

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 traiter 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 : 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

Laquelle de ces deux boucles ne se termine pas ?

Boucle 1

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

Boucle 2

```
x = 10
while x >= 0:
    x = x - 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 quantité exprimée à l'aide des variables de l'algorithme qui **reste positive et entière** tout au long de l'exécution de la boucle et **décroit strictement** à chaque itération. Comme il n'existe pas de suite infinie à valeurs dans l'ensemble des entiers naturels qui soit strictement décroissante cela prouve alors que le nombre d'itérations est fini.
- ☞ On appelle **variant** de la boucle une telle quantité.



Exercice 2

Démontrer que l'algorithme de division euclidienne dans \mathbb{N} se termine (avec $a \geq 0$ 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)
```

.....

.....

.....

.....

.....

.....

.....

.....



Exercice 3

Démontrer que l'algorithme de recherche séquentielle d'un élément e dans un tableau `tab` se termine :

```
def appartient(e, tab):  
    """Renvoie la valeur de 'e in tab' ."""  
    n = len(tab)
```

```
i = 0
while i < n and tab[i] != e:
    i = i + 1
return i != n
```

.....

.....

.....

.....

.....

.....

.....

.....

Exercice 4

Soit n un entier positif. Le nombre m de chiffres de n en base deux est le plus petit entier m tel que $2^m \geq n$.

1. Compléter la fonction `nombre_chiffres_binaire(n)` qui prend en paramètre un entier positif n et qui renvoie son nombre de chiffres en base deux.

```
def nombre_chiffres_binaire(n):
    """Renvoie le nombre de chiffres de n en base 2."""
    m = 0
    .....
    .....
    .....
    .....
    .....
```

2. Démontrer que si $n \geq 0$ alors `nombre_chiffres_binaire(n)` se termine.

.....

.....

.....

.....

.....

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 invariant.
- ☞ Ensuite on associe à chaque itération i de boucle un **invariant**. C'est une propriété \mathcal{P}_i , évaluée en fin de l'itération i de boucle, qui doit vérifier deux caractéristiques :
 - **Initialisation** : \mathcal{P}_0 est vraie avant la première itération de boucle.
 - **Transmission** : si \mathcal{P}_{i-1} est vraie avant l'itération i de boucle et que l'itération i de boucle s'exécute alors \mathcal{P}_i est vraie.
- ☞ Supposons que la dernière itération de boucle ait pour indice k , la correction s'obtient au terme d'une chaîne d'implications logiques :
 - \mathcal{P}_0 est vraie par *initialisation*;
 - \mathcal{P}_0 vraie donc \mathcal{P}_1 vraie par *transmission*;
 - ...
 - \mathcal{P}_{i-1} vraie donc \mathcal{P}_i vraie par *transmission*;
 - ...
 - \mathcal{P}_{k-1} vraie donc \mathcal{P}_k vraie par *transmission*.

On en déduit que \mathcal{P}_k est vraie.

Si on a choisi judicieusement l'invariant, l'expression de \mathcal{P}_k doit prouver la **correction** de l'algorithme.

Exercice 5

Montrer que la fonction puissance(x , n) où x est un flottant et n un entier positif renvoie bien x^n .

```
def puissance(x, n):  
    """Renvoie x ** n ."""  
    assert n >= 0  
    p = 1  
    for k in range(n):  
        p = p * x  
    return p
```

.....

.....

.....

.....

.....

.....

Exercice 6

Démontrer que l'algorithme de division euclidienne implémenté par la fonction `division_euclidienne(a, b)` de l'exercice 2 est correct.

.....

.....

.....

.....

.....

.....

.....

Exercice 7

1. Écrire une fonction `recherche_maximum(tab)` qui prend en paramètre un tableau d'entiers `tab` et renvoie le maximum de `tab`.

.....

.....

.....

.....

.....

2. Démontrer que l'algorithme implémenté par cette fonction est correct.

.....

.....

.....

.....

.....

3 Retour sur la recherche dichotomique et le tri par sélection.

Exercice 8 Recherche dichotomique

La fonction `recherche_dicho(val, tab)` détermine si l'entier `val` appartient au tableau d'entiers `tab` trié dans l'ordre croissant, par recherche dichotomique.

```
def recherche_dicho(val, tab):  
    """  
    Renvoie la valeur de 'val in tab'.  
    Précondition tab dans l'ordre croissant.  
    """  
    g = 0  
    d = len(tab) - 1  
    trouve = False  
    while g <= d:  
        m = (g + d) // 2  
        if tab[m] == val:  
            trouve = True  
            g = d + 1 # sortie de boucle  
        elif tab[m] > val: # on continue la recherche dans tab[g:m]  
            d = m - 1  
        else: # on continue la recherche dans tab[m + 1:d]  
            g = m + 1  
    return trouve
```

On définit les quantités suivantes :

- $g_0 = 0$ est la valeur de la variable g avant la boucle, $d_0 = 0$ est la valeur de la variable d avant la boucle et $m_0 = (g_0 + d_0) // 2$.
- à la fin l'itération $k \geq 1$ (et donc avant l'itération $k + 1$) de la boucle, on note g_k la valeur de la variable g , d_k celle de la variable d et $m_k = (g_k + d_k) // 2$.

1. Démontrer que la quantité $d_k - g_k$ est un variant de boucle.

.....

.....

.....

.....

.....

.....

.....

On a déterminé un variant de boucle donc la terminaison de l'algorithme est prouvée.


```
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 $1 \leq k < \text{len}(t)$ on définit la propriété vérifiée à la fin de l'itération d'indice k de la boucle. \mathcal{P}_0 désigne un état avant l'entrée dans la boucle.



Dans le code, les indices de boucle commencent à 0 et on aura pour $k \geq 1, k = i + 1$.

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

Démontrons que \mathcal{P}_k est un invariant de boucle.

.....
.....
.....
.....
.....
.....
.....

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