
TD2 - Terminaison et correction d'un algorithme

1 Rappel de cours

1.1 Terminaison d'un algorithme

Pour montrer qu'un algorithme se termine (c'est-à-dire que l'exécution de l'algorithme sur n'importe quelle entrée s'arrête au bout d'un temps fini), il faut montrer qu'une condition de terminaison est obligatoirement vérifiée à un moment donné de l'exécution. Par exemple, qu'on va sortir de la boucle principale, ou bien qu'on ne fait pas une infinité d'appels récursifs.

Pour une boucle, on peut utiliser un *convergent* pour montrer que la suite des valeurs de la variable principale (celle du test de terminaison de la boucle) converge vers la valeur qui interrompt la boucle.

Pour un algorithme récursif, on peut montrer que la suite des valeurs des appels récursifs converge vers le cas de base de la fonction récursive.

1.2 Correction d'un algorithme : les invariants de boucle

Il s'agit de montrer que l'algorithme est correct, c'est à dire qu'il calcule bien toujours le résultat attendu.

Pour montrer qu'un algorithme utilisant une boucle principale est correct, on peut utiliser un *invariant de boucle* : en notant par j le numéro d'itération de la boucle, c'est une propriété $P(j)$ qui est vraie avant la boucle ($j = 0$) ou bien après le premier tour de boucle ($j = 1$), qui reste vraie après chaque itération de la boucle (on passe de $P(j)$ à $P(j + 1)$), et qui, pour la dernière itération, montre que le résultat calculé par l'algorithme est correct. On montre le passage de $P(j)$ à $P(j + 1)$ généralement par récurrence sur j .

Pour un algorithme récursif, on utilise naturellement une preuve par récurrence sur les appels récursifs.

2 Exercices

Exercice 1 (Terminaisons d'algorithmes itératifs).

Pour chacun des quatre algorithmes suivants, dites s'il se termine ou pas. Si oui, mettez en évidence un convergent et prouvez la terminaison. Si non, précisez les cas particuliers dans lesquels l'algorithme se termine.

Algo1(n)

Entrée : un entier n

Sortie : un entier

- Tant que $n > 0$ faire :
 - * Si n est pair alors :
 $n = 2n$
 - * Sinon :
 $n = n - 1$
- Retourner n

Algo2(n)

Entrée : un entier n

Sortie : un entier

- Tant que $n > 1$ faire :
 - * Si n est pair alors :
 $n = n/2$
 - * Sinon :
 $n = n - 1$
- Retourner n

Algo3(n)

Entrée : un entier n

Sortie : un entier

- Tant que $n \neq 11$ faire :
 - * Si n est pair alors :
 $n = n/2$
 - * Sinon :
 $n = n - 2$
- Retourner n

Algo4(n)

Entrée : un entier n

Sortie : un entier

- Tant que $n > 1$ faire :
 - * Si n est pair alors :
 $n = n/2$
 - * Sinon :
 $n = n + 1$
- Retourner n

Exercice 2 (Terminaisons d'algorithmes récursifs).

Dans le cas de la terminaison d'algorithmes récursifs, la notion de *convergent* s'adapte facilement : il s'agit d'une quantité qui diminue strictement à *chaque appel récursif* et qui prend ses valeurs dans un ensemble bien fondé.

Pour chacun des quatre algorithmes suivants, dites s'il se termine ou pas. Si oui, mettez en évidence un convergent et prouvez la terminaison. Si non, précisez les cas particuliers dans lesquels l'algorithme se termine.

Algo5(n)

Entrée : un entier n

Sortie : un entier

- Retourner $n + Algo5(n - 1)$

Algo6(n)

Entrée : un entier n

Sortie : un entier

- Si $n == 10$ alors :
Retourner 1
- Retourner $n - Algo6(n - 1)$

Algo7(n)

Entrée : un entier n

Sortie : un entier

- Si $n == 10$ alors :
Retourner 1
- Si $n < 10$ alors :
Retourner $n - Algo7(10)$
- Retourner $n - Algo7(n - 1)$

Algo8(n)

Entrée : un entier n

Sortie : un entier

- Si $n == 10$ alors :
Retourner 1
- Si $n < 10$ alors :
Retourner $n - Algo8(25)$
- Retourner $n - Algo8(n - 1)$

Exercice 3 (Dans la vraie vie : le bug de Zune).

Les baladeurs MP3 Zune 30 de Microsoft ont été commercialisés à partir de 2006. Le 31 décembre 2008, ils ont été paralysés par un bug (cf. <https://fr.wikipedia.org/wiki/Zune>). Ce bug provenait d'un bout de code qui convertissait un nombre de jours absolu (compté depuis le 1er janvier 1980) en un couple formé d'une année, et d'un nombre de jours relatif (depuis le 1er janvier l'année en cours). Par exemple : $380 \mapsto (1981, 14)$. En effet, 1980 était une année bissextile (366 jours) donc la décomposition est $380 = 366 (1 \text{ an}) + 14 (\text{jours})$. Attention, le premier jour de l'année porte le numéro 1 et non 0 : ainsi, $367 \mapsto (1981, 1)$ (1er janvier 1981).

Le code en question (traduit en python) est le suivant :

```
def jour_de_l_annee(jour):
    annee = 1980
    while jour > 365:
        if bissextile(annee):
            if jour > 366:
                jour = jour - 366
                annee = annee + 1
        else:
            jour = jour - 365
            annee = annee + 1
    return (annee, jour)
```

1. Expliquez le bug.
2. Proposez une correction et prouvez la terminaison du programme corrigé.

Exercice 4 (Maximum itératif).

Montrer que l'algorithme suivant se termine. Montrer qu'il est correct en utilisant un invariant de boucle et un raisonnement par récurrence.

`maxiter(T)`: Recherche du maximum d'un tableau T

Entrée : un tableau $T[1..n]$ non trié de n entiers

Sortie : le plus grand entier de T

- $\text{max} = T[1]$
- Pour i allant de 1 à n :
 - ★ Si $T[i] > \text{max}$ alors :
 $\text{max} = T[i]$
- Retourner max

Exercice 5 (Maximum récursif).

Pour montrer la correction d'un algorithme récursif, on adapte la notion d'invariant de boucle sous la forme d'un raisonnement par récurrence portant sur les paramètres des appels récursifs.

Montrer que l'algorithme suivant se termine. Montrer qu'il est correct en utilisant un raisonnement par récurrence sur la taille du tableau.

`maxrec(T)`: Recherche du maximum d'un tableau T

Entrée : un tableau $T[1..n]$ non trié de n entiers

Sortie : le plus grand entier de T

- Si $n == 1$ alors :
 Retourner $T[1]$
- Sinon :
 Retourner le maximum entre `maxrec(T[1...n-1])` et $T[n]$

Exercice 6 (Tri par sélection).

Dans le tri par sélection, on trie de gauche à droite et on choisit le plus petit élément dans la partie non triée et on le met en fin de la partie triée. Montrer que l'algorithme se termine. Montrer qu'il est correct en utilisant des invariants de boucle. Il faudra raisonner séparément sur les deux boucles imbriquées (par récurrence à chaque fois). Pour la boucle interne, un invariant de boucle possible est $Q(k)$: « à l'itération numéro k de la boucle imbriquée, `valeurMin` contient le minimum des valeurs de $T[i] \dots T[i+k]$ ». Pour la boucle principale, vous pouvez utiliser $P(i)$: « $T[1..i]$ est trié et aucune valeur $T[k]$ pour $k > i$ n'est strictement inférieure à $T[i]$ ».

`triSelection(T)`: tri d'un tableau T d'entiers par ordre croissant

Entrée : un tableau $T[1..n]$ non trié de $n \geq 1$ entiers

Sortie : le tableau T trié

- Pour i allant de 1 à $n-1$ faire :
 - * `indiceMin = i`
 - * `valeurMin = T[i]`
 - * Pour j allant de $i+1$ à n faire :
 - Si $T[j] < \text{valeurMin}$ alors :
 - `indiceMin = j`
 - `valeurMin = T[j]`
 - * `T[indiceMin] = T[i]`
 - * `T[i] = valeurMin`
- Retourner T

Exercice 7 (Pour s'entraîner).

Étudier la terminaison et la correction des trois programme Python ci-dessous.

```
def Mult(a,b):
    assert b>=0
    r=0
    while b>0:
        b=b-1
        r=r+a
    return r
```

```
import random

def Alea(l):
    debut = 0
    fin = len(l) - 1
    resultat = []
    while debut <= fin:
        if random.randint(0, 2) == 1:
            resultat.append(l[debut])
            debut = debut + 1
        else:
            resultat.append(l[fin])
            fin = fin - 1
    return resultat
```

```

def Div(a, b):
    r = a
    q = 0
    if b == 0:
        raise ZeroDivisionError("division by zero")
    else:
        while r >= b:
            r = r - b
            q = q + 1
    return (q, r)

```

3 Exercices supplémentaires

Exercice 8 (Euclide soustractions).

Montrer que l'algorithme ci-dessous se termine. Montrer qu'il calcule correctement $PGCD(a, b)$ en utilisant un invariant de boucle.

```

def EuclideBis(a,b):
    x=a
    y=b
    if x==0 or y==0:
        x=max(x,y)
    else:
        while x!=y:
            if x>y:
                x=x-y
            else:
                y=y-x
    return x

```

Exercice 9 (Tri par insertion).

Dans l'algorithme de tri par insertion, on trie le tableau de gauche à droite, en insérant la valeur courante à la bonne place dans la partie déjà triée. Montrer que l'algorithme se termine. Montrer qu'il est correct en utilisant un invariant de boucle. Il faudra raisonner séparément sur les deux boucles imbriquées (par récurrence dans les deux cas). Pour la boucle intérieure, un invariant possible est $Q(k)$: « à l'itération numéro k de la boucle imbriquée, valeurInsertion est inférieur ou égal à $T[i - k] \dots T[i]$ ». Pour la boucle principale, vous pouvez utiliser $P(i)$: « $T[1 \dots i]$ est trié ».

triInsertion(T): tri d'un tableau T d'entiers par ordre croissant

Entrée : un tableau $T[1..n]$ non trié de n entiers

Sortie : le tableau T trié

- Pour i allant de 2 à n faire :
 - * $j = i - 1$
 - * valeurInsertion = $T[i]$
 - * Tant que $j > 0$ et $T[j] > \text{valeurInsertion}$ faire :
 - $T[j+1] = T[j]$
 - $j = j - 1$
 - * $T[j+1] = \text{valeurInsertion}$
- Retourner T

Exercice 10 (Recherche dichotomique récursive).

Montrer que l'algorithme suivant se termine. Montrer qu'il est correct par un raisonnement par récurrence sur $n - m$.

rechercheDicho(T, m, n, x): calcul d'une position de x dans T

Entrée : Un tableau d'entiers $T[m..n]$ triés par ordre croissant, un entier x , deux positions m et n

Sortie : 0 si $x \notin T$, et une position de x dans T (entre m et n) sinon

- Si $m == n$:
 - * Si $T[m] == x$:
Retourner m
 - * Sinon :
Retourner 0
- Sinon :
 - * $k = \lfloor \frac{m+n}{2} \rfloor$
 - * Si $T[k] < x$:
Retourner rechercheDicho($T, k + 1, n, x$)
 - * Sinon :
Retourner rechercheDicho(T, m, k, x)

Exercice 11 (Couplage stable – Gale-Shapley, 1962).

Étant donnés deux ensembles A et B , un *couplage* est un ensemble de paires de $A \times B$, deux à deux disjointes. On applique cette notion par exemple pour appairer des étudiants à des places en formation, des donneurs d'organes à des receveurs, etc. On représente un couplage par une application $c : A \rightarrow B$ qui associe à chaque élément a de A un élément $c(a)$ de B . Si $c(a) = b$, on notera $a = c^{-1}(b)$.

Lorsque les éléments de A et B ont des listes de préférence, on parle de *couplage stable* $c : A \rightarrow B$ si pour toute paire (a, b) de $A \times B$ non-appariée, a ne préfère pas b par rapport à son élément apparié $c(a)$ et b ne préfère pas a à son élément apparié $c^{-1}(b)$.

Gale et Shapley ont démontré qu'il existe toujours un couplage stable, grâce à l'algorithme ci-dessous qui permet d'en produire un.

Montrer que l'algorithme termine, et qu'il est correct.

`couplageStable(A, B)`: calcul d'un couplage stable entre A et B

Entrée : Deux ensembles A et B et pour chaque élément x , une liste ordonnée de préférences $L(x)$ d'éléments de l'autre ensemble

Sortie : Un couplage stable de A, B

- Tant qu'il existe un élément de A non apparié avec une liste de préférences non vide :
 - * Choisir un tel élément a de A
 - * Soit b l'élément de $L(a)$ préféré par a
 - * Si b n'est pas apparié :
Apparier a et b
 - * Sinon (b est apparié à $c^{-1}(b)$), si b préfère a à $c^{-1}(b)$:
Dés-apparier b et $c^{-1}(b)$
Apparier a et b
 - * Sinon :
Enlever b de $L(a)$
- Retourner le couplage