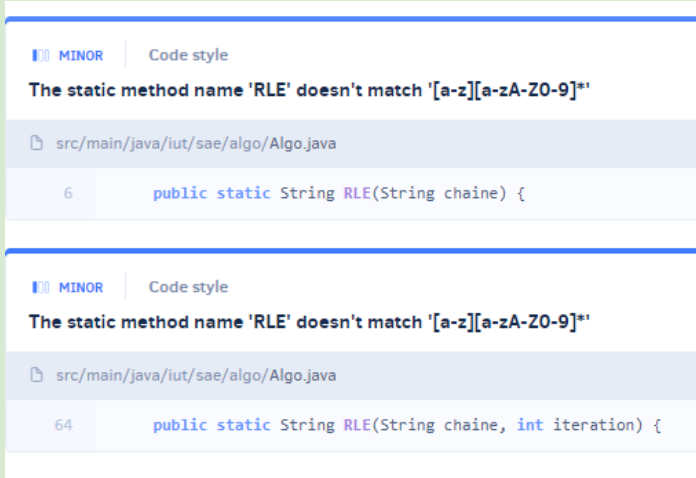
Note finale : 15

Moyenne pour le classement : **19.85** -> $((20*50)+(20*40)+(18*5)+(19*5))/100$

Bilan : une fois corriger quelques problèmes de throw et de package, l'algorithme 12efficacite.java passe tous les tests et mérite une bonne note cependant il y a des soucis évitables facilement (ce qui ne l'empêche pas d'être très bien placé dans le classement...)

Détails de la notation de 12efficacite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	✓ ▾
testRLERecuratif	Test l'encodage récursif	✓ ▾
testUnRLE	Test le décodage	✓ ▾
testUnRLERecuratif	Test le décodage récursif	✓ ▾
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	✓ ▾
testRLERecuratifGrand	Test le comportement pour une itération = 5	✓ ▾
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	~108ms
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	~210ms

Détails notation qualité du code	
Note attribuée	Explication notation qualité de code
	 <p><i>Extrait Codacy</i></p> <pre> C:\Users\firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\ java\iut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text [WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argument -r <file> to output the report to a file instead. [WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2. 0/pmd-userdocs/incremental-analysis.html C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:4: UseUtilityClass: This utility class has a non-private constructor C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:6: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:64: MethodNamingConv ventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' </pre> <p>Très peu de problèmes mineurs détectés par les outils d'évaluation de la qualité de ce code.</p>
Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
20	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p> <p>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</p> <p>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</p>

Notation de l'algorithme : 14efficacite.java

14efficacite.java.java			
Notation du programme			
Notation		Remarque / explication	Note
* <div>Passe vos tests supplé... ▾</div>		Problème de nom de classe "Efficacite" -> "Algo" : malus -2. Problème d'importation des packages.	20-2
Notation du programme pour le classement Efficacité			
Coefficient	Notation	Remarque / explication	Note
5	Lisibilité du code	Notation subjective, facilité à comprendre le code.	20
5	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	17
40	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k)	20
50	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE.	20

Note finale : **18**Moyenne pour le classement : **19.85** -> $((20*50)+(20*40)+(17*5)+(20*5))/100$

Détails de la notation de 14efficacite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	✓
testRLERecuratif	Test l'encodage récursif	✓
testUnRLE	Test le décodage	✓
testUnRLERecuratif	Test le décodage récursif	✓
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	✓
testRLERecuratifGrand	Test le comportement pour une itération = 5	✓
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	~112ms
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	~202ms
Détails notation qualité du code		
Note attribuée	Explication notation qualité de code	

17

```

MINOR | Code style
The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'

src/main/java/iut/sae/algo/Algo.java
15      public static String RLE(String in){

MINOR | Code style
The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'

src/main/java/iut/sae/algo/Algo.java
36      public static String RLE(String in, int iteration) throws AlgoException{

```

Extrait Codacy

```

C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:7: UseUtilityClass:
This utility class has a non-private constructor
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:15:      MethodNamingConv
entions:      The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:36:      MethodNamingConv
entions:      The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*'
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:38:      ControlStatement
Braces: This statement should have braces
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:40:      ControlStatement
Braces: This statement should have braces
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:55:      ControlStatement
Braces: This statement should have braces
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:71:      ControlStatement
Braces: This statement should have braces
C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:73:      ControlStatement
Braces: This statement should have braces

```

Extrait PMD

Uniquement des petites améliorations notables en utilisant les accolades lors des boucles ou des if.

Détails détermination de la complexité algorithmique

Note attribuée

Explication complexité

20

RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.

RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.

unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.

unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.

Notation de l'algorithme : 28efficacite.java

28efficacite.java			
Notation du programme			
Notation		Remarque / explication	Note
* Fonctionne mais ne pas... ▾		Problème de nom de classe "efficacite1" -> "Algo" : malus -2. Problème d'importation des packages. Aucune méthode unRLE : malus -2	12-4
Notation du programme pour le classement Efficacité			
Coefficient	Notation	Remarque / explication	Note
5	Lisibilité du code	Notation subjective, facilité à comprendre le code.	16
5	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	14
40	Efficacité	Méthode "normal" : RLE : O(n) Méthode récursive : RLE : O(n * k)	8 (manque la moitié des codes - 10 et RLE récursif est copié collé donc redondance d'un même programme -2)







50	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE.	10 (manque unRLE...)
----	----------------------------------	---	----------------------

Note finale : 8

Moyenne pour le classement : 9.7-> $((10*50)+(8*40)+(14*5)+(16*5))/100$

Bilan : manque la moitié des méthodes. RLE récursif ne réutilise pas RLE() mais plutôt est un copié collé modifié pour intégrer la récursion.

Détails de la notation de 28efficacite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecuratif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecuratif	Test le décodage récursif	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecuratifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	~182ms
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	/
Détails notation qualité du code		

Note attribuée	Explication notation qualité de code
14	 <p><i>Extrait Codacy</i></p> <pre> C:\Users\Firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\ java\ut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text [WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argument -r <file> to output the report to a file instead. [WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2. 0/pmd_userdocs_incremental_analysis.html C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\ut\sae\algo\Algo.java:3: UseUtilityClass: This utility class has a non-private constructor C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\ut\sae\algo\Algo.java:9: MethodNamingConventions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\ut\sae\algo\Algo.java:10: CompareObjectsWi thEquals: Use equals() to compare object references. C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\ut\sae\algo\Algo.java:10: UseEqualsToCompa reStrings: Use equals() to compare strings instead of '==' or '!=' C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\ut\sae\algo\Algo.java:50: MethodNamingConv entions: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\ut\sae\algo\Algo.java:57: CompareObjectsWi thEquals: Use equals() to compare object references. C:\Users\Firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\ut\sae\algo\Algo.java:57: UseEqualsToCompa reStrings: Use equals() to compare strings instead of '==' or '!=' </pre> <p><i>Extrait PMD</i></p> <p>Utilisation de "==" pour comparer des strings. Méthode RLE récursive n'utilise pas RLE() donc code redondant.</p>
Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
8	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p>

Notation de l'algorithme : 59efficacite.java







59efficacite.java			
Notation du programme			
Notation		Remarque / explication	Note
* Fonctionne mais ne pas... ▾		Problème de nom de classe "Efficacite" -> "Algo" : malus -2. Test RLE : Expected [9H4W] but was [9W5W] Test10Caractere : Expected [9a1a] but was [9a2a] testRLERecursifGrand : Expected [13211913211W13211413211W'] but was [13211913211W13211513211W]	13 - 2
Notation du programme pour le classement Efficacité			
Coefficient	Notation	Remarque / explication	Note
5	Lisibilité du code	Notation subjective, facilité à comprendre le code.	19
5	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code. Utilisation de l'outil FindBugs afin de détecter de potentiels problèmes / bugs dans le programme. Utilisation de l'outil PMD pour analyser et détecter des défauts tels que des variables inutilisées, des blocs catch vides, la création d'objets inutiles...	19
40	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives : RLE : O(n * k) UnRLE : O(n * k)	20

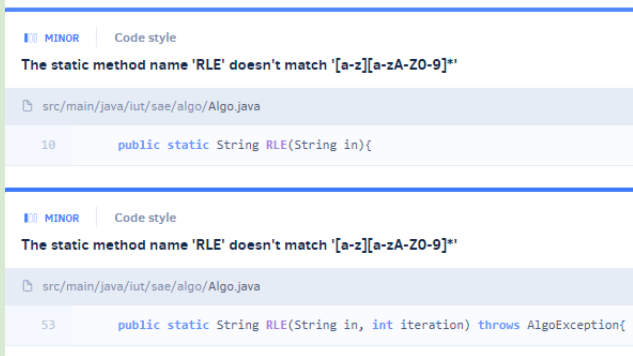
50	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE.	18 (Certains cas ne fonctionnent pas bien : -2)
----	----------------------------------	---	--

Note finale : 11

Moyenne pour le classement : **18.9** -> $((18*50)+(20*40)+(19*5)+(19*5))/100$

Détails de la notation de 59efficacite.java

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecuratif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecuratif	Test le décodage récursif	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecuratifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	~115ms
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	~202ms

Détails notation qualité du code	
Note attribuée	Explication notation qualité de code
19	<div>  <p><i>Extrait Codacy</i></p> <pre> C:\Users\firew\Downloads\pmd-bin-7.2.0\bin>pmd check -d C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java -R rulesets/java/quickstart.xml -f text [WARN] Progressbar rendering conflicts with reporting to STDOUT. No progressbar will be shown. Try running with argument -r <file> to output the report to a file instead. [WARN] This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-doc-7.2.0/pmd_userdocs_incremental_analysis.html C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:3: UseUtilityClass: This utility class has a non-private constructor C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:10: MethodNamingConvention: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:13: ControlStatementBraces: This statement should have braces C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:53: MethodNamingConvention: The static method name 'RLE' doesn't match '[a-z][a-zA-Z0-9]*' C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:57: ControlStatementBraces: This statement should have braces C:\Users\firew\Documents\SAEALGO\sae2024-2-02-Firewelll\src\main\java\iut\sae\algo\Algo.java:79: ControlStatementBraces: This statement should have braces </pre> <p><i>Extrait PMD</i></p> <p>Problèmes mineurs d'accolades dans les boucles / if.</p> </div>
Détails détermination de la complexité algorithmique	
Note attribuée	Explication complexité
20	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p> <p>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</p> <p>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</p>

Classement Efficacité

Position	Algorithme	Note classement
1er 🏆 exaequo	12efficacite.java	19.85
1er 🏆 exaequo	14efficacite.java	19.85
3ème 🥉	59efficacite.java	18.9
4ème 🙌	06efficacite.java	18.8
5ème 🙌	28efficacite.java	9.7

Algorithme sobriété

Notation de l'algorithme : 13sobriete.c

x.java			
Notation du programme			
Notation		Remarque / explication	Note
<div>* Ne fonctionne pas (reto... ▾)</div>		<p>Mauvaise signature d'algorithme ? Les fonctions prennent 2 chaînes de caractères en paramètre au lieu d'une chaîne ou d'une chaîne et d'un entier. (Ici les fonctions sont renommées par moi même mais ce ne sont pas leur nom dans le fichier original).</p> <p>Pour ce programme j'ai adapté les certains tests pour qu'il soit en accord avec les fonctions.</p> <pre>void RLE(char *s, char *r) void unRLE(char *r, char *s) void RLE_R(char *s, char *r) void unRLE_R(char *r, char *s)</pre>	6 + 2 parce que je vois l'effort de faire en C etc et qu'il est possible que ce soit ma propre incompréhension
Notation du programme pour le classement Sobriété			
Coefficient	Notation	Remarque / explication	Note
20	Lisibilité du code	Notation subjective, facilité à comprendre le code.	12
20	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code.	20
40	Efficacité	Méthodes "normal" : RLE : O(n) UnRLE : O(n) Méthodes récursives hors sujet	10







20	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE.	10
----	----------------------------------	---	----

Note finale : 8

Moyenne pour le classement : **12.4** -> $((10*20)+(10*40)+(20*20)+(12*20))/100$

Bilan : Algorithme sûrement prometteur mais les signatures des fonctions ne sont pas attendues comme ceci.

Détails de la notation de 13sobriete.c

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecursif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecursif	Test le décodage récursif	
test10Caractere	Test le comportement pour 10 caractères identiques d'affilé	
testRLERecursifGrand	Test le comportement pour une itération = 5	
Test de rapidité RLE	Test qui lance l'algorithme RLE sur un nombre de 50 itérations Si RLE ne marche pas alors ce test est inutilisable.	/
Test de rapidité unRLE	Test qui lance l'algorithme unRLE sur un nombre de 50 itérations. (Lance également RLE pour avoir un résultat) Si RLE ou unRLE ne marche pas alors ce test est inutilisable.	/
Détails notation qualité du code		
Note attribuée	Explication notation qualité de code	
20	<div><div>Issues 0</div><div>Extrait Codacy</div><div>Aucun problème de qualité.</div></div>	
Détails détermination de la complexité algorithmique		
Note attribuée	Explication complexité	

10	<p>RLE(String in, String in) : $O(n)$, où n est la longueur de la chaîne d'entrée..</p> <p>unRLE(String in, String in) : $O(n)$, où n est la longueur de la chaîne codée.</p>
----	---

Notation de l'algorithme : 37sobriete.c

37sobriete.c.java			
Notation du programme			
Notation		Remarque / explication	Note
* Fonctionne mais ne pas... ▾		Test le décodage récursif échoue à partir d'un entier supérieur à 2, étrange...	16
Notation du programme pour le classement Sobriété			
Coefficient	Notation	Remarque / explication	Note
20	Lisibilité du code	Notation subjective, facilité à comprendre le code.	16
20	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code.	16
40	Efficacité	Méthodes "normal" : RLE : $O(n)$ UnRLE : $O(n)$ Méthodes récursives : RLE : $O(n * k)$ UnRLE : $O(n * k)$	16
20	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE.	20

Note finale : 20

Moyenne pour le classement : **16,8** -> $((20*20)+(16*40)+(16*20)+(16*20))/100$

Détails de la notation de 37sobriete.c

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	<div>✓</div>
testRLERecursif	Test l'encodage récursif	<div>✓</div>
testUnRLE	Test le décodage	<div>✓</div>
testUnRLERecursif	Test le décodage récursif	<div>✗</div>
Détails notation qualité du code		
Note attribuée	Explication notation qualité de code	
16	<div><div>Issues 4</div><div>Filter All categories All levels All patterns</div><div><div><div>MINOR</div><div>Security</div><div>The "strlen" family of functions does not handle strings that are not null terminated.</div></div><div>12 if (strlen(s) == 0) {</div></div><div><div>MINOR</div><div>Security</div><div>Does not handle strings that are not \0-terminated; if given one it may perform an over-read (it could cause a crash if unprotected) (CWE-126).</div></div><div>12 if (strlen(s) == 0) {</div><div><div>MINOR</div><div>Security</div><div>The "strlen" family of functions does not handle strings that are not null terminated.</div></div><div>17 int tailleHsgEncode = strlen(s) * 4;</div><div><div>MINOR</div><div>Security</div><div>Does not handle strings that are not \0-terminated; if given one it may perform an over-read (it could cause a crash if unprotected) (CWE-126).</div></div><div>17 int tailleHsgEncode = strlen(s) * 4;</div></div>	
Détails détermination de la complexité algorithmique		
Note attribuée	Explication complexité	
	<div><div>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</div><div>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</div><div>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</div><div>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</div></div>	

Notation de l'algorithme : 44sobriete.c

44sobriete.java			
Notation du programme			
Notation		Remarque / explication	Note
* Passe vos tests supplé... ▾		Aucune remarque particulière.	20
Notation du programme pour le classement Sobriété			
Coefficient	Notation	Remarque / explication	Note
20	Lisibilité du code	Notation subjective, facilité à comprendre le code.	19
20	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code.	14
40	Efficacité	Méthodes "normal" : RLE : $O(n)$ UnRLE : $O(n)$ Méthodes récursives : RLE : $O(n * k)$ UnRLE : $O(n * k)$	20
20	Temps d'exécution de RLE / unRLE	Temps d'exécution pour 50 itérations de RLE et unRLE.	20

Note finale : 20

Moyenne pour le classement : 18.6 -> $((19*20)+(20*40)+(14*20)+(20*20))/100$

Détails de la notation de 44sobriete.c

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	<div>✓</div>
testRLERecursif	Test l'encodage récursif	<div>✓</div>
testUnRLE	Test le décodage	<div>✓</div>
testUnRLERecursif	Test le décodage récursif	<div>✓</div>
Détails notation qualité du code		
Note attribuée	Explication notation qualité de code	
14	<div><div>Issues 10 Duplication 2</div><div>Extrait Codacy</div></div>	
Détails détermination de la complexité algorithmique		
Note attribuée	Explication complexité	
20	<p>RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.</p> <p>RLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne d'entrée après toutes les itérations.</p> <p>unRLE(String in) : $O(n)$, où n est la longueur de la chaîne codée.</p> <p>unRLE(String in, int iteration) : $O(k * p)$, où k est le nombre d'itérations et p est la longueur de la chaîne codée après toutes les itérations.</p>	





Notation de l'algorithme : 19sobriete.py

19sobriete.py			
Notation du programme			
Notation		Remarque / explication	Note
* Fonctionne mais ne pas... ▾		Il n'y a que la fonction RLE, tout le reste est absent.	5 (¼ de 20)
Notation du programme pour le classement Sobriété			
Coefficient	Notation	Remarque / explication	Note
20	Lisibilité du code	Notation subjective, facilité à comprendre le code.	20
20	Qualité du code	Utilisation de Codacy pour déterminer la qualité du code.	20
40	Efficacité	Méthodes "normal" : RLE : O(n)	20
20	Temps d'exécution de RLE / unRLE	Pas d'algo récursif.	0

Note finale : 5

Moyenne pour le classement : 16 -> $((0*20)+(20*40)+(20*20)+(20*20))/100$

Détails de la notation de 19sobriete.py

Détails des tests		
Nom du test	Explication du test	État du test
testRLE	Test l'encodage	
testRLERecursif	Test l'encodage récursif	
testUnRLE	Test le décodage	
testUnRLERecursif	Test le décodage récursif	
Détails notation qualité du code		
Note attribuée	Explication notation qualité de code	
20	<div><div><div><div><div></div><div></div><div></div></div><div>MINOR</div><div>Code Style</div><div>Trailing whitespace</div></div><div><div>36</div></div></div><div>Extrait Codacy</div><p>Problème mineur.</p></div>	
Détails détermination de la complexité algorithmique		
Note attribuée	Explication complexité	
8	RLE(String in) : $O(n)$, où n est la longueur de la chaîne d'entrée.	

Classement Sobriété

Position	Algorithme	Note classement
1er 🏆	44sobriete.c	18.6
2ème 🥈	37sobriete.c	16,8
3ème 🥉	19sobriete.py	16
4ème 🙌	13sobriete.c	12.4



Bouyssou Melvin 3A