TD 3 : Invariant, Variant

Info1.Algo1

2022-2023 Semestre Impair

Rappels

Exercice 1

1) Expliquer la méthode permettant, à partir de la donnée d'un invariant et d'une post-condition, de déterminer la condition d'arrêt ainsi que la condition de boucle d'un algorithme de répétition.
2) Appliquer cette méthode aux cas suivants :
a) Invariant : `tab[i]<=0`
Post-condition : `tab[i]<=0 and tab[i+1]>0`
b) Invariant : `tab[i]<=0 and tab[j]>0`
Post-condition : `tab[i]<=0 and tab[i+1]>0`

Exercice 2

La fonction `somme_entiers` ci-dessous effectue la somme des entiers de `0` à `n`, où `n` est l'entier positif passé en paramètre.

```
def somme_entiers(n):
  k = 0
  s = 0
  while k<n:
    k += 1
    s += k
  return s
```

1) Compléter ce programme en écrivant les 4 assertions correspondant à la pré-condition, la post-condition et l'invariant.
Rappel : La somme des entiers de 0 à k (avec $k \geq 0$) est donnée par :

$$\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$$

2) Donner un variant pour cet algorithme.

Exercice 3

On souhaite écrire la fonction `log2_entier` qui accepte en paramètre un entier `n` strictement positif et retourne l'unique entier `k` postif tel que `2**k<=n<2**(k+1)`

1) a) Écrire la **pré-condition** de cette fonction.
b) Écrire la **post-condition** de cette fonction.
2) On suppose que **l'invariant** est : `k>=0 and 2**k<=n`.
En déduire la **condition d'arrêt** ainsi que la **condition de boucle**.
3) Écrire la fonction `log2_entier`.
4) Donner un **variant** pour cet algorithme.
5) (Pour aller plus loin) Pour éviter d'avoir à calculer une puissance dans la condition de boucle, on décide d'effectuer les modifications suivantes :

— **Post-condition :** `k>=0 and p==2**k and p<=n<2*p`
— **Invariant :** `k>=0 and p==2**k and p<=n`

Modifier la fonction `log2_entier` de manière à respecter ces modifications.

Exercice 4

On fournit la fonction auxiliaire `est_majorant` ci-dessous qui accepte en paramètre un tableau d'entiers `tab` et 3 entiers `m`, `d` et `f`. La fonction retourne le booléen indiquant si la valeur `m` majore la tranche `tab[:f]` :

```
def est_majorant(tab,m,f):
  for i in range(f):
    if tab[i]>m:
      return False
  return True
```

On souhaite dans cet exercice écrire la fonction `indice_maximum` qui accepte en entrée un tableau d'entier `tab` (de longueur `n`) et retourne un indice `i_max` du maximum du tableau `tab`.
Dans un premier temps, on ne suppose pas l'unicité de la solution. Ainsi pour le tableau :
`[1,6,9,8,0,7,9,4,9,2]`
la fonction pourra retourner tout indice auquel se trouve la valeur 9, c'est-à-dire 2, 6 ou 8.
1) Donner une **pré-condition** pertinente pour la fonction `indice_maximum`.
2) On suppose que l'algorithme adopté est un **parcours gauche-droite** et on donne les propriétés suivants :

— **post-condition :** `0<=i_max<n and est_majorant(tab,tab[i_max],n)`
— **invariant :** `0<=i_max<f and est_majorant(tab,tab[i_max],f)`
où `f` est la borne supérieure de la tranche explorée.

a) Déterminer la **condition d'arrêt** ainsi que la **condition de boucle**.
b) Écrire la fonction `indice_maximum`
3) (Pour aller plus loin)
a) Quelle condition peut-on rajouter à la pré-condition afin de garantir que la solution retourné soit unique ?
b) Écrire une fonction auxiliaire permettant de vérifier cette condition sur le tableau `tab`
c) Réécrire la pré-condition correspondante.

Exercice 5

On considère un tableau d'entiers trié par ordre croissant (variable `tab`) dont le premier élément est négatif ou nul et le dernier est strictement positif.

On souhaite écrire la fonction `dernier_negatif` qui détermine l'indice `i` de l'élément négatif ou nul du tableau qui est suivi par un élément strictement positif (c'est donc le dernier négatif ou nul).
1) On suppose fournie la fonction utilitaire `est_croissant` qui accepte en paramètre un tableau `tab` et retourne le booléen indiquant si le tableau est en ordre croissant *(à savoir faire)*.
a) Écrire une pré-condition pertinente de la fonction `dernier_negatif`
b) Écrire la post-condition de cette fonction.
2) On suppose dans cette question que modèle de solution est un parcours gauche-droite. L'invariant est alors : `0<=i<len(tab)-1 and tab[i]<=0`.
a) Déterminer la condition d'arrêt et la condition de boucle.
b) Écrire la fonction `dernier_negatif` dans ce cas.
c) Donner un variant pour cet algorithme.
3) On suppose dorénavant que modèle de solution est algorithme de dichotomie. L'invariant est alors : `0<=i<j<len(tab) and tab[i]<=0 and tab[j]>0`.
a) Déterminer la condition d'arrêt et la condition de boucle.
b) Écrire la fonction `dernier_negatif` dans ce cas.
c) Donner un variant pour cet algorithme.
4) **(pour aller plus loin)** Quels éléments dans la pré-condition permettent de :
- Garantir l'existence d'une solution ?
- Garantir l'unicité de la solution ?

Dans chacun des cas, prendre un exemple qui mette en défaut le critère (existence ou unicité) voulu.

Exercice 6

Dans cet exercice on caractérise des propriétés d'une tranche d'un tableau `tab` délimitée par deux indices `d` (inclus) et `f` (exclus). Les deux indices doivent caractériser une tranche valide du tableau, donc on considèrera que `d` et `f` vérifient `0<=d<=f<=len(tab)`.
1) Dans cette question on souhaite écrire des fonctions auxiliaires qui serviront à formuler des propriétés (pré- et post-condition, invariant, ...). On ne demande donc pas d'étudier les pré- et post-condition de ces fonctions.
a) Écrire la fonction auxiliaire `est_tranche_centree(tab,d,f)` qui accepte en paramètres :
- Un tableau `tab`.
- Deux entiers `d` et `f` délimitant une tranche valide.

et retourne un booléen indiquant si la tranche est centrée (autant d'éléments sur sa gauche que sur sa droite).
b) Écrire la fonction auxiliaire `est_tranche_palindrome(tab,d,f)` qui accepte les mêmes paramètres qu'à la question **a** et retourne un booléen indiquant si la tranche est un palindrome.
2) Dans cette question on souhaite écrire la fonction `palindrome_centre` qui accepte en paramètre un tableau `tab` et retourne le tuples d'entiers `(d,f)` délimitant le plus grand palindrome centré dans le tableau `tab`.
On donne les éléments suivants :
- Post-condition :

```
0<=d<=f<=len(tab)
and est_tranche_centree(tab,d,f)
and est_tranche_palindrome(tab,d,f)
and (d==0 or tab[d-1]!=tab[f])
```

- Invariant :

```
0<=d<=f<=len(tab)
and est_tranche_centree(tab,d,f)
and est_tranche_palindrome(tab,d,f)
```

a) Donner une pré-condition pertinente.
b) À l'aide des éléments donnés, déterminer la condition d'arrêt et la condition de boucle de cet algorithme.
c) Écrire la fonction `palindrome_centre`

Exercice 7

Le problème du drapeau Hollandais a été proposé par Edsger Wybe Dijkstra (1930-2002).
Le drapeau hollandais est un drapeau à trois bandes horizontales rouge, blanc, bleu, couleurs qui seront représentées par les caractères `'R'` (rood), `'W'` (wit) et `'B'` (blauw). Le problème est de trier, dans l'ordre `'R'`, `'W'` et `'B'` un tableau de caractères ne contenant que ces trois caractères dans le désordre. La solution proposée ne doit parcourir le tableau qu'une fois, fonctionne de manière assez similaire à la méthode du pivot.
L'objectif de l'exercice est d'écrire la fonction `drapeau_hollandais` **qui accepte en paramètre le tableau et modifie ce tableau en conservant ses éléments afin de constituer le drapeau hollandais.**
1) a) Déterminer le résultat final dans les cas suivants :

W	R	B	R	B	W	W	W	B

B	W	B	W	B	B	B	W

b) Écrire la **pré-condition** de la fonction `drapeau_hollandais`.
On pourra introduire la fonction auxiliaire `contient_RWB`.
2) a) Écrire la fonction auxiliaire `est_monochrome` qui accepte en paramètres :
- Un tableau de caractères `tab`
- Deux entiers `d` et `f` vérifiant `0<=d<=f<=len(tab)`
- Un caractère `car`

et retourne le booléen indiquant si la tranche tab[d :f] contient bien uniquement le caractère `car`.
b) On décide d'écrire la **post-condition** de la fonction `drapeau_hollandais` en utilisant deux indices `i` et `j` :

indice					i			j		
caractère	R	R	...	R	W	...	W	B	...	B

En déduire une formulation de cette post-condition.
c) On propose d'écrire l'**invariant** en utilisant trois indices `i`, `j` et `k` :

indice					i			j				k		
caractère	R	R	...	R	W	...	W	?	?	?	?	B	...	B

(les ? désignent les éléments à traiter)
En déduire une formulation de cet invariant.
d) En déduire la condition d'arrêt et la condition de boucle.
3) On propose le **modèle de solution** suivant :
À chaque étape de la boucle, on traite l'élément d'indice `j` :
- S'il est **rouge**, on le permute avec celui d'indice `i`, les indices `i` et `j` sont actualisés.
- S'il est **blanc**, aucune permutation n'est effectuée, et `j` est actualisé.

— S'il est **bleu**, on permute avec celui d'indice `k-1`, et `k` est actualisé.

Préciser l'état du tableau et des variables `i`, `j` et `k` une étape après les configurations suivantes :

a)

indice			i	j					k
caractère	R	R	W	R	B	W	W	W	B

b)

indice			i		j			k	
caractère	R	R	W	W	W	R	B	B	B

c)

indice		i		j				k	
caractère	R	W	W	B	B	W	R	B	B

4) Écrire la fonction `drapeau_hollandais` (avec les assertions) qui prend en entrée le tableau `tab` et applique le modèle de solution ci-dessus.

Contrainte : Ne modifier le tableau qu'en utilisant une fonction de permutation pour garantir que les éléments du tableau sont conservés.

5) Donner un variant de cet algorithme.