2024

S2.02-Exploration algo.

DA CHAO ROMAIN

iut Midi-Pyrénées BLAGNAC

Sommaire

Introduction

Dans cette SAE 2.02, nous allons entreprendre une évaluation comparative de divers algorithmes en nous concentrant sur trois critères principaux : la simplicité, l'efficacité et la

sobriété numérique. Les algorithmes que nous allons analyser sont implémentés en Python, Java et C, et utilisent principalement des techniques de compression comme le Run-Length Encoding (RLE). Pour mener à bien cette évaluation, nous utiliserons des outils comme Codacy pour mesurer la complexité cyclomatique et Intellij pour déterminer les temps d'exécution. Nous observerons également la consommation des ressources CPU et RAM à l'aide du gestionnaire des tâches Windows. L'objectif est de classer ces algorithmes selon leurs performances et de tirer des conclusions sur les meilleures pratiques de développement.

Liste des algorithmes à évaluer

Lien du dépôt : dépôt SAE 2.02

1. simplicite

Programmes	**Language**
35simplicite.py	Python
40simplicite.py	Python
61simplicite.java	Java
07simplicite.java	Java
47simplicite.java	Java

2. efficacité

01efficacite.c	C
24efficacite.java	Java
46efficacite.java	Java
61efficacite.java	Java
68efficacite.java	Java

3. sobriété

06sobriete.java	Java
15sobriete.java	Java
17sobriete.c	C
50sobriete.java	Java

Evaluation des algorithmes

Explication de la méthode et des outils d'évaluation :

- Utilisation de Codacy (pour la complexité cyclomatique)
- Utilisation d'Intellij (pour mesurer le temps d'exécution)
- RLE récursif comportant 60 itérations pour évaluer le temps d'exécution avec plus de précision.
- Gestionnaire des tâches windows (analyse CPU, RAM) sur une machine qui n'a quasiment aucun programme tournant en fond

Simplicité

L'évaluation de la simplicité des algorithmes se base sur leur capacité à compiler correctement, respecter l'anonymat, et suivre les consignes spécifiques comme l'utilisation des méthodes de java.util. Il est crucial que les algorithmes fonctionnent comme prévu, passent tous les tests, et soient bien commentés pour être facilement compréhensibles et maintenables. La complexité cyclomatique est également mesurée pour évaluer la simplicité du flux logique.

35simplicite.py

L'algorithme compile ?	Oui
L'anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L'algorithme fonctionne-t-il ?	Oui
L'algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	4

Compréhension des commentaires	Aucun commentaire
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. De plus, il est bien optimisé avec un indice de complexité cyclomatique de 4. Cependant, bien qu'il soit fonctionnel et efficace, il **manque de commentaires**. Pour améliorer la simplicité et la lisibilité du code, il serait judicieux d'ajouter des commentaires afin que le code soit plus facilement compréhensible par d'autres développeurs.

40simplicite.py

L'algorithme compile ?	Oui
L'anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L'algorithme fonctionne-t-il ?	Oui
L'algorithme passe-t-il les tests ?	Non (ne passe pas le cas où la chaine de caractères est vide)
Complexité cyclomatique	4
Compréhension des commentaires	Aucun commentaire
Total /20	10/20

L'algorithme compile correctement, fonctionne comme prévu mais ne passe pas tous les tests fournis (**ne passe pas le cas où la chaine de caractères est vide et il manque RLE récursif, unRLE et unRLE récursif**). Il est quand même bien optimisé avec un indice de complexité cyclomatique de 4. Cependant, bien qu'il soit fonctionnel et efficace, il **manque de commentaires**. Pour améliorer la simplicité et la lisibilité du code, il serait judicieux d'ajouter des commentaires afin que le code soit plus facilement compréhensible par d'autres développeurs.

07simplicite.java

L'algorithme compile ?	Oui
L'anonymat est il respecté ?	Oui

La consigne des méthodes java.util est-elle respectée ?	Oui
L'algorithme fonctionne-t-il ?	Oui
L'algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	9
Compréhension des commentaires	19/20 (clarté, cohérence et concision)
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **pourrait être mieux optimisé** quand on voit que sa complexité est de 9. L'algorithme est très bien commenté et lisible, c'est ce qu'on attend principalement d'un algorithme de simplicité.

61simplicite.java

L'algorithme compile ?	Oui
L'anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L'algorithme fonctionne-t-il ?	Non
L'algorithme passe-t-il les tests ?	Non (ne passe pas RLE et unRLE)
Complexité cyclomatique	7
Compréhension des commentaires	18/20 (clarté, cohérence à améliorer et concision)
Total /20	5/20

L'algorithme compile correctement, mais ne fonctionne pas bien en ne passant pas tout les tests fournis. Il **pourrait être mieux optimisé** quand on voit que sa complexité est de 7. L'algorithme est tout de même bien commenté et lisible.

47simplicite.java

L'algorithme compile ?	Oui

L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Oui
L’algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	9
Compréhension des commentaires	19/20 (clarté, cohérence et concision)
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **pourrait être mieux optimisé** quand on voit que sa complexité est de 9. L’algorithme est tout de même très bien commenté et lisible.

Efficacité

Pour l'efficacité, les algorithmes doivent compiler sans erreur, respecter l'anonymat, et suivre les consignes. Ils doivent fonctionner correctement et passer tous les tests. La complexité cyclomatique est mesurée pour comprendre la complexité logique, et le temps d'exécution est évalué pour déterminer la performance en termes de vitesse. Ces critères combinés permettent de classer les algorithmes selon leur efficacité globale.

01efficacite.c

L’algorithme compile ?	Oui
L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Oui
L’algorithme passe-t-il les tests ?	Non (manque RLE récursif et unRLE) 15/20 (car les tests sont tout de même bons)
Complexité cyclomatique	2
Temps d'exécution	0.55 s
Total /20	15/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. De plus, il est bien optimisé avec un indice de complexité cyclomatique de 2.
Il lui **manque juste le RLE récursif et unRLE**. Son temps d'exécution est très rapide.

46efficacite.java

L'algorithme compile ?	Oui
L'anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L'algorithme fonctionne-t-il ?	Oui
L'algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	9
Temps d'exécution	1,2 s
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **pourrait être mieux optimisé** quand on voit que sa complexité est de 9. Son temps d'exécution reste tout de même excellent.

61efficacite.java

L'algorithme compile ?	Oui
L'anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L'algorithme fonctionne-t-il ?	Oui
L'algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	6
Temps d'exécution	1,35 s
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **est bien optimisé sans trop l'être** avec sa complexité de 6. Son temps d'exécution est bon. Il est également bien commenté.

24efficacite.java

L’algorithme compile ?	Oui
L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Oui
L’algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	5
Temps d'exécution	indéfini (+ de 7 min)
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **est bien optimisé sans trop l’être** avec sa complexité de 5. Son **temps d’exécution est catastrophique** dans notre cas. Il est également bien commenté.

68efficacite.java

L’algorithme compile ?	Oui
L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Oui
L’algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	7
Temps d'exécution	1,65 s
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **manque un peu d’optimisation** avec sa complexité de 7. Son temps d’exécution est bon dans notre cas. Il est également bien commenté.

Sobriété

La sobriété est évaluée par la consommation de mémoire et l'utilisation des ressources CPU et RAM. Les algorithmes sont testés individuellement, et leur consommation est observée pour établir un classement. Les algorithmes doivent compiler, passer les tests, et utiliser les ressources de manière efficace. Ceux qui échouent à compiler ou à passer les tests sont classés en dernier.

50sobriete.java

L’algorithme compile ?	Oui
L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Oui
L’algorithme passe-t-il les tests ?	Oui
Complexité cyclomatique	5
Temps d'exécution	indéfini (+ de 7 min)
Consommation	Modérée
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **est plutôt bien optimisé** avec sa complexité de 5. Son **temps d’exécution est catastrophique** dans notre cas, les ressources de la machine sont donc sollicités en continue, **ce qui n’est pas très sobre**. Il est également bien commenté.

15sobriete.java

L’algorithme compile ?	Oui
L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Non
L’algorithme passe-t-il les tests ?	Non

Complexité cyclomatique	1
Temps d'exécution	Non défini car infonctionnel
Consommation	inconnue car non fonctionnel
Total /20	5/20

L'algorithme compile correctement, mais ne fonctionne pas.

17sobriete.c

L’algorithme compile ?	Non (#include <stdlib.h> manquante)
L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Non
L’algorithme passe-t-il les tests ?	Non
Complexité cyclomatique	8
Temps d'exécution	Non défini car infonctionnel
Consommation	inconnue car non fonctionnel
Total /20	5/2 = 2,5/20

L'algorithme ne compile pas en raison d’un oubli d’import. Il **pourrait être mieux optimisé** avec sa complexité de 8. Il est cependant bien commenté.

06sobriete.java

L’algorithme compile ?	Oui
L’anonymat est il respecté ?	Oui
La consigne des méthodes java.util est-elle respectée ?	Oui
L’algorithme fonctionne-t-il ?	Oui
L’algorithme passe-t-il les tests ?	Oui

Complexité cyclomatique	7
Temps d'exécution	1,5 s
Consommation	Faible
Total /20	20/20

L'algorithme compile correctement, fonctionne comme prévu et passe tous les tests fournis. Il **pourrait être mieux optimisé** avec sa complexité de 7. Son **temps d'exécution est bon**.

Classement des algorithmes

En ayant pris en compte les divers critères d'évaluation et avis précédents, voici le classement des algorithmes par catégories (le 1er étant le meilleur) :

Simplicité

1. 07simplicite.java
2. 47simplicite.java
3. 35simplicite.py
4. 40simplicite.py
5. 61simplicite.java

Efficacité

1. 01efficacite.c
2. 68efficacite.java
3. 46efficacite.java
4. 61efficacite.java
5. 24efficacite.java

Sobriété

1. 06sobriete.c
2. 50sobriete.java
3. 15sobriete.java
4. 17sobriete.c

Conclusion

Après avoir évalué les algorithmes, nous avons identifié ceux qui excellent dans les domaines de la simplicité, de l'efficacité et de la sobriété numérique. En termes de simplicité, certains algorithmes ont été remarquables par leur clarté et leur facilité de maintenance. Sur le plan de l’efficacité, nous avons noté des algorithmes performants par leur rapidité d'exécution et leur faible complexité cyclomatique. Concernant la sobriété numérique, certains se sont distingués par une consommation minimale des ressources mémoire et CPU, illustrant les avantages des solutions optimisées pour la gestion des ressources. Ces évaluations nous offrent une compréhension approfondie des forces et des faiblesses des différentes approches, orientant ainsi les choix technologiques et les techniques de programmation pour des projets futurs.