Validité et terminaison d'un algorithme itératif

Alix Munier-Kordon Maryse Pelletier

LIP6
Sorbonne Université
Paris

Module 2I003 Algorithmique Elémentaire

Outline

Problème et Instance

Definition

Un problème P est identifié par **un nom**, **un ensemble de paramètres**, et **une description de la solution**.

Definition

Une instance I d'un problème P est une valeur possible pour les paramètres du problème.

Problème de Tri

Définition du problème de tri :

Nom : Tri

Paramètres : un entier n et un tableau $tab[0 \cdots n-1]$ de n entiers.

Solution : Le tableau $tab[0 \cdots n-1]$ trié par ordre croissant.

Définition d'une instance du problème de tri :

Un entier $n = 5$

Un tableau $tab[0 \cdots 4] = [6, 3, 1, 7, 5]$.

Programme

Definition

Un programme est composé de :

1. un ensemble fini de variables,
2. des opérations élémentaires portant sur les variables (arithmétiques, logiques, affectations),
3. des structures de contrôles (if, while,$\cdots$etc) qui permettent de définir un ordre sur les opérations élémentaires.

Un programme est souvent exprimé dans un langage de programmation ou un langage algorithmique.

Algorithme et fonction

Definition

Un algorithme $\mathcal{A}$ qui résoud un problème P est un programme qui vérifie les deux propriétés suivantes :

Terminaison : L'algorithme appliqué à toute instance du problème effectue un nombre fini d'instructions.

Validité : L'algorithme résoud le problème P pour toute instance (de P).

Definition

Une fonction est un algorithme qui renvoie une valeur.

Trois questions pour l'évaluation d'un algorithme

Soit $\mathcal{A}$ un algorithme pour résoudre un problème P.

1. Est-ce que l'algorithme $\mathcal{A}$ se termine ?
   Terminaison
2. Est-ce que l'algorithme $\mathcal{A}$ résoud le problème P ?
   Validité
3. Est-ce que l'algorithme $\mathcal{A}$ est efficace en temps de calcul ?
   Complexité

Algorithme itératif de calcul de la suite de Fibonacci

Suite de Fibonacci :

- $F_0 = 0$, $F_1 = 1$;
- $\forall n \geq 2$, $F_n = F_{n-1} + F_{n-2}$.

```
def FiboIt (n):
    if (n==0):
        return 0
    x = 0; y = 1;  i = 1
    while (i<n):
        z = x+y; x = y
        y = z; i = i+1
    return y
```

Terminaison de la fonction FiboIt

- Le corps de boucle est composé d'instructions élémentaires;
- La boucle est effectuée $n - 1$ fois.

Donc FiboIt(n) se termine pour toute valeur $n \in \mathbb{N}$.

Comment varient les variables x et y ?

1. A l'initialisation, $x_1 = 0$ et $y_1 = 1$.
2. $\forall i \in \{2, \cdots, n\}$, x_i est la valeur de x à la fin du corps de boucle pour la valeur i.
3. $\forall i \in \{2, \cdots, n\}$, y_i est la valeur de y à la fin du corps de boucle pour la valeur i.

Pour $n = 6$:

i	1	2	3	4	5	6
x_i	0	1	1	2	3	5
y_i	1	1	2	3	5	8

La fonction retourne 8.

Propriétés des suites x_i et y_i

$\mathcal{P}(i)$, $i \in \{1, \cdots, n\}$: $x_i = F_{i-1}$ et $y_i = F_i$.
Par récurrence faible :

Base Pour $i = 1$, $x_1 = 0 = F_0$ et $y_1 = 1 = F_1$.
Donc $\mathcal{P}(1)$ est vérifiée.

Iteration Supposons que $\mathcal{P}(k)$ soit vérifiée pour une valeur $k \geq 1$ fixée. Alors,
$y_{k+1} = x_k + y_k = F_{k-1} + F_k = F_{k+1}$ et
$x_{k+1} = y_k = F_k$.
Donc $\mathcal{P}(k+1)$ est vérifiée.

Conclusion Comme $\mathcal{P}(1)$ est vérifiée, et que, pour tout $k \in \{1, \cdots, n-1\}$, $\mathcal{P}(k) \Rightarrow \mathcal{P}(k+1)$, on en déduit que $\forall i \in \{1, \cdots, n\}$, $\mathcal{P}(i)$ est vérifiée.

Validité de la fonction FiboIt

FiboIt est valide si, pour tout $n \in \mathbb{N}$, FiboIt(n) retourne F_n.

- Si $n = 0$, le résultat retourné est bien égal à F_0.
- Sinon, en sortie de boucle, $i = n$ et donc $y_n = F_n$. La fonction retourne donc bien la valeur F_n.

On en déduit que la fonction FiboIt est valide.

Comment démontrer la terminaison d'une boucle ?

1. Vérifier que toute exécution du corps de boucle se termine;
2. Vérifier que le corps de boucle est exécuté un nombre borné de fois;

Ces deux démonstrations ne se font pas par récurrence !!

Comment démontrer la validité d'une boucle ?

1. Déterminer un invariant de boucle si possible à la fin du corps de boucle;
2. Démontrer l'invariant de boucle par récurrence sur le nombre d'itérations;
3. Etudier la sortie de boucle pour conclure à la validité de la boucle.

Notion d'invariant de boucle

Definition

Un invariant d'une boucle est une propriété qui est vérifiée à chaque exécution du corps de cette boucle. Cette propriété est de plus vraie à un endroit précis du corps de boucle et permet de démontrer la validité de la boucle.

Pour la fonction FiboIt, la propriété $\mathcal{P}(i)$, $i \in \{1, \cdots, n\}$ est vérifiée à la fin du corps de boucle.

Recherche de l'élément minimum d'un tableau

$tab[0 \cdots n-1]$ est un tableau de n entiers;
RechercheMin(tab, d, f) retourne l'indice d'un élément de $tab[d \cdots f]$ de valeur minimum pour $0 \leq d \leq f < n$.

```
def RechercheMin(tab, d, f):
    imin=d; i=d+1
    while i<=f:
        if tab[i]<tab[imin]:
            imin=i
        i=i+1
    return imin
```

Terminaison de RechercheMin

- Le corps de boucle est composé d'instructions élémentaires;
- La boucle est effectuée $f - d$ fois au plus;

Donc RechercheMin(tab, d, f) se termine pour toute valeur $0 \leq d \leq f < n$.

Validité de RechercheMin(*tab*, *d*, *f*)

Soit $imin_{d+1} = d$ et $imin_i$, pour $i \in \{d+2, \cdots, f+1\}$, la valeur de la variable *imin* à la fin du corps de la boucle.

Theorem

Pour tout $i \in \{d+1, \cdots, f+1\}$, $imin_i \in \{d, \cdots, i-1\}$ et $\forall k \in \{d, \cdots, i-1\}$, $tab[k] \geq tab[imin_i]$.

Corollary

La fonction RechercheMin(tab, d, f) est valide pour un tableau de n valeurs avec $0 \leq d \leq f < n$.

RechercheMin est utilisée pour le tri par sélection.

Exemple d'exécution du tri par sélection

$i = 0$		6	3	1	7	5	$imin = 2$
$i = 1$	1		3	6	7	5	$imin = 1$
$i = 2$	1	3		6	7	5	$imin = 4$
$i = 3$	1	3	5		7	6	$imin = 4$
$i = 4$	1	3	5	6	7		fin de l'algorithme

Tri par sélection

```
def TriParSelection(tab):
    i=0; n=len(tab)
    while (i!= n):
        k = RechercheMin(tab, i, n-1)
        if (k!=i):
            z = tab[i]
            tab[i]=tab[k]
            tab[k]=z
        i=i+1
```

Terminaison du tri par sélection

- Les paramètres des appels à RechercheMin vérifient $0 \leq i \leq n-1 < n$. Donc, tous ces appels se terminent.
- Les autres instructions du corps de boucle sont élémentaires.
- Le corps de boucle est exécuté $n-1$ fois.

On en déduit que TriParSelection se termine.

Validité de `TriParSelection`

$tab_0 = tab$ à l'initialisation et pour $i \in \{1, \cdots, n\}$, soit tab_i le tableau *tab* obtenu en fin du corps de boucle pour la valeur i.

Theorem

Pour tout $i \in \{0, \cdots, n\}$, tab_i contient tous les éléments de tab et $tab_i[0 \cdots i-1]$ est constitué des i plus petits éléments de tab triés en ordre croissant.

Corollary

TriParSelection(tab) trie les éléments de tab par ordre croissant.

Conclusion

1. **Validité** et **terminaison** permettent de s'assurer qu'un algorithme est correct pour toutes les valeurs admissibles des paramètres.
2. La **terminaison** d'une boucle simple peut souvent se réduire à majorer le nombre de tours de boucles et s'assurer de la terminaison des instructions du corps de boucle.
3. La **validité** s'obtient en démontrant par récurrence l'invariant de boucle, puis en étudiant la sortie de boucle.