Guibert
Rémy

SAÉ Exploration algorithmique

Analyses et comparaisons

Synthèse :

Notes	Algo	Lisibilité	Qualité	Temps d'exécution	Complexité	Sobriété	Passe les tests
1	simplicite_35.java	+++++	++++	N/A	N/A	N/A	[ ]
1	simplicite_56.java	+++	++++	N/A	N/A	N/A	[ ]
1	simplicite_73.java	++++	++++	N/A	N/A	N/A	[ ]
5	simplicite_84.java	+++++	+++++	N/A	N/A	N/A	[x]
5	efficacite_143.java	N/A	N/A	++++	++++	N/A	[x]
0	efficacite_3.java	N/A	N/A	-	-	N/A	[ ]
1	efficacite_90.java	N/A	N/A	+++++	++++	N/A	[ ]
4	efficacite_13.py	N/A	N/A	+++	++++	N/A	[x]
1	sobriete_150.java	N/A	N/A	-	++++	+++	[ ]
4	sobriete_102.py	N/A	N/A	++++	++++	++++	[x]
5	sobriete_136.c	N/A	N/A	+++++	++++	+++++	[x]

Simplicité

simplicite_35.java :

Lisibilité du code :
On comprend ce que fait l'algo, il créer un String et concatène les caractères différents des espaces ou les espace consécutifs, néanmoins on remarque que les espaces en fin de String sont ignorés, donc il ne fonctionne pas dans le cas où il y a un ou plusieurs espaces en fin de chaîne. La fonction est commentée (Javadoc).

Qualité du code :
Une erreur mineur sur Codacy.

Passage des tests :
Il ne passe pas tous les tests.

simplicite_56.java :

Lisibilité du code :
On comprend ce que fait l'algo, il créer un String et concatène les caractères différents des espaces ou les espace consécutifs, néanmoins on remarque que les espaces en fin de String sont ignorés, donc il ne fonctionne pas dans le cas où il y a un ou plusieurs espaces en fin de chaîne. La fonction est commentée (instruction par instruction).

Qualité du code :
Une erreur mineur sur Codacy.

Passage des tests :
Il ne passe pas tous les tests.

simplicite_73.java :

Lisibilité du code :
On comprend ce que fait l'algo, il créer un String et concatène les caractères différents des espaces ou les espace consécutifs, néanmoins on remarque que les espaces en début de String causeront une exception. La fonction est commentée (instruction par instruction).

Qualité du code :
Une erreur mineur sur Codacy.

Passage des tests :
Il ne passe pas tous les tests.

simplicite_84.java :

Lisibilité du code :
On comprend ce que fait l'algo, il créer un String et concatène les caractères différents des espaces ou les espace consécutifs, il n'y a visiblement pas de problèmes. La fonction n'est pas commentée.

Qualité du code :
Aucune erreur sur Codacy.

Passage des tests :
Il n'y a pas de problèmes lors des tests.

Efficacité

efficacite_143.java :

Efficacité :
L'algorithme tourne autour d'une boucle for de la taille de la chaîne en paramètre, à mon avis la complexité est O(n) avec n la taille de la chaîne.

Temps d'exécution :
Sur 15 exécution d'un fichier texte de 100 000 caractères, cela a pris en moyenne 5 ms.

Passage des tests :
L'algorithme a passé tous les tests.

efficacite_3.java :

Algorithme disqualifié, pour deux raisons, la première c'est qu'au lieu de renvoyer un objet String il l'affiche, la deuxième c'est qu'il efface les espaces blancs en début et fin de la chaîne donnée en paramètre, ce qui ne devrait pas être fait, puisque si il y a plusieurs espaces ils ne devraient pas être retirés, finalement l'algorithme est basé sur le fait que ni en début ni en fin il n'y a d'espace donc il est erroné.

efficacite_90.java :

Efficacité :
L'algorithme utilise la méthode replaceAll de Java, qui est d'un complexité linéaire donc O(n) avec n la taille de la chaîne.

Temps d'exécution :
Sur 15 exécution d'un fichier texte de 100 000 caractères, cela a pris en moyenne 3 ms.

Passage des tests :
L'algorithme n'a pas passé tous les tests.

efficacite_13.py :

Efficacité :
L'algorithme tourne autour d'une boucle for de la taille de la chaîne en paramètre, à mon avis la complexité est O(n) avec n la taille de la chaîne.

Temps d'exécution :
Sur 15 exécution d'un fichier texte de 100 000 caractères, cela a pris en moyenne 22 ms.

Passage des tests :
L'algorithme a passé tous les tests.

Sobriété

sobriete_150.java :

Complexité :
L'algorithme tourne autour d'une boucle for de la taille de la chaîne en paramètre, l'oppération significative est la concaténaton de String, donc la complexité est O(n) avec n la taille de la chaîne.

Temps d'exécution :
Sur 15 exécution d'un fichier texte de 100 000 caractères, cela a pris en moyenne 2520 ms.

Sobriété :

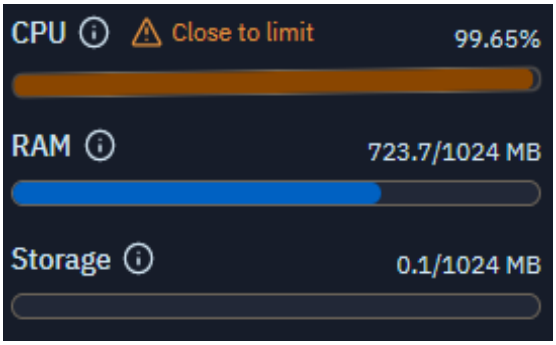

Figure 1: Utilisation CPU/RAM lors des 15 exécutions

Passage des tests :
L'algorithme n'a même pas passé le premier test !

sobriete_102.py :

Efficacité :
L'algorithme utilise une méthode équivalente à replaceAll de Java, qui est d'un complexité linéaire donc O(n) avec n la taille de la chaîne.

Temps d'exécution :
Sur 15 exécution d'un fichier texte de 100 000 caractères, cela a pris en moyenne 3 ms.

Sobriété :

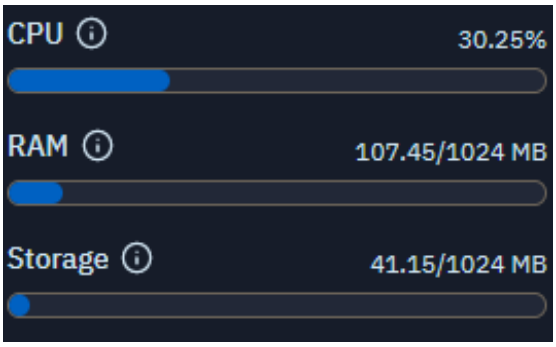

Figure 2: Utilisation CPU/RAM lors des 15 exécutions

Passage des tests :
L'algorithme a passé tous les tests.

sobriete_136.c :

Complexité :
L'algorithme tourne autour d'une boucle for de la taille du tableau de caratères en paramètre, donc la complexité est O(n) avec n la taille du tableau.

Temps d'exécution :
Sur 15 exécution d'un fichier texte de 100 000 caractères, cela s'exécute très rapidement, je n'ai pas de valeur mais c'est de l'ordre des millisecondes.

Sobriété :

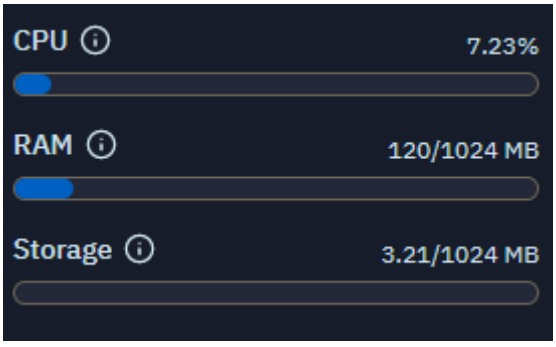

Figure 3: Utilisation CPU/RAM lors des 15 exécutions

Passage des tests :
L'algorithme a passé tous les tests.