Algorithmique Avancée : Séance 4 TP

Mardi 22 octobre 2024

Pour chaque fonction écrite, décrire la complexité au meilleur des cas et la complexité au pire des cas.

Exercice 1 (Tris)**.** Codons les tris vus en classes. Il faut les essayer pour s'assurer que le programme fonctionne correctement.

1. Écrire une fonction `tri_insertion` qui trie une liste par ordre décroissant.
2. Écrire une fonction `tri_bulle` qui trie une liste par ordre décroissant.

Exercice 2 (Taleaux 2D)**.** Travaillons sur les tableaux à deux dimensions !

1. Écrire une fonction `init_lignes(n)` qui prend en paramètre un entier `n`, et qui renvoie une liste contenant n listes vides.
2. Écrire une fonction `init_colonnes(tab, n)` qui prend en paramètre `tab` une liste de listes vides et un entier `n`, qui remplie les listes contenues dans `tab` par n zéros chacune, et qui renvoie la liste ainsi obtenue.
3. Écrire une fonction `affiche_lignes(tab)` qui prend en paramètre `tab` une liste de listes, et qui affiche les listes de `tab` à hauteur d'une liste par ligne.
4. Écrire une fonction `affiche_tab(tab)` qui prend en paramètre `tab` une liste de listes, et qui affiche le contenu de `tab` sous forme de tableau (sans les crochets qui apparaissent dans l'exécution de la fonction d'avant).
5. Écrire une fonction `put(tab, i, j, element)` qui prend en paramètre `tab` une liste de listes, et trois entiers `i`, `j` et `element`. La fonction `put(tab, i, j, element)` écrit `element` dans la case ligne i colonne j du tableau et le renvoie.
6. Écrire une fonction `get(tab, i, j)` qui prend en paramètre `tab` une liste de listes, et deux entiers `i` et `j`. La fonction `get(tab, i, j)` renvoie le contenu de la case ligne i colonne j du tableau.
7. Écrire une fonction `somme_tab(tab)` qui prend en paramètre `tab` une liste de listes, qui renvoie la somme de tous les éléments du tableau.
8. Écrire une fonction `cherche(tab, element)` qui prend en paramètre `tab` un tableau à 2 dimensions, et qui renvoie les coordonnées (ligne, colonne) d'`element` dans le tableau (ou $(-1, -1)$ si non présent).

9. Écrire une fonction `minimum(tab)` qui prend en paramètre `tab` un tableau à 2 dimensions, et qui renvoie le minimum du tableau.

10. Écrire une fonction `minimum_lignes(tab)` qui prend en paramètre `tab` un tableau à 2 dimensions, et qui renvoie une liste contenant le minimum de chaque ligne du tableau.

11. Écrire une fonction `minimum_colonnes(tab)` qui prend en paramètre `tab` un tableau à 2 dimensions, et qui renvoie une liste contenant le minimum de chaque colonne du tableau.

12. Écrire une fonction `max_ligne_col(tab)` qui prend en paramètre un tableau `tab` de taille $n \times n$ et qui renvoie le maximum de chaque ligne et le maximum de chaque colonne.

13. Écrire une fonction `max_par_ligne_colonne(tab)` qui prend en paramètre `tab` un tableau à 2 dimensions et qui renvoie deux listes : la première contenant le maximum de chaque ligne et la seconde contenant le maximum de chaque colonne.

14. Bonus : Réécrire la fonction précédente en itérant une seule fois sur le tableau.

15. Écrire une fonction `trier(tab)` qui utilise soit `tri_insertion` soit `tri_bulle` pour trier le tableau à 2 dimensions.

    Indice : Transformer le tableau 2D en un tableau 1D avant de le trier, puis retransformer en 2D.

Exercice 3 (Sudoku)**.** Écrire le code pour des sudoku de taille 9×9, mais analyser la complexité pour du $n \times n$ (en supposant que n est un carré).

1. Écrire une fonction `sudoku_correct(tab)` qui prend en paramètre un tableau à deux dimensions `tab` et qui vérifie si c'est un sudoku résolu.

2. Écrire une fonction `ecrire_sudoku(tab, f)` qui écrit un sudoku (un tableau à deux dimensions) dans un fichier `f` sous le format suivant : une virgule sépare les éléments de la même ligne, et un point-virgule sépare une ligne de la suivante.

3. Écrire une fonction `lire_sudoku(f)` qui lit et renvoie un sudoku d'un fichier `f` sous le même format qu'auparavant.

Aide pour lire et écrire dans les fichiers

Pour lire :

1. `f = open("nom_fichier", "r")` pour lire en lecture
2. `contenu = f.read()` pour lire le contenu de `f` dans `contenu`
3. `f.close()` pour fermer le fichier

Pour écrire :

1. `f = open("nom_fichier", "w")` pour lire en écriture
2. `f.write(contenu)` pour écrire `contenu` dans `f`
3. `f.close()` pour fermer le fichier