Semaine complexité

Matière
- Première notion de complexité d'un algorithme : O(1)//O(N)

Objectifs
- Découvrir qu'il est possible d'écrire des méthodes très optimales si on tient compte de la spécificité de la table.

Exercices obligatoires

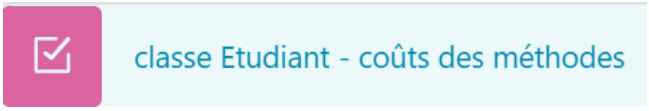
A L'étudiant et sa table de cotes

a) Complétez la classe *EtudiantCotesNonTriees*.
Chaque étudiant possède un numéro de matricule et une table avec ses cotes (réels compris entre 0 et 20). Cette table n'est pas triée.
La classe *TestEtudiantCotesNonTriees* permet de tester cette classe.

b) Complétez la classe *EtudiantCotesTriees* dans laquelle la table des cotes de l'étudiant est triée par ordre croissant.
Toutes les méthodes que vous avez écrites dans la classe *EtudiantCotesNonTriees* peuvent être reprises sans changement MAIS elles peuvent généralement être rendues plus efficaces !
Par exemple, pensez à ceci :
- Où se trouve la cote la plus basse ?
- Où se trouve la cote la plus élevée ?
- En cas d'échec(s), où se place(nt)-t-il(s) dans la table ?

Essayez d'écrire des méthodes de façon optimale en tenant compte du fait que la table est triée
La classe *TestEtudiantCotesTriees* permet de tester cette classe.
Attention celle-ci ne permet pas de vérifier l'efficacité des méthodes.

c) Faites le test moodle :



B Les drapeaux

B1 Un drapeau belge d'enfants.

L'école a demandé à tous les enfants de venir habillés en noir, jaune ou rouge afin de constituer un immense drapeau belge.

Chaque écolier va être placé un à un dans le drapeau.
Pour éviter le gros chambardement, il faut que ces ajouts engendrent le moins de déplacements possibles.

Imaginez toutes des chaises alignées.
Pour placer un enfant habillé en noir sur la première chaise, il faudrait que tous les enfants déjà installés se décalent (O(N)).
On pourrait simplement déplacer le premier enfant habillé en rouge sur la première chaise inoccupée en fin de drapeau.
Ce déplacement libère une chaise sur laquelle, on pourrait placer le premier enfant habillé en jaune.
Ce déplacement libère une place pour l'enfant habillé en noir. Avec cette solution, seulement 2 enfants devront se déplacer ! (O(1))

Exemple : on veut ajouter nick habillé en noir

tableTriee avant :

nora	nico	noel	nestor	julie	jo	rene	remi	robin	

Pour ajouter nick, 2 déplacements sont nécessaires :

nora	nico	noel	nestor	julie	jo	rene	remi	robin	

(2) (1)

tableTriee après ajout de nick :

nora	nico	noel	nestor	**nick**	jo	**julie**	remi	robin	**rene**

Il faut également prévoir le désistement d'un enfant.
Comme on ne veut pas de chaise inoccupée en plein drapeau, il faudrait que tous les enfants qui le suivent se décalent d'une chaise (O(N)). En réfléchissant un peu, on se rend compte qu'il y a moyen d'occuper la chaise libérée en ne faisant bouger que 3 enfants maximum ! (O(1))

Pour cet exercice nous vous donnons les classes suivantes :

Ecolier, qui permet de représenter un enfant ayant une couleur. Chaque écolier possède un nom et une couleur parmi noir, jaune ou rouge.

NoirJauneRouge, qui permet de gérer le drapeau. Elle contient un tableau d'écoliers et le nombre d'écoliers de chaque couleur.
Le nombre d'écoliers est limité à `NOMBRE_MAX_ECOLIERS`.
On ne peut retrouver 2 écoliers qui ont le même nom.
La particularité de cette classe est qu'elle maintient le tableau trié suivant les couleurs : d'abord tous les noirs, ensuite tous les jaunes, ensuite tous les rouges.
L'ordre des écoliers d'une même couleur n'a pas d'importance.
Vous allez compléter les **méthodes d'ajout et de suppression** en respectant la *JavaDoc*.
Les algorithmes sont imposés. Les principes de base sont expliqués et illustrés dans le document *DrapeauBelge*.

Test6CasNoirJauneRouge, qui reprend les exemples repris dans le document *DrapeauBelge* qui se trouve sur moodle. **Attention, seuls quelques exemples sont testés**. Ce n'est donc pas parce que quelques exemples passent que forcément vos méthodes sont correctes.

TestPersoNoirJauneRouge qui permet d'introduire vos propres tests.
Pensez à tester des cas particuliers : ajout d'un nom existant, ajout d'un 11ème écolier, …

Pensez à toujours bien tester vos classes.

Exercices supplémentaires

B2 Drapeau bicolore à trier

On remplit aléatoirement toutes les cellules d'un tableau de n cases avec des entiers.

43	18	39	76	27	85	52

On trie ensuite ce tableau afin d'obtenir tous les nombres pairs en premier ensuite les impairs.

52	18	76	27	85	39	43

On vous demande de compléter la classe *DrapeauBicolore* qui contient un tableau d'entiers.
Vous devez pour cette classe compléter la méthode `triBicolore()` qui réorganise le tableau pour que les pairs apparaissent avant les impairs.

L'algorithme de tri est imposé.
Il est suggéré dans le code de la méthode à compléter.
Essayez de le comprendre sans consulter le document *TriBicolore*.
Le document *TriBicolore* reprend toutes les étapes qui ont permis de trier le tableau ci-dessus.

La classe *TestTriBicolore* permet de tester la classe.
Vous devez vérifier les affichages.

Exercices défi

B2 Améliorez la méthode `triBicolore()`.
On se rend compte que beaucoup de permutations ont été faites pour rien en utilisant l'algorithme proposé ! Les impairs sont systématiquement déplacés, alors qu'ils ne le devraient pas nécessairement.

B3 

Le drapeau Hollandais - Dijksta

L'algorithme du néerlandais E.W. Dijkstra répond au problème suivant :
On remplit aléatoirement toutes les cellules d'un tableau de n cases avec des boules bleues, blanches et rouges.
On trie ensuite ce tableau afin d'obtenir le drapeau hollandais.
Pour cela on doit :

- Utiliser ce seul tableau et le parcourir une seule fois.
- Évaluer pour chaque boule, chaque couleur une seule fois au maximum.

Dans un premier temps, essayez d'imaginer l'algorithme. Ensuite, essayez de comprendre l'algorithme dans le document *AlgorithmeDrapeauHollandais*.

La classe *DrapeauTricolore* contient un tableau de char : b (blue), r (red) et w (white).
Le constructeur de cette classe crée un tableau et le remplit complètement de boules dont les couleurs sont choisies aléatoirement parmi les trois couleurs suivantes : b, r et w
Complétez la méthode `triTricolore()` qui réorganise le tableau pour que les boules apparaissent dans l'ordre : b, w, r.
Complétez la classe *TestDrapeauTricolore* à votre guise.