

SAf 2.02

Sommaire

1. Auteur(s)	1
2. Objectifs	1
3. Description	2
4. Livrables	4
5. G�n�ralit�s, notation de la SAf et r�sultat du concours	5

1. Auteur(s)

1.1. Du sujet 

  Jean-Michel Bruel

  Version: 2023.01 (BUT1 2023)

  Dur e : 12 heures (3 heures   domicile, 1 TD et 1 TP encadr s, 3 cr neaux en autonomie)

Merci   Saadia Albane pour l d e du probl me   r soudre

1.2. De la solution 

  LAST NAME : DOE

  First name : John

  TD group :

! 1

" 2

! 3

! 4

2. Objectifs

L objectif de cette SAf (Situation d'Apprentissage et d' valuation) est d approfondir la r flexion sur l approche algorithmique des probl mes rencontr s pendant les phases de d veloppement. (cf. <docs/sae2.02.pdf>).

Plus pr cis ment :

  Participer   un concours de codage

  Lire, comprendre et  valuer un code qui n est pas le sien

- ¥ Comparer des algorithmes sur un critère précis
- ¥ Justifier de manière objective ses comparaisons et son classement

3. Description

Cette SAf se déroule en 2 phases.

3.1. Phase 1 : concours d'algorithme

Vous allez devoir soumettre un algorithme qui résout un problème simple (niveau BUT S1) mais qui peut se régler avec plusieurs solutions différentes. Vous avez la semaine 22 (non encadrée, mais questions sur Discord bienvenues) pour réaliser et soumettre votre (ou vos) solutions.



Cette 1^{re} phase est individuelle.

Le problème est le suivant :

Un explorateur a découvert un texte d'un peuple ancien (**texte**) rempli de mots (ensembles contigus de lettres). Il pense avoir trouvé l'ensemble des lettres premières utilisées, ainsi que l'ordre (**ordre**) utilisé par ce peuple pour les classer et ainsi réaliser leurs dictionnaires. Il vous demande d'écrire une fonction (java ou C) qui classe les mots d'un **texte** en fonction de **ordre** (comme il n'est pas sûr de lui il veut faire plusieurs essais).

Exemple d'input

```
texte = "Il fait beau aujourd'hui comme en aout"
```

```
ordre = ['f', 'l', 'z', 'u', 'k', 'a', 'b', 'o']
```

Exemple d'output

```
["fait", "Il", "aujourd", "aout", "beau", "hui", "comme", "en"]
```

Les contraintes sont les suivantes :

- ¥ votre algorithme doit être écrit dans l'un des langages suivants au choix : java, ou C
- ¥ il doit permettre à l'un des 2 programmes principaux fournis (java ou C) de fonctionner (respect donc des noms de classes, méthodes ou fonctions en conséquence). Le choix du nom de la fonction n'est donc pas libre!
- ¥ le texte est donné sous forme d'une chaîne de caractères (sans accents pour éviter les soucis)
- ¥ l'ordre est donné sous forme de liste de lettres
- ¥ si l'ordre n'est pas complet, tout mot commençant par une lettre "inconnue" est placé après le dernier classé, sans contrainte d'ordre vis-à-vis des autres non-classés

Vu qu'il existe de nombreuses façons de résoudre ce problème, vous devrez soumettre, pour chaque catégorie, votre meilleure solution et votre pire solution.

Simplicité

Ici il s'agit de faire un code facile à maintenir, lisible par des humains. Pas forcément efficace, mais très facile à lire et à réutiliser. Toute méthode de `java.util` existante est autorisée.

Efficacité

Peu importe le code source, c'est l'efficacité de son exécution qui est recherchée (complexité maîtrisée, temps d'exécution minimal, etc.). Ici aucune méthode complexe (de type `split()` ou `sort()`) ne devra être utilisée (contrairement à celles de type `size()` ou `length()` qui sont autorisées).

Sobriété numérique

L'algorithme consomme le moins de ressources possible (mémoire, calcul, etc.).

■

Vous pouvez soumettre plusieurs algorithmes dans plusieurs catégories pour maximiser vos chances de gagner le concours et obtenir des points bonus.

!

Nous sommes conscients que vous pouvez vous aider de ChatGPT ou des codes de vos collègues, mais la notation qui a le plus gros coefficient est la plus finale. Si vous êtes incapable d'expliquer vos propres résultats, cette note s'approchera de 0.

3.1.1. Dépôt

Vous devrez déposer sur [WebEtud](#), avant samedi 3 juin à 23h59, vos fichiers de solutions en les nommant ainsi (pour le dépôt) : `[efficacité|sobriété|simplicité]-[meilleur|pire].[java|c]`.

Par exemple pour votre meilleur algorithme java en simplicité, vous le déposerez avec le nom `simplicité-meilleur.java`.

■

⚠ Si vous en déposez plusieurs d'une même catégorie/type, numérotez-les (`simplicité-meilleur1.java` et `simplicité-meilleur2.java`)

⚠ Ne mettez aucun commentaire ou signe qui permettent de vous identifier dans le code!

⚠ Pensez à déposer aussi les `.h` pour les fonctions C.

3.2. Phase 2 : comparaison et Évaluation des solutions

Dans cette deuxième phase, (avec séances encadrées et libres), vous devrez comparer des solutions entre elles, et les classer en justifiant vos analyses.

!

Cette deuxième phase est en binôme (de votre choix)

Vous vous verrez affecter, pour chaque catégorie d'algorithmes (Simplicité, Efficacité, Sobriété) un certain nombre de solutions au hasard parmi celles soumises en phase 1.

Il vous faudra évaluer chaque algorithme selon des critères et les classer ensuite.

”

On vous impose au minimum les critères ci-dessous mais vous pourrez en rajouter. Êtes-vous de les utiliser judicieusement pour les catégories les plus appropriées.

3.3. Critères de comparaison

Lisibilité du code

Ce critère est subjectif. Il se base sur la facilité à comprendre ce que fait le code.

Qualité du code

Vous utiliserez des outils open source de mesure de qualité de code (e.g., [Codacy](#)).

Efficacité

Il s'agit d'évaluer la complexité algorithmique de la solution ($O(n^2)$ ou $O(n \log(n))$). Si on double par exemple la taille de la donnée en entrée, est-ce qu'on double le temps de calcul ?

Sobriété numérique

Cela devient un critère de plus en plus important. Certains outils permettent de donner une mesure de la consommation en ressources d'un algorithme (e.g., [Joular](#)).

Temps d'exécution

Il s'agit de mesurer le temps d'exécution.

!

Il conviendra de prendre des mesures sur des données plus ou moins grandes, certains algorithmes étant plus rapides que d'autres en fonction de la taille des données en entrée (beaucoup de mots dans la chaîne initiale), ou de leur variété (beaucoup de grands mots).

4. Livrables

Vous utiliserez le dépôt initial qui vous aura été attribué via classroom pour pousser vos codes et vos livrables (en plus des dépôts moodle).

4.1. Phase 1 (deadline : samedi 3 juin 2023 à minuit)

! Votre ou vos algorithmes en précisant les éléments du tableau ci-dessous :

#	lien	langage	catégorie	Type
1	meilleur java	Java	Simplicité	Meilleur
2	pire java	Java	Efficacité	Pire

4.2. Phase 2 (deadline : vendredi 16 juin 2023 à minuit)

! Le rapport d'évaluation des algorithmes (e.g., asciidoc ou PDF). Pour chaque catégorie, vous

devrez d signer qui est 1er, 2 me, 3 me,   (avec possibilit  d ex-aequo si le hasard vous a attribu  des algos similaires). Il doit se trouver dans le r pertoire **rapport** de votre d p t.

- ! Les codes de test, d valuation ou de mesure. Ils doivent se trouver dans le r pertoire **analyse** de votre d p t.
- ! Les r f rences des biblioth ques/outils utilis s (pour ceux non fournis). Elles doivent  tre list es dans la sous-section (R f rences) ci-dessous.
- ! La cha ne de compilation et ex cutable, ou packaging selon les standards du langage (comment ex cuter vos codes d valuation). Cette description doit se trouver dans vos rapports.



Les r pertoires et fichiers existants devront  tre compl t s et mis   jour sans  tre renomm s. Les binaires de compilation (r pertoire **bin** ou **target** par exemple) ne devront pas  tre pouss s sur le d p t.

4.3. Pr -requis

Voici les pr -requis pour ex cuter nos codes d valuation.

  Java v.x.y.z

   

4.4. Reproductibilit 

  Pour reproduire nos analyses :

1. Installez X
2. Lancez Y
3.  

4.5. R f rences

  [Mon super outil XYZ](#)

   

5. G n ralit s, notation de la SAf et r sultat du concours

5.1. G n ralit s

  Vous pouvez vous entraider pour les outils d analyse et de performance, voire vous inspirer de ChatGPT

  N h sitez pas   solliciter vos enseignants des ressources impliqu es par cette SAf (salon [#sae_2_02_qualit ](#) du serveur discord).

5.2. Notation

- ¥ 90% de la notation portera sur votre rapport de la phase 2 et vos analyses (vřracitř, pertinence, qualitř, ajout de critřres pertinents, É). Lřvaluation comportera un oral en semaine 25 (lors des sřances encadrřes).
- ¥ 10% de la notation portera sur le classement de votre algorithme de la phase 1 (pertinence de la catřgorie choisie, řvaluation/classement par les pairs, É)
- ¥ Bonus pour les 10 premiers de chaque catřgorie du concours de codage et ce, pour chaque "type" (les 1^ meilleurs, et les 10 pires)
- ¥ Bonus pour ceux qui auront proposřs plusieurs algos diffřrents (indřpendamment de leur classement final)
- ¥ Bonus supplřmentaire pour ceux qui auront proposřs des versions en langages diffřrents de leur(s) algo(s) (indřpendamment de leur classement final)

5.3. Divers

- ¥ Pour le rřsultat du concours, les algorithmes de la catřgorie "performances" seront rřcompensřs par langage et par "type".