Séance 4: Exercices sur les chaînes de caractères

Université Paris-Diderot

Exercice 1 (Parcourir un tableau, ⋆)

En guise d'échauffement, écrire une fonction `afficher(int[] tab)` *qui prend en paramètre un tableau d'entiers et qui les affiche séparés par des points virgules.*

```
afficher({1, 2, 6, 5, 3, 12, 0}) // 1;2;6;5;3;12;0;
afficher({}) // n'affiche rien
afficher({1, 4, 3, 4, 2}) // 1;4;3;4;2;
```

□

Exercice 2 (Fonction *maximum*, ⋆⋆)

Le but de cet exercice est de proposer une fonction `maximum(int[] tab)` *qui prend en paramètre un tableau d'entiers positifs et qui renvoie la valeur maximale du tableau. Si le tableau est vide, elle renvoie zéro.*

```
maximum({1, 2, 6, 5, 3, 12, 0}) // 12
maximum({}) // 0
maximum({1, 4, 3, 4, 2}) // 4
```

Au quotidien... La fonction qui trouve le maximum d'un tableau est implémentée dans tous les langages et utilisée dans la grande majorité des logiciels. Par exemple, dans un projet où une intelligence artificielle joue aux échecs, on peut utiliser la fonction maximum pour déterminer le coup associé à la plus forte probabilité de victoire. □

Exercice 3 (Décalage vers la droite, ⋆⋆)

Proposer une fonction `decale(int[] tab, int d)` *qui prend en paramètre un tableau d'entiers positifs et qui renvoie le tableau décalé de* `d` *positions vers la droite. Les valeurs sortant du tableau par la droite sont envoyées à gauche. On pourra s'aider d'une fonction* `decale1(int[] tab)` *qui décale les éléments d'un cran vers la droite.*

```
decale({0, 1, 2, 3}, 1) // 3, 0, 1, 2
decale({0, 1, 2, 3}, 2) // 2, 3, 0, 1
decale({0, 1, 2, 3}, 3) // 1, 2, 3, 0
decale({0, 1, 2, 3}, 4) // 0, 1, 2, 3
```

Au quotidien... Dans un jeu à plusieurs joueurs, on peut utiliser une fonction de décalage pour implanter le système de tour-par-tour. Le joueur en première position du tableau est le joueur actif, et on décale le tableau d'un cran vers la gauche pour passer au second joueur. Le premier joueur se retrouve alors à la fin de la liste, et doit attendre un cycle complet pour rejouer. □

Exercice 4 (Compter les éléments différents, ⋆⋆)
Proposer une fonction `compteDifferent(int[] tab)` *qui prend en paramètre un tableau d'entiers positifs et qui renvoie le nombre d'éléments différents dans le tableau. On pourra s'aider d'un tableau "mémoire" qui retient les éléments déjà vus.*

```
compteDifferent({0, 1, 2, 3}) // 4
compteDifferent({0, 1, 2, 0}) // 3
compteDifferent({0, 1, 1, 1}) // 2
compteDifferent({}) // 0
```

Au quotidien... Dans un système de commentaires où chaque utilisateur peut écrire plusieurs avis, on peut utiliser un tableau contenant pour chaque commentaire le pseudonyme de l'utilisateur associé. Compter les éléments dans ce tableau en ne comptant qu'une fois chaque utilisateur permet de connaître le nombre d'utilisateurs ayant interagi au total. □

Bonus

Exercice 5 (Une fonction de tri, ⋆⋆⋆)
Dans cet exercice on va implémenter le tri par sélection *– un algorithme qui trie un tableau de longueur n en temps proportionnel à n^2.*

1. *Écrire une fonction* `swap(int[] a, int i, int j)` *qui prend en paramètre un tableau d'entiers* `a`*, deux indices* `i` *et* `j`*, et qui échange les éléments situés aux positions* `i` *et* `j` *dans le tableau* `a`*.*
2. *Écrire une fonction* `int findMinIndex(int[] a, int startPos)` *qui prend en paramètre un tableau d'entiers positifs* `a` *(de longueur n), un entier* `startPos` *inférieur à n, et qui renvoie l'indice i du plus petit élément situé après l'indice* `startpos`*. Autrement dit, i est un indice entre* `startPos` *et $n-1$ tel que* `a[i]`$\leq$`a[j]` *pour tous* `j` *tels que* `startPos` $\leq$ `j` $< n-1$*.*
3. *Utiliser les fonctions* `swap` *et* `findMinIndex` *pour écrire la fonction* `sort(int[] a)` *qui prend en paramètre un tableau d'entiers positifs* `a` *et qui trie le tableau* `a` *dans un ordre croissant. Indice : que fait la ligne* `swap(a, i, findMinIndex(a, i))` *?*

```
findMinIndex({5, 4, 5, 6}, 0) // 1
findMinIndex({5, 4, 5, 6}, 1) // 1
findMinIndex({5, 4, 5, 6}, 2) // 2
findMinIndex({5, 4, 5, 6}, 3) // 3

tri({2, 1, 3, 3}) // 1, 2, 3, 3
tri({0, 1, 2, 0}) // 0, 0, 1, 2
tri({1, 1, 0, 2}) // 0, 1, 1, 2
tri({}) //
```

Au quotidien... Les fonctions de tri sont omniprésentes en algorithmique et en programmation, car elles ont un nombre d'utilisations immense. Lorsqu'on essaie d'organiser une large collection d'objets (des livres numériques, des fichiers, des données bancaires, des joueurs...), les fonctions de tri facilitent grandement le travail, ce qui permet d'accéder rapidement aux éléments les plus "intéressants". □