TD 4 (algorithmique) - Compromis performances / mémoire

Exercice 1

Il est souvent utile de stocker les résultats de certains calculs que l'on effectue, en vue de les réutiliser pour accélérer des calculs ultérieurs. Pour les deux questions qui suivent, il faut initialiser une liste **globale** à laquelle on accédera pour accélérer les calculs.

(a) Écrivez une fonction `factorielle(n)` calculant la factorielle de $n \in \mathbb{N}$ en $O(1)$ si une factorielle plus grande a déjà été calculée, et en $O(n)$ sinon.

(b) Le nombre de Fibonacci de rang $n \in \mathbb{N}$, noté F_n, est défini par :

$$F_n = \begin{cases} n & \text{si } n \leq 1, \\ F_{n-1} + F_{n-2} & \text{sinon.} \end{cases}$$

Écrivez une fonction `fibonacci(n)` renvoyant F_n en $O(1)$ si une valeur supérieure a déjà été calculée, et en $O(n)$ sinon.

Exercice 2

Pour rappel, un mot est un *anagramme* d'un autre mot s'il contient exactement les mêmes lettres mais dans un ordre différent. Par exemple, le mot `'imaginer'` est un anagramme du mot `'migraine'`. Écrivez une fonction en $O(n)$ renvoyant `True` si les deux chaînes qu'on lui passe en paramètres sont des anagrammes, et `False` sinon.

Remarque : si les dictionnaires n'ont pas encore été vus, on supposera que les mots ne contiennent que des lettres minuscules non accentuées. Vous aurez alors besoin de la fonction `ord(c)`, qui renvoie la valeur numérique du caractère `c` passé en paramètre. Ces valeurs étant toutes consécutives pour les minuscules, on utilisera la relation `ord(c) - ord('a')` pour envoyer ces valeurs sur l'intervalle entier $[0, 25]$.

Exercice 3

Soit A et B deux listes de naturels, de tailles respectives m et n, et $V = \max(\max(A), \max(B))$. Vous pouvez supposer que ni A ni B ne contient de doublons, mais A et B peuvent avoir des éléments en commun. Pour les questions qui suivent, vous ne pouvez pas utiliser `set` ni `dict`.

(a) Écrivez une fonction renvoyant l'union de A et B sous la forme d'une troisième liste sans doublon. La complexité de la fonction **doit** être $O(m + n + V)$, où V est la plus grande valeur apparaissant dans A et B.

(b) Écrivez une fonction renvoyant l'intersection de A et B sous la forme d'une troisième liste sans doublon. La complexité de la fonction **doit** être $O(m + n + V)$, où V est la plus grande valeur apparaissant dans A et B.

(c) Si ce n'est déjà fait, modifiez vos solutions pour que la liste renvoyée dans les deux sous-questions précédentes soit triée. Vous ne pouvez pas utiliser `sorted` ni `list.sort`.

Exercice 4

Écrivez une fonction permettant d'insérer un élément e à la position i dans une liste "à trous" comme vu au cours. Cette liste sera représentée par une matrice $2 \times n$ dont la première ligne contient les données et la seconde les positions libres. La fonction doit en outre renvoyer `True` si l'insertion s'est bien passée et `False` dans le cas contraire. Si la place i n'est pas libre, on peut faire l'insertion à n'importe quelle autre place libre.

Exercice 5

Écrivez une fonction `compression(M)` renvoyant dans une liste le contenu d'une matrice symétrique sans ses éléments superflus. Par exemple :

```
>>> M = [[1, 2, 3, 4], [2, 5, 6, 7], [3, 6, 8, 9], [4, 7, 9, 10]]
>>> compression(M)
[1, 2, 5, 3, 6, 8, 4, 7, 9, 10]
```

Exercice 6

Soit L le résultat de la fonction `compression` sur une certaine matrice symétrique M à laquelle on n'a pas accès. Écrivez une fonction renvoyant, sur base de L, le nombre n de lignes de la matrice M de départ.

Remarque : il est possible d'effectuer cette tâche en $O(1)$.