Compte rendu SAE Algo

18 JUIN

Créé par : Enzo Fournet

Table des matières

Introduction

Dans ce compte rendu nous allons expliquer les méthodes d'évaluation et évaluer des algorithmes.

1 Préambule

D'abords nous avons choisi de ne pas évaluer les algorithmes qui ne passe pas les tests de bases puisqu'il semble insensé de comparer des algorithmes alors qu'ils ne sont pas capables de fournir un résultat pertinent.
Cependant la lisibilité en ce qui concerne la simplicité, nous évalueront la lisibilité du code et la qualité de la nomenclature des variables et/ou méthodes utilisé.

2 Méthode d'évaluation choisi

2.1 Efficacité

2.1.1 Qu'est-ce que c'est ?

L'efficacité d'un programme se réfère à sa capacité d'exécuter des tâches rapidement. Un programme est considéré comme efficace s'il peut accomplir son travail avec le moins de temps de traitement possible.

2.1.2 Comment allons-nous mesurer ce paramètre ?

Pour l'efficacité nous avons décidé de seulement mesurer la rapidité d'exécution du programme en utilisant.
Avec la Class TestEffi et sa méthode main. Visible ici.

2.2 Simplicité

2.2.1 Qu'est-ce que c'est ?

La simplicité d'un programme se réfère à la clarté et à la lisibilité de son code. Un programme simple est facile à comprendre, à maintenir et à modifier. Il évite la complexité inutile, utilise des noms de variables descriptifs, suit les conventions de codage standard et est bien documenté.

2.2.2 Comment allons-nous mesurer ce paramètre ?

Pour la simplicité nous allons vérifier que le code est lisible et commenté de façon suffisante. Puis nous vérifieront que la nomenclature des variables et/ou méthodes utilisé est pertinente est logique.

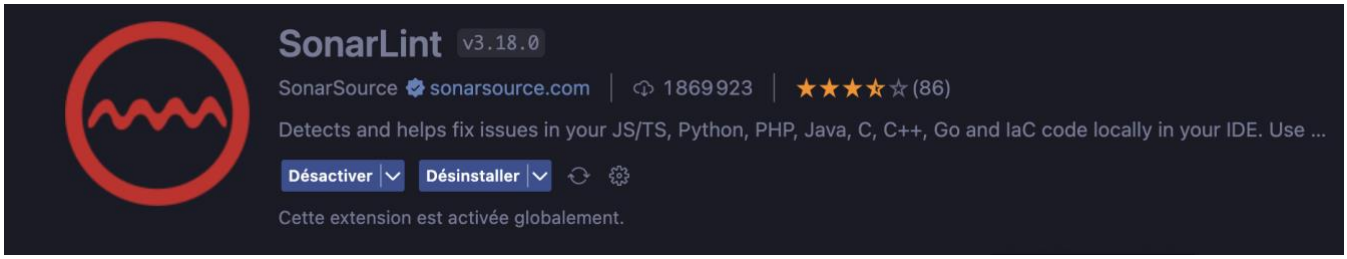
2.3 Sobriété

2.3.1 Qu'est-ce que c'est ?

La sobriété d'un programme fait référence à son minimalisme, c'est-à-dire qu'il n'inclut que les fonctionnalités et les composants strictement nécessaires pour accomplir sa tâche. Un programme sobre évite l'encombrement et la complexité inutiles, rendant le code plus propre, plus facile à comprendre et à maintenir.

2.3.2 Comment allons-nous mesurer ce paramètre ?

En ce qui concerne la sobriété, nous regarderont et étudierons le code pour vérifier qu'il ne contient rien d'inutile qui pourrai nuire à sa sobriété.
Ceci à l'aide de SonarLint une extension VSCODE.



2.4 Évaluation des algorithmes

2.4.1 Efficacité Meilleur

2.4.1.1 20 - 2ème

Je n'ai pas la capacité de réalisé mes tests d'efficacité en C.

2.4.1.2 53 – 1er

Le programme a mis 71 millisecondes à s'exécuter avec notre méthode de test.

```
Temps d'exécution : 71968 micro secondes
Temps d'exécution : 71.968334 millisecondes
```

Nous n'avons pour le moment pas de point de comparaison mais le programme semble très lent.

2.4.2 Efficacité Pire

2.4.2.1 16 – 2ème

Ce programme n'est pas évaluable puisqu'il ne passe pas les tests de base il est donc impossible. Il sera donc dernier de sa catégorie.

Cependant il tri de la mauvaise façon en 2 millisecondes ce qui est très rapide comparé aux autres algos.

```
st, error, eaque, et, explicabo, enim, eos]
Temps d'exécution : 2110 micro secondes
Temps d'exécution : 2.110959 millisecondes
```

2.4.2.2 66 – 1er

Le programme met 10 millisecondes à s'exécuter et à trier correctement les mots.

Cependant il est assez illogique de se rendre compte qu'un algorithme de la catégorie pire est plus rapide qu'un algo de la catégorie meilleur.

Il est donc premier étant le seul fonctionnel.

```
ntore, veritatis, quasi, vitae, Nemo, ipsam, quia, vo
Temps d'exécution : 10497 micro secondes
Temps d'exécution : 10.497833 millisecondes
```

2.4.3 Simplicité meilleure

2.4.3.1 19 – 1er

Cet algorithme, utilise une nomenclature claire accompagner d'une JavaDoc et de quelque commentaire utile.
Chaque méthode a son propre rôle défini clairement.
Les méthodes utilisées sont simples et compréhensibles.
Il est simplement dommage que le nom des méthodes soit en anglais et les commentaires en Français il semble nécessaire de faire un choix clair de langue à utiliser.
Il sera donc premier de sa catégorie

2.4.3.2 22 – 3ème

Celui-ci ne fonctionne pas il ne sera donc évidemment dernier de cette catégorie.
Au-delà de ça le code est plus complexe à comprendre la nomenclature moins claire.
Pas de JavaDoc.

2.4.3.3 29 – 2ème

Cet algo fonctionne mais est plus complexe que le 19 il ne présente pas de javadoc et une nomenclature en français et anglais, il est donc 2ème.

2.4.4 Simplicité pire

2.4.4.1 27 - 1er

Ce code ne contient aucun commentaire, il est illisible indenter n'importe comment.
Il est impossible de comprendre facilement ce qu'il fait.
Il est donc exæquo avec le 35.

2.4.4.2 35 - 1er

Ce code ne contient aucun commentaire, il est écrit sur une seule ligne.
Il est impossible de comprendre facilement ce qu'il fait.
Il est donc exæquo avec le 27.

2.4.5 Sobriété meilleure

2.4.5.1 3 – 1er

Ce code n'importe que les librairies importantes et ne semble pas faire d'action inutile.
Mais il ne passe pas les tests.
Il est donc exæquo avec le 56.

2.4.5.2 56 – 1er

Ce code n'importe que les librairies importantes et ne semble pas faire d'action inutile. Mais il ne passe pas les tests.
Il est donc exæquo avec le 3.

2.4.6 Sobriété pire

2.4.6.1 17 – 1er

Cet algo passe les tests mais import que le nécessaire. Sa sobriété semble bonne.
Il ne fait pas non plus d'action inutile. Il est donc premier à cause de la qualité du 48.

2.4.6.2 48 – 2ème

Cet algo a des gros soucis de nomenclature, d'import qui ne sont même pas présent, on ne peut donc même pas l'exécuter de base. Il n'a rien de moins que le 17 il ne fonctionne tout simplement pas il est donc deuxième.