TP 7 - VARIANT - INVARIANT

Info1.Algo1 - 2022-2023 Semestre Impair

Variant

Rappels

On appelle **variant d'une boucle** toute variable ou expression :

- entière.
- positive.
- strictement décroissante à chaque itération.

La suite formée par les valeurs successives d'un variant est nécessairement finie, donc :

> **Si une boucle possède un variant, alors cette boucle se termine.**

La vérification expérimentale peut s'effectuer de la façon suivante :

```
while ...:
  v_debut = ... # <expression du variant>
  assert v_debut>=0, 'Variant (positif)'
  # Corps de la boucle
  ...
  v_fin = ... # <expression du variant>
  assert v_fin<v_debut, 'Variant (decroissant)'
```

Exercice 1 - Détecter une boucle infinie ★

Dans le fichier **ex01_somme_chiffres.py** est fournie la fonction `somme_chiffres` qui accepte en paramètre un entier positif ou nul et retourne la somme des chiffres qui le compose.

Comme vous allez le constater, cette fonction a un problème: elle ne termine pas.

1) Compléter les assertions relatives au variant pour confirmer le problème.

2) Corriger le code pour que la fonction `somme_chiffres` retourne la somme des chiffres de son paramètre.

Exercice 2 - Somme (déterminer un variant) ★

Dans le fichier **ex02_somme.py** est fournie la fonction `somme` qui accepte en paramètre un tableau d'entiers et retourne la somme `s` des valeurs de ce tableau.

On souhaite vérifier expérimentalement la terminaison de la boucle mise en oeuvre.

1) Déterminer un variant pour cette boucle.

2) Compléter les lignes `v_debut = ...` et `v_fin = ...` avec l'expression de ce variant (la même sur les deux lignes). Vérifier qu'il n'y a aucune erreur d'assertions lorsqu'on exécute la fonction de test.

Exercice 3 - Indice du maximum (instrumentation du variant) ★

Dans le fichier **ex03_indice_maximum.py** sont fournies :

- la fonction auxiliaire `est_majorant` qui accepte en paramètres un tableau d'entiers `tab` ainsi qu'un entier `m` et retourne `True` si `m` est un majorant des valeurs du tableau, `False` sinon.
- la `indice_maximum` qui accepte en paramètre un tableau d'entiers non vide `tab` et retourne l'indice `i` du maximum.

On souhaite vérifier expérimentalement la terminaison de l'algorithme quelque peu original mis en oeuvre ici, et qui consiste à un parcours à deux indices `i` et `j` sur le tableau `tab`.

1) Déterminer un variant pour la boucle de la fonction `indice_maximum`.

2) Ajouter les 4 lignes de codes permettant la vérification expérimentale de la terminaison. Vérifier qu'il n'y a aucune erreur d'assertions lorsqu'on exécute la fonction de test.

Exercice 4 - Le baladeur Zune ★

Le baladeur mp3 ***Zune 30*** *de chez Microsoft a été mis sur le marché en novembre 2006. Pendant les premiers mois de son utilisation, aucun problème notable n'est survenu. Soudain, le 31 décembre 2008, tous les baladeurs Zune 30 ont été paralysés pendant 24 heures.*

1) Le fichier **ex04__annee__jour__q1.py** fournit la fonction `calculer_annee_jour` qui représente la traduction mot-à-mot d'un code en langage C extrait du pilote de ces baladeurs.

Cette fonction accepte en paramètre un entier positif `jours` qui représente le nombre de jours écoulés depuis début 1980 et retourne les entiers `annee` (l'année courante), et `jours` (le nombre de jours écoulés depuis le début de cette année).

a) Que semble-t-il se passer lorsque l'on teste la fonction sur l'entier `10593` correspondant à la date du 31 décembre 2008?

b) On choisit la variable `jours` comme variant de la boucle de la fonction `calculer_annee_jour`. Ajouter les 4 lignes de codes permettant la vérification expérimentale de la terminaison.

c) Interpréter l'erreur d'assertion obtenue.

2) L'objectif de cette question est de réécrire dans le fichier **ex04__annee__jour__q2.py** la fonction `calculer_annee_jour` afin qu'elle termine dans tous les cas.

a) Écrire la fonction auxiliaire `nb_jours_annee` qui accepte en paramètre l'entier `annee` et retourne le nombre de jours de cette année.

b) Écrire la fonction `calculer_annee_jour`, y compris les 4 lignes de codes permettant la vérification expérimentale de la terminaison. Le variant devra être la variable `jours`.

Invariant et algorithmes de réarrangement

Introduction

Dans les exercices de cette partie, on s'intéresse à des algorithmes dont l'objectif est de **réarranger les valeurs d'un tableau selon un critère de comparaison** *(valeurs négatives puis positives, inférieures à une valeur de référence, ...)*.

- Le premier de cette série d'exercices vise à écrire les fonctions auxiliaires qui seront utilisées.
- Les suivants ont pour objectif l'écriture des algorithmes proprement dits.

Si vous maîtrisez les mécanismes d'import de module, vous pourrez importer les fonctions auxiliaires grâce à l'instruction :

```
from ex05_fonctions_auxiliaires import *
```

Sinon, il suffira de recopier les fonctions auxiliaires dans les fichiers où elles sont nécessaires.

Exercice 5 - Fonctions auxiliaires ★

1) On souhaite s'assurer que, lorsque l'on réarrange les éléments du tableau, aucun élément ne disparaît ou n'apparaît dans le tableau.

Dans le fichier **ex05_fonctions_auxiliaires.py**, écrire la fonction `permuter` qui accepte en paramètre un tableau d'entiers `tab` et deux indices `i` et `j` valides. L'**effet** de cette fonction est de permuter les éléments d'indice `i` et `j` de `tab`.

> **Dans les exercices suivants, toute modification sur les tableau devra donc s'effectuer en utilisant la fontion `permuter`.**

2) Afin d'exprimer les post-conditions et les invariants des exercices suivants, on souhaite tester que la valeur de tout élément d'un tableau `tab` dont l'indice est compris entre deux indices `g` (inclus) et `d` (exclu) est inférieure (ou supérieure) à une valeur `v` donnée (au sens large `<= >=` ou au sens strict `< >`)

Écrire les fonctions auxiliaires `est_infegal`, `est_inf`, `est_supegal`, `est_sup` et `est_egal` qui acceptent en paramètres le tableau `tab`, deux indices `g` et `d` et une valeur `v`, et retournent le booléen indiquant si la propriété est vérifiée pour tous les éléments d'indice `i` tel que `g<=i<d`.

Ainsi :

Le booléen ...	... vaut `True` si et seulement si tous les éléments d'indice `i` tel que `g<=i<d` sont
`est_infegal(tab,g,d,v)`	inférieurs ou égaux à `v`
`est_inf(tab,g,d,v)`	strictement inférieurs à `v`
`est_supegal(tab,g,d,v)`	supérieurs ou égaux à `v`
`est_sup(tab,g,d,v)`	strictement supérieurs à `v`
`est_egal(tab,g,d,v)`	égaux à `v`

> **Dans les exercices suivants, ces fonctions seront utilisées uniquement à des fins de vérification de la post-condition et de l'invariant et ne devront donc pas être utilisées dans l'algorithme proprement dit.**

Exercice 6 - Négatifs et positifs ★★

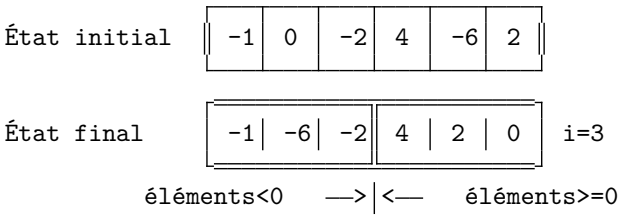
1) Dans cette première question, on souhaite compléter dans le fichier **ex06_negatifs_positifs.py** la fonction `negatifs_positifs` qui accepte en paramètre un tableau d'entiers `tab` et retourne un indice `i` et le tableau `tab` (modifié uniquement à l'aide la fonction `permuter`), tels que les éléments de `tab` à gauche de `i` (exclu) soient tous **strictement négatifs**, et ceux à droite de `i` (inclus) soient tous **positifs ou nuls**.

Spécification :

- **Pré-condition :** *aucune*
- **Post-condition** :

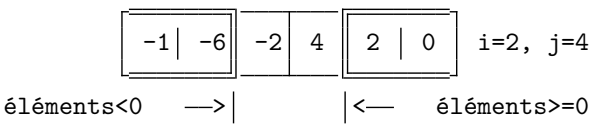
```
0<=i<=n and est_inf(tab,0,i,0) and est_supegal(tab,i,n,0)
```

Exemple



Invariant

L'invariant proposé nécessite deux indices `i` et `j` tels que tous les éléments à gauche de `i` (exclu) sont tous **strictement négatifs**, et ceux à droite de `j` (inclus) sont tous **positifs ou nuls**.



Autrement dit, l'invariant est :

```
0<=i<=j<=n and est_inf(tab,0,i,0) and est_supegal(tab,j,n,0)
```

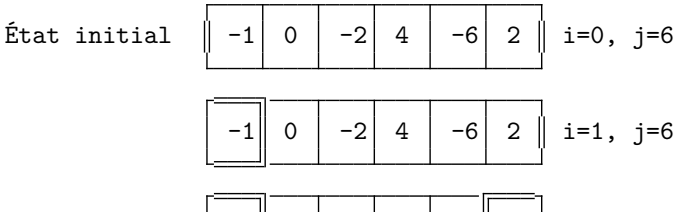
Modèle de solution

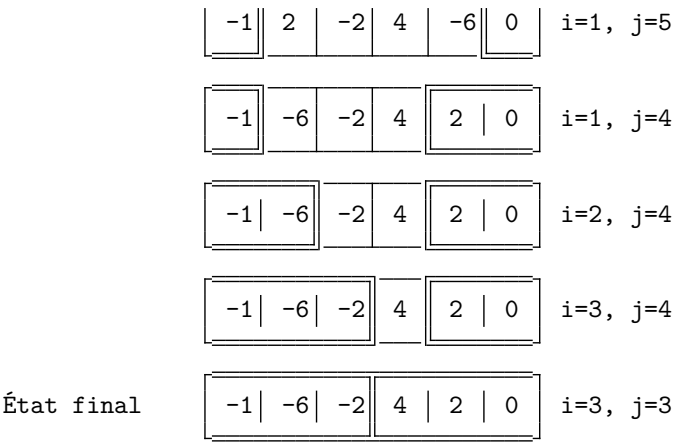
À chaque itération de la boucle, on traite l'élément d'indice `i` :

- Si celui-ci est strictement négatif, on incrément `i`.
- Sinon on le permute avec l'élément d'indice `j-1` et on décrément `j`.

Exemple de déroulé

Les étapes ci-dessous correspondent au modèle de solution ci-dessus :





2) Dans cette seconde version, on réarrange le tableau selon trois critères au lieu de deux : strictement négatif, nul et strictement positif. On souhaite donc compléter la fonction `negatifs_nuls_positifs` qui accepte en paramètre un tableau d'entiers `tab` et retourne deux indices `i` et `j` et le tableau `tab` (modifié uniquement à l'aide la fonction `permuter`), tels que les éléments de `tab` à gauche de `i` (exclu) soient tous **strictement négatifs**, ceux compris entre `i` (inclus) et `j` (exclu) soient **nuls** et ceux à droite de `j` (inclus) soient tous **strictement positifs**

Spécification :

- **Pré-condition :** *aucune*
- **Post-condition** :

```
0<=i<=j<=n and est_inf(tab,0,i,0)
and est_egal(tab,i,j,0) and est_sup(tab,j,n,0)
```

Invariant

L'invariant proposé nécessite trois indices `i`,`j` et `k` tels que tous les éléments à gauche de `i` (exclu) sont tous **strictement négatifs**, ceux compris entre `i` (inclus) et `j` (exclu) sont **nuls** et ceux à droite de `k` (inclus) sont tous **strictement positifs**.

Autrement dit, l'invariant est :

```
0<=i<=j<=k<=n and est_inf(tab,0,i,0)
and est_egal(tab,i,j,0) and est_sup(tab,k,n,0)
```

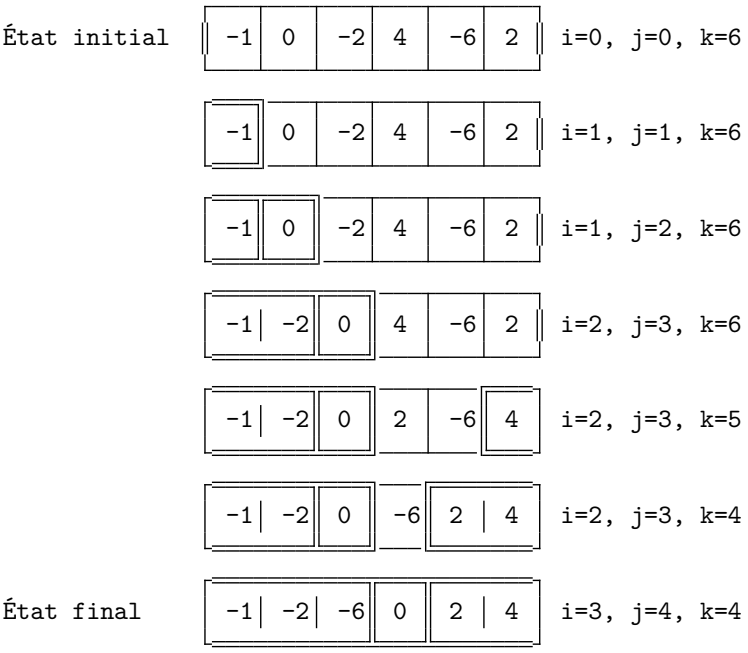
Modèle de solution

À chaque itération de la boucle, on traite l'élément d'indice `j` :

- S'il est strictement négatif, on le permute avec celui d'indice `i` et on incrémente `i` et `j`.
- S'il est nul, on incrémente `j`.
- Sinon on le permute avec l'élément d'indice `k-1` et on décrémente `k`.

Exemple de déroulé

Les étapes ci-dessous correspondent au modèle de solution ci-dessus :



Exercice 7 - Méthode du pivot ★★

La méthode du pivot est un algorithme qui sert de base à l'algorithme dit de **tri rapide** (en anglais : *quicksort*) qui sera vu ultérieurement. L'objectif de cet exercice est d'implémenter cet algorithme sous la forme d'une fonction `partitionner_pivot`.

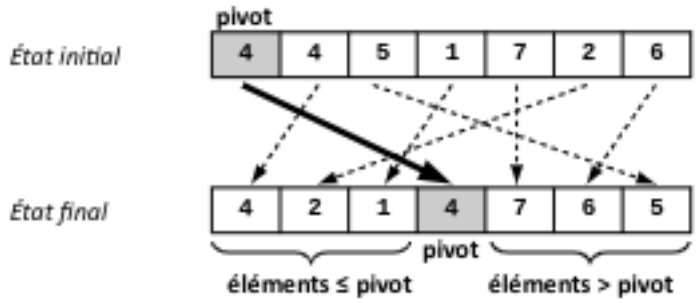
Principe général

Étant donné un **tableau non vide** d'entiers et un élément de ce tableau (appelé **pivot**), le principe général de la méthode du pivot est de déplacer par **permutations successives** les éléments du tableau, de telle sorte que :

- Tous les éléments du tableau qui sont **inférieurs ou égaux** au pivot se retrouvent à sa **gauche**.

- Tous les éléments du tableau qui lui sont **strictement supérieurs** se retrouvent à sa **droite**.

Ici, on choisit le premier élément du tableau comme pivot.



Spécification

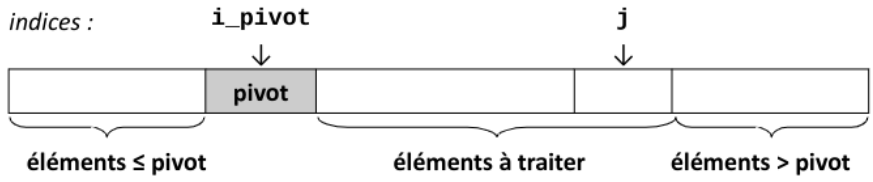
La fonction `partitionner_pivot` accepte en paramètre un tableau non vide d'entiers `tab` de longueur `n` et retourne l'entier `i_pivot` ainsi que le tableau `tab` (modifié uniquement à l'aide la fonction `permuter`)

- **Pré-condition :** `n>0`
- **Post-condition** :

```
0<=i_pivot<=j<n and tab[i_pivot]==pivot
and est_infegal(tab,0,i_pivot,pivot) and est_sup(tab,i_pivot+1,n,pivot)
```

Invariant

On peut décrire l'invariant de la méthode du pivot par le schéma suivant :



Autrement dit, l'invariant est :

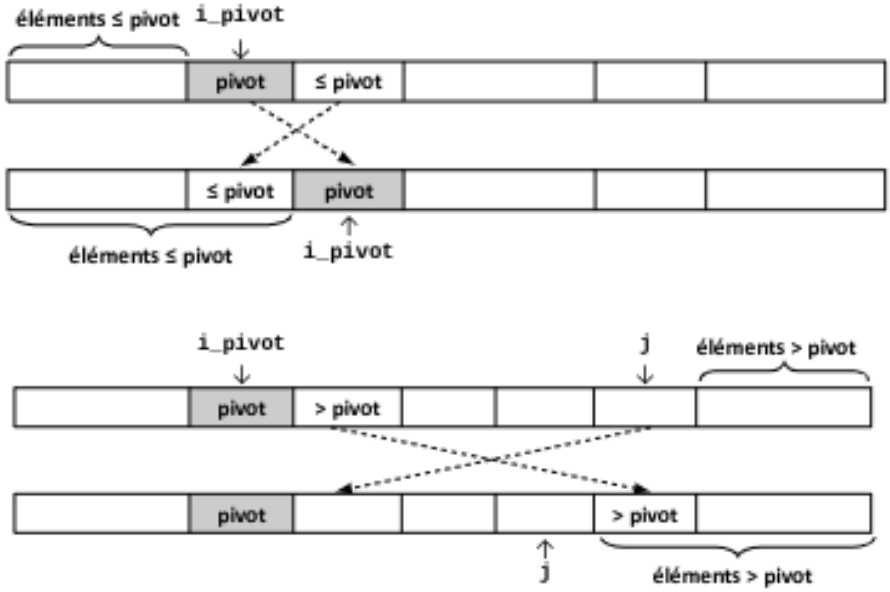
```
0<=i_pivot<=j<n and tab[i_pivot]==pivot
and est_infegal(tab,0,i_pivot,pivot) and est_sup(tab,j+1,n,pivot)
```

Modèle de solution

À chaque itération de la boucle, on traite l'élément à droite du pivot :

- Si celui-ci est inférieur ou égal au pivot, on le permute avec le pivot.
- Sinon on le permute avec l'élément d'indice `j`.

On actualise ensuite la valeur de `i_pivot` ou `j`.



Exemple de déroulé

État initial	**4**	4	5	1	7	2	**6**	i_pivot=0, j=6
	4	**4**	5	1	7	2	**6**	i_pivot=1, j=6
	4	**4**	6	1	7	**2**	5	i_pivot=1, j=5
	4	**4**	2	1	**7**	6	5	i_pivot=1, j=4
	4	2	**4**	1	**7**	6	5	i_pivot=2, j=4
	4	2	1	**4**	**7**	6	5	i_pivot=3, j=4
État final	4	2	1	**4**	7	6	5	i_pivot=3, j=3

Travail à effectuer

Écrire dans le fichier **ex07_pivot.py** la fonction `partitionner_pivot` selon la spécification et le modèle de solution donnés ci-dessus.