INVARIANTS ET TESTS

Exercice 1: Un exemple de tri quadratique, le tri par insertion:

Pour un entier `N>1`, on note T une liste de N entiers.

On parcourt la liste et on insère l'élément courant dans la partie déjà triée à gauche, le reste de la liste n'est pas modifié. Pour commencer, le premier élément est bien placé car le slice T[0 :1] n'a qu'un seul élément donc est déjà trié. On peut visualiser ce tri : animation (tri du jeu de cartes)

On commence par une fonction nommée **`insertion (t,k)`** qui prend en entrée :

- un indice k vérifiant 0<k<N
- une liste T de N nombres telle que le slice T[:k] est trié

et qui insère la valeur T[k] dans la partie déjà triée T[:k] de sorte que le slice t[:k+1] soit trié.

La spécification en triplet de Hoare est :

#PE: N>1 ET 0<k<N ET (T[:k],<=)

#insertion(t,k)

#PS: (t[:k+1],<=) ET t[:k+1]=permut(T[:k+1])

On va écrire 2 versions de cette fonction insertion :

1. commencez par écrire une première version sans contrainte particulière :
2. écrire une deuxième version en complétant le schéma ci-dessous : (technique d'écriture avec preuve) :

L'algorithme commenté est :

```
def insertion(t,k):
    N=len(t)
    #PE 1
    T=t[:]  #T est une copie utilisée pour garder les valeurs initiales
    if t[k]<t[k-1]: #sinon il n'y a rien à faire
        ...
        ...
        t[k]=t[k-1]
        #INV: 0<=j<k et x=T[k] et t[j]>x et t[j+1:k+1]==T[j:k]  et t[:j]==T[:j]  ← INIT 2
        while j>0 and t[j-1]>x:
            #INV et CC
            ...
            ...
            #INV  ← FIN d'ITERATION 3
        #INV ET  (j=0 ou t[j-1]<=x) ← SORTIE DE BOUCLE 4
        t[j]=x
    #PS 5
```

a) Écrire une fonction booléenne **inv(j,k,t,T,x)**

b) remplacer les 5 lignes : **1 2 3 4 5** par des assert utilisant la fonction inv

c) compléter les lignes manquantes

d) tester votre fonction sur quelques exemples.

Écrire ensuite la fonction **tri-insertion(t)** qui renvoie la liste t triée : il s'agit d'une seule boucle for dans laquelle vous appelez la fonction insertion.

Tests : faciles en python grâce à la fonction `sorted()`qui renvoie une liste triée en conservant la liste initiale. La fonction **test_tri** suivante permet de tester tous les algorithmes de tri que vous aurez l'occasion d'écrire cette année. Elle prend en paramètre **le nom de la fonction de tri à tester tri_a_tester** et produit un affichage en cas de problème :

```
from copy import deepcopy
from random import randrange
#E : une fonction de tri : tri_a_tester
#S : affichage d'un message
def test_tri( tri_a_tester ):
    pb=False
    for i in range(1000) : #nombre de tris
        nb=randrange(10)  #longueur de la liste a trier
        #génération d'une liste aléatoire de nb nombres de [-10,10]
        T=[randrange(21)-10 for p in range(nb)]
        t=deepcopy(T)
        if sorted(T)!=tri_a_tester(t) #liste triée avec la fonction à tester
                pb=True
                print('Pb :')
                print(T)
                print(t)
    if pb==False :
        print( ''Ok'')


#appel de la fonction test_tri pour le tri_insertion
test_tri(tri_insertion)
```

Attention : Pour que le test fonctionne, la fonction `tri_a_tester` doit prendre pour seul argument la liste à trier et renvoyer la liste triée !

Exercice 2 : Le drapeau hollandais (Edsger Dijkstra)

Le problème consiste à réorganiser une collection d'éléments identifiés par leur couleur, sachant que trois couleurs seulement sont possibles (par exemple, rouge, blanc, bleu, si on considère le drapeau des Pays-Bas).

C'est un problème de tri qui peut être résolu grâce à **un seul parcours de la liste**. Le principe est le suivant :

On part des deux extrémités. On s'intéresse à la case courante du tableau T, dont on teste la couleur, et selon le résultat on procède à des échanges, de sorte qu'on ait à chaque étape à gauche une zone de bleus, puis une zone de blancs, puis une zone inconnue et enfin, à droite une zone de rouges. On va utiliser 3 variables :

- b (blue) : indice de la première case après la zone bleue connue,
- w (white) : indice de la première case après la zone blanche connue,
- r (red) : indice de la première case avant la zone rouge connue.

A chaque étape, on réduit la zone inconnue comprise entre les bornes w et r par test de la couleur de la case w.

1. Spécifier en triplet de Hoare
2. Déterminer l'invariant de boucle
3. Écrire la fonction **`drapeau_hollandais(t)`**
4. Justifier la bonne terminaison de l'algorithme
5. Tester en adaptant la fonction **`test_tri.`**