DÉPARTEMENT INFORMATIQUE - BUT INFORMATIQUE
1ER ANNÉE 2023-2024
IUT Blagnac - Université Toulouse Jean Jaurès

S2.02
Exploration algorithmique d'un problème

python™

Fait par : Clavel Simon

SOMMAIRE

Introduction :

Cette Situation d'Apprentissage et d'Évaluation (SAÉ) se concentre sur l'approche algorithmique dans le développement logiciel. Son objectif est d'approfondir la réflexion et les compétences nécessaires pour résoudre efficacement des problèmes complexes à travers la programmation. Les participants auront l'opportunité de participer à un concours de codage, d'évaluer des codes différents, de comparer des algorithmes selon des critères précis, et de justifier objectivement leurs choix et classements.

I - Outils d'évaluation

- codacy (pour utiliser cette outils j'ai du copier tout le projet dans un nouveau projet sur github afin d'avoir des resultats).
- Utilisation d'une classe qui calcule le temps des fonctions (donnée avec les tests sur le dépôt Git).

II - les tests

Rajout de 5 tests (donnés avec les tests sur le dépôt Git) :

- testRLESupplementaire()
- testRLERecursifSupplementaire()
- testUnRLESupplementaire()
- testUnRLERecursifSupplementaire()
- testJavaUtil()

Utilisation d'une classe AlgotestEfficacite afin de savoir quels sont les algorithmes les plus efficaces en calculant leur temps d'exécution.

Pour le python et le c j'ai testé directement dans la classe.

III - notes

Simplicité :

Fichier :	Notes :	Explication :
31simplicite.java	20	Tout marche, passe tous les tests.
34simplicite.java	20	Tout marche, passe tous les tests.
40simplicite.java	20	Tout marche, passe tous les tests.
51simplicite.py	5	Qu'une seule fonction et avec le mauvais nom qui de plus ne fonctionne pas (problème avec les répétitions).
54simplicite.java	10	Changement de nom de RLE avec itération et seulement deux fonctions (il manque les fonctions unRLE). Il n'y a que les tests RLE sans itération marche.

efficacité :

Fichier :	Notes :	Explication :
20efficacite.java	20	Tout marche, passe tous les tests.
65efficacite.java	10	Les tests RLE et RLE avec iterations ne marchent pas. Le test ne passe pas avec une chaîne vide "".
53efficacite.java	14	Manque la fonction unRLE avec les itérations, ne passe donc pas les tests unRLE avec itérations.
55efficacite.java	20	Tout marche, passe tous les tests.
63efficacite.java	20	Tout marche, passe tous les tests.

sobriété :

Fichier :	Notes :	Explication :
12sobriete.java	20	Tout marche, passe tous les tests.
61sobriete.c	20	Tout marche.
37sobriete.c	20	Tout marche.
50sobriete.java	18	Ne passe pas les tests supplémentaires RLE et RLE avec itérations.

IV - classement des algorithmes

Simplicité :

resultat des test d'efficacité :

résultat fait a partir de l'outil codacy et aussi à partir de ma facilité personnel à pouvoir comprendre le code.

classement :

1 → 31.java
2 → 34.java
3 → 54.java
4 → 40.java
5 → 51.py

Ce classement à été fait en fonction de la lisibilité des algorithmes.

efficacité :

resultat des test d'efficacité :

20efficacite.java

Testing RLE(String in) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 5351900 nanosecondes
Testing RLE(String in, int iteration) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 60153000 nanosecondes
Testing unRLE(String in) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 17500 nanosecondes
Testing unRLE(String in, int iteration) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 28514700 nanosecondes

Somme : 5351900 +60153000 + 17500 + 28514700 = 94037100

65efficacite.java

Les tests ne passent pas de base donc bas de classement, pas possible de noter.

53efficacite.java

Les tests ne passent pas de base donc bas de classement, pas possible de noter.

55efficacite.java

Testing RLE(String in) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 1373900 nanosecondes
Testing RLE(String in, int iteration) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 30563300 nanosecondes
Testing unRLE(String in) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 167000 nanosecondes
Testing unRLE(String in, int iteration) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 36917600 nanosecondes

Somme : 1373900 + 30563300 + 167000 + 36917600 = 69021800

S2.02 - Exploration algorithmique d'un problème
2023 / 2024

63efficacite.java

Testing RLE(String in) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 584999 nanosecondes
Testing RLE(String in, int iteration) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 33191900 nanosecondes
Testing unRLE(String in) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 14999 nanosecondes
Testing unRLE(String in, int iteration) method:
Chaîne de caractères: ABBBCCCCDDDDDDEEEEEEEE, Temps d'exécution: 17968000 nanosecondes

Somme : 584999 + 33191900 + 14999 + 17968000 = 51759898

classement :

1 → 63efficacite.java
2 → 55efficacite.java
3 → 20efficacite.java
4 → 65efficacite.java
4 → 53efficacite.java

sobriété :

resultat des test d'efficacité :

résultat fait a partir de l'outil codacy

classement :

1 → 61sobriete.c
2 → 37sobriete.c
3 → 12sobriete.java
4 → 50sobriete.java

Conclusion :

En conclusion, cette SAÉ m'a offert une occasion précieuse d'approfondir mes compétences en algorithmique dans le cadre du développement logiciel. Participer au concours de codage, évaluer divers codes, comparer des algorithmes et justifier mes choix de manière objective ont renforcé ma capacité à aborder des défis techniques avec méthode et assurance. Cette expérience enrichissante m'a mieux préparé(e) à appliquer mes connaissances dans des environnements professionnels où la précision algorithmique et la prise de décision informée sont cruciales.