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

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

Boucle 2

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

• Boucle 1 : x vaut initialement 0 et augmente de 1 à chaque itération donc la condition x >= 0 est toujours vraie et la boucle 1 ne se termine pas

• Boucle 2 : On trace l'évolution de la variable x en sortie de boucle

itération	1	2	3	4	5	6	7	8	9	10	11
valeur de x	9	8	7	6	5	4	3	2	1	0	-1

A la fin de la 11ème itération, x vaut -1 donc la condition x >= 0 n'est plus vérifiée et la boucle se termine avant la 12ème itération

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.

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`.

```
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)
```

.....

.....

.....

.....

.....

.....

.....

Montrons que la grandeur $V = r - b$ variant de la boucle :

Condition 1 : avant la boucle $V = r - a$ est un entier

Condition 2 : une itération de boucle s'exécute ssi $r \geqslant b$ c'est-à-dire $V = r - b \geqslant 0$

Condition 3 : On note V_k la valeur de V avant l'exécution de l'itération k. A la fin de l'itération, on a $V_{k+1} = V_k - b$

donc V_{k+1} est un entier

et $V_{k+1} < V_k$

Conclusion : V vérifie les 3 conditions caractéristiques d'un variant donc V est un variant de boucle et la boucle se termine.

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 != ?

Tester l'égalité de 2 flottants n'est pas une bonne idée, la boucle ne va pas se terminer.

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

Montrons que $V = 10 - 10x$ est un variant de boucle :
Condition 1 : Avant la boucle $V = 10 - 10 \times 0 = 10$
Condition 2 : Une itération de boucle s'exécute ssi
$1 - x > 0 \iff 10 - 10x > 0 \iff V > 0$
Condition 3 : Si on note V_k la valeur au début de l'exécution de l'itération k, on a à la fin de l'itération :
$V_{k+1} = V_k - 1$ est un entier et $V_{k+1} < V_k$
Conclusion : V vérifie les 3 conditions caractéristiques d'un variant de la boucle, c'est un variant et la boucle se termine

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 $2^m > n$.
↳ inégalité stricte

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
    p = 1
    while p <= n :
        p = p * 2
        m = m + 1
    return m
```

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

Montrons que $V = n - p$ est un variant de boucle
Condition 1 : avant la boucle $V = n - 1$ est un entier

Condition 2 : 1 itération s'exécute ssi : $p \leq n \Leftrightarrow n - p \geq 0 \Leftrightarrow V \geq 0$

Condition 3 : Si on note V_k la valeur de V au début de l'itération k on a $V_{k+1} = V_k - p$ donc V_{k+1} est un entier et $V_{k+1} < V_k$

Conclusion : V vérifie les 3 conditions caractéristiques d'un variant donc c'est un variant et la boucle 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 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.

```
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
```

Initialisation : avant la boucle $p = x^0$ donc $\mathscr{P}_0$ est vraie

Préservation : On suppose que $\mathscr{P}_k$ est vraie avant l'itération $k+1$, c'est-à-dire que $p = x^k$.
Au cours de cette itération p est multipliée par x, donc à la fin de l'itération $p = x^{k+1}$ et donc $\mathscr{P}_{k+1}$ est vraie

Conclusion : $\mathscr{P}_k$ est donc un invariant de boucle.
En sortie de boucle $\mathscr{P}_n$: "$p = x^n$" est vraie ce qui prouve la correction de l'algorithme

Exercice 6

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

On considère $\mathscr{P}_k$: "$a = qb + r$"
Initialisation : avant la boucle, on a : $q = 0$ et $r = a$ donc $a = qb + r$ donc $\mathscr{P}_0$ vraie

Préservation : On suppose que $\mathscr{P}_k$ est vraie avant l'itération $k+1$, c'est à dire que $a = q_k \times b + r_k$
A la fin de l'itération $k+1$, on a : $q_{k+1} = q_k + 1$
et $r_{k+1} = r_k - b$ donc $q_{k+1} \times b + r_{k+1} = q_k \times b + r_k = a$
donc $\mathscr{P}_{k+1}$ est vraie

Conclusion : $\mathscr{P}_k$ est un invariant de boucle et en sortie de boucle $a = qb + r$ avec $0 \leq r < b$ à cause de la terminaison

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`.

1) def recherche_maximum(tab):
    assert len(tab) > 0
    maxi = tab[0]
    for k in range(1, len(tab)):
        if tab[k] > maxi:
            maxi = tab[k]
    return maxi

2) Notons tab[0:k+1] le sous-tableau de tab contenant tous les éléments d'indice i compris entre 0 et k inclus.

$\mathcal{P}_k$: "maxi = max(tab[0:k+1])"

Montrons que $\mathcal{P}_k$ est un invariant de la boucle

Initialisation : avant la boucle
maxi = tab[0] = max(tab[0:0+1])
donc $\mathcal{P}_0$ est vraie.

Préservation : On suppose que $\mathcal{P}_k$ est vraie avant l'itération k+1, c'est-à-dire que :
maxi = max(tab[0:k+1])
A la fin de l'itération :
maxi = max(max(tab[0:k+1]), tab[k+1])
donc maxi = max(tab[0:k+2])
donc $\mathcal{P}_{k+1}$ est vraie

Conclusion : $\mathcal{P}_k$ est un invariant de la boucle (inconditionnelle donc se termine)

En sortie de boucle f(tab)-1 est vraie
donc max = max(tab[0:len(tab)])
donc max est le maximum du tableau 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 da

```
def recherche_dicho(v, t):
    """Renvoie si 'v in t' Précondition t 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
    # (si 0<=i<g alors t[i]<v) et (si d<i<=len(t)-1 alors v<t[i])
    # 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] == v:
            return True
        elif v < 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 is,
m
O la

- 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.

voir page suivante

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`.

1) La boucle peut se terminer avec une sortie prématurée Si celle-ci ne se produit jamais, montrons que d-g est un variant de boucle

Condition 1: Avant la boucle $d_0 - g_0 = \text{len(tab)} - 1$ est un entier

Condition 2: l'itération k+1 s'exécute ssi $g_k \leq d_k$ c'est-à-dire $d_k - g_k \geq 0$

Condition 3: A la fin de l'itération k+1 on a

1er cas: $g_{k+1} = m_{k+1} + 1$ et $d_{k+1} = d_k$

or $m_{k+1} = (g_k + d_k) // 2$ donc $g_k \leq m_{k+1} < d_k$

ainsi on a $g_k < m_{k+1} + 1 = g_{k+1}$.

et donc $\boxed{d_{k+1} - g_{k+1} = d_k - (m_{k+1} + 1) < d_k - g_k}$

2ème cas: $d_{k+1} = m_{k+1} - 1$ et $g_{k+1} = g_k$

$m_{k+1} < d_k$ donc $m_{k+1} - 1 < d_k$

et donc $\boxed{d_{k+1} - g_{k+1} = m_{k+1} - 1 - g_k < d_k - g_k}$

Petit oubli: dans les 2 cas $d_{k+1} - g_{k+1}$ reste un entier

Conclusion: d-g vérifie les 3 caractéristiques d'un variant donc c'est un variant de la boucle et la boucle se termine même s'il n'y a pas de sortie prématurée

2) Avant chaque itération $k+1$ de boucle on définit la propriété :

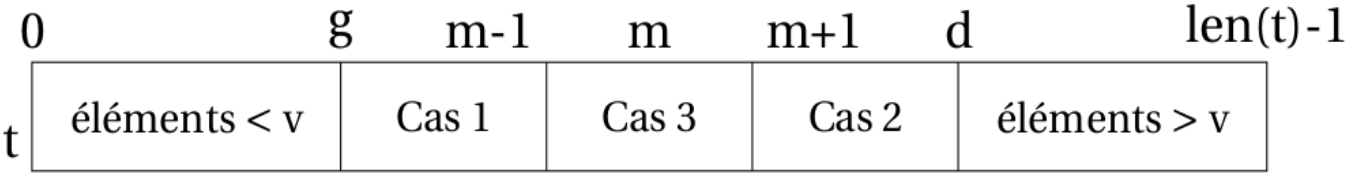
J_k = " (Si $0 \leq i < g_k$ alors $t[i] < v$)

ET (Si $d_k < i \leq \text{len}(t)-1$ alors $v < t[i]$)"

Autrement-dit les éléments du tableau t à gauche de la zone de recherche sont inférieurs à v et ceux à droite sont supérieurs à v.

Initialisation : J_0 est vraie car avant la boucle $g_0 = 0$ et $d_0 = \text{len}(t)-1$ donc la zone de recherche couvre tout le tableau.

Hérédité : Supposons que J_k est vraie et que l'itération $k+1$ se déroule.



Par hypothèse : • Si $0 \leq i < g_k$ alors $t[i] < v$
• Si $d_k < i \leq \text{len}(t)-1$ alors $v < t[i]$

· On calcule d'abord $m_{k+1} = (g_k + d_k) // 2$

On a donc $g_k \leq m_{k+1} < d_k$

Cas 1 : $v < t[m_{k+1}]$ $g_{k+1} = g_k$ et $d_{k+1} = m_{k+1} - 1$

• Si $0 \leq i < g_k = g_{k+1}$ alors $t[i] < v$ car J_k vraie

- Si $d_{k+1} < i \leq len(t) - 1$ alors $m_{k+1} - 1 < i \leq len(t) - 1$

Car t trié dans l'ordre croissant donc

$$t[m_{k+1}] \leq t[i]$$

et comme $v < t[m_{k+1}]$

on en déduit que $v < t[i]$

- Donc J_{k+1} est vraie dans ce cas

Cas 2 : $t[m_{k+1}] < v$ symétrique au cas 1

Cas 3 : $t[m_{k+1}] == v$

Sortie prématurée de la boucle et de la fonction les valeurs de g et d ne sont pas modifiées, donc J_{k+1} est vraie

Conclusion propriété : On a démontré que la J_k est un invariant de la boucle

Correction

1er cas : v n'est pas dans t

L'algorithme ne renvoie True que si on tombe sur une correspondance entre v et une valeur de t, donc il ne peut pas renvoyer True. De plus la boucle se termine et la valeur renvoyée en sortie de boucle est False. Donc l'algo est correct

2ème cas : v dans t

L'invariant de boucle garantit qu'à chaque itération, la zone de recherche contient effectivement égales à v. Comme la boucle se termine, on aura forcément une correspondance entre v

et une valeur de t, l'algo renvoie alors True, et il est correct.

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):
        imin = recherche_index_min(t, i)
        echange(t, i, imin)
```

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

la boucle externe for est bornée (ou inconditionnelle) donc elle se termine. La boucle interne de recherche_index_min est aussi bornée donc elle se termine. echange exécute au plus 3 instructions.

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.

Initialisation : $\mathscr{P}_0$ est vraie puisque la partie triée t[0:i] est vide

Préservation : Supposons $\mathscr{P}_i$ vraie

Par hypothèse t[0:i] trié et t[0:i] ≤ t[i:len(t)]

trié non trié

Après recherche_index_min et echange, en première position de la partie non triée on a son minimum qui est aussi ≤ à tous les éléments de la partie triée. On peut étendre la partie triée avec cet élément et à la fin de l'itération i+1 on aura t[0:i+1] trié et t[0:i+1] ≤ t[i+1:len(t)]

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.

et donc $\mathscr{P}_{i+1}$ vraie