IUT d'Orsay, Département Informatique *Année scolaire 2009-2010*

Formation Initiale 2ème année PEC ***Projet***

Tri & Complexité

Une chaîne industrielle de production d'œufs est constituée de trois machines, qui travaillent le long d'un plateau sur lequel les œufs sont alignés l'un derrière l'autre:

- La première machine dépose en temps réel au fur et à mesure de leur ponte, l'un derrière l'autre, les œufs pondus par une batterie de poules.
- La seconde machine marque les œufs d'une étiquette de couleur et du numéro d'arrivée de l'œuf dans la file. La couleur de l'étiquette est **bleu** pour les gros œufs, **blanc** pour les moyens, et **rouge** pour les petits.
- Une fois le plateau rempli, la troisième machine a pour fonction de trier les œufs: elle place les gros (bleus) en tête, les moyens (blancs) ensuite, et les petits (rouges) en queue. Les œufs non encore triés se situent entre les moyens et les petits.

BLEU	BLANC	A TRIER *(la pioche)*	ROUGE

Le **tri** effectué par cette 3ème machine est **stable**, c'est-à-dire que le placement des œufs d'une même couleur conserve l'ordre chronologique de ponte.

Exemple :
avant le tri :

(1, Rouge)	(2, Bleu)	(3, Rouge)	(4, Blanc)	(5, Blanc)	(6, Bleu)	(7, Rouge)

après le tri :

(2, Bleu)	(6, Bleu)	(4, Blanc)	(5, Blanc)	(1, Rouge)	(3, Rouge)	(7, Rouge)

Le plateau sera représenté par un tableau. Le sous-tableau non trié est appelé *la pioche*.
Les indices *finBleu, finBlanc, finPioche* et *dernier* désignent la dernière case, **respectivement**, des emplacements des œufs gros, moyens, non triés, et petits. L'indice (*finBlanc+1*) adresse donc la première cellule de *la pioche*, où se trouve l'élément en cours de traitement.

La procédure de tri est la suivante :
Les œufs non triés sont traités dans l'ordre de leur numéro.
Soit P l'œuf à traiter.

- Si P est blanc, aucun œuf n'est déplacé.
- Si P est rouge, on décale les zones *pioche* et *rouge* d'un cran vers la tête et on dépose P en queue de file.
- Si P est bleu, et si la zone *blanche* n'est pas vide, on décale la zone *blanche* de un cran vers la queue et on dépose P sur la case libérée. Si P est bleu, et si la zone *blanche* est vide, aucun œuf n'est déplacé.

Un *œuf* est implémenté par une classe ayant pour attributs le numéro d'ordre (entier) et la couleur (chaîne de caractères).
La *file* des œufs est implémentée sous forme d'une classe qu'on appellera *File*.
On utilise deux méthodes de la classe *File* :

- *dernier()* donne l'indice *dernier* de la dernière case de la file.
- UnOeuf(int i) renvoie l'oeuf placé à la case d'indice fourni en argument.

1. Écrire **en pseudo-langage** un algorithme **itératif** qui traduit cette procédure. Vous insèrerez un compteur des écritures dans le tableau, il aura pour rôle d'évaluer la complexité de l'algorithme
   Respectez les règles de syntaxe et de présentation de votre algorithme, en y insérant les commentaires qui le rendent immédiatement compréhensible.
2. Implémentez votre algorithme sous la forme d'un progamme en *Java* (avec commentaires pour le rendre lisible). Incluez le compteur des écritures dans le tableau, et une fonction de calcul du temps d'exécution.
3. Créez un petit sous-programme de test qui vérifie que le tableau après tri est bien correctement trié.
4. Créez une méthode de *File* qui génère aléatoirement un tableau d'oeufs non triés, la taille du tableau étant en paramètre. Testez quelle est la limite supérieure de la taille du tableau au delà de laquelle votre programme échoue.
5. Votre programme principal en *Java* demandera tout d'abord la taille du tableau, puis exécutera en boucle 1000 fois votre programme de tri, chaque tri étant effectué sur un nouveau tableau généré aléatoirement. Testez votre tri comme prévu à la question 3 sur 1000 tableaux de taille 100.
6. La complexité moyenne pour 1000 tris sera alors calculée. Vous exécuterez ce protocole pour une taille de tableau égale successivement à 10, 20, 50, 100, 200, 500, 1000.
   Vous inscrirez vos résultats dans un tableau, et vous représenterez la courbe donnant la complexité en fonction de la taille n du tableau. (vous générerez cette courbe par le moyen de votre choix, Excel par exemple). Dans le même repère, représentez la fonction n -> a n^2, où vous déterminerez le coefficient réel *a* qui permet de s'approcher au mieux de la courbe de complexité.
7. Reprendre les questions 2 et 3 pour une version **récursive** du programme (récursivité terminale). Testez quelle est la limite supérieure de la taille du tableau au delà de laquelle votre programme échoue.
8. Reprendre les questions 6 et 7 pour la version récursive, dans la limite de la taille autorisée pour le tableau. Comparez avec la version itérative.
9. Si l'on dispose d'un second plateau (une seconde file) pour déposer les œufs triés, présenter rapidement (**en français**), une **idée simple d'algorithme**, pour le même résultat de tri, mais dont la complexité serait très inférieure. Vous évaluerez l'ordre de grandeur de cette complexité en justifiant cette évaluation.
10. Implémenter l'algorithme précédent.

Regroupez tous les programmes demandés ci-dessus dans un seul ***main avec*** un menu de choix.

Par mail envoyé avant le **vendredi 5 mars à minuit**, dont le champ *objet* contiendra : ***projet 1 PEC*** ,
vous enverrez vos sources dans un fichier compressé. Vous enverrez en plus, hors de ce fichier compressé, un compte-rendu **au format pdf** qui répond à toutes les demandes de l'énoncé (excepté les programmes *Java)*.