Introduction

On poursuit ici l'étude théorique des algorithmes entreprise dans le chapitre traitant de la complexité. On se pose maintenant la question de savoir si un algorithme donné répond bien au problème qu'il est censé traité dans sa spécification. *Il se pose alors deux grandes questions :*

1. *Se termine-t-il ? C'est la question de la* terminaison.
2. *Résout-il bien le problème qu'il est censé traiter ? C'est la question de la* correction.

Le but de ce chapitre est d'introduire les méthodologies qui permettent de traiter ces problèmes.

Source d'inspiration : cours de mon collègue Pierre Duclosson.

1 Terminaison

Objectif 1

Pour un algorithme ou une partie d'algorithme qui ne comporte pas de boucles ou seulement des boucles inconditionnelles, la question de la terminaison ne se pose, a priori, pas. Le cas des boucles conditionnelles est plus délicat : la condition est censée être vraie au départ (sinon c'est du code mort) et cette même condition doit finir par être fausse sinon les itérations ont lieu indéfiniment.

Exercice 1

Pour chacune des boucles déterminer si elle se termine ?

Boucle 1

```
x = 0
while x >= 0:
    x = x + 1
```

Boucle 2

```
x = 10
while x >= 0:
    x = x - 1
```

Boucle 3

```
x = 1
while x != 0:
    x = x - 0.1
```

Boucle 4

```
x = 0
while x > 1:
    x = x - 0.1
```

Point de cours 1 *Variant de boucle et terminaison d'algorithme*

- On appelle **itération** d'une boucle **une** exécution des instructions qui figure dans le corps de la boucle.
- Une boucle inconditionnelle `for` se termine nécessairement.
- Pour démontrer qu'une boucle conditionnelle (`while`) se termine, il suffit de déterminer une grandeur exprimée à l'aide des variables de l'algorithme qui vérifie les trois conditions suivantes :
  - Condition 1 : cette grandeur a une valeur entière avant la boucle ;
  - Condition 2 : une itération de boucle ne s'exécute que si la grandeur est positive ;
  - Condition 3 : chaque exécution d'une itération de boucle fait décroître strictement la grandeur et la maintient dans l'ensemble des entiers.

  Comme il n'existe pas de suite infinie à valeurs dans l'ensemble des entiers positifs qui soit strictement décroissante (pas de *descente infinie* dans l'ensemble des entiers naturels) cela prouve alors que la grandeur ne peut prendre qu'un nombre fini de valeurs positives et que le nombre d'itérations est fini.
- On appelle **variant** de la boucle une telle quantité.

Exercice 2

La fonction `division_euclidienne` ci-dessous calcule le quotient et le reste de la division euclidienne de a par b (avec $a \geqslant 0$ et $b > 0$).

```
def division_euclidienne(a, b):
    """Renvoie le quotient et le reste de la division euclidienne de a par
        b ."""
    assert (a >= 0) and (b > 0)
    q = 0
    r = a
    while r >= b :
        r = r - b
        q = q + 1
    return (q, r)
```

On considère un index k tel que $k = 0$ désigne l'état du programme avant l'exécution de la boucle et $k \geqslant 1$ est l'état après l'itération k de la boucle.
On définit la quantité $v_k = r_k - b_k$ où r_k est l'état de la variables r à l'étape k du programme.
Démontrer que v_k est un variant de la boucle de `division_euclidienne` et conclure sur la terminaison de cet algorithme.

.....

.....

.....

.....

.....

.....

.....

Exercice 3

On considère l'algorithme implémenté par la fonction `boucle` ci-dessous.
À l'aide d'un variant de boucle, démontrer que l'algorithme se termine
Et si on remplace l'opérateur de comparaison < par `!=` ?

```
def boucle():
    x = 0
    while x < 1:
        x = x + 0.1
    return
```

.....

.....

.....

.....

.....

.....

.....

2 Correction

Objectif 2

La terminaison d'un algorithme est une condition nécessaire mais pas suffisante. On souhaite s'assurer que lorsque l'algorithme se termine, le traitement effectué soit correctement réalisé.

Point de cours 2 *Invariant de boucle et correction d'algorithme*

Pour démontrer la **correction** d'un algorithme, les difficultés se posent dans les boucles (quel qu'en soit le type, conditionnelles ou inconditionnelles).

- ☞ Avant d'analyser la **correction** d'un algorithme, on démontre sa **terminaison** à l'aide d'un variant.
- ☞ Ensuite on associe à chaque itération i de boucle un **invariant**. C'est une propriété $\mathscr{P}_i$, évaluée en fin de l'itération i de boucle, qui doit vérifier deux caractéristiques :
  - **Initialisation :** $\mathscr{P}_0$ est vraie avant la première itération de boucle.

- **Transmission :** si $\mathscr{P}_i$ est vraie en fin d'itération i et donc avant l'itération $i+1$ de boucle et que l'itération $i+1$ de boucle s'exécute alors $\mathscr{P}_{i+1}$ est vraie.

☞ Supposons que la dernière itération de boucle ait pour indice $k+1$, la correction s'obtient au terme d'une chaîne d'implications logiques :

- $\mathscr{P}_0$ est vraie par *initialisation*;
- $\mathscr{P}_0$ vraie donc $\mathscr{P}_1$ vraie par *transmission*;
- ...
- $\mathscr{P}_i$ vraie donc $\mathscr{P}_{i+1}$ vraie par *transmission*;
- ...
- $\mathscr{P}_k$ vraie donc $\mathscr{P}_{k+1}$ vraie par *transmission.*

On en déduit que $\mathscr{P}_{k+1}$ est vraie.

Si on a choisi judicieusement l'invariant, l'expression de $\mathscr{P}_{k+1}$ doit prouver la **correction** de l'algorithme.

Une preuve d'invariant est similaire à une preuve par récurrence en mathématiques.

Exercice 4

On considère la propriété $\mathscr{P}_k$: « *La valeur de* `p` *en fin d'itération* k *et avant l'itération* $k+1$ *de la boucle est* x^k ».
Démontrer que $\mathscr{P}_k$ est un invariant de la boucle de la fonction `puissance(x, n)` spécifiée ci-dessous.
En déduire que `puissance(x, n)` renvoie bien x^n et que l'algorithme est correct.

```
def puissance(x, n):
    """Renvoie x ** n, où x est un flottant et n un entier."""
    assert n >= 0
    p = 1
    for k in range(1, n + 1):
        p = p * x
    return p
```

.....

.....

.....

.....

.....

.....

Exercice 5

Pour la fonction `division_euclidienne` de l'exercice 2, on considère un index k tel que $k = 0$ désigne l'état du programme avant l'exécution de la boucle et $k \geqslant 1$ est l'état après l'itération k de la boucle. On définit la propriété $\mathscr{P}_k$: « $a_k = q_k \times b + r_k$ » où a_k, q_k et r_k sont les états des variables a, q et r à l'étape k du programme.
Démontrer que $\mathscr{P}_k$ est un invariant de la boucle de `division_euclidienne`. En déduire que cet algorithme est correct.

.....

.....

.....

.....

.....

.....

.....

3 Retour sur le tri par sélection.

Les algorithmes du programme : recherche linéaire de maximum dans un tableau, recherche dichotomique dans un tableau trié, tris par sélection ou insertion, se terminent et son corrects. On va se limiter au cas du tri par sélection. Voir https://fjunier.forge.apps.education.fr/tnsi/Bac/C0_Algorithmes_de_r%C3%A9f%C3%A9rence/C0_Algorithmes_de_r%C3%A9f%C3%A9rence/ pour les algorithmes de référence en NSI.

Exercice 6 *Tri par sélection*

On suppose qu'on dispose de deux fonctions dont la terminaison et la correction sont prouvées :

- `recherche_index_min(t, i)` renvoie un index du minimum d'un tableau d'entiers `t` à partir de l'index `i < len(t)`.
- `echange(t, i, imin)` permute les éléments d'index `i` et `imin` dans un tableau d'entiers `t`.

La fonction `tri_selection(t)` trie en place par sélection un tableau d'entiers `t`.

```
def tri_selection(t):
    """Trie en place par sélection un tableau d'entiers."""
    n = len(t)
    for i in range(0, n):
        imin = recherche_index_min(t, i)
        echange(t, i, imin)
```

1. Justifier la terminaison de l'algorithme implémenté par `tri_selection(t)`.

.....

.....

2. Pour tout indice `0 <= i < len(t)` on définit la propriété $\mathscr{P}_i$ vérifiée avant chaque itération d'indice i de la boucle. $\mathscr{P}_0$ désigne un état avant l'entrée dans la boucle.

Ici on a un décalage entre l'indice de boucle et celui de la propriété. A la fin de première itération (pour l'indice de boucle `i = 0`, la propriété $\mathscr{P}_{0+1}$ est vraie.

$\mathscr{P}_i$:= *le sous-tableau `t[0:i]` est trié dans l'ordre croissant et si `t[0:i]` est non vide et `t[i:]` non vide alors `t[i-1]` est inférieur ou égal à tous les éléments de `t[i:]`.*

Démontrons que $\mathscr{P}_i$ est un invariant de boucle.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

La boucle se termine au tour d'indice `i = len(t) - 1`, donc $\mathscr{P}_{i+1} = \mathscr{P}_{\texttt{len(t)}}$ est vrai ce qui se traduit par `t[0:len(t)]` est trié dans l'ordre croissant, ce qui prouve la correction de l'algorithme.