

Preuves de programme

Quentin Fortier

March 2, 2022

Preuve de programme

Soit f une fonction.

On veut montrer 2 choses :

- f **termine** pour chaque entrée : ne fait pas boucle infinie ou appels récursifs infinis

Preuve de programme

Soit f une fonction.

On veut montrer 2 choses :

- f **termine** pour chaque entrée : ne fait pas boucle infinie ou appels récursifs infinis
- f est **correct** : la valeur renvoyée par f est bien celle qu'on veut

Pour montrer qu'une boucle while (ou fonction récursive) termine :

- utiliser une suite d'entiers strictement décroissante (à chaque itération du while ou appel récursif)
- montrer que la boucle s'arrête lorsque la suite devient négative

```
def pgcd(a, b):  
    if b == 0:  
        return a  
    else:  
        return pgcd(b, a mod b)
```

Question

Comment montrer que $\text{pgcd}(a, b)$ termine si $a \geq b$?

```
def pgcd(a, b):  
    if b == 0:  
        return a  
    else:  
        return pgcd(b, a mod b)
```

Question

Comment montrer que $\text{pgcd}(a, b)$ termine si $a \geq b$?

Soient a_n et b_n les valeurs de a et b après n appels récursifs.

```
def pgcd(a, b):  
    if b == 0:  
        return a  
    else:  
        return pgcd(b, a mod b)
```

Question

Comment montrer que $\text{pgcd}(a, b)$ termine si $a \geq b$?

Soient a_n et b_n les valeurs de a et b après n appels récursifs.

On montre par récurrence sur n que $a_n \geq b_n$, $a_n \searrow \searrow$ et $b_n \searrow \searrow$.

```
def dichotomie(e, L):  
    i, j = 0, len(L) - 1  # i et j sont les indices de L entre les  
    while i <= j: # tant qu'il reste au moins 1 élément entre les  
        m = (i + j)//2 # milieu de i et j  
        if e < L[m]:  
            j = m # regarder dans la partie gauche  
        elif e > L[m]:  
            i = m # regarder dans la partie droite  
        else:  
            return True # on a trouvé e  
    return False # e n'a pas été trouvé
```

Question

Donner un exemple où cette version (erronée) de la recherche par dichotomie ne termine pas.


```
def dichotomie(e, L):  
    i, j = 0, len(L) - 1  # i et j sont les indices de L entre les  
    while i <= j: # tant qu'il reste au moins 1 élément entre les  
        m = (i + j)//2 # milieu de i et j  
        if e < L[m]:  
            j = m # regarder dans la partie gauche  
        elif e > L[m]:  
            i = m # regarder dans la partie droite  
        else:  
            return True # on a trouvé e  
    return False # e n'a pas été trouvé
```

Question

Donner un exemple où cette version (erronée) de la recherche par dichotomie ne termine pas.

dicho([0], 1) fait boucle infinie

```
def dichotomie(e, L):  
    i, j = 0, len(L) - 1  
    while i <= j:  
        m = (i + j)//2  
        if e < L[m]:  
            j = m - 1  
        elif e > L[m]:  
            i = m + 1  
        else:  
            return True  
    return False
```

Question

Comment montrer que la boucle `while` termine ?

```
def dichotomie(e, L):  
    i, j = 0, len(L) - 1  
    while i <= j:  
        m = (i + j)//2  
        if e < L[m]:  
            j = m - 1  
        elif e > L[m]:  
            i = m + 1  
        else:  
            return True  
    return False
```

Question

Comment montrer que la boucle `while` termine ?

Montrer que $j - i$ décroît strictement

Terminaison

m est égal à $\lfloor \frac{i+j}{2} \rfloor$ donc vérifie par définition :

$$\frac{i+j}{2} - 1 < m \leq \frac{i+j}{2}$$

Terminaison

m est égal à $\lfloor \frac{i+j}{2} \rfloor$ donc vérifie par définition :

$$\frac{i+j}{2} - 1 < m \leq \frac{i+j}{2}$$

- Si $e < L[m]$:

Terminaison

m est égal à $\lfloor \frac{i+j}{2} \rfloor$ donc vérifie par définition :

$$\frac{i+j}{2} - 1 < m \leq \frac{i+j}{2}$$

- Si $e < L[m]$: on remplace j par m

$$m - i \leq \frac{i+j}{2} - i = \frac{j-i}{2} < j - i$$

- Si $e > L[m]$:

Terminaison

m est égal à $\lfloor \frac{i+j}{2} \rfloor$ donc vérifie par définition :

$$\frac{i+j}{2} - 1 < m \leq \frac{i+j}{2}$$

- Si $e < L[m]$: on remplace j par m

$$m - i \leq \frac{i+j}{2} - i = \frac{j-i}{2} < j - i$$

- Si $e > L[m]$: on remplace i par $m + 1$

$$j - (m + 1) < j - \frac{i+j}{2} = \frac{j-i}{2} < j - i$$

Terminaison

m est égal à $\lfloor \frac{i+j}{2} \rfloor$ donc vérifie par définition :

$$\frac{i+j}{2} - 1 < m \leq \frac{i+j}{2}$$

- Si $e < L[m]$: on remplace j par m

$$m - i \leq \frac{i+j}{2} - i = \frac{j-i}{2} < j - i$$

- Si $e > L[m]$: on remplace i par $m + 1$

$$j - (m + 1) < j - \frac{i+j}{2} = \frac{j-i}{2} < j - i$$

Ainsi $j - i$ est une suite d'entiers strictement décroissante donc qui devient négatif, ce qui termine la boucle **while**.

```
while m != n:
    if m > n:
        m = m - n
    else:
        n = n - m
```

Question

Est-ce que cette boucle **while** termine pour n et m dans \mathbb{N}^* ?

Pour prouver qu'une fonction est correcte (renvoie bien le bon résultat), on utilise presque toujours un **raisonnement par récurrence** :

- Boucle **while** : récurrence sur le nombre d'itération, ce qu'on appelle aussi **invariant de boucle**
- Fonction récursive : récurrence sur le nombre d'appels récursifs

Correction

```
def exp_rapide(a, n):  
    if n == 0:  
        return 1  
    else:  
        b = exp_rapide(a, n/2)  
        if n % 2 == 0:  
            return b*b  
        else:  
            return a*b*b
```

```
def exp_rapide(a, n):  
    if n == 0:  
        return 1  
    else:  
        b = exp_rapide(a, n/2)  
        if n % 2 == 0:  
            return b*b  
        else:  
            return a*b*b
```

Récurrance forte sur $\mathcal{H}(n)$: `exp_rapide a n` renvoie a^n

```
def dichotomie(e, L):  
    i, j = 0, len(L) - 1  
    while i <= j:  
        m = (i + j)//2  
        if e < L[m]:  
            j = m - 1  
        elif e > L[m]:  
            i = m + 1  
        else:  
            return True  
    return False
```

```
def paysan(a, b):  
    res = 0  
    while b != 0:  
        if b % 2 == 0:  
            a *= 2  
            b /= 2  
        else:  
            res += a  
            b -= 1  
    res
```

Exercice

Dire ce que fait cette fonction et le prouver en donnant un invariant de boucle.