SAE 2.02

Exploration Algorithmique D'un problème

IUT Informatique de Blagnac

2023

1. Introduction

Cette SAE a pour objectif de développer et évaluer des solutions algorithmiques pour le problème de compression de données en utilisant l'algorithme Run-Length Encoding (RLE). Cette SAE se divise en deux phases distinctes : une phase individuelle où chaque participant doit soumettre des solutions dans différentes catégories d'évaluation (Simplicité, Efficacité, Sobriété), et une phase en binôme où les solutions seront comparées et évaluées selon plusieurs critères prédéfinis.

2. Présentation des outils d’évaluation

Pour noter et classer ces différents algorithmes nous allons utiliser plusieurs outils:

- Junit: pour effectuer des tests et vérifier que le résultat est bien le résultat attendu pour les fichiers Java
- PyUnit: pour effectuer des tests et vérifier que le résultat est bien le résultat attendu pour les fichiers Python
- CUnit: pour effectuer des tests et vérifier que le résultat est bien le résultat attendu pour les fichiers C
- Codacy: afin de récupérer des informations sur la qualité du code comme la complexité cyclique, le nombre de lignes commentées et le nombre de lignes de code dupliquées
- IntelliJ: pour lancer nos tests
- Un ordinateur portable fonctionnant sous Linux Ubuntu 22.04 avec lequel nous allons lancer les programmes et regarder leur consommation mémoire et CPU. Nous utiliserons un ordinateur sous Linux plutôt que sous Windows afin d’éviter le plus possible les tâches indésirables en arrière-plan qui pourraient fausser les tests. Bien sûr aucun programme comme Discord, Firefox ou autre ne tournera sur l’ordinateur pendant la phase de test.

Les tests qu'ils soient Junit, PyUnit ou CUnit seront équivalents avec les mêmes opérations demandées

3. Liste des algorithmes

3.1 Algorithmes de Simplicité

nom du fichier	langage	Lien vers le fichier
32simplicite.py	Python	lien
42simplicite.java	Java	lien
57simplicite.java	Java	lien
04simplicite.java	Java	lien
25simplicite.java	Java	lien

3.2 Algorithmes d'efficacité

nom du fichier	langage	Lien vers le fichier
10efficacite.java	Java	lien
31efficacite.java	Java	lien
30efficacite.java	Java	lien
26efficacite.java	Java	lien
51efficacite.c	C	lien

3.3 Algorithmes de sobriété

nom du fichier	langage	Lien vers le fichier
54sobriete.java	Java	lien
51sobriete.java	Java	lien
28sobriete.c	C	lien
12sobriete.java	Java	lien

4. Notation

4.1 Notation Algorithmes de simplicité

nom fichier	32simplicite.py
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 24
- Le code aurait pu être un peu plus aéré (sauts de lignes)

Note: 20/20

nom fichier	42simplicite.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 11

Note: 20/20

nom fichier	57simplicite.java
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 44
- présence de la Javadoc

Note: 5/20

nom fichier	04simplicite.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui

Remarques:

- noms de variables corrects mais à améliorer
- nombre de lignes commentées: 2 (insuffisant)
- Le code aurait pu être un peu plus aéré (sauts de lignes)

Note: 20/20

nom fichier	25simplicite.java
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 41
- Le code aurait pu être un peu plus aéré (sauts de lignes)

Note: 5/20

4.2 Notation Algorithmes d'efficacité

nom fichier	10efficacite.java
Compilation	non
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui
Complexité	null
Durée de passage des tests	null

Remarques:

- noms de variables à revoir
- nombre de lignes commentées: 1 (insuffisant)
- Code qui ne compile pas, ne passe pas les tests fournis, illisible et absolument pas optimisé avec la présence d'un return avant un break

Note: 2.5/20

nom fichier	31efficacite.java

Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	5
Durée de passage des tests	2.3s

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 16
- manque la Javadoc mais code très lisible

Note: 20/20

nom fichier	30efficacite.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	6
Durée de passage des tests	2.16s

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 6 (insuffisant)
- manque la Javadoc mais code très lisible

Note: 20/20

nom fichier	26efficacite.java
Compilation	oui

Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	6
Durée de passage des tests	2.4s

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 5 (insuffisant)
- manque la Javadoc mais code très lisible

Note: 20/20

nom fichier	51efficacite.c
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	3
Durée de passage des tests	0.2s

Remarques:

- noms de variables corrects mais en anglais
- nombre de lignes commentées: 0 (insuffisant)
- uniquement la fonction RLE mais mal nommée (pas de pénalité car les autres fonctions n'étaient pas obligatoire)

Note: 20/20

4.2 Notation Algorithmes de sobriété

nom fichier	54sobriete.java

Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui
Complexité	null
Durée de passage des tests	null
Nombre de caractères RLE	1110

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 9
- Pas de fonction unRLE et fonction RLE récursive mal nommée

Note: 5/20

nom fichier	51sobriete.java
Compilation	oui
Tests fournis	non
Tests supplémentaires	non
Anonymat	oui
Complexité	null
Durée de passage des tests	null
Nombre de caractères RLE	592

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 0
- Renvoie 13w au lieu de 9w3w

Note: 5/20

nom fichier	12sobriete.java
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	7
Durée de passage des tests	2.1s
Nombre de caractères RLE	869

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 0

Note: 20/20

nom fichier	28sobriete.c
Compilation	oui
Tests fournis	oui
Tests supplémentaires	oui
Anonymat	oui
Complexité	6
Durée de passage des tests	0.2s
Nombre de caractères RLE	2277

Remarques:

- noms de variables corrects
- nombre de lignes commentées: 34
- utilisation d'import de librairies de tests inutiles

Note: 20/20

5. Classement des algorithmes

5.1 Algorithmes simples

Pour classer ces algorithmes nous allons prendre en compte la lisibilité du code, la clarté des noms de variables, la qualité des commentaires et la facilité de compréhension, les codes ne passant pas les tests ou qui ne compilent pas seront en fin de classement.

1. 32simplicite.py
2. 42simplicite.java
3. 04simplicite.java
4. 57simplicite.java
5. 25simplicite.java

5.2 Algorithmes efficaces

Pour classer ces algorithmes nous allons prendre en compte la complexité cyclique, la durée de passage des tests et l'optimisation du code, les codes ne passant pas les tests ou qui ne compilent pas seront en fin de classement.

1. 51efficacite.c
2. 31efficacite.java
3. 30efficacite.java
4. 26efficacite.java
5. 10efficacite.java

5.3 Algorithmes sobres

Pour classer ces algorithmes nous allons prendre en compte la consommation mémoire, l'utilisation des ressources CPU et la performance globale en termes de sobriété., les codes ne passant pas les tests ou qui ne compilent pas seront en fin de classement. Les évaluations de puissance CPU et mémoire ont été effectuées en lançant les programmes 1 par 1 sur le PC de test, en observant le graphique de consommation on peut en déduire un classement mais pas de données précises.

1. 28sobriete.c

2. 12sobriete.java
3. 54sobriete.java
4. 51sobriete.java

6. Conclusion

En conclusion, ce projet nous a permis d'explorer en profondeur l'approche algorithmique du problème de compression par RLE à travers en développant, évaluant et de en comparant des solutions.

Cette SAE nous a non préparés à relever des défis algorithmiques concrets, mais elle nous a également sensibilisés à l'importance de la lisibilité, de l'efficacité et de la sobriété numérique dans le développement logiciel moderne.