CHAPITRE 7
ÉLÉMENTS DE RÉCURSIVITÉ

Programmation en Python

2

La fonction factorielle

Définition

- En informatique, une fonction est dite **récursive** si cette fonction fait appel à elle-même dans sa définition.
- Au premier abord, on pourrait penser que les fonctions récursives sont mal définies (définition cyclique).
- Le plus célèbre exemple d'une fonction récursive est la fonction **factorielle**.

La fonction factorielle

- On rappelle que la **factorielle** d'un nombre entier positif n, noté **n!**, est le produit des nombres entiers strictement positifs inférieurs ou égaux à n (avec 0! = 1 par convention).
  - **Exemples :**

    $0! = 1$ (par convention)

    $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$

    $7! = 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5040$

    $11! = 11 \cdot 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 39'916'800$

La fonction factorielle

- Il existe une définition récursive simple de la fonction factorielle :
- $$\text{Fact}(n) = \begin{cases} 1 & \text{si } n = 0 \\ n * \text{Fact}(n-1) & \text{si } n > 0 \end{cases}$$
- On voit que cette définition de la fonction factorielle s'appelle elle-même dans sa propre définition.

La fonction factorielle

- Exemple d'exécution de la fonction récursive factorielle :
- $$\text{Fact}(n) = \begin{cases} 1 & \text{si } n = 0 \\ n * \text{Fact}(n-1) & \text{si } n > 0 \end{cases}$$

Fact(4)	**24**
4 * Fact(3)	= 4 * 6 = 24
3 * Fact(2)	= 3 * 2 = 6
2 * Fact(1)	= 2 * 1 = 2
1 * Fact(0)	= 1 * 1 = 1

La fonction factorielle en Python


```python
def facto(n):
    """Fonction factorielle"""
    if n == 0:
        return 1
    else:
        return n * facto(n-1)
```

La fonction factorielle en Python


```
# Exemple (faire démo):
for x in range(10):
        print facto(x)
```

Exécution

```
1
1
2
6
24
120
720
5040
40320
362880
```

9

Suite de Fibonacci

Suite de Fibonacci

- La **suite de Fibonacci** est une suite infinie de nombre définie récursivement de la manière suivante :

  $$fib(0) = fib(1) = 1$$

  $$fib(n) = fib(n-1) + fib(n-2), \text{ pour tout } n > 1$$

- Cette suite est dite récursive car sa définition fait appel à elle-même…
- Ainsi, on a les premiers termes suivants :

  $fib(0) = 1$, $fib(1) = 1$, $fib(2) = 1 + 1 = 2$, $fib(3) = 2 + 1 = 3$, $fib(4) = 3 + 2 = 5$, $fib(5) = 5 + 3 = 8$, etc.

Suite de Fibonacci en Python


```python
def fibonacci(n):
    """Suite de Fibonacci"""
    if n == 0 or n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Suite de Fibonacci en Python


```
# Exemple (faire démo):
for x in range(10):
    print fibonacci(x)
```

Exécution

```
1
1
2
3
5
8
13
21
34
55
```

Algorithme de conversion décimal-binaire

Rappels

- Il existe une procédure récursive simple pour convertir un nombre entier positif décimal en binaire.
- On rappelle que dans le système décimal, les positions successives des chiffres de droite à gauche représentent les puissances successives de 10 (10^0, 10^1, 10^2, 10^3, etc.).
- **Exemple :**

  $29453 = 3 * 10^0 + 5 * 10^1 + 4 * 10^2 + 9 * 10^3 + 2 * 10^4$

Rappels

- **Exemples :**
  - Que signifie le nombre décimal (base 10) 5478 ?

… $10^4=10000$ $10^3=1000$ $10^2=100$ $10^1=10$ $10^0=1$

… … 5 4 7 8

  - Que signifie le nombre décimal (base 10) 362'458 ?

$10^5=100000$ $10^4=10000$ $10^3=1000$ $10^2=100$ $10^1=10$ $10^0=1$

3 6 2 4 5 8

Rappels

- Dans le système binaire, les positions successives des chiffres de droite à gauche représentent les puissances successives de 2 (2^0, 2^1, 2^2, 2^3, etc.).
- **Exemple :**

$$10011 = 1 * 2^0 + 1 * 2^1 + 0 * 2^2 + 0 * 2^3 + 1 * 2^4$$

Système binaire

- **Exemples :**
  - Que signifie le nombre binaire (base 2) 1101 ?

…	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
…	…	1	1	0	1

  - Que signifie le nombre binaire (base 2) 101101 ?

…	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
…	1	0	1	1	0	1

Conversion binaire - décimal

- **Exemples :**
  - Que vaut en décimal le nombre binaire 1101 ?

…	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
…	…	1	1	0	1

Donc 1101 vaut 8 + 4 + 1 = 13 (en décimal)

  - Que vaut en décimal le nombre binaire 10101101 ?

$2^7=128$	$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
1	0	1	0	1	1	0	1

Donc 10101101 vaut 128 + 32 + 8 + 4 + 1 = 173 (en décimal)

Conversion décimal - binaire

- **Exemples :**
  - Que vaut en binaire le nombre décimal 267 ?

$2^8=256$	$2^7=128$	$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
1	0	0	0	0	1	0	1	1

Donc 267 vaut 100001011 (en binaire)

  - Que vaut en binaire le nombre décimal 45 ?

$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
…	1	0	1	1	0	1

Donc 45 vaut 101101 (en binaire)

Algo de conversion décimal - binaire

- On divise successivement n par 2 jusqu'à obtenir 0, et à chaque pas, on retient le reste de cette division. La suite de ces restes lue dans l'ordre inverse donne la représentation binaire de n.
  - **Exemple :** prenons n = 26

    26 / 2 = 13 reste 0
    13 / 2 = 6 reste 1
    6 / 2 = 3 reste 0
    3 / 2 = 1 reste 1
    1 / 2 = 0 reste 1

    Au final, la représentation binaire de 26 est 11010.

Algorithme en Python (non récursif)


```python
def DecimalToBinary(n):
    if n == 0:
        res = str(0) # 0 converti en chaîne
    else:
        res = ""     # chaîne vide
    while n != 0:
        res = str(n % 2) + res
        n = n / 2   # division entière (en Python 2)
    return res
```

Algorithme en Python (non récursif)


```
# Exemple (faire démo):
print DecimalToBinary(26)
print DecimalToBinary(45)
print DecimalToBinary(267)
```

Exécution:

```
11010
101101
100001011
```

Algorithme en Python (récursif)


```python
def DecimalToBinaryRec(n):
    if n > 0:
        return DecimalToBinaryRec(n / 2) + str(n % 2)
    else:
        return ""
```

Division entière

- Exemple d'exécution :

f(26)	**11010**
f(13) & 0	11010
f(6) & 1	1101
f(3) & 0	110
f(1) & 1	11
f(0) & 1	" " & 1 = 1

Algorithme en Python (non récursif)


```python
# Exemple (faire démo):
print DecimalToBinaryRec(26)
print DecimalToBinaryRec(45)
print DecimalToBinaryRec(267)
```

```
Exécution:
11010
101101
100001011
```

25

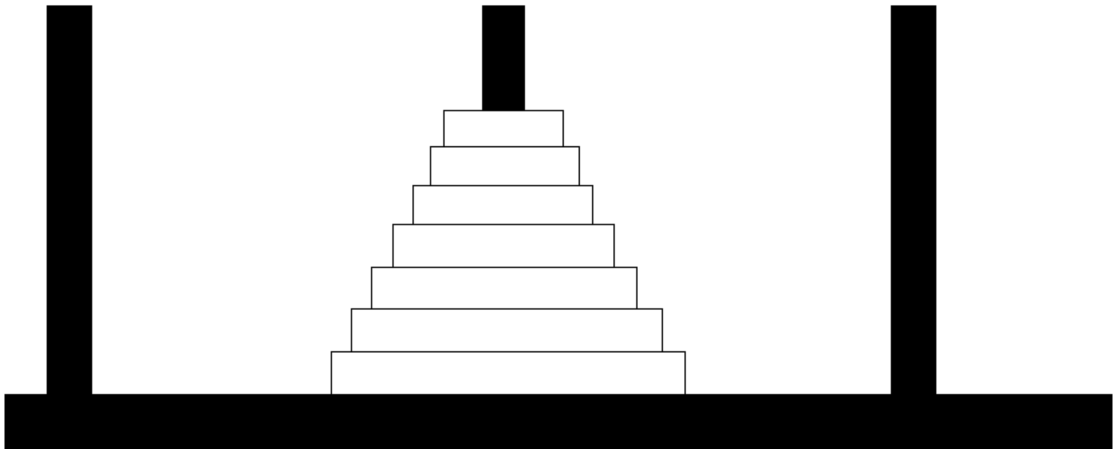
Tours de Hanoï

Tours de Hanoï

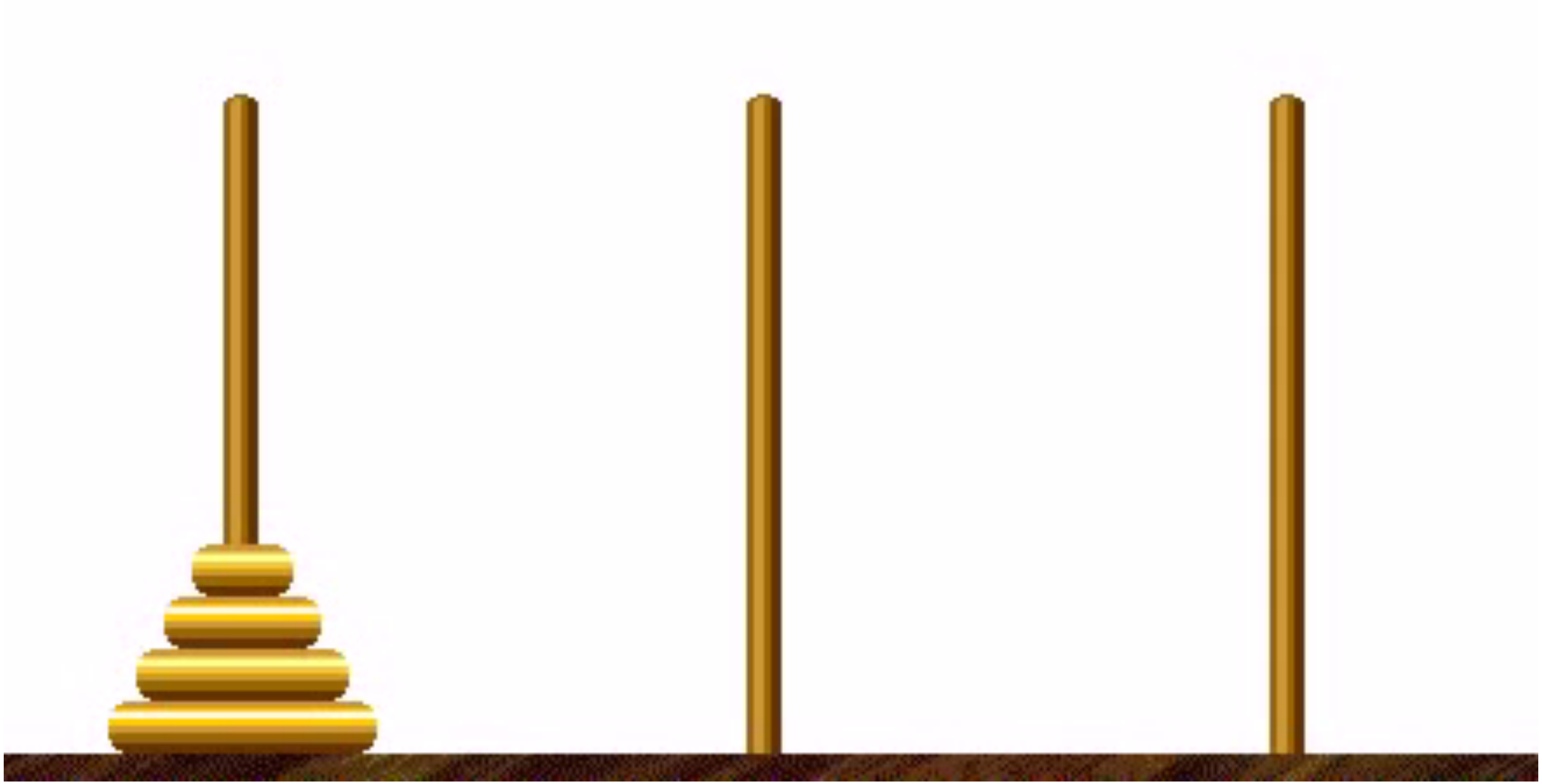
- Le problème des **tours de Hanoï** est un jeu de réflexion imaginé par le mathématicien français Édouard Lucas.
- Il consiste à déplacer en un minimum de coups des disques de diamètres différents d'une tour de départ à une tour d'arrivée en passant par une tour intermédiaire.
- Règles à respecter :
  - on ne peut pas déplacer plus d'un disque à la fois.
  - on ne peut placer un disque que sur un disque plus grand ou sur un emplacement vide.

Tours de Hanoï

- La situation la plus courante correspond à 7 disques, mais le nombre de disques peut être quelconque.



Tours de Hanoï



Algorithme

- On démontre par récurrence que pour une situation avec n disques, il faut 2^n - 1 coups au minimum pour parvenir à ses fins.
  - Pour 6 disques, il faut au minimum 63 coups.
  - Pour 7 disques, il faut au minimum 127 coups.
  - Pour 8 disques, il faut au minimum 255 coups.
- Ainsi, le problème devient rapidement très difficile à résoudre de tête.
- Toutefois, il existe une procédure récursive très simple permettant de résoudre le problème informatiquement.

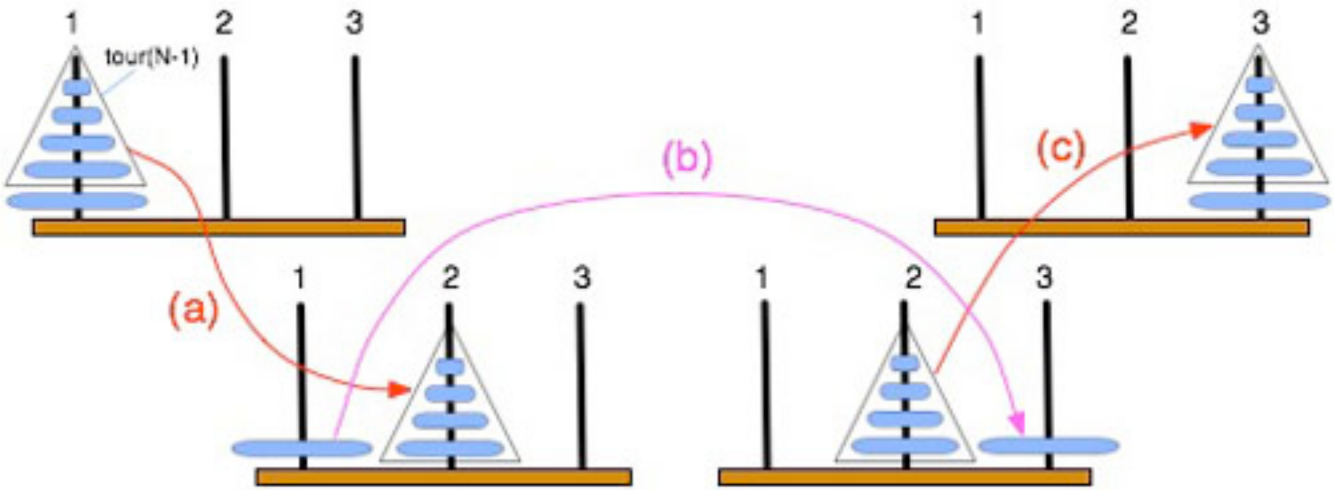
Algorithme

- Solution via une procédure récursive :
  - D'abord, on remarque que le problème est trivial pour le cas de zéro disque. En effet, il n'y a pas de jeu dans ce cas, donc rien à faire !
  - Supposons qu'on sache résoudre le problème pour le cas de n-1 disques. Alors on peut très facilement en déduire une solutions pour le cas de n disques, en trois étapes seulement (c.f. slide suivant) !

Algorithme

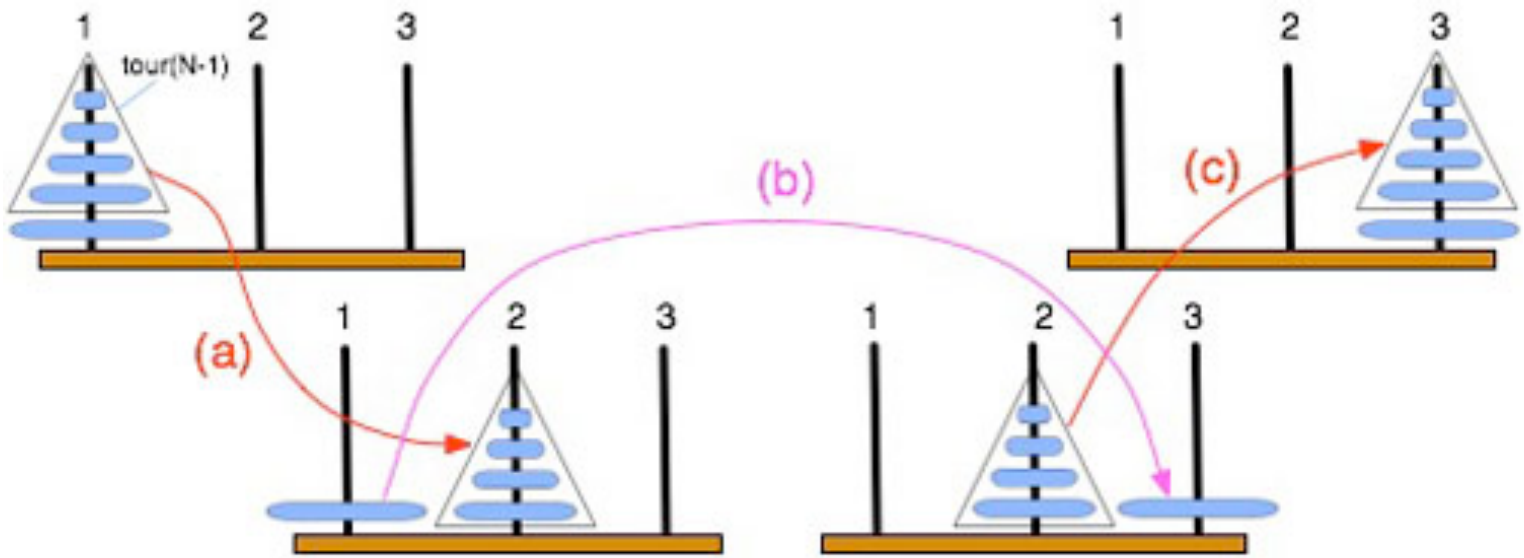
- La solution pour le cas de n disques se déduit très facilement de la solution pour le cas de n-1 disques.
- Pour déplacer n disques du pilier 1 ver le le pilier 3, il faut :

a) Déplacer n-1 disques du pilier 1 vers le pilier 2
b) Bouger un disque du pilier 1 vers le pilier 3
c) Déplacer n-1 disques du pilier 2 vers le pilier 3



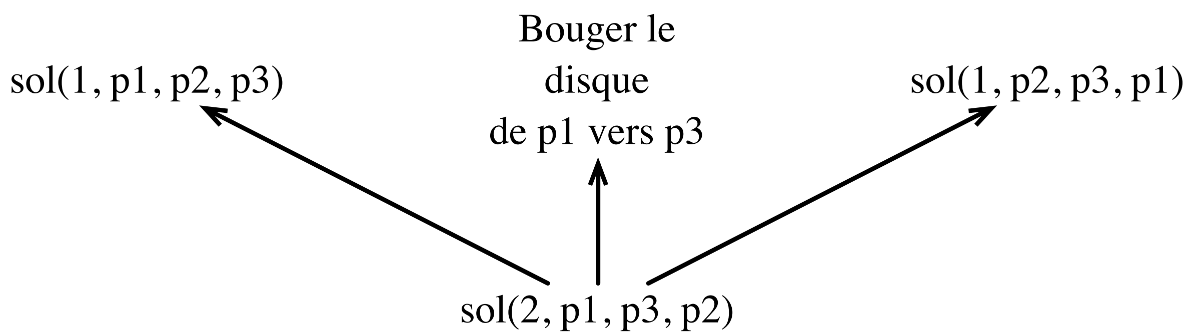
Algorithme

- On a donc une situation récursive :
  - La solution pour le cas de zéro disque est triviale.
  - La solution pour le cas de n disques se déduit très facilement de la solution pour le cas de n-1 disques.



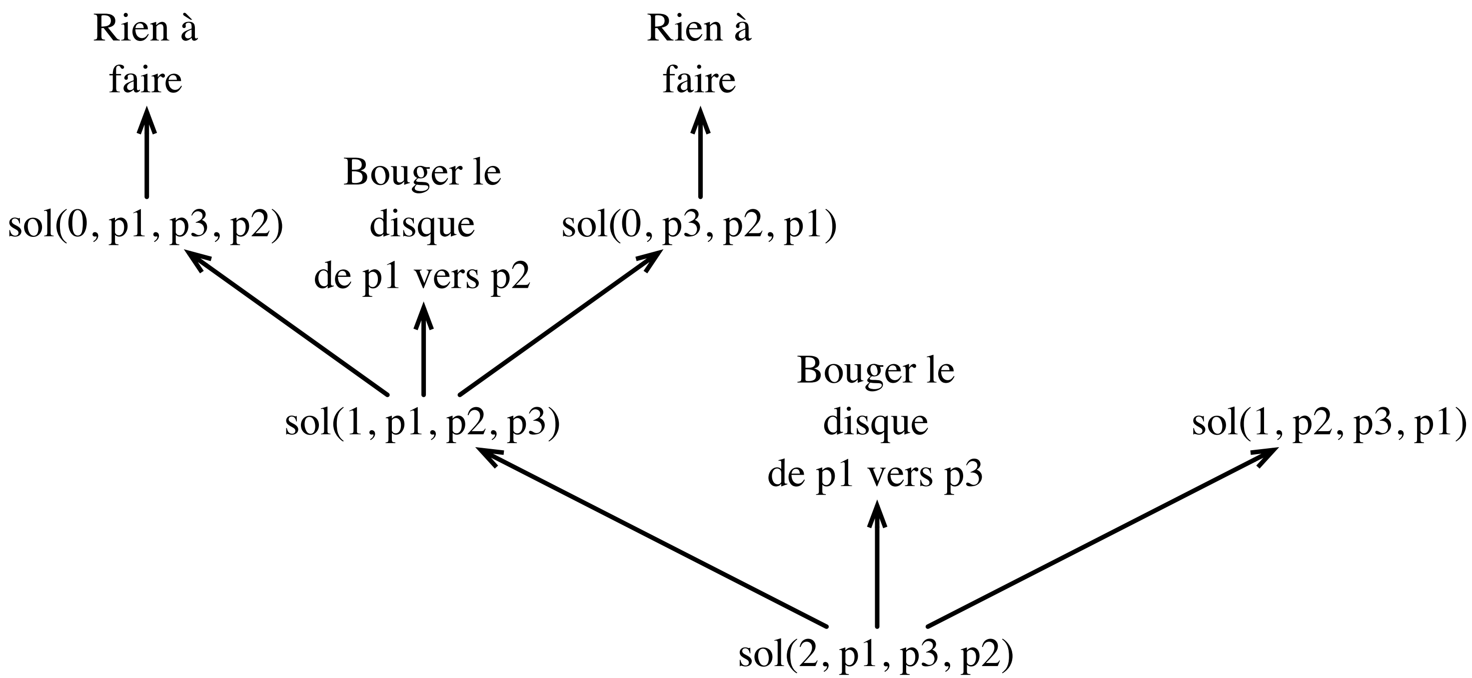
Algorithme

- Arbre d'exécution de la procédure Solution pour le cas n = 2, p1 = 1[er] pilier, p2 = 2[ème] pilier, p3 = 3[ème] pilier.



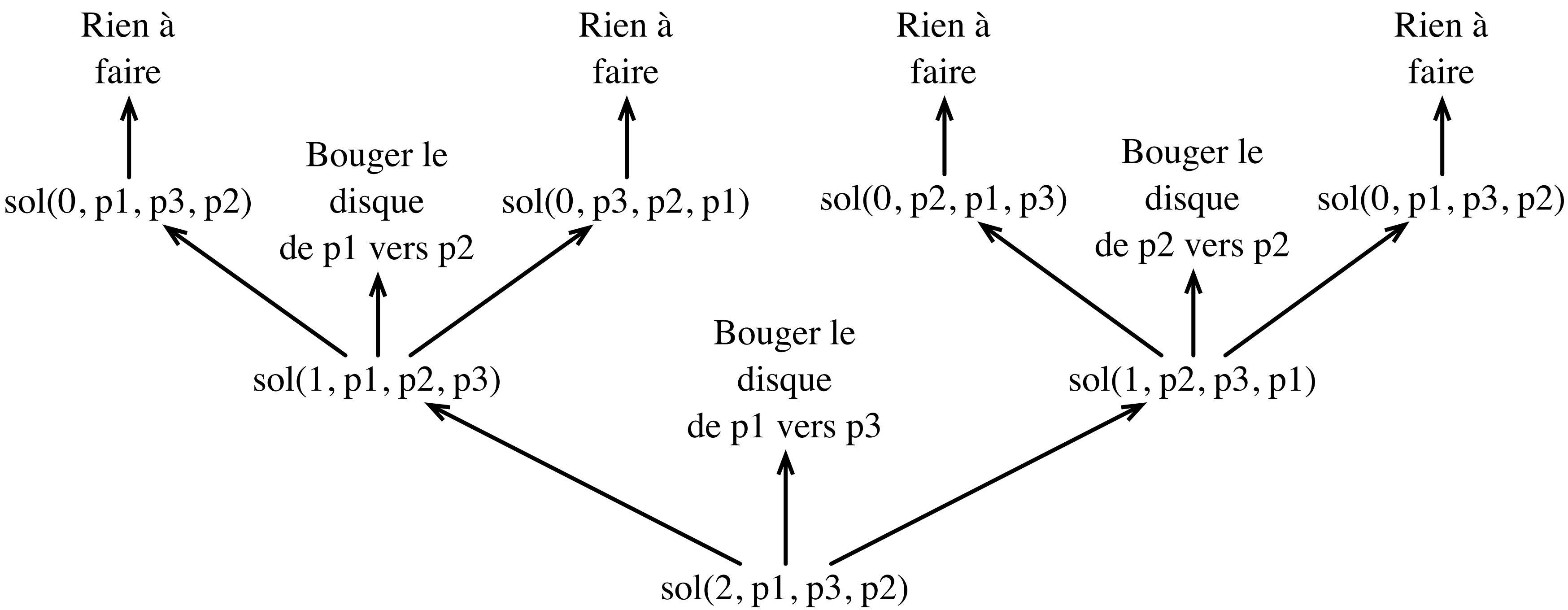
Algorithme

- Arbre d'exécution de la procédure Solution pour le cas $n = 2$, $p1 = 1^{er}$ pilier, $p2 = 2^{ème}$ pilier, $p3 = 3^{ème}$ pilier.



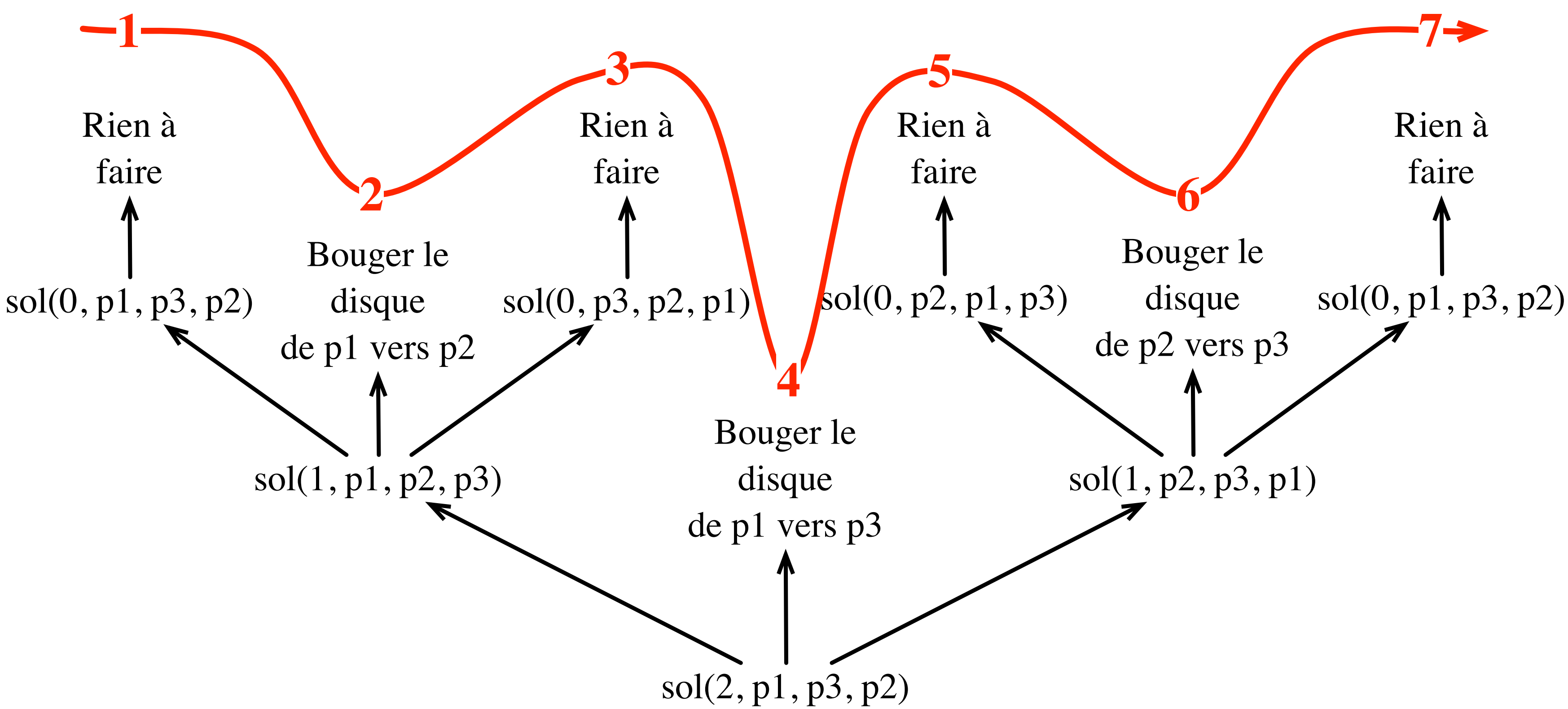
Algorithme

- Arbre d'exécution de la procédure Solution pour le cas n = 2, p1 = 1er pilier, p2 = 2ème pilier, p3 = 3ème pilier.



Algorithme

- Arbre d'exécution de la procédure Solution pour le cas n = 2, p1 = 1er pilier, p2 = 2ème pilier, p3 = 3ème pilier.



Algorithme

- Pseudocode de l'algorithme récursif qui permet de donner la solution pour le cas de `n` disques à déplacer du pilier `p1` vers le pilier `p3` en passant par le pilier `p2`

```
Hanoi(n, p1, p3, p2)
  Si n > 0 :
    Hanoi(n-1, p1, p2, p3) # appel récursif
    print « Bouger le disque de p1 vers p3 »
    Hanoi(n-1, p2, p3, p1) # appel récursif
  Sinon :
    rien faire
```

Algorithme en Python


```
def Hanoi(n, p1, p3, p2):
    if n > 0 :
        Hanoi(n-1, p1, p2, p3)
        print "Bouger le disque de " + str(p1) + \
              " vers " + str(p3)
        Hanoi(n-1, p2, p3, p1)
    else:
        pass
```

Algorithme en Python


```
# Exemple (faire démo):
Hanoi(7, 1 ,3 ,2)
```

Algorithme en Python

- Pour mieux visualiser ce qui se passe, on peut ajouter une impression des piliers de la configuration des piliers dans le terminal.
- On se concentre uniquement le cas de 7 disques.

Algorithme en Python


```
# liste des piliers...
# chaque pilier est une liste de nombres
# au début, le pilier 1 est rempli et les deux autres vides
piliers = [[7,6,5,4,3,2,1], [], []]
```

Algorithme en Python


```
def bouger_disque(p1,p2):
    print "\n"
    print "Bouger le disque de " + str(p1) + " vers " + str(p2)
    # mise à jour des piliers
    piliers[p2-1].append(piliers[p1-1].pop())
    # impression des piliers (dans le cas n=7)
    for i in range(7):
        print "" # pour aller a la ligne
        for p in piliers:
            # on complete le pilier avec des points
            p_mod = p + ["."]*(7-len(p))
            # on les imprime depuis la fin vers le debut
            # pour avoir le petits disques en dessus
            # et les gros en dessous
            print p_mod[-i-1], "   ",
```

Algorithme en Python


```
def Hanoi_bis(n, p1, p3, p2):
    if n > 0 :
        Hanoi_bis(n-1, p1, p2, p3)
        bouger_disque(p1,p3) # appel nouvelle fonction...
        Hanoi_bis(n-1, p2, p3, p1)
    else:
        pass

# Exemple (faire démo):
Hanoi_bis(7, 1 ,3 ,2)
```