Soutien en algorithmique et programmation

Séance 4 : tris de tableaux

Introduction

On va travailler sur des tris classiques sur des tableaux. On manipule de simples tableaux d'entiers, que l'on veut trier par ordre croissant. **Attention**, dans tous les exercices ci-dessous, on considèrera qu'**on travaille sur des tableaux et non des vecteurs** : **on travaillera donc toujours par échange d'éléments** et en ne modifiant jamais la taille du tableau et sans créer de tableau intermédiaire.

Pour tester les fonctions à implanter, on peut utiliser le programme principal ci-dessous, ainsi que la fonction permettant de créer un tableau rempli de chiffres aléatoires :

```
def creer(taille):
    """
      Cree un tableau rempli de valeurs aleatoires
    """
    return [randint(1, 9) for _ in range(taille)]

def main():
    """
      Fonction principale
    """
    for taille in range(6): # on teste sur des tableaux de taille 0 à 5 inclus
        print("-- Taille =", taille, "--")
        tab = creer(taille)
        tab_orig = tab[:] # copie du CONTENU du tableau
        print("Tableau initial        :", tab)
        trier_nain(tab)
        print("Tri du nain            :", tab)
        tab = tab_orig[:]
        print("Tableau initial        :", tab)
        trier_min(tab)
        print("Tri par sélect. du min :", tab)
        tab = tab_orig[:]
        print("Tableau initial        :", tab)
        trier_ins(tab)
        print("Tri par insertion      :", tab)
        print()
```

Attention : on rappelle qu'en Python, si `tab1` est un tableau, et qu'on écrit `tab2 = tab1`, on ne recopie pas le contenu de `tab1` dans `tab2` : l'affectation crée un *alias*, c'est à dire un deuxième nom pour le tableau `tab1`. Si on veut recopier le contenu d'un tableau, on peut utiliser la syntaxe `tab2 = tab[:]`.

Autre rappel important : si on utilise la syntaxe Python permettant de désigner des tranches de tableaux (par exemple `tab[debut:fin]`), la borne supérieure est **exclue** (dans l'exemple, on travaille donc sur la tranche de tableau dont les indices sont dans l'intervalle `[debut..fin[`).

Tri du nain de jardin

On va travailler dans cet exercice sur un tri itératif appelé le tri du nain de jardin. Son principe est très simple : un nain de jardin cherche à trier des pots de fleurs par taille croissante. Il regarde le pot devant lui : s'il est plus petit que le pot à sa droite, le nain avance d'un pas vers la droite (s'il n'est pas arrivé à la fin de la file de pots). Si le pot devant lui est plus grand que le pot à sa droite, le nain échange les deux pots, et recule d'un pas vers la gauche (s'il n'est pas revenu au début de la file de pots).

Vous implanterez donc une fonction `trier_nain(tab)` qui trie le tableau par ordre croissant en utilisant l'algorithme du nain de jardin.

Tri par sélection du minimum

On va maintenant implanter un tri par sélection (recherche) du minimum. Si on parcours le tableau à l'aide d'un indice courant `idx`, on garantit la propriété (invariant) suivante pour chaque pas de l'itération :

- tous les éléments dans `tab[0:idx]` sont déjà triés par ordre croissant ;
- tous les élément dans `tab[idx:len(tab)]` sont dans un ordre inconnu et doivent encore être traités.

A chaque pas de l'itération, on va donc chercher le minimum dans la tranche de tableau `tab[idx:len(tab)]` puis l'échanger avec l'élément courant `tab[idx]` : la propriété sera donc préservée et on pourra incrémenter `idx`.

Vous implanterez :

- une fonction `rechercher_min(tab, debut)` qui renvoie l'indice du minimum dans le tableau en partant de l'indice `debut` ;
- une fonction `trier_min(tab)` qui trie le tableau par ordre croissant en utilisant l'algorithme du tri par sélection du minimum.

Tri par insertion

On va enfin implanter un tri par insertion. Son principe est un peu la réciproque de celui du tri par sélection du minimum. On doit préserver la même propriété à chaque tour de la boucle, mais au lieu de chercher l'élément minimum dans la partie droite du tableau pour l'échanger avec l'élément courant, on va plutôt insérer l'élément courant à sa place dans la partie gauche (déjà triée) du tableau.

Pour insérer un élément dans la partie gauche du tableau, on doit bien sûr décaler les éléments présents d'une case vers la droite. Comme on travaille sur des tableaux (et pas des vecteurs), on s'interdit d'utiliser les méthodes `del` et `insert` dans cet exercice.

Vous implanterez donc:

- une fonction `decaler_inserer(tab, idx_val)` qui parcours le tableau de la droite vers la gauche à partir de l'indice `idx_val - 1` pour trouver la case où insérer la valeur `tab[idx_val]` (c'est à dire : reculer dans le tableau tant que `tab[idx]` est strictement supérieur à la valeur à insérer), puis insère cette valeur dans la case libérée ;
- une fonction `trier_ins(tab)` qui trie le tableau par ordre croissant en utilisant l'algorithme du tri par insertion.