

Groupe 4b

KHALIL Ahmad

AL-MASRI Marwan

SAE 2.02 Exploration algorithmique d'un problème

Sommaire :

Introduction	2
Présentation des outils d'évaluation	2
Choix des critères d'évaluation	4
Classement des algorithmes Marwan	6
Classement des algorithmes Ahmad	9
Notation des algorithmes - Marwan	12
Notation des algorithmes - Ahmad	14


Introduction

La présente Situation d'Apprentissage et d'Évaluation (SAÉ) vise à approfondir la réflexion sur l'approche algorithmique dans le cadre du développement informatique. Les étudiants doivent soumettre un algorithme permettant de résoudre un problème simple mais offrant plusieurs solutions différentes. Le problème en question porte sur le classement des mots d'un texte selon un ordre spécifique donné. Les participants sont tenus d'écrire une fonction en Java ou en C pour effectuer ce classement, en respectant les contraintes précisées. Dans ce compte-rendu, nous examinerons les différentes solutions des participants dans le but d'établir un classement des algorithmes dans les différentes catégories de la SAÉ. Pour établir ce classement, nous utiliserons des outils d'évaluation appropriés que nous détaillerons dans la première partie.

Présentation des outils d'évaluation

Codacy

Codacy est un outil d'évaluation de code largement utilisé dans l'industrie du développement logiciel. Il offre une analyse avancée qui permet d'identifier les erreurs, les vulnérabilités et les mauvaises pratiques de programmation dans votre code source. Codacy prend en charge de nombreux langages de programmation populaires tels que Java, Python, JavaScript, C# et bien d'autres. Il propose une intégration fluide avec des plateformes de gestion de code source telles que GitHub, Bitbucket et GitLab, ce qui permet une évaluation continue du code tout au long du processus de développement. Codacy fournit également des informations détaillées sur les problèmes détectés, y compris des suggestions de correction, ce qui facilite la correction des erreurs et l'amélioration de la qualité du code. Nous utiliserons cet outil pour estimer la complexité des algorithmes.

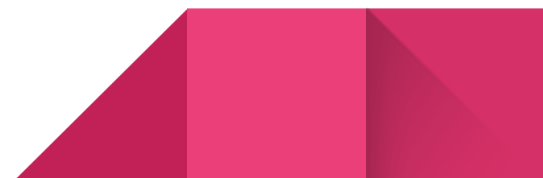


JArchitect

JArchitect est un outil d'évaluation de code puissant et polyvalent, largement utilisé dans l'industrie du développement logiciel. Il offre des fonctionnalités avancées pour l'analyse, la mesure et la visualisation de la qualité du code source. JArchitect permet aux développeurs de comprendre la structure de leur code, d'identifier les zones problématiques, de détecter les violations des bonnes pratiques et de prendre des décisions éclairées pour améliorer la qualité et la maintenabilité de leur code. Avec JArchitect, les développeurs peuvent gagner du temps dans l'analyse de leur code, identifier les problèmes potentiels et prendre des mesures proactives pour maintenir un code de qualité supérieure. Nous utiliserons cet outil pour mesurer la qualité du code des algorithmes.

Joular

Joular ou Joularjx est un outil d'évaluation de code qui surveille la consommation d'énergie de chaque méthode lors de l'exécution d'un programme. Il utilise un agent Java intégré et cela lui permet de fournir des données précises de puissance. Il offre en temps réel la consommation d'énergie de chaque méthode surveillée et l'énergie totale consommée par chaque méthode à la sortie du programme. Cet outil permet aux développeurs d'optimiser l'efficacité énergétique de leur code, réduisant la consommation d'énergie, ici nous l'utiliserons donc pour mesurer la consommation de chaque algorithme.



Choix des critères d'évaluation

Lisibilité du code :

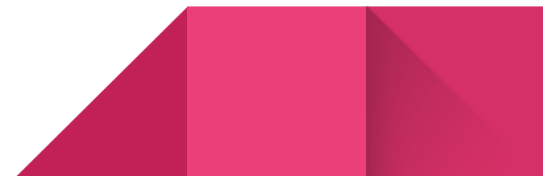
Ce critère est essentiel car il concerne la compréhension et la maintenabilité du code. Un code bien structuré, avec des noms de variables et de fonctions significatifs, facilite la collaboration entre développeurs, permet une détection plus rapide des erreurs et facilite les modifications ultérieures. La lisibilité du code contribue à la qualité globale du logiciel, en réduisant les risques d'erreurs et en améliorant l'efficacité du développement.

Qualité du code :

La qualité du code est un critère objectif qui peut être évalué à l'aide d'outils tels que Codacy. Ces outils analysent le code source pour détecter des problèmes potentiels tels que les erreurs de syntaxe, les violations des bonnes pratiques de programmation, les duplications de code, etc. Évaluer la qualité du code est important car un code de qualité réduit les bugs, facilite la maintenance, améliore la performance et la sécurité du logiciel.

Simplicité / Efficacité / Sobriété (Selon catégorie) :

La simplicité de l'algorithme est un critère important car elle permet de le rendre plus facile à comprendre, à implémenter et à maintenir. Un algorithme simple est souvent préférable car il réduit les risques d'erreurs, facilite la collaboration entre développeurs et permet une évolution plus fluide.



L'efficacité de l'algorithme est un critère clé pour évaluer sa capacité à traiter les données rapidement et de manière optimale. Un algorithme efficace est celui qui utilise de manière efficiente les ressources disponibles, minimise les opérations inutiles et offre des performances élevées même avec des ensembles de données volumineux.

La catégorie sobriété implique d'analyser les performances de l'algorithme en termes de temps d'exécution et d'utilisation des ressources, notamment en fonction de la taille des données en entrée.

Consommation électrique :

Il est essentiel de mesurer l'impact de l'algorithme sur le temps de calcul et la consommation en ressources en électricité, tout cela dans le but de garantir des performances optimales du logiciel. Nous jugerons ce critère en mesurant ses consommations des ressources via l'outil Joular.

Temps d'exécution :

Le temps d'exécution est un critère essentiel pour évaluer les performances d'un algorithme. Il mesure la durée nécessaire à l'algorithme pour traiter les données en entrée et produire les résultats attendus. Un temps d'exécution court est généralement préférable, car il permet d'obtenir des résultats plus rapidement. Nous effectuerons cette mesure dans les programmes de test.



Classement des algorithmes Marwan

Catégorie simplicité meilleur :

Identifiant	Lisibilité du code /10	Qualité du code /10	Note de l'algorithme /20	Classement
1	7.5	9	16.5	Hors concours
36	6.5	6	12.5	1
16	8	8	16	Hors concours

Catégorie simplicité pire :

Identifiant	Lisibilité du code /10	Qualité du code /10	Note de l'algorithme /20	Classement
3	9.5	3.5	13	1
48	8	3.5	11.5	Hors concours

Catégorie efficacité meilleur :

Identifiant	Qualité du code /10	Temps d'exécution sur un texte court	Temps d'exécution sur un texte long	Note de l'algorithme /20	Classement
10	6	1,60 ms	8,6 ms	18	1
43	5	1,7 ms	15,3 ms	15	2

Catégorie efficacité pire :

Identifiant	Qualité du code /10	Temps d'exécution sur un texte court	Temps d'exécution sur un texte long	Note de l'algorithme /20	Classement
22	5	10,9 ms	> à 3 min	16	1
9	4	2,3 ms	7,6 ms	10	2

Catégorie sobriété meilleur :

Identifiant	Qualité du code /10	Consommation électrique	Note de l'algorithme /20	Classement
41	-	-	-	Hors concours
68	5	-	-	Hors concours

Catégorie sobriété pire :

Identifiant	Qualité du code /10	Consommation électrique	Note de l'algorithme /20	Classement
23	8	7,46 Joules Program consumed 7,46 joules	15	2
64	8.5	4,35 Joules Program consumed 4,35 joules	18	1

Classement des algorithmes Ahmad

Catégorie simplicité meilleur :

Identifiant	Lisibilité du code /10	Qualité du code /10	Note de l'algorithme /20	Classement
63	9	9.5	18,5	1
58	6	9	15	2
7	6	7	13	3

Catégorie simplicité pire :

Identifiant	Lisibilité du code /10	Qualité du code /10	Note de l'algorithme /20	Classement
48	5	6.5	11,5	2
38	9	10	19	1

Catégorie efficacité meilleur :

Identifiant	Qualité du code /10	Temps d'exécution sur un texte court	Temps d'exécution sur un texte long	Note de l'algorithme /20	Classement
32	9	15,5 ms	18,5 ms	15	1
65	7	—	—	—	Hors concours

Catégorie efficacité pire :

Identifiant	Qualité du code /10	Temps d'exécution sur un texte court	Temps d'exécution sur un texte long	Note de l'algorithme /20	Classement
22	6	23,5 ms	> à 3 min	15	1
40	4	—	—	—	Hors concours

Catégorie sobriété meilleur :

Identifiant	Qualité du code /10	Consommation électrique	Note de l'algorithme /20	Classement
5	6	5,46 Joules Program consumed 5,46 joules	16	2
8	7	4,67 Joules Program consumed 4,67 joules	17	1

Catégorie sobriété pire :

Identifiant	Qualité du code /10	Consommation électrique	Note de l'algorithme /20	Classement
40	10	9,21 Joules Program consumed 9,21 joules	20	1
64	9	6,59 Joules Program consumed 6,59 joules	18	2

Notation des algorithmes - Marwan

Identifiant	Catégorie	Justification de la note	Note finale /20
1	Simplicité-meilleur	Ne compile pas, On obtient l'erreur suivante pour <code>array.sort</code> : The method <code>sort(T[], Comparator<? super T>)</code> in the type <code>Arrays</code> is not applicable for the arguments <code>(String[], ComparateurOrdreFixe)</code>	5
16	Simplicité-meilleur	Ne compile pas, On obtient l'erreur suivante : <code>texte cannot be resolved</code>	5
36	Simplicité-meilleur	Tous les tests fonctionnent.	20
48	Simplicité-pire	Ne compile pas, manque des imports. Mais après avoir rajouté les imports, compile et réussit tous les test sauf celui du test de la virgule.	16
3	Simplicité-pire	Fonctionne et rate deux tests, tous les mots ne sont pas retournés car le seul séparateur est espace, les virgules ne sont donc pas prises en compte.	15
10	Efficacité-meilleur	Fonctionne mais ne passe pas certains tests initiales, tous les mots ne sont pas retournés car le seul séparateur est espace, les virgules ne sont donc pas prises en compte	17
43	Efficacité-meilleur	Fonctionne et réussit tous les tests.	20

9	Efficacité-pire	Fonctionne et réussit tous les tests.	20
22	Efficacité-pire	Fonctionne et réussit tous les tests.	20
41	Sobriété-meilleur	Ne compile pas et donne un code qui n'a aucun rapport avec l'exercice. <pre>void solution(char** result, int tailleResult, char* texte, char* ordre, int tailleOrdre) { result[0] = "JMB"; result[1] = "Coucou"; }</pre>	0
68	Sobriété-meilleur	Ne compile pas, mauvais type.	5
23	Sobriété-pire	Compile mais ne passe aucun test.	10
64	Sobriété-pire	Compile et réussit tous les tests.	20

Notation des algorithmes - Ahmad

Identifiant	Catégorie	Justification de la note	Note finale /20
63	Simplicité-meilleur	L'algorithme passe les tests initiales mais pas les tests supplémentaires, l'algorithme ne trie les mots que par leur première lettre.	18
58	Simplicité-meilleur	L'algorithme passe les tests initiales mais pas les tests supplémentaires, l'algorithme ne trie les mots que par leur première lettre.	18
7	Simplicité-meilleur	Fonctionne mais ne passe pas certains tests initiales, tous les mots ne sont pas retournés car le seul séparateur est espace, les virgules ne sont donc pas prises en compte.	16
38	Simplicité-pire	Excellent PIRE algorithme, on obtient une erreur nous disant que la mémoire est insuffisante. En revanche, tous les mots ne sont pas retournés car le seul séparateur est espace, les virgules ne sont donc pas prises en compte.	17
48	Simplicité-pire	Imports manquants. Tous les tests fonctionnent. L'algorithme passe les tests initiales mais pas les tests supplémentaires, l'algorithme ne trie les mots que par leur première lettre.	16
32	Efficacité-meilleur	L'algorithme passe les tests initiales mais pas les tests supplémentaires, l'algorithme ne trie les mots que par leur première lettre.	18
65	Efficacité-meilleur	Nom de la méthode invalide, type de retour de la méthode principale invalide (void), impossible de tester l'algorithme.	5
22	Efficacité-pire	Tous les tests fonctionnent.	20

40	Efficacité-pire	L'algorithme utilise une classe extérieure qui n'est pas fournie, impossible donc de tester le programme. Le code est cependant plutôt lisible.	5
5	Sobriété-meilleur	Fonctionne mais ne passe pas certains tests initiales, tous les mots ne sont pas retournés car le seul séparateur est espace, les virgules ne sont donc pas prises en compte.	17
8	Sobriété-meilleur	Fonctionne mais ne passe pas certains tests initiales, tous les mots ne sont pas retournés car le seul séparateur est espace, les virgules ne sont donc pas prises en compte.	17
40	Sobriété-pire	L'algorithme passe les tests initiales mais pas les tests supplémentaires, l'algorithme ne trie les mots que par leur première lettre.	18
64	Sobriété-pire	Tous les tests fonctionnent.	20