AUSSENAC
Thomas

Rapport d’évaluation des Algorithmes

1. Introduction

L'objectif de cette SAE est de concevoir et évaluer des solutions algorithmiques pour la compression de données en utilisant l'algorithme Run-Length Encoding (RLE). Le projet se déroule en deux phases distinctes : une première phase individuelle où chaque participant doit proposer des solutions dans différentes catégories d'évaluation (Simplicité, Efficacité, Sobriété), suivie d'une phase en binôme où les solutions seront comparées et évaluées selon plusieurs critères prédéfinis.

2. Présentation des outils d’évaluation

Pour évaluer et classer ces différents algorithmes, nous utiliserons plusieurs outils spécifiques à chaque langage :

- JUnit : pour effectuer des tests et vérifier la conformité des résultats pour les fichiers Java.
- PyUnit : pour effectuer des tests et vérifier la conformité des résultats pour les fichiers Python.
- CUnit : pour effectuer des tests et vérifier la conformité des résultats pour les fichiers C.

Nous utiliserons également Codacy pour obtenir des informations sur la qualité du code, telles que la complexité cyclomatique, le nombre de lignes commentées et de lignes de code dupliquées.

Pour l'exécution des tests, nous utiliserons IntelliJ. Nos tests seront exécutés sur un ordinateur portable fonctionnant sous Linux Ubuntu 22.04. Ce choix permettra de minimiser les interruptions causées par les tâches indésirables en arrière-plan par rapport à un environnement Windows. Aucun programme tel que Discord, Firefox ou autre ne sera en cours d'exécution sur l'ordinateur pendant la phase de test.

Les tests, qu'ils soient JUnit, PyUnit ou CUnit, seront équivalents et impliqueront les mêmes opérations requises.

3. Liste des algorithmes

3.1 Algorithmes de Simplicité

nom du fichier	langage	Lien vers le fichier
simplicite19.java	Java	lien
simplicite36.java	Java	lien
simplicite52.java	Java	lien
simplicite58.java	Java	lien
simplicite62.java	Java	lien

3.2 Algorithmes d'efficacité

nom du fichier	langage	Lien vers le fichier
efficacite34.c	C	lien
efficacite37.c	C	lien
efficacite41.java	Java	lien
efficacite55.java	Java	lien
efficacite62.java	Java	lien

3.3 Algorithmes de sobriété

nom du fichier	langage	Lien vers le fichier
19sobriete.py	Python	lien
34sobriete.py	Python	lien
sobriete62.java	Java	lien
sobriete66.java	Java	lien

4. Notation

4.1 Notation Algorithmes de simplicité

nom fichier	simplicite19.java
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui

Remarques:

- Il manque 3 méthodes sur 4
- La construction est inversé (caractère puis entier)
- Code trop espacé pour rien
- Aucune Javadoc ni commentaires

Note: 5/20

nom fichier	simplicite36.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui

Remarques:

- Noms de variables corrects
- Pas de commentaires
- Pas de Javadoc

Note: 20/20

nom fichier	simplicite52.java
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui

Remarques:

- Il manque unRLE(String,int)
- Utilisation d'une ArrayList inefficiente
- Ne vérifie pas si la liste en entrée est vide

Note: 5/20

nom fichier	simplicite58.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui

Remarques:

- Noms de variable peu explicite
- Aucune Javadoc ni commentaires

Note: 20/20

nom fichier	simplicite62.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui

Remarques:

- Noms de variables peu explicites
- Modification de l'input dans les unRLE
- Aucune Javadoc ni commentaires

Note: 20/20

4.2 Notation Algorithmes d’efficacité

nom fichier	efficacite34.c
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui
Complexité	null
Durée moyenne pour 65 itération “abc”	null

Remarques:

- Noms de variables corrects
- Code propre et assez bien commenter

Note: 10/20

nom fichier	efficacite37.c
Compilation	non
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui
Complexité	null
Durée moyenne pour 65 itération “abc”	null

Remarques:

- Noms de variables corrects
- Mauvaise condition d'arrêt pour le for
- Ne compile pas

Note: 2.5/20

nom fichier	efficacite41.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	6
Durée moyenne pour 65 itération “abc”	4.130s

Remarques:

- Noms de variables améliorable
- Aucune Javadoc ni commentaires

Note: 20/20

nom fichier	efficacite55.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	6
Durée moyenne pour 65 itération “abc”	4.366s

Remarques:

- Noms de variables corrects
- Aucune Javadoc
- Peu de commentaires

Note: 20/20

nom fichier	efficacite62.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	5
Durée moyenne pour 65 itération “abc”	3.992s

Remarques:

- Noms de variables corrects
- Code bien commenté
- Aucune Javadoc

Note: 20/20

4.2 Notation Algorithmes de sobriété

nom fichier	19sobriete.py
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui
Complexité	6
Durée moyenne pour 65 itération “abc”	null
Nombre de caractères RLE	563

Remarques:

- Aucune doc ni commentaires
- Absence de RLE(str, int), de unRLE et unRLE(str, int)

Note: 5/20

nom fichier	35sobriete.py
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui
Complexité	4
Durée moyenne pour 65 itération “abc”	null
Nombre de caractères RLE	401

Remarques:

- Aucune doc ni commentaires
- Renvoie 10A au lieu de 9A1A
- Absence de RLE(str, int), de unRLE et unRLE(str, int)

Note: 5/20

nom fichier	sobriete62.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	4
Durée moyenne pour 65 itération “abc”	null
Nombre de caractères RLE	804

Remarques:

- Noms de variables corrects
- L’appel récursif en stockant le résultat à chaque fois détruit les performances de l’algorithme
- Aucune Javadoc

Note: 20/20

nom fichier	sobriete66.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	5
Durée moyenne pour 65 itération “abc”	5.094s
Nombre de caractères RLE	572

Remarques:

- Noms de variables corrects
- Il manque un throw dans RLE(String, int)
- Aucune Javadoc ni commentaires

Note: 20/20

5. Classement des algorithmes

5.1 Algorithmes simples

Dans notre évaluation des algorithmes, nous privilégierons les aspects suivants : la lisibilité du code, la clarté des noms de variables, la qualité des commentaires, et la facilité de compréhension. Les algorithmes qui échouent aux tests ou ne compilent pas seront relégués au bas du classement.

1. simplicite52.java
2. simplicite19.java
3. simplicite36.java
4. simplicite62.java
5. simplicite58.java

5.2 Algorithmes efficaces

Dans notre classement des algorithmes, nous évaluerons la complexité cyclomatique, la durée d'exécution des tests et le niveau d'optimisation du code. Les algorithmes qui ne passent pas les tests ou qui ne compilent pas seront positionnés en bas du classement.

1. efficacite62.java
2. efficacite41.java
3. efficacite55.java

5.3 Algorithmes sobres

Dans notre classement des algorithmes, nous évaluerons la consommation mémoire, l'utilisation des ressources CPU, et la performance globale en termes de sobriété. Les algorithmes qui ne passent pas les tests ou qui ne compilent pas seront positionnés en bas du classement. Les évaluations ont été effectuées en exécutant les programmes individuellement sur le PC de test, et le classement repose sur l'observation des graphiques de consommation sans fournir de données précises.

1. sobriete66.java
2. 35sobriete.py
3. 19sobriete.py
4. sobriete62.java

6. Conclusion

En conclusion, ce projet nous a offert une exploration approfondie de l'approche algorithmique du problème de compression par RLE. Nous avons développé, évalué et comparé différentes solutions tout au long de cette SAE.

Cette expérience ne nous a pas seulement préparés à relever des défis algorithmiques concrets, mais elle nous a aussi sensibilisés à l'importance cruciale de la lisibilité, de l'efficacité et de la sobriété numérique dans le développement logiciel moderne.