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 :* 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

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

Boucle 2

```python
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 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 naturels qui soit strictement décroissante 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

Démontrer que l'algorithme de division euclidienne dans $\mathbb{N}$ se termine (avec $a \geqslant 0$ et $b > 0$).
On exprimera un variant à l'aide des variables `r`, `a`, `b` ou `q`.

```python
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

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 `!=` ?

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

....

....

....

....

....

....

....

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 $2^m \geqslant n$.

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.

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

2. Démontrer que si `n >= 0` alors `nombre_chiffres_binaire(n)` se termine. On déterminera un variant de boucle.

   ....

. . . .

. . . .

. . . .

. . . .

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.

Exercice 5

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.

```python
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(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` non vide 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, t)` détermine si l'entier `val` appartient au tableau d'entiers `t` trié dans l'ordre croissant, par recherche dichotomique.

```python
def recherche_dicho(val, t):
    """Renvoie si 'val in t' Précondition tab dans l'ordre croissant."""
    g, d = 0, len(t) - 1
    # Propriété P(k) vraie en fin d'itération k et avant l'itération k + 1
    # (val not in t) ou ((val in t) et (il existe g<=i<=d avec t[i]==val))
    # Initialisation : P(0) vraie
    while g <= d:
        # P(k) vraie en fin d'itération k et avant l'itération k + 1
        m = (g + d) // 2
        if t[m] == val:
            return True
        elif val < t[m]: # on continue la recherche dans t[g:m]
            d = m - 1
        else:             # on continue la recherche dans t[m + 1:d]
            g = m + 1
        # Préservation : P(k+1) vraie à la fin de l'itération k + 1
    return False
```

La boucle peut se terminer avec une sortie prématurée, supposons que celle-ci ne se produise jamais, montrons que la boucle se termine alors à l'aide d'un variant de boucle.
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 $k \geqslant 1$ (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.

2. Démontrons que la propriété $\mathscr{P}_k$ définie en commentaire dans le code est un invariant de boucle.

....

....

....

....

....

....

....

....

....

....

Conclure sur la correction de l'algorithme. On raisonnera par disjonction des cas :

- Premier cas : Si `trouve` vaut `True` en sortie de boucle alors `val` est dans `t` ;
- Second cas : Si `trouve` vaut `False` en sortie de boucle alors `val` n'est pas dans `t`.

....

....

....

....

Exercice 9 *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'entierss `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):
        kmin = 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 vérifiée avant chaque l'itération d'indice i de la boucle. $\mathscr{P}_0$ désigne un état avant l'entrée dans la boucle.

$\mathscr{P}_i$:= le sous-tableau `t[0:i]` est trié dans l'ordre croissant et si `t[0:i]` est non vide et `t[i:]` non vides 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.