Programmation et fonctions

INFO1 - semaines 39 à 42

Guillaume CONNAN

septembre 2015

Iut de Nantes - Dpt d'informatique

Sommaire

Sommaire

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE ⊗ CODOMAINE

...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE ⊗ CODOMAINE

...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE ⊗ CODOMAINE

...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE
Arrivée CODOMAINE
Une fonction est une relation
C'est donc une partie (sous-ensemble) de DOMAINE ⊗ CODOMAINE
...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE ⊗ CODOMAINE

...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE
Arrivée CODOMAINE
Une fonction est une relation
C'est donc une partie (sous-ensemble) de DOMAINE ⊗ CODOMAINE
...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE
Arrivée CODOMAINE
Une fonction est une relation
C'est donc une partie (sous-ensemble) de DOMAINE $\otimes$ CODOMAINE
...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE
Arrivée CODOMAINE
Une fonction est une relation
C'est donc une partie (sous-ensemble) de DOMAINE $\otimes$ CODOMAINE
...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE
Arrivée CODOMAINE
Une fonction est une relation
C'est donc une partie (sous-ensemble) de DOMAINE $\otimes$ CODOMAINE
...mais il n'y a pas deux couples ayant le même premier élément

Donc la fonction *double* de domaine $\{1, 2, 3, ...\}$ est

$$\{(1, 2), (2, 4), (3, 6), ...\}$$

Donc la fonction *double* de domaine $\{1, 2, 3, ...\}$ est

$$\{(1, 2), (2, 4), (3, 6), ...\}$$

La fonction *addition* de domaine $\{1, 2, 3, ...\} \otimes \{1, 2, 3, ...\}$ est

$$\{((1, 1), 2), ((1, 2), 3), ...((5, 7), 12), ((5, 8), 13), ...\}$$

La fonction *addition* de domaine $\{1, 2, 3, ...\} \otimes \{1, 2, 3, ...\}$ est

$$\{((1, 1), 2), ((1, 2), 3), ...((5, 7), 12), ((5, 8), 13), ...\}$$

Mais bon, ça manque un peu de dynamisme.

Départ $\longrightarrow$ Arrivée

Départ $\longrightarrow$ Arrivée

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

```
In [62]: ?chr
Docstring:
chr(i) -> Unicode character

Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
Type:      builtin_function_or_method
```

```
In [63]: ?ord
Docstring:
ord(c) -> integer

Return the integer ordinal of a one-character string.
Type:      builtin_function_or_method
```

```
Prelude Data.Char> :t chr
chr :: Int -> Char
Prelude Data.Char> :t ord
ord :: Char -> Int
```

```
In [62]: ?chr
Docstring:
chr(i) -> Unicode character

Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
Type:      builtin_function_or_method
```

```
In [63]: ?ord
Docstring:
ord(c) -> integer

Return the integer ordinal of a one-character string.
Type:      builtin_function_or_method
```

```
Prelude Data.Char> :t chr
chr :: Int -> Char
Prelude Data.Char> :t ord
ord :: Char -> Int
```

```
In [62]: ?chr
Docstring:
chr(i) -> Unicode character

Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
Type:      builtin_function_or_method
```

```
In [63]: ?ord
Docstring:
ord(c) -> integer

Return the integer ordinal of a one-character string.
Type:      builtin_function_or_method
```

```
Prelude Data.Char> :t chr
chr :: Int -> Char
Prelude Data.Char> :t ord
ord :: Char -> Int
```

```python
def car_en_maj(car) :
    ascii = ord(car)
    if 97 <= ascii <= 97 + 26 :
        return chr(ascii - 32)
    else :
        return car
```

```python
def car_en_maj(car) :
    ascii = ord(car)
    if 97 <= ascii <= 97 + 26 :
        return chr(ascii - 32)
    return car
```

```python
def car_en_maj(car) :
    ascii = ord(car)
    if 97 <= ascii <= 97 + 26 :
        return chr(ascii - 32)
    else :
        return car
```

```python
def car_en_maj(car) :
    ascii = ord(car)
    if 97 <= ascii <= 97 + 26 :
        return chr(ascii - 32)
    return car
```

Sommaire

Print

Et pourquoi pas **`print`** ?

`ord` prend un argument de type **`string`**, retourne un argument de type **`int`** ET NE FAIT QUE ÇA.

C'est une fonction PURE

Print

Et pourquoi pas **print** ?
ord prend un argument de type **string**, retourne un argument de type **int** ET NE FAIT QUE ÇA.
C'est une fonction PURE

Print

Et pourquoi pas **print** ?
ord prend un argument de type **string**, retourne un argument de type **int** ET NE FAIT QUE ÇA.
C'est une fonction PURE

Print

Et pourquoi pas **print** ?
ord prend un argument de type **string**, retourne un argument de type **int** ET NE FAIT QUE ÇA.
C'est une fonction PURE


8
VALID
MORROW, JEROME
011010100-09564
ED GTACATGACTAAGTTAC EYES:BLUE TACCTGTCA
GNQ9.8612 = VALIDITY JE7342DAN

Print

```
In [34]: type(print(2))
2
Out[34]: NoneType

In [35]: print(print(2))
2
None

In [36]: print(print(2),print('GNU'))
2
GNU
None None
```

```
In [37]: [ print(a) for a in range(5) ]
0
1
2
3
4
Out[37]: [None, None, None, None, None]
```

```
In [38]: b = [ print(a) for a in range(5) ]
0
1
2
3
4

In [39]: b
Out[39]: [None, None, None, None, None]
```

print est IMPURE !

```
In [38]: b = [ print(a) for a in range(5) ]
0
1
2
3
4

In [39]: b
Out[39]: [None, None, None, None, None]
```

print est IMPURE !


IN-VALID
010011001-28253
TACATGACTAAGTTAC MYOPIA: TACCTGTCATT
GQ 3.4071 = DEFICIENCY LI
*SUSP. DE-GENE-ERATE

`print` renvoie toujours `None` mais a un EFFET SECONDAIRE (*side effect*) qui affiche quelque chose dans le monde extérieur.

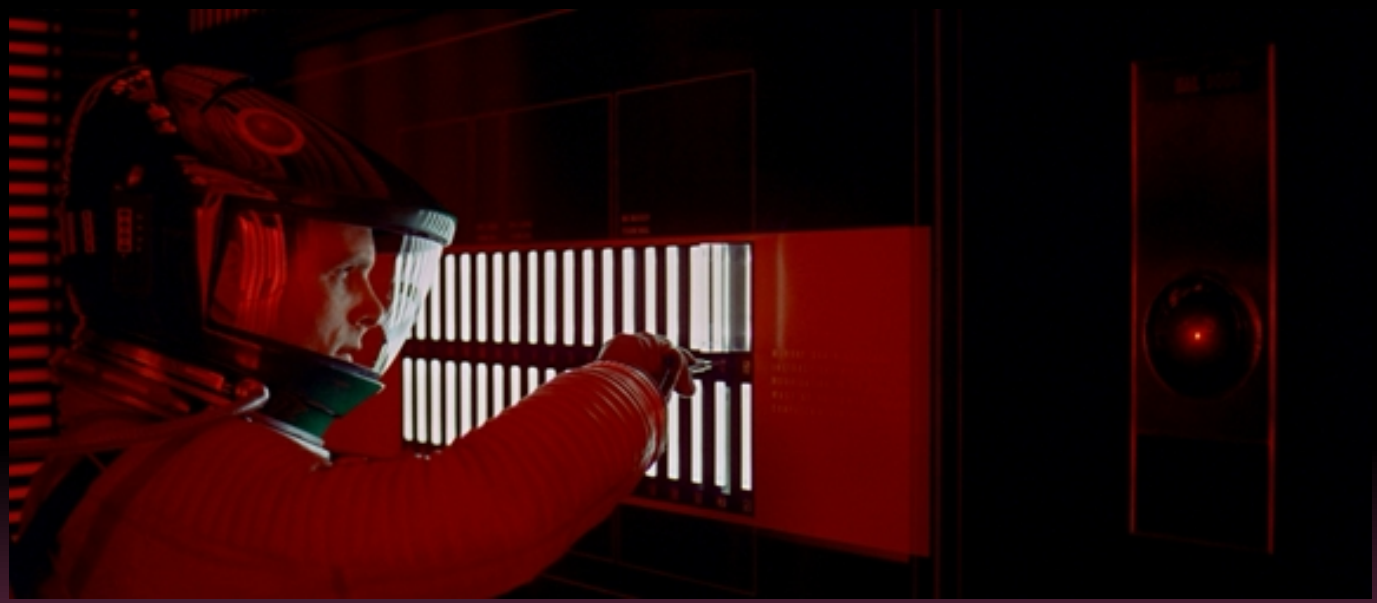
```python
def plus_chaine(x) :
    global a
    return x + ' ' + a
```

```
In [102]: a = 'Dave'

In [103]: plus_chaine('Bonjour')
Out[103]: 'Bonjour Dave'

In [104]: a = 'HAL'

In [105]: plus_chaine('Bonjour')
Out[105]: 'Bonjour HAL'
```

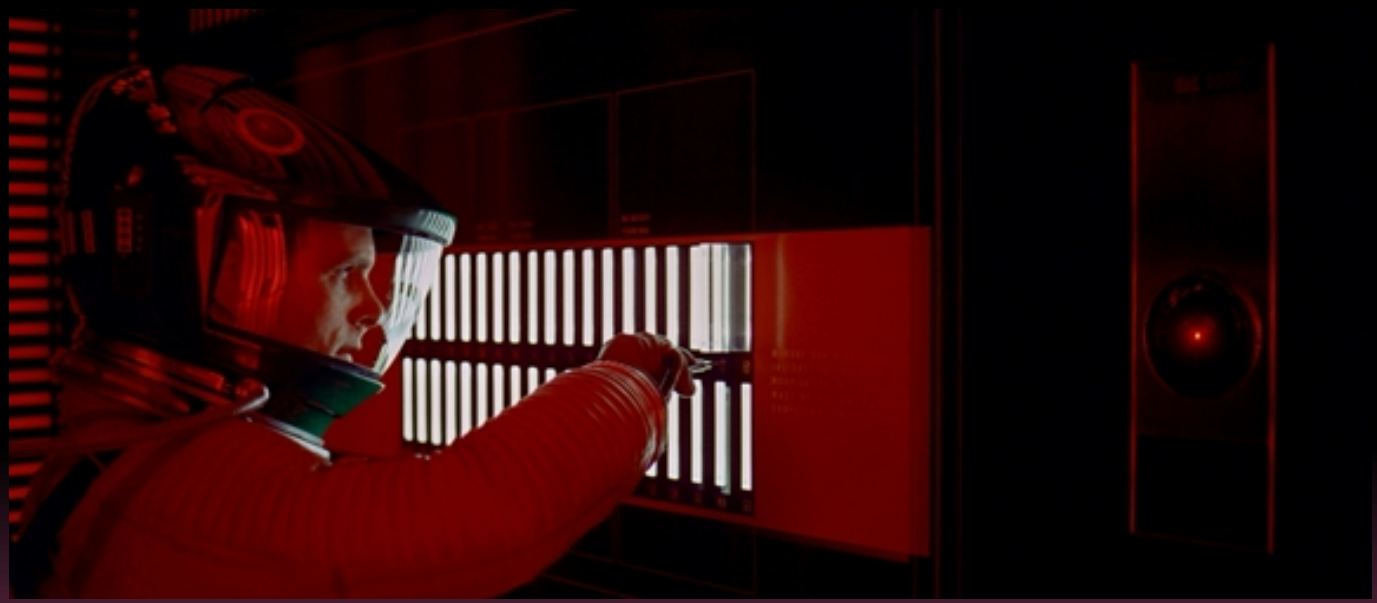


```
In [102]: a = 'Dave'

In [103]: plus_chaine('Bonjour')
Out[103]: 'Bonjour Dave'

In [104]: a = 'HAL'

In [105]: plus_chaine('Bonjour')
Out[105]: 'Bonjour HAL'
```




IN-VALID
010011001-28253
TACATGACTAAGTTAC MYOPIA: TACCTGTCATT
GQ 3.4071 = DEFICIENCY LI
*SUSP. DE-GENE-ERATE

http://golang.org

Rust

mozilla

Sommaire

Les fonctions sont des expressions comme les autres.

```
def transforme_une_lettre(transformation, lettre) :
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')
Out[119]: 'G'
```

```
def cesar(lettre) :
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')
Out[120]: 'j'
```

```
def transforme_une_lettre(transformation, lettre) :
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')
Out[119]: 'G'
```

```
def cesar(lettre) :
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')
Out[120]: 'j'
```

```
def transforme_une_lettre(transformation, lettre) :
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')
Out[119]: 'G'
```

```
def cesar(lettre) :
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')
Out[120]: 'j'
```

```
def transforme_une_lettre(transformation, lettre) :
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')
Out[119]: 'G'
```

```
def cesar(lettre) :
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')
Out[120]: 'j'
```

λ-expressions

$$x \mapsto 2x + 1$$

$$\lambda x \cdot 2x + 1$$

```
    lambda              x          :          2*x + 1
"Une fonction qui    prend x    et retourne    2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
Out[121]: 11
```

$$x \mapsto 2x + 1$$

$$\lambda x \cdot 2x + 1$$

```
    lambda              x          :          2*x + 1
"Une fonction qui    prend x   et retourne    2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
Out[121]: 11
```

$$x \mapsto 2x + 1$$

$$\lambda x \cdot 2x + 1$$

```
    lambda            x         :          2*x + 1
"Une fonction qui   prend x   et retourne    2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
Out[121]: 11
```

$$x \mapsto 2x + 1$$

$$\lambda x \cdot 2x + 1$$

```
    lambda              x          :           2*x + 1
"Une fonction qui    prend x    et retourne    2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
Out[121]: 11
```

```
In [1]: transforme_une_lettre(lambda c : chr( ord(c) + 5 ), 'a')
Out[1]: 'f'
```

```
def cesar(decalage) :
    return lambda lettre : chr( ord(lettre) + decalage )
```

```
In [127]: cesar(3)
Out[127]: <function __main__.cesar.<locals>.<lambda>>
```

```
In [128]: cesar(3)('a')
Out[128]: 'd'

In [129]: cesar(5)('a')
Out[129]: 'f'
```

```
def cesar(decalage) :
    return lambda lettre : chr( ord(lettre) + decalage )
```

```
In [127]: cesar(3)
Out[127]: <function __main__.cesar.<locals>.<lambda>>
```

```
In [128]: cesar(3)('a')
Out[128]: 'd'

In [129]: cesar(5)('a')
Out[129]: 'f'
```

```
def cesar(decalage) :
    return lambda lettre : chr( ord(lettre) + decalage )
```

```
In [127]: cesar(3)
Out[127]: <function __main__.cesar.<locals>.<lambda>>
```

```
In [128]: cesar(3)('a')
Out[128]: 'd'

In [129]: cesar(5)('a')
Out[129]: 'f'
```

Sommaire

Composition de fonctions

Défi ISI n° 1 :
On attendra dimanche 27 septembre ;-)

Dans un vrai langage de programmation :

```haskell
(.) :: (b -> c) -> (a -> b) -> a -> c
f . g = \x -> f (g x)
```

Défi

Écrire une fonction qui renvoie la longueur du mot le plus long d'une chaîne de caractères donnée en argument.

```
In [1]: long_mot_plus_long("Vos beaux yeux d'amour mourir me font Belle
          ↪   Marquise")
Out[1]: 8
```

- Il faudrait une fonction qui éclate la phrase en liste de mots
- Il faudrait une fonction qui renvoie la liste des longueurs de ces mots
- Il faudrait une fonction qui calcule le maximum des éléments d'une liste

- Il faudrait une fonction qui éclate la phrase en liste de mots
- Il faudrait une fonction qui renvoie la liste des longueurs de ces mots
- Il faudrait une fonction qui calcule le maximum des éléments d'une liste

- Il faudrait une fonction qui éclate la phrase en liste de mots
- Il faudrait une fonction qui renvoie la liste des longueurs de ces mots
- Il faudrait une fonction qui calcule le maximum des éléments d'une liste

$$\texttt{String} \xrightarrow{\texttt{eclate}} [\texttt{String}] \xrightarrow{\texttt{longueurs}} [\texttt{Int}] \xrightarrow{\texttt{maximum}} \texttt{Int}$$

$$\texttt{s} \xrightarrow{\texttt{eclate}} \texttt{eclate s} \xrightarrow{\texttt{longueurs}} \texttt{longueurs (eclate s)} \xrightarrow{\texttt{maximum}} \texttt{maximum (longueurs (eclate s))}$$

$$\texttt{String} \xrightarrow{\texttt{maximum} \circ \texttt{longueurs} \circ \texttt{eclate}} \texttt{Int}$$

$$\texttt{String} \xrightarrow{\texttt{eclate} \mid \texttt{longueurs} \mid \texttt{maximum}} \texttt{Int}$$

$$\texttt{String} \xrightarrow{\texttt{eclate}} [\texttt{String}] \xrightarrow{\texttt{longueurs}} [\texttt{Int}] \xrightarrow{\texttt{maximum}} \texttt{Int}$$

$$\texttt{s} \xrightarrow{\texttt{eclate}} \texttt{eclate s} \xrightarrow{\texttt{longueurs}} \texttt{longueurs (eclate s)} \xrightarrow{\texttt{maximum}} \texttt{maximum (longueurs (eclate s))}$$

$$\texttt{String} \xrightarrow{\texttt{maximum} \circ \texttt{longueurs} \circ \texttt{eclate}} \texttt{Int}$$

$$\texttt{String} \xrightarrow{\texttt{eclate} \mid \texttt{longueurs} \mid \texttt{maximum}} \texttt{Int}$$

$$\texttt{String} \xrightarrow{\texttt{eclate}} [\texttt{String}] \xrightarrow{\texttt{longueurs}} [\texttt{Int}] \xrightarrow{\texttt{maximum}} \texttt{Int}$$

$$\texttt{s} \xrightarrow{\texttt{eclate}} \texttt{eclate s} \xrightarrow{\texttt{longueurs}} \texttt{longueurs (eclate s)} \xrightarrow{\texttt{maximum}} \texttt{maximum (longueurs (eclate s))}$$

$$\texttt{String} \xrightarrow{\texttt{maximum} \circ \texttt{longueurs} \circ \texttt{eclate}} \texttt{Int}$$

$$\texttt{String} \xrightarrow{\texttt{eclate} \mid \texttt{longueurs} \mid \texttt{maximum}} \texttt{Int}$$

$$\texttt{String} \xrightarrow{\texttt{eclate}} [\texttt{String}] \xrightarrow{\texttt{longueurs}} [\texttt{Int}] \xrightarrow{\texttt{maximum}} \texttt{Int}$$

$$\texttt{s} \xrightarrow{\texttt{eclate}} \texttt{eclate s} \xrightarrow{\texttt{longueurs}} \texttt{longueurs (eclate s)} \xrightarrow{\texttt{maximum}} \texttt{maximum (longueurs (eclate s))}$$

$$\texttt{String} \xrightarrow{\texttt{maximum} \circ \texttt{longueurs} \circ \texttt{eclate}} \texttt{Int}$$

$$\texttt{String} \xrightarrow{\texttt{eclate} \mid \texttt{longueurs} \mid \texttt{maximum}} \texttt{Int}$$

```
def long_mot_plus_long(cs) :
   return maximum( longueurs( eclate( cs ) ) )
```

```
In [14]: eclate("Vos beaux yeux")
Out[14]: ['Vos', 'beaux', 'yeux']
```

```python
def eclate(chaine) :
    return chaine.split()
```

```
In [14]: eclate("Vos beaux yeux")
Out[14]: ['Vos', 'beaux', 'yeux']
```

```
def eclate(chaine) :
    return chaine.split()
```

```
In [15]: longueurs(['Vos', 'beaux', 'yeux'])
Out[15]: [3, 5, 4]
```

```python
def longueurs(xs) :
    return [ len(x) for x in xs ]
```

```
In [15]: longueurs(['Vos', 'beaux', 'yeux'])
Out[15]: [3, 5, 4]
```

```python
def longueurs(xs) :
    return [ len(x) for x in xs ]
```

```python
def maximum(xs) :
    maxi = xs[0]
    for elmt in xs[1:] :
        if elmt > maxi :
            maxi = elmt
    return maxi
```

```python
def eclate(chaine) :
    return chaine.split()

def longueurs(xs) :
    return [ len(x) for x in xs ]

def maximum(xs) :
    maxi = xs[0]
    for elmt in xs[1:] :
        if elmt > maxi :
            maxi = elmt
    return maxi

def long_mot_plus_long(cs) :
    return maximum(longueurs(eclate(cs)))
```

```python
def compose2(f,g) :
    return lambda x: f(g(x))
```

```python
compose2 = lambda f,g : lambda x : f(g(x))
```

```
In [28]: compose2(longueurs,eclate)("un deux trois")
Out[28]: [2, 4, 5]
```

```python
long_mot_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

```python
def compose2(f,g) :
    return lambda x: f(g(x))
```

```python
compose2 = lambda f,g : lambda x : f(g(x))
```

```
In [28]: compose2(longueurs,eclate)("un deux trois")
Out[28]: [2, 4, 5]
```

```python
long_mot_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

```python
def compose2(f,g) :
    return lambda x: f(g(x))
```

```python
compose2 = lambda f,g : lambda x : f(g(x))
```

```
In [28]: compose2(longueurs,eclate)("un deux trois")
Out[28]: [2, 4, 5]
```

```python
long_mot_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

```
def compose2(f,g) :
    return lambda x: f(g(x))
```

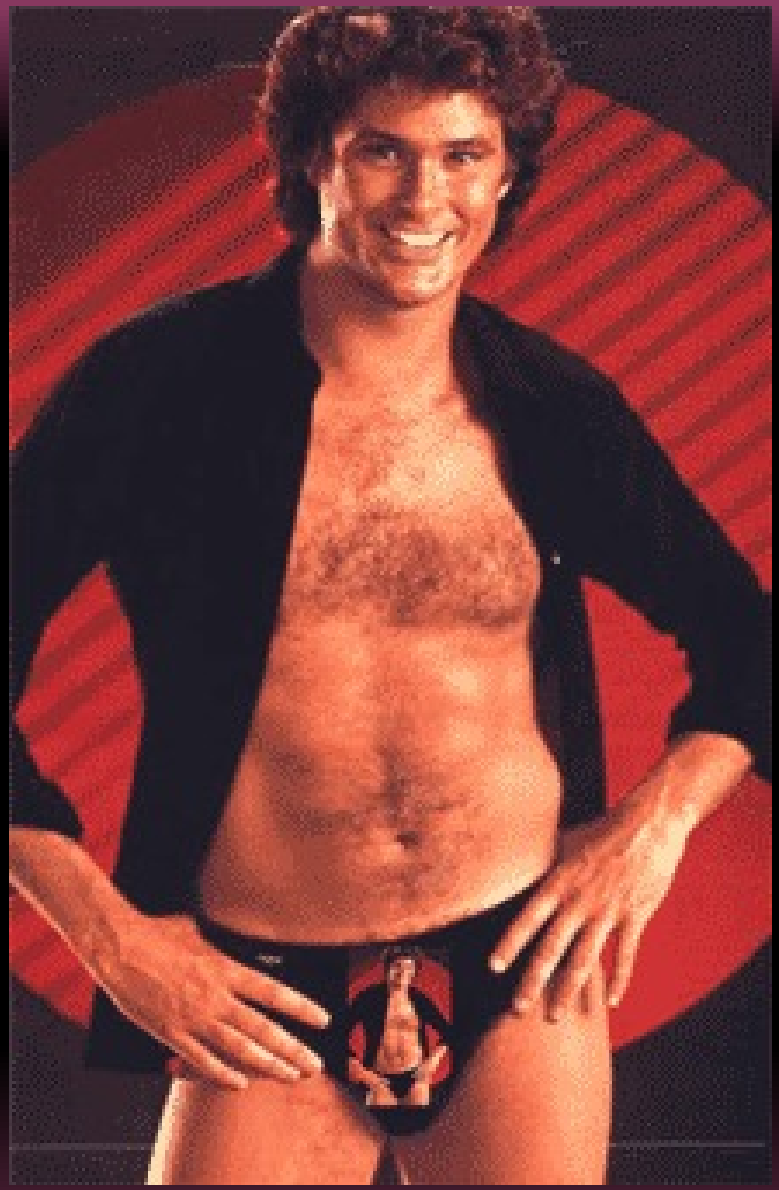
```
compose2 = lambda f,g : lambda x : f(g(x))
```

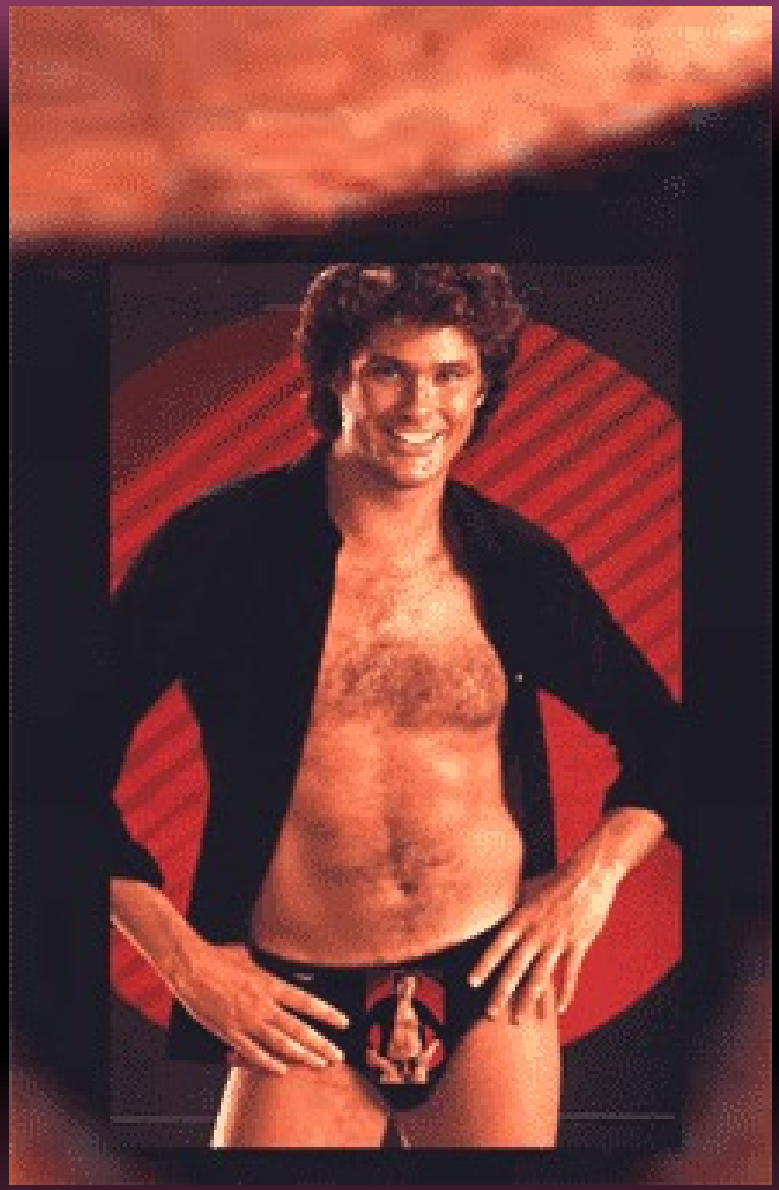
```
In [28]: compose2(longueurs,eclate)("un deux trois")
Out[28]: [2, 4, 5]
```

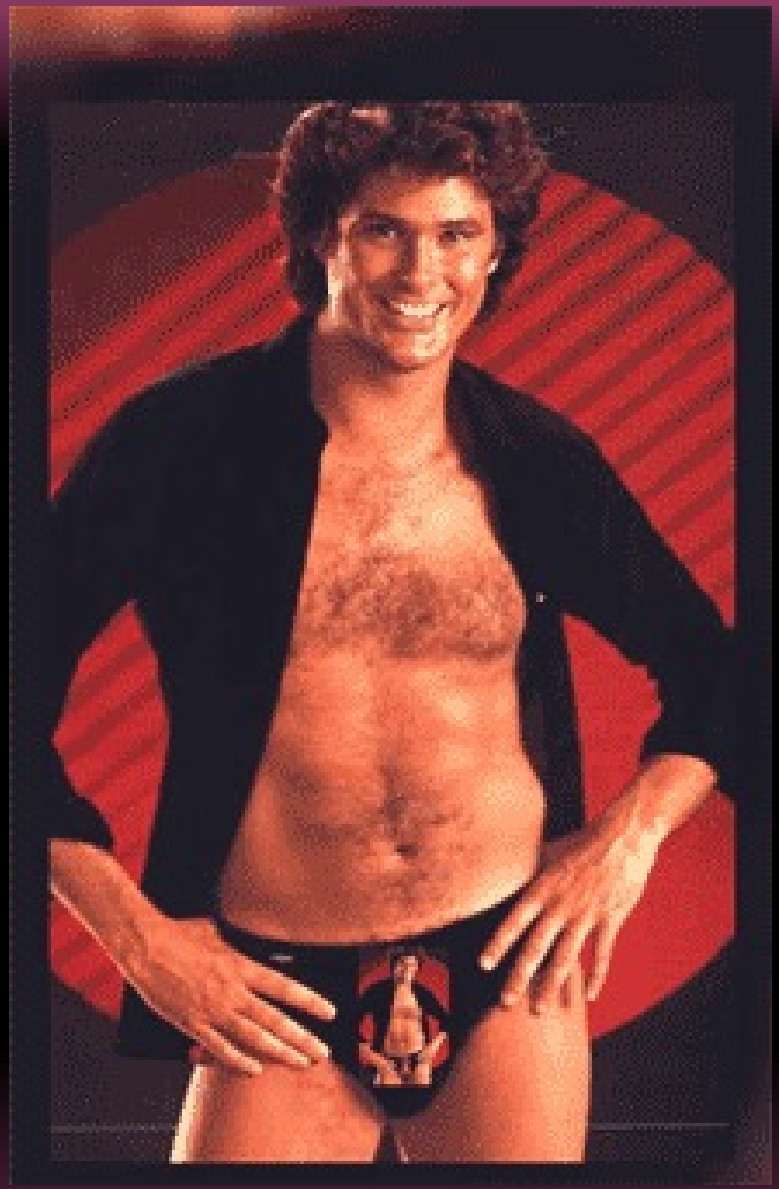
```
long_mot_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

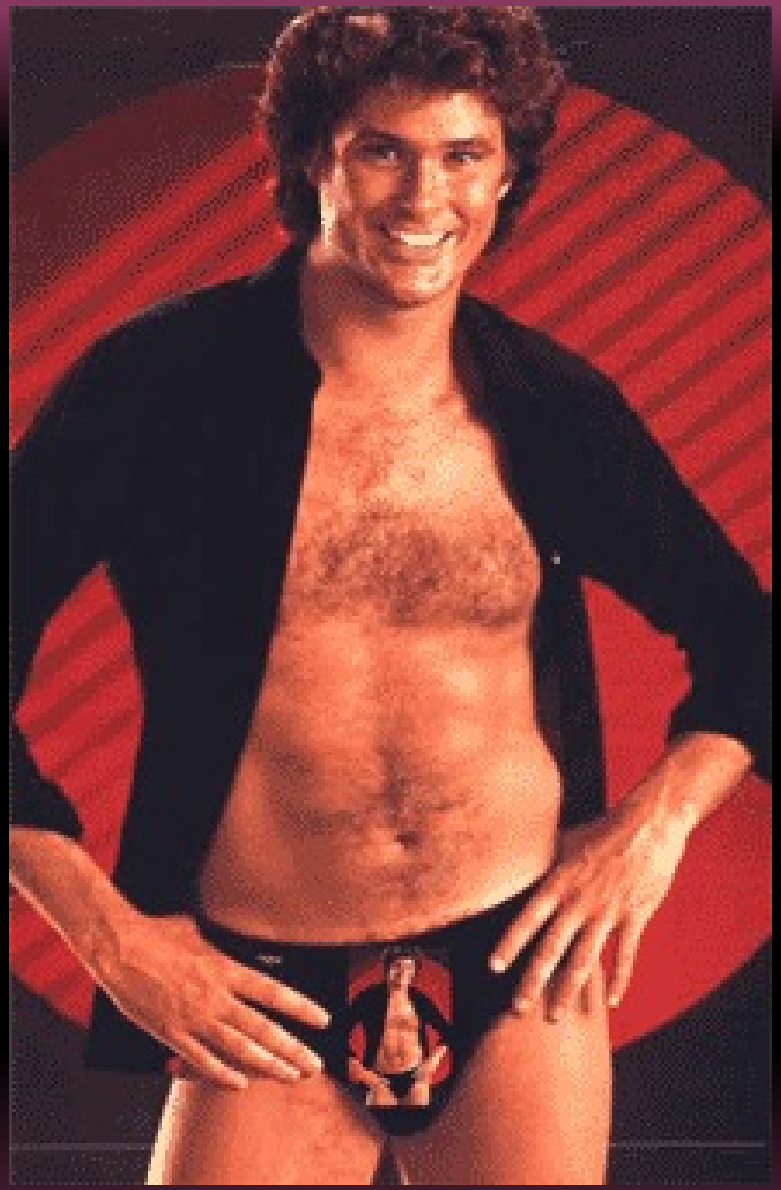
Sommaire

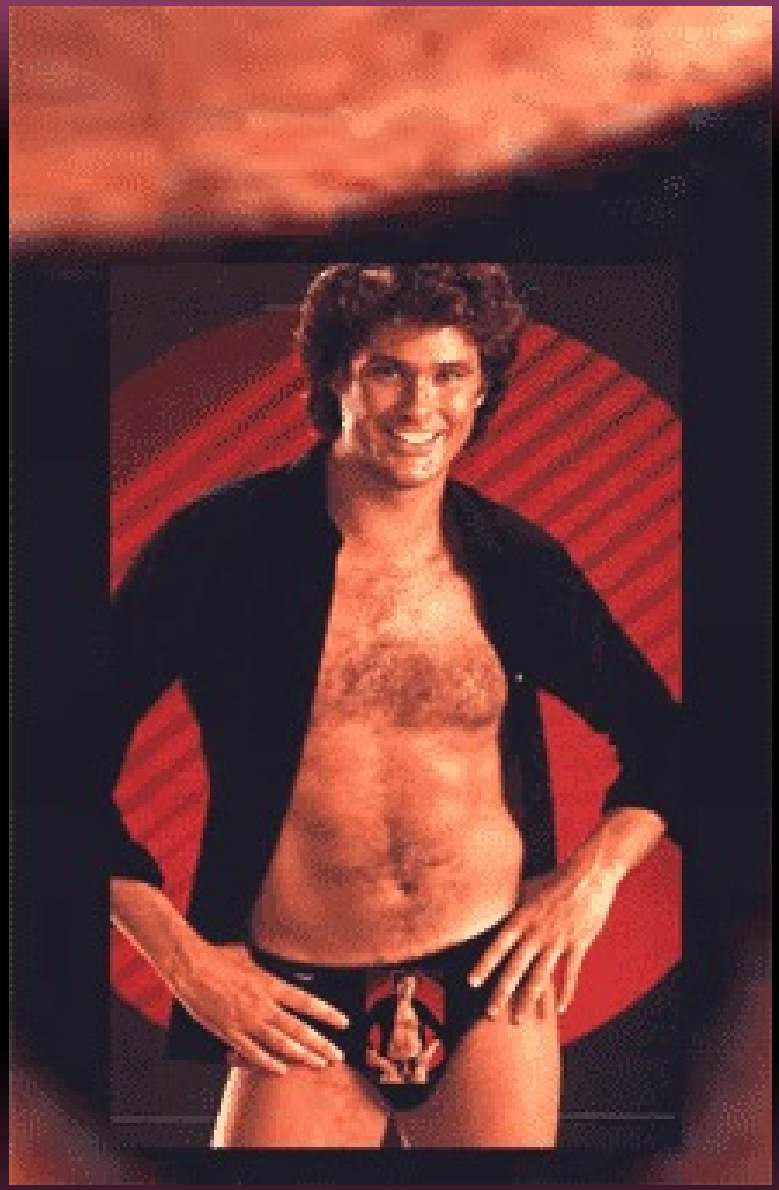

La Vache qui rit

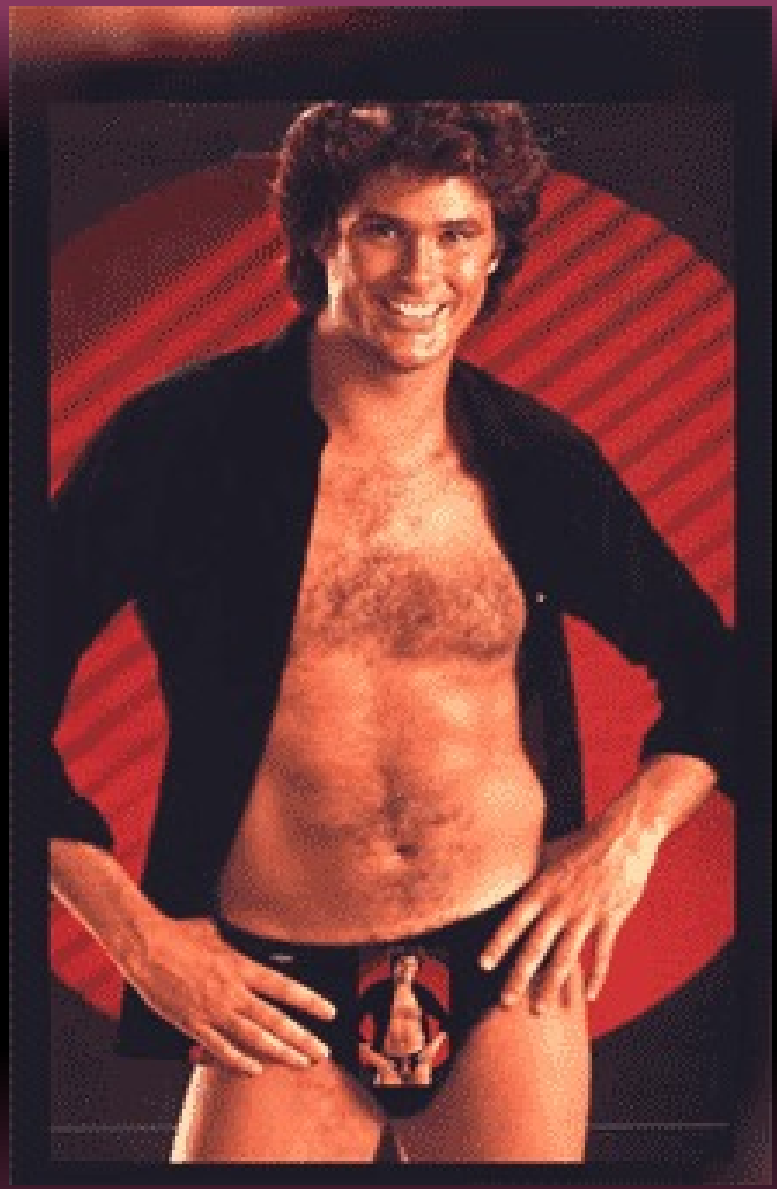


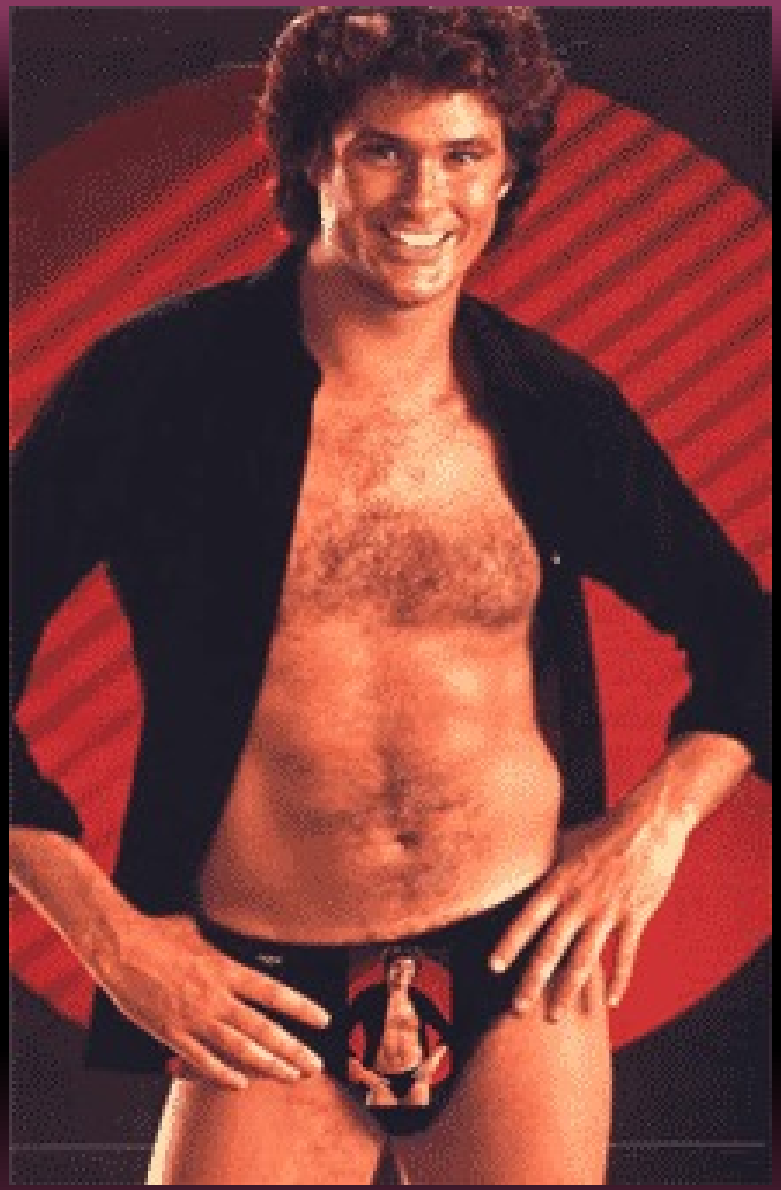


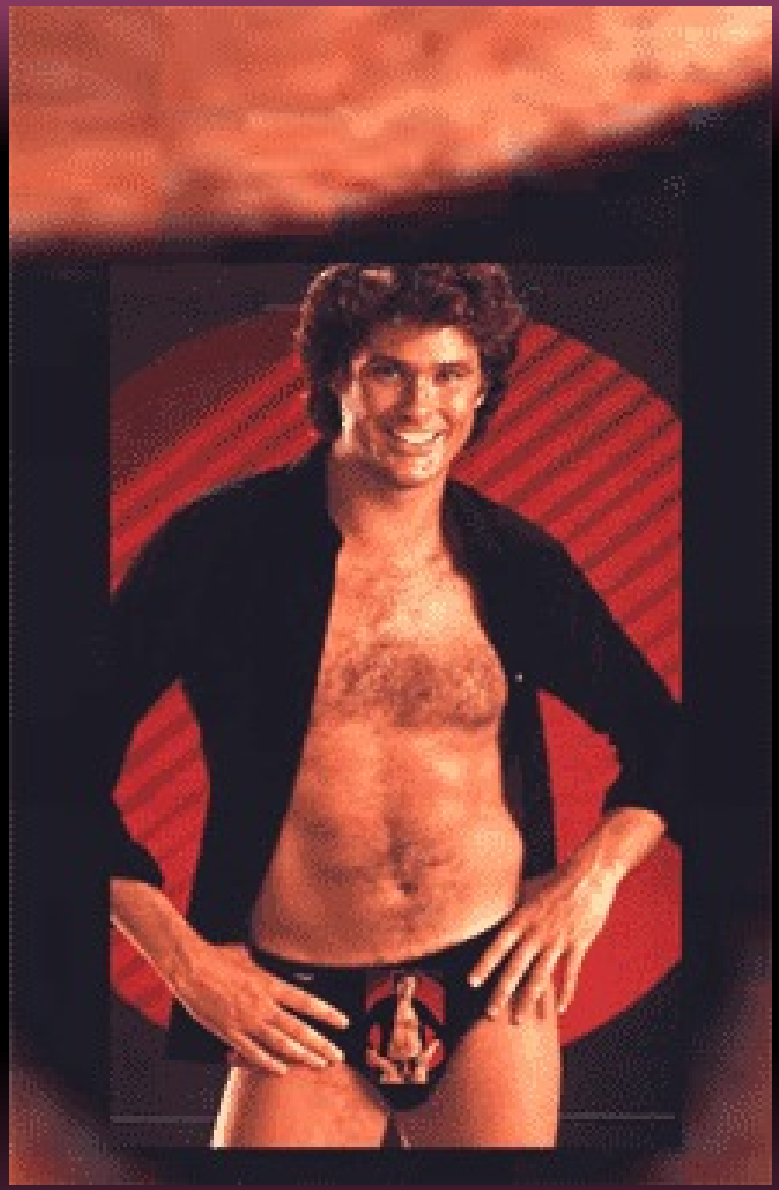


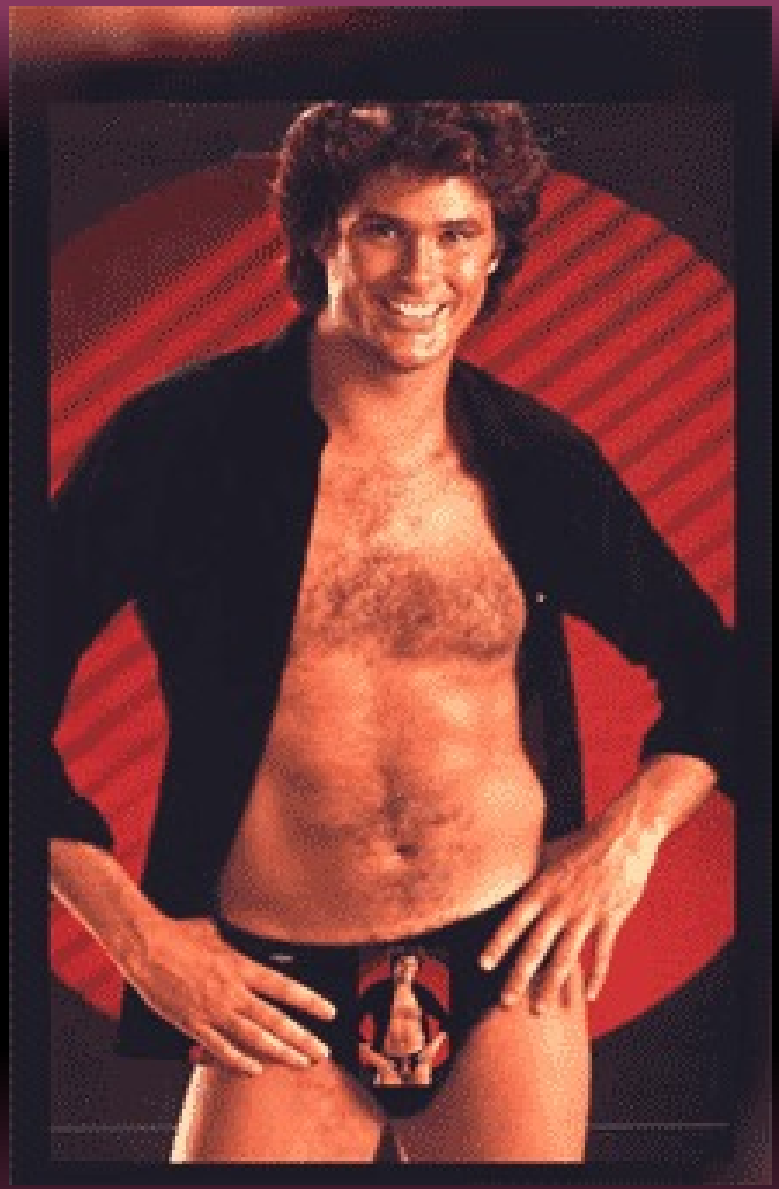


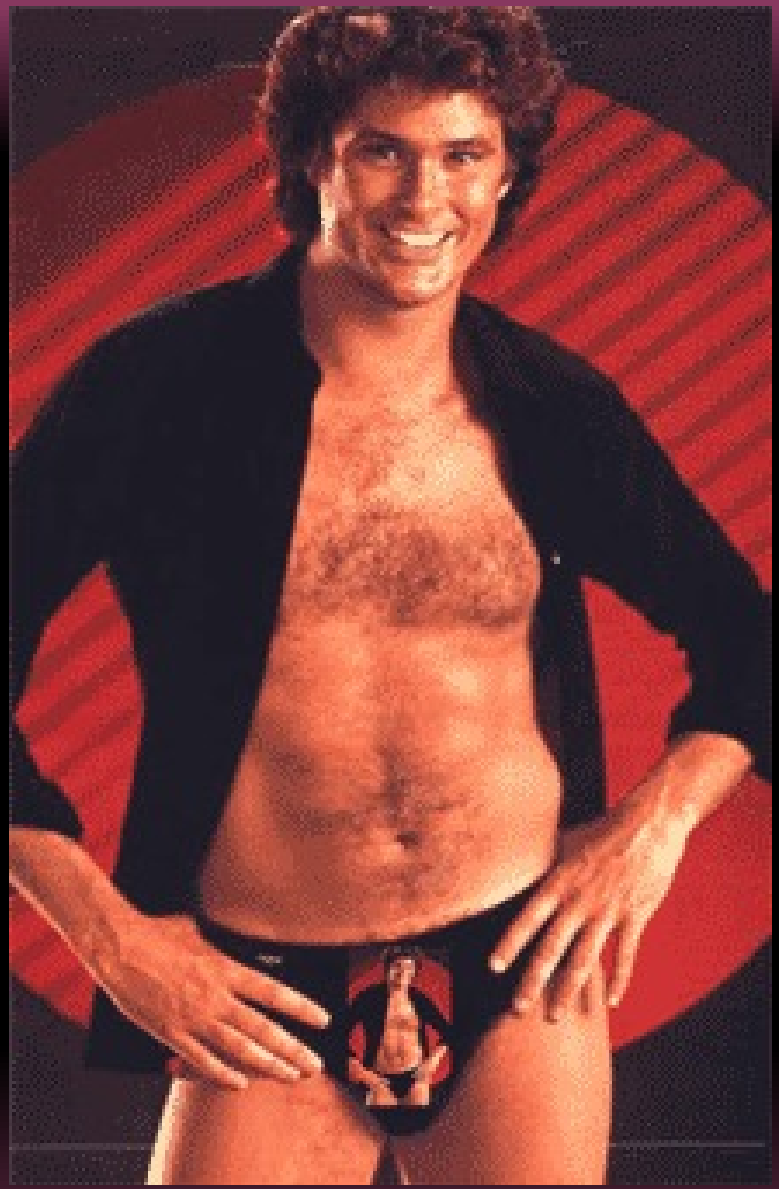


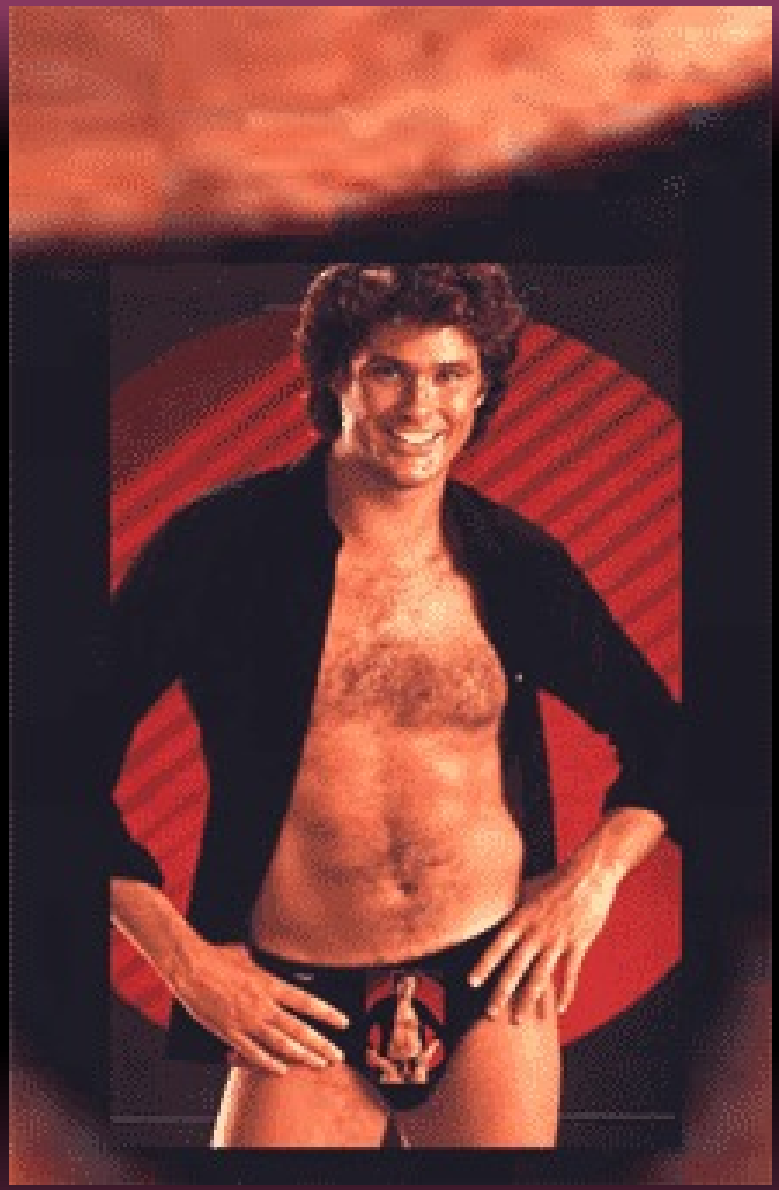


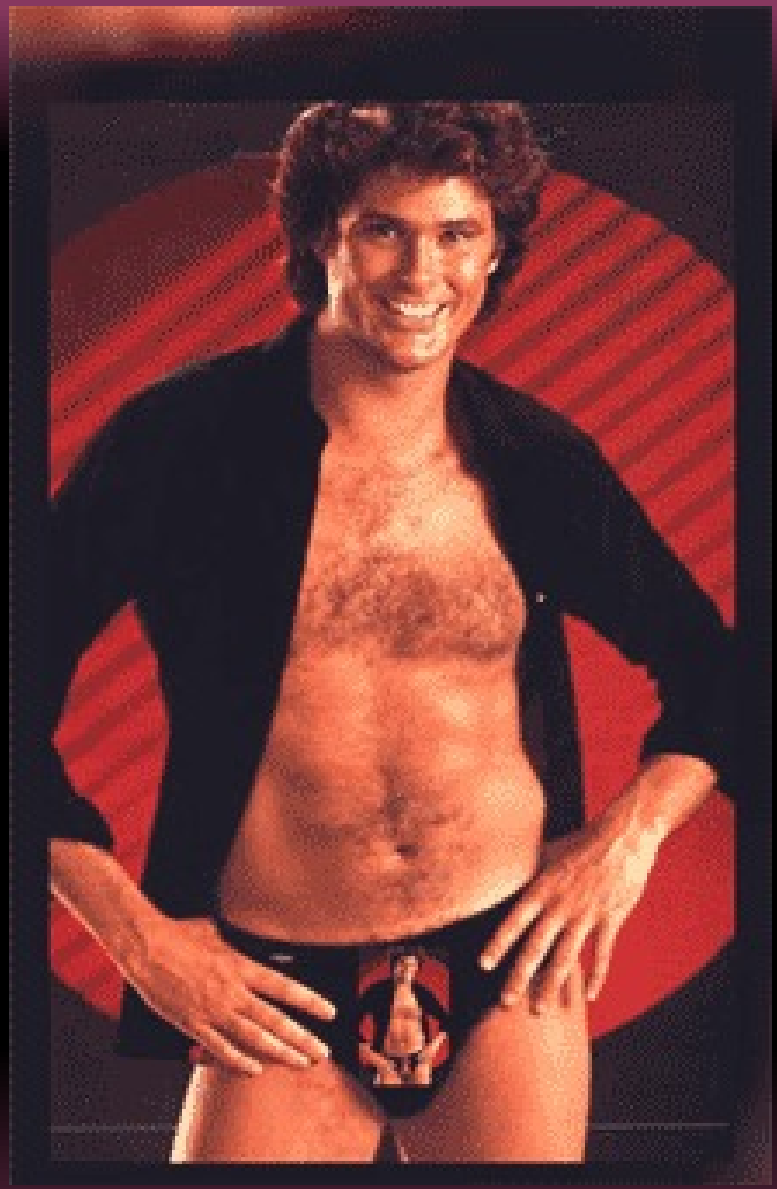












- induction
- induction mathématique (raisonnement par récurrence)
- fonction récurive et type récursif

- induction
- induction mathématique (raisonnement par récurrence)
- fonction récurive et type récursif

- induction
- induction mathématique (raisonnement par récurrence)
- fonction récurive et type récursif

Le retour de Max

```python
def max2(a,b) :
    return a if a >= b else b
```

```python
def maxi(xs) :
    assert xs != [], "Liste vide !"
    if len(xs) == 1 :
        return xs[0]
    return max2( xs[0], maxi(xs[1:]) )
```

```python
def max2(a,b) :
    return a if a >= b else b
```

```python
def maxi(xs) :
    assert xs != [], "Liste vide !"
    if len(xs) == 1 :
        return xs[0]
    return max2( xs[0], maxi(xs[1:]) )
```

```
In [54]: maxi("agfdjuol")
maxi <- ('agfdjuol',)
 maxi <- ('gfdjuol',)
  maxi <- ('fdjuol',)
   maxi <- ('djuol',)
    maxi <- ('juol',)
     maxi <- ('uol',)
      maxi <- ('ol',)
       maxi <- ('l',)
       maxi -> l
      maxi -> o
     maxi -> u
    maxi -> u
   maxi -> u
  maxi -> u
 maxi -> u
maxi -> u
Out[54]: 'u'
```

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

... ???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

$\cdots$???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

$\cdots$???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

$\cdots$???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

```python
def pow1(a, n) :
    assert n >= 0, "L'exposant doit être entier naturel"
    if n == 0 :
        return 1
    else :
        return a * pow1(a, n - 1)
```

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```python
def pow2(a, n) :
    assert n >= 0, "L'exposant doit être entier naturel"
    def accumule(k, acc) :
        if k == 0 :
            return acc
        else :
            return accumule(k - 1, a*acc)
    return accumule(n, 1)
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1,4)
accumule(0,8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1,4)
accumule(0,8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1,4)
accumule(0,8)
8
```

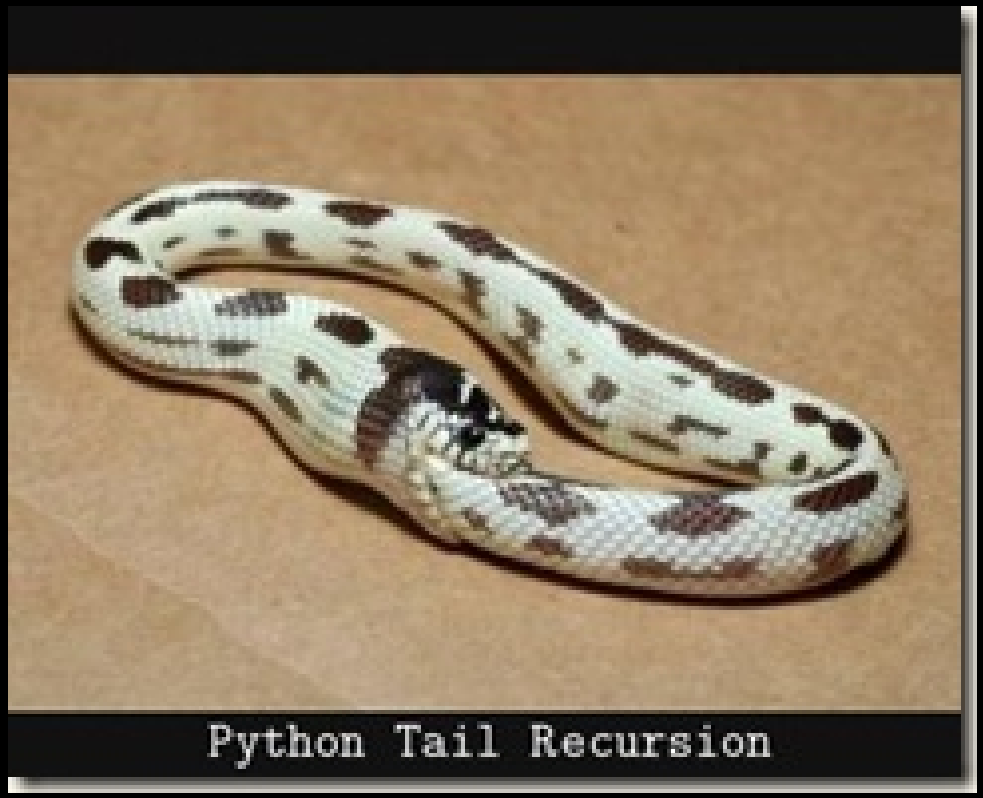
```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1,4)
accumule(0,8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1,4)
accumule(0,8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1,4)
accumule(0,8)
8
```

```
In [24]: pow1(2,3)
pow1 <- (2, 3)
 pow1 <- (2, 2)
  pow1 <- (2, 1)
   pow1 <- (2, 0)
   pow1 -> 1
  pow1 -> 2
 pow1 -> 4
pow1 -> 8
Out[24]: 8
```

```
In [27]: pow2(2,3)
pow2 <- (2, 3)
accumule <- (3, 1)
 accumule <- (2, 2)
  accumule <- (1, 4)
   accumule <- (0, 8)
   accumule -> 8
  accumule -> 8
 accumule -> 8
accumule -> 8
pow2 -> 8
Out[27]: 8
```

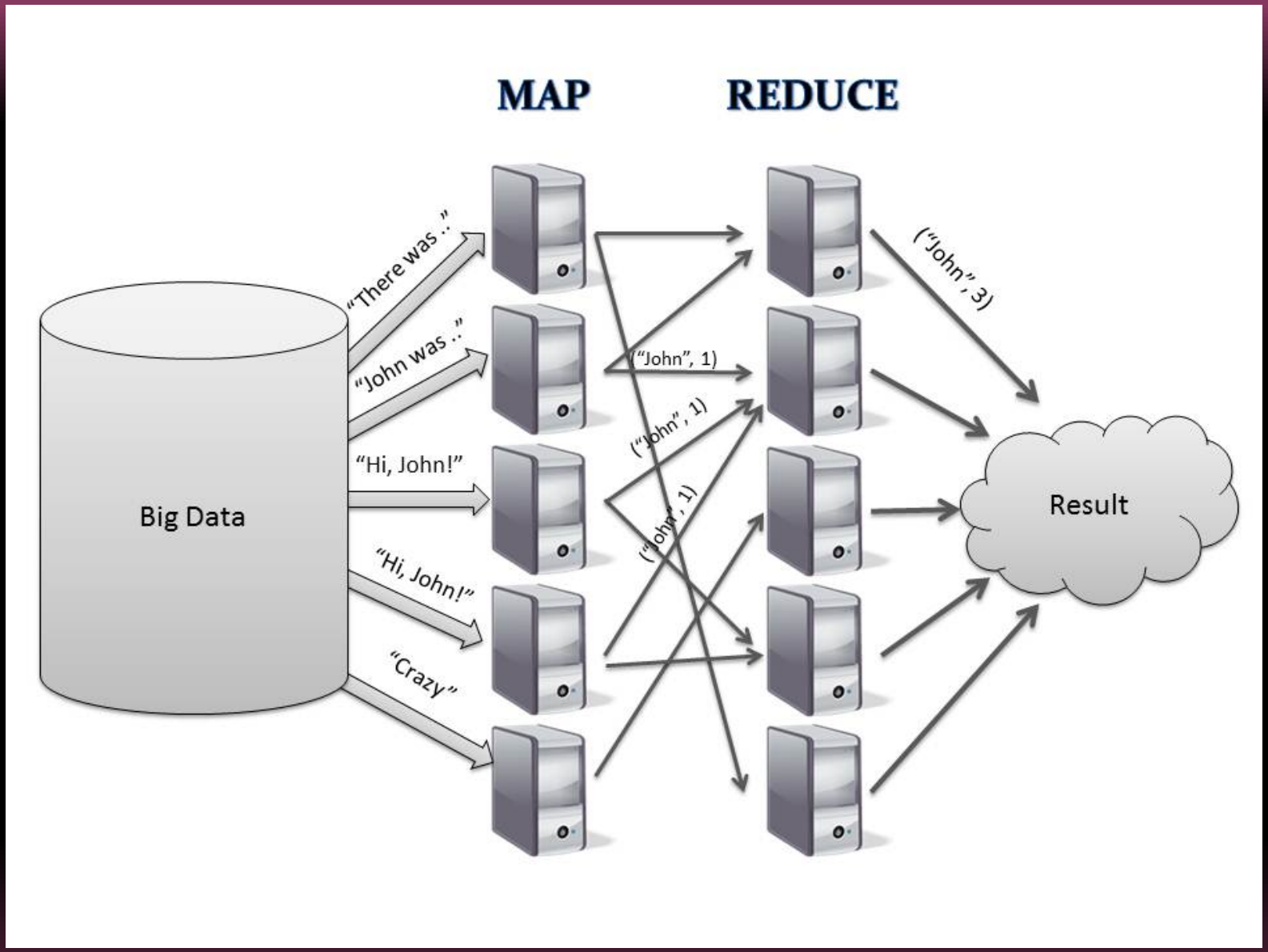

Python Tail Recursion

```python
def pow3(a, n) :
    assert n >= 0, "L'exposant doit être entier naturel"
    accu = 1
    k = 0
    while k < n :
        accu *= a
        k += 1
    return accu
```

```python
def pow4(a, n) :
    assert n >= 0, "L'exposant doit être un entier naturel"
    accu = 1
    for k in range(n) :
        accu = a*accu
    return accu
```

```python
from functools import reduce

def pow5(a, n):
    return reduce(lambda accu, k : a*accu, range(n), 1)
```


MAP
REDUCE
Big Data
"There was .."
"John was .."
"Hi, John!"
"Hi, John!"
"Crazy"
("John", 1)
("John", 1)
("John", 1)
("John", 3)
Result

Le secret de la réduction

Le secret de la réduction

$$f(f(f(\cdots f(f(accu0, x_0), x_1)\cdots), x_{n-1}), x_n)$$

```
def reduit(f, xs, accu0) :
    accu = accu0
    for x in xs :
        accu = f(accu, x)
    return accu
```

```
reduce(f, xs, accu0)
```

Le secret de la réduction

$$f(f(f(\cdots f(f(accu0, x_0), x_1)\cdots), x_{n-1}), x_n)$$

```
def reduit(f, xs, accu0) :
    accu = accu0
    for x in xs :
        accu = f(accu, x)
    return accu
```

```
reduce(f, xs, accu0)
```

Le secret de la réduction

$$f(f(f(\cdots f(f(accu0, x_0), x_1)\cdots), x_{n-1}), x_n)$$

```
def reduit(f, xs, accu0) :
    accu = accu0
    for x in xs :
        accu = f(accu, x)
    return accu
```

```
reduce(f, xs, accu0)
```

Le retour de Max

```
maxi = lambda xs: reduce(lambda max_tmp, x: x if x > max_tmp else max_tmp, xs)
```

```
max2 = lambda a, b: a if a >= b else b

maxi = lambda xs: reduce(max2, xs)
```

```
def max2(a,b) :
    return a if a >= b else b

def maxi(xs) :
    return reduce(max2, xs)
```

```
maxi = lambda xs: reduce(lambda max_tmp, x: x if x > max_tmp else max_tmp, xs)
```

```
max2 = lambda a, b: a if a >= b else b

maxi = lambda xs: reduce(max2, xs)
```

```
def max2(a,b) :
    return a if a >= b else b

def maxi(xs) :
    return reduce(max2, xs)
```

```
maxi = lambda xs: reduce(lambda max_tmp, x: x if x > max_tmp else max_tmp, xs)
```

```
max2 = lambda a, b: a if a >= b else b

maxi = lambda xs: reduce(max2, xs)
```

```
def max2(a,b) :
    return a if a >= b else b

def maxi(xs) :
    return reduce(max2, xs)
```

```
In [23]: maxi([1,12,5,47,86,9,4])
Out[23]: 86

In [24]: maxi([])
TypeError: reduce() of empty sequence with no initial value
```

Défi

Construisez une fonction prend une liste d'entiers et renvoie le produit des carrés des nombres pairs de cette liste. Une réponse en une petite ligne serait appréciée...

Défi

Construisez une fonction compose qui renvoie la composée d'une suite de fonctions :

```
In [35]: compose(maximum,longueurs,eclate)("Trois deux un")
Out[35]: 5
```

Défi

Construisez une fonction prend une liste d'entiers et renvoie le produit des carrés des nombres pairs de cette liste. Une réponse en une petite ligne serait appréciée...

Défi

Construisez une fonction compose qui renvoie la composée d'une suite de fonctions :

```
In [35]: compose(maximum,longueurs,eclate)("Trois deux un")
Out[35]: 5
```

```
In [29]: %timeit pow5(17,100)
100000 loops, best of 3: 12.3 µs per loop

In [30]: %timeit pow4(17,100)
100000 loops, best of 3: 6.67 µs per loop

In [31]: %timeit pow3(17,100)
100000 loops, best of 3: 9.24 µs per loop

In [32]: %timeit pow2(17,100)
10000 loops, best of 3: 18.6 µs per loop

In [33]: %timeit pow1(17,100)
10000 loops, best of 3: 17.8 µs per loop
```

```python
def pow6(a, n):
    if n == 0 :
        return 1
    if n == 1 :
        return a
    if n % 2 == 0 :
        return pow6(a*a, n//2)
    return a * pow6(a, n-1)
```

```
In [42]: %timeit pow6(17,100)
100000 loops, best of 3: 2.35 µs per loop
```

```python
def pow6(a, n):
    if n == 0 :
        return 1
    if n == 1 :
        return a
    if n % 2 == 0 :
        return pow6(a*a, n//2)
    return a * pow6(a, n-1)
```

```
In [42]: %timeit pow6(17,100)
100000 loops, best of 3: 2.35 µs per loop
```

```
In [45]: pow6(2,10)
pow6 <- (2, 10)
 pow6 <- (4, 5)
  pow6 <- (4, 4)
   pow6 <- (16, 2)
    pow6 <- (256, 1)
    pow6 -> 256
   pow6 -> 256
  pow6 -> 256
 pow6 -> 1024
pow6 -> 1024
Out[45]: 1024
```

Défi

Dérécursifiez la fonction `pow6`


ROMA
ROMA

OCaml

```
let rec pow a = function
    | 1 -> 1
    | n -> a * (pow a (n - 1)) ;;
```

```
let powt a n =
   let rec pow_iter acc k =
       if k == 0 then acc
       else pow_iter (a*acc) (k-1)
   in pow_iter 1 n ;;
```

```
# pow 17 100_000;;
Stack overflow during evaluation (looping recursion?).

# powt 17 100_000;;
- : int = -573761023
```

Sommaire

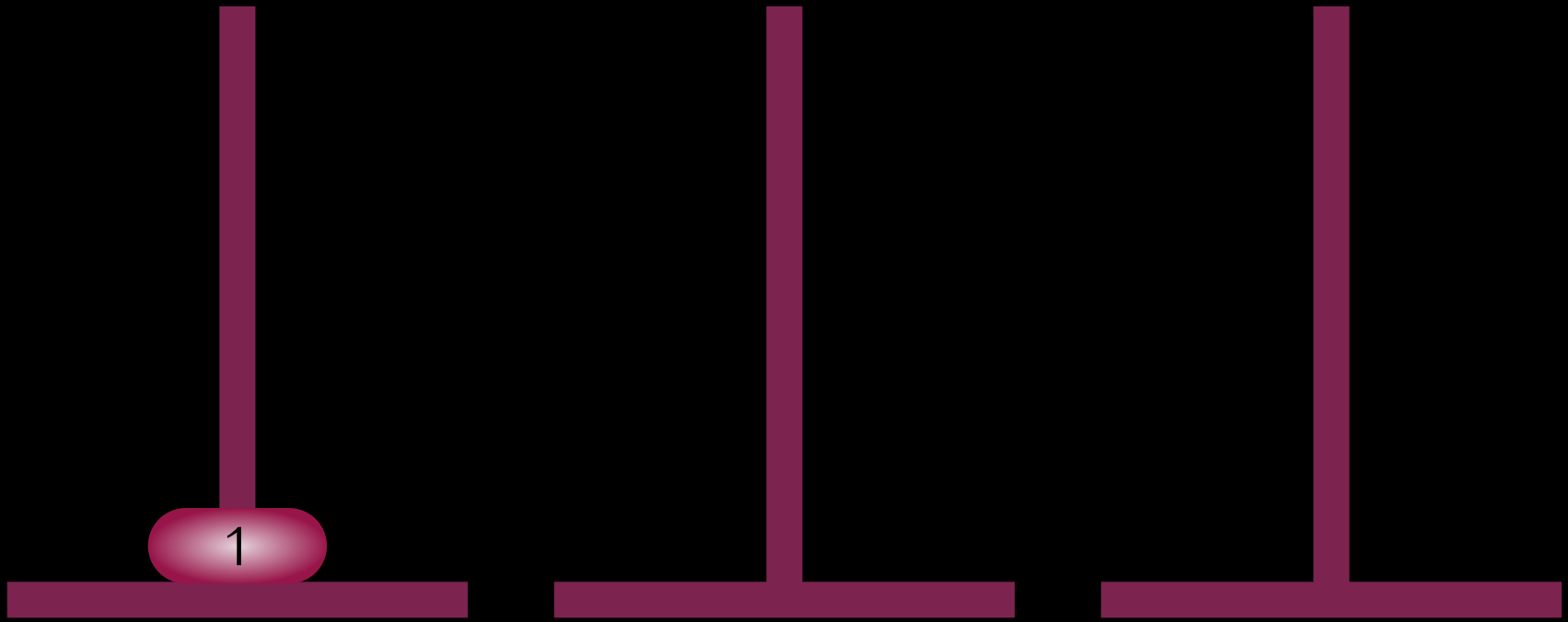
Pour prouver qu'une propriété P_n dépendant uniquement d'un paramètre n est vraie pour tout $n \geqslant n_0$, il faut vérifier que :

- P_{n_0} est vraie (on parle parfois d'initialisation) ;
- pour tout $n \geqslant n_0$, $P_n \rightarrow P_{n+1}$ (on parle parfois d'hérédité).

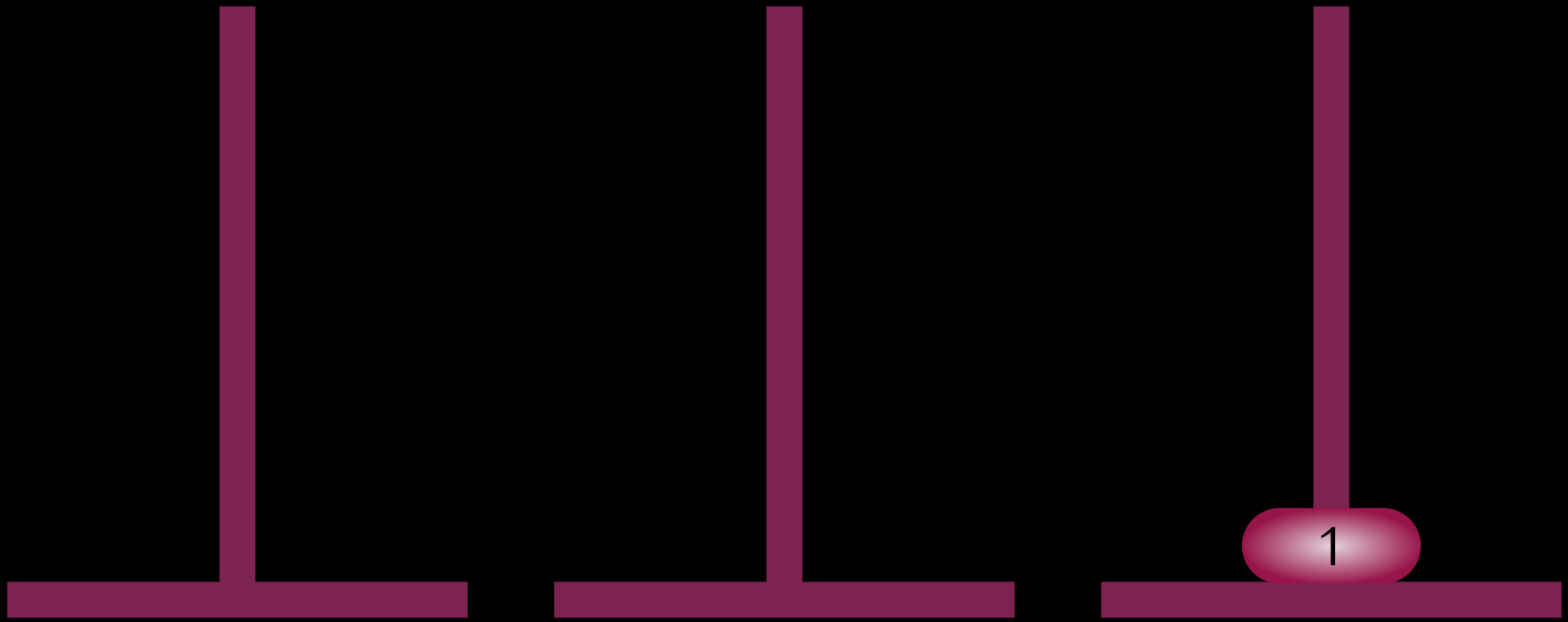
Pour prouver qu'une propriété P_n dépendant uniquement d'un paramètre n est vraie pour tout $n \geqslant n_0$, il faut vérifier que :

- P_{n_0} est vraie (on parle parfois d'initialisation) ;
- pour tout $n \geqslant n_0$, $P_n \rightarrow P_{n+1}$ (on parle parfois d'hérédité).

Tours de Hanoi – 1 Disque

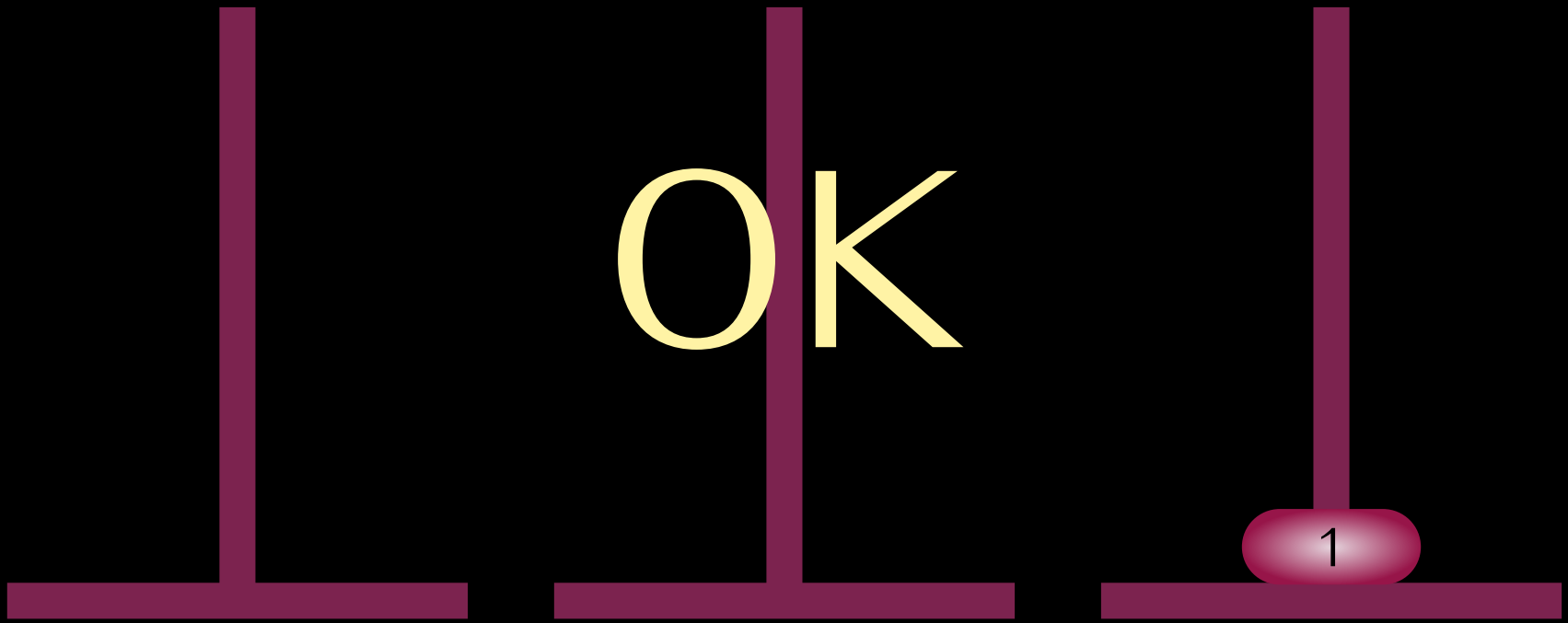


Tours de Hanoi – 1 Disque

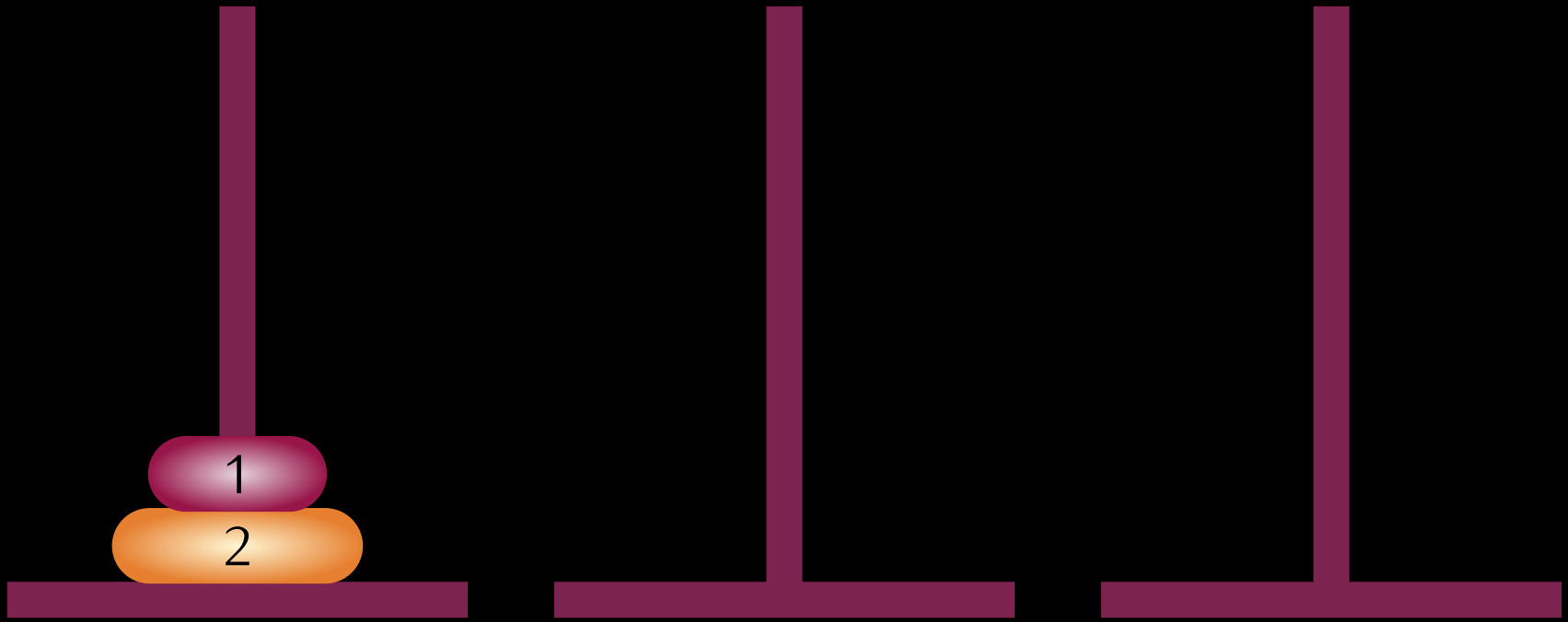


On déplace le disque de la tige 1 vers la tige 3.

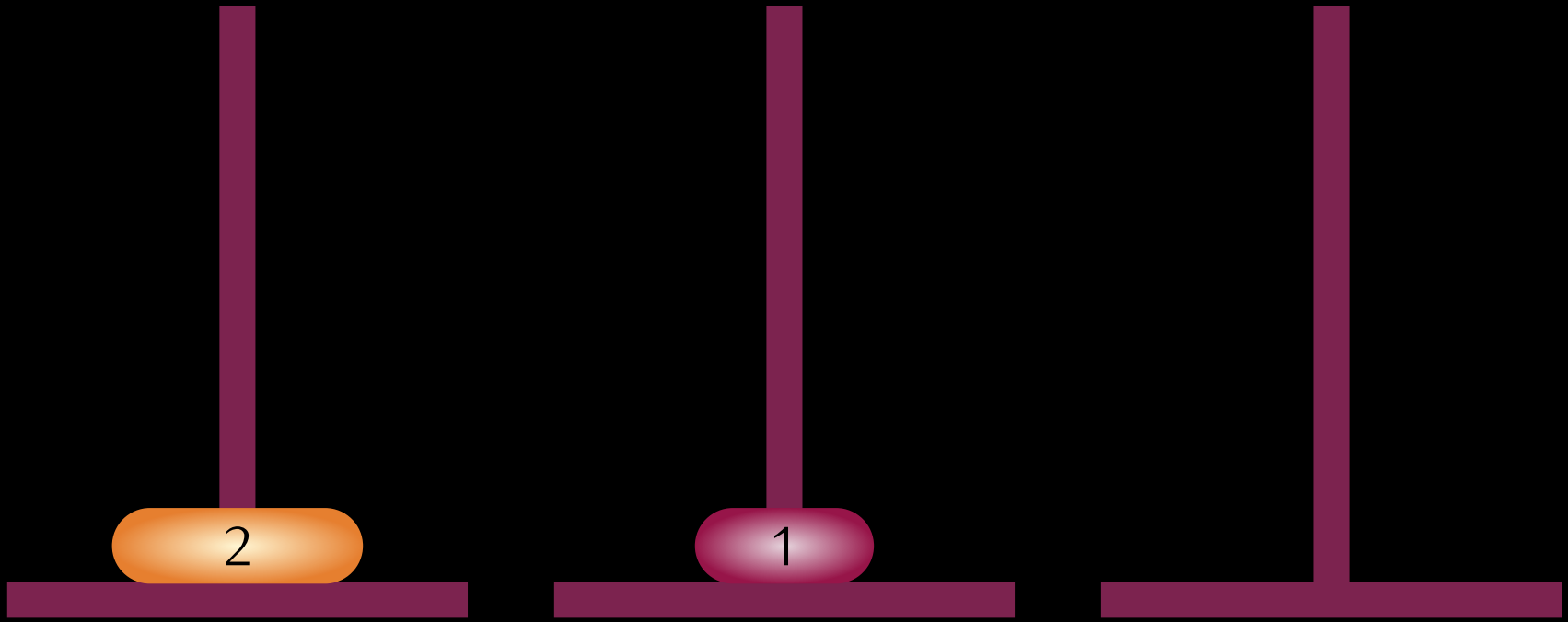
Tours de Hanoi – 1 Disque



Tours de Hanoi – 2 Disques

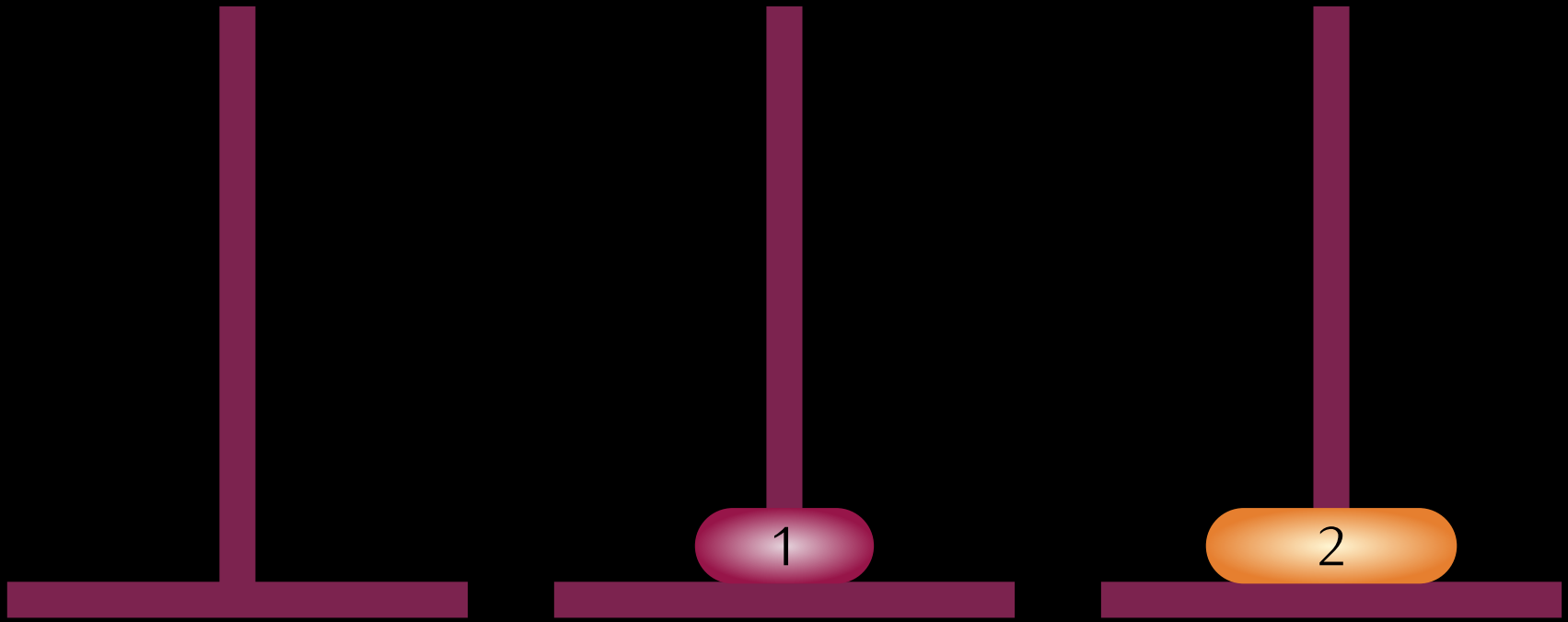


Tours de Hanoi – 2 Disques



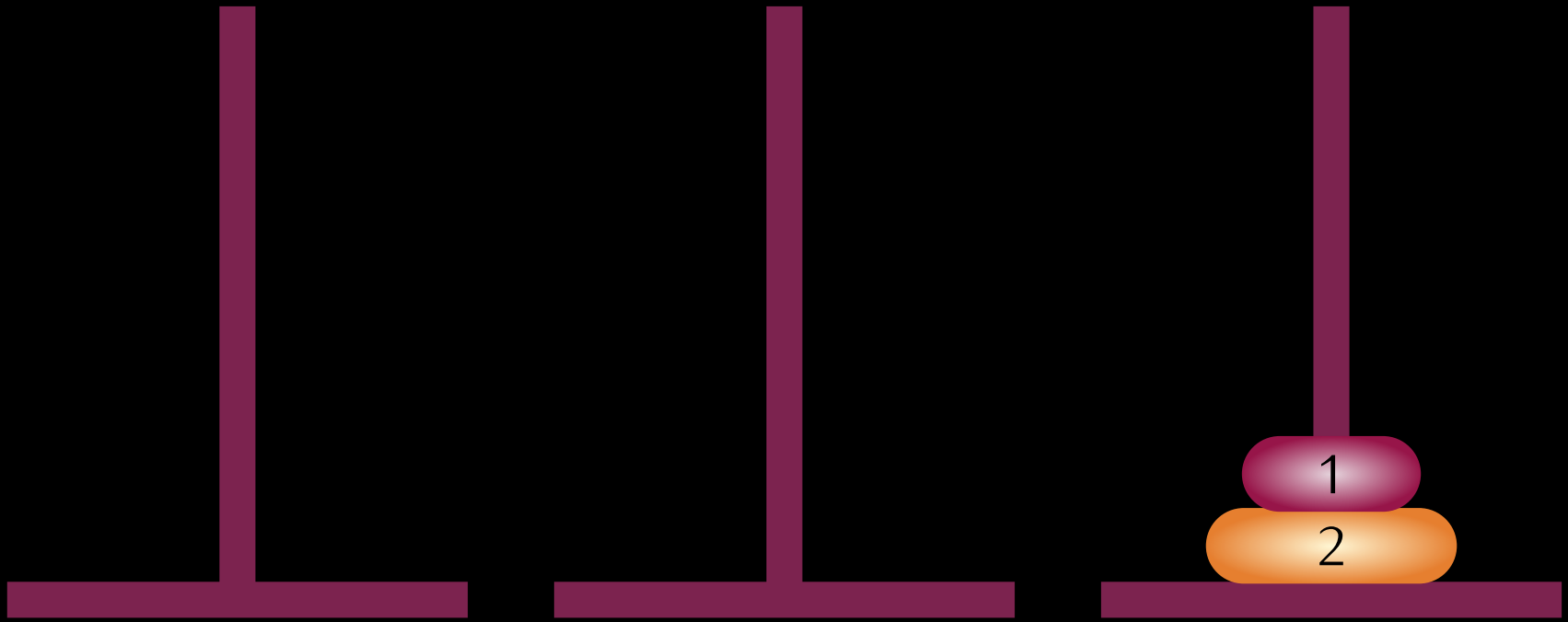
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 2 Disques



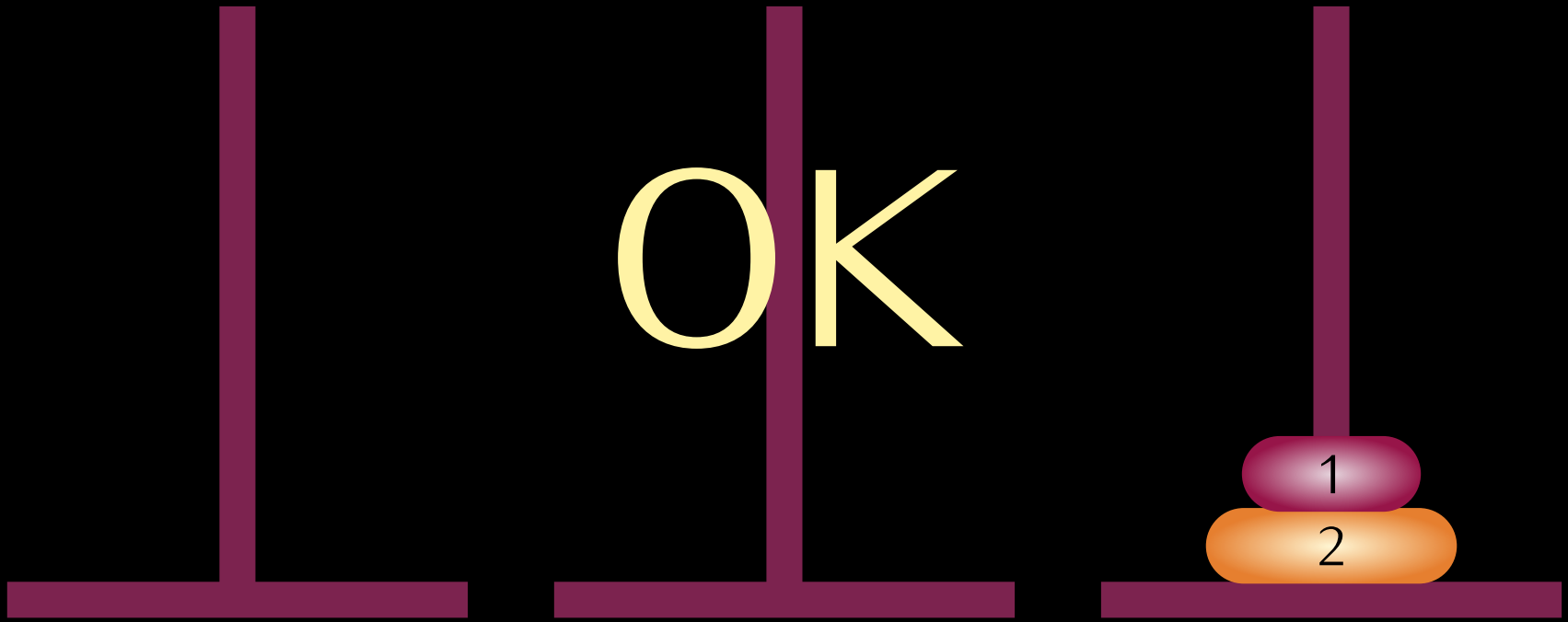
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 2 Disques

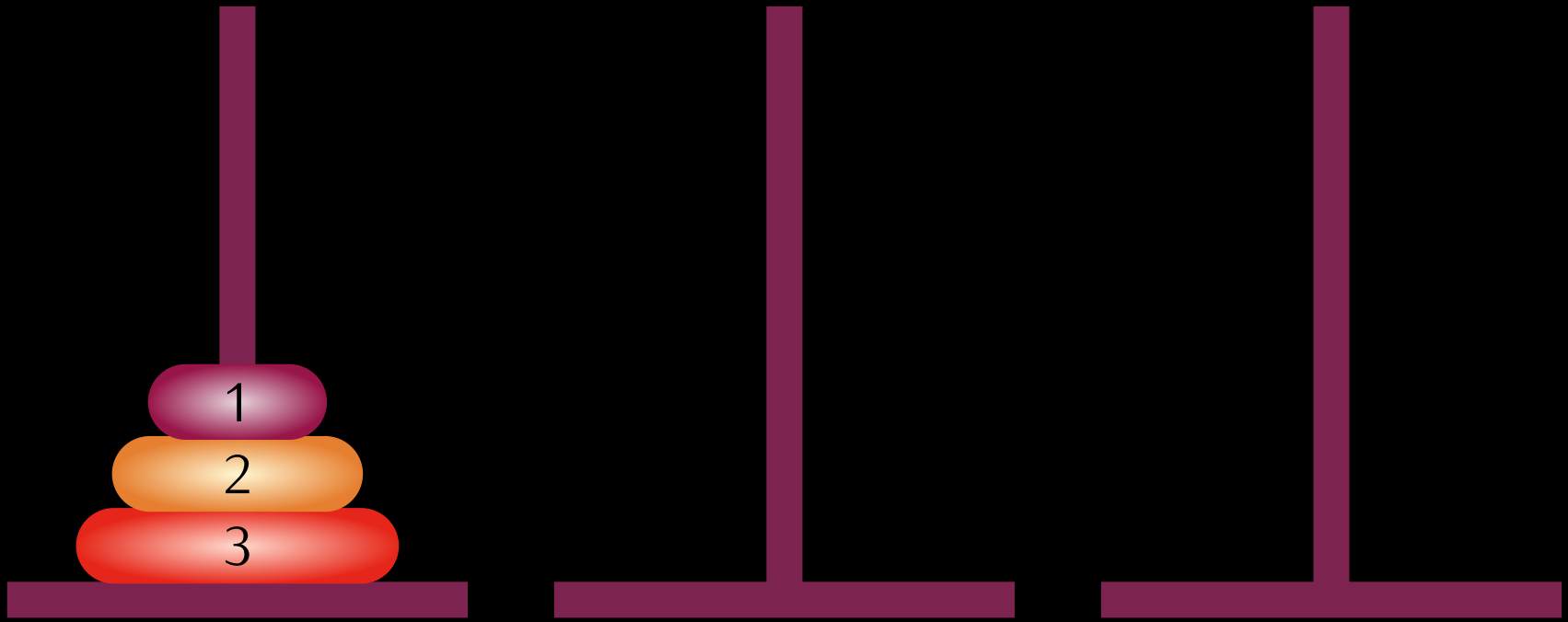


On déplace le disque de la tige 2 vers la tige 3.

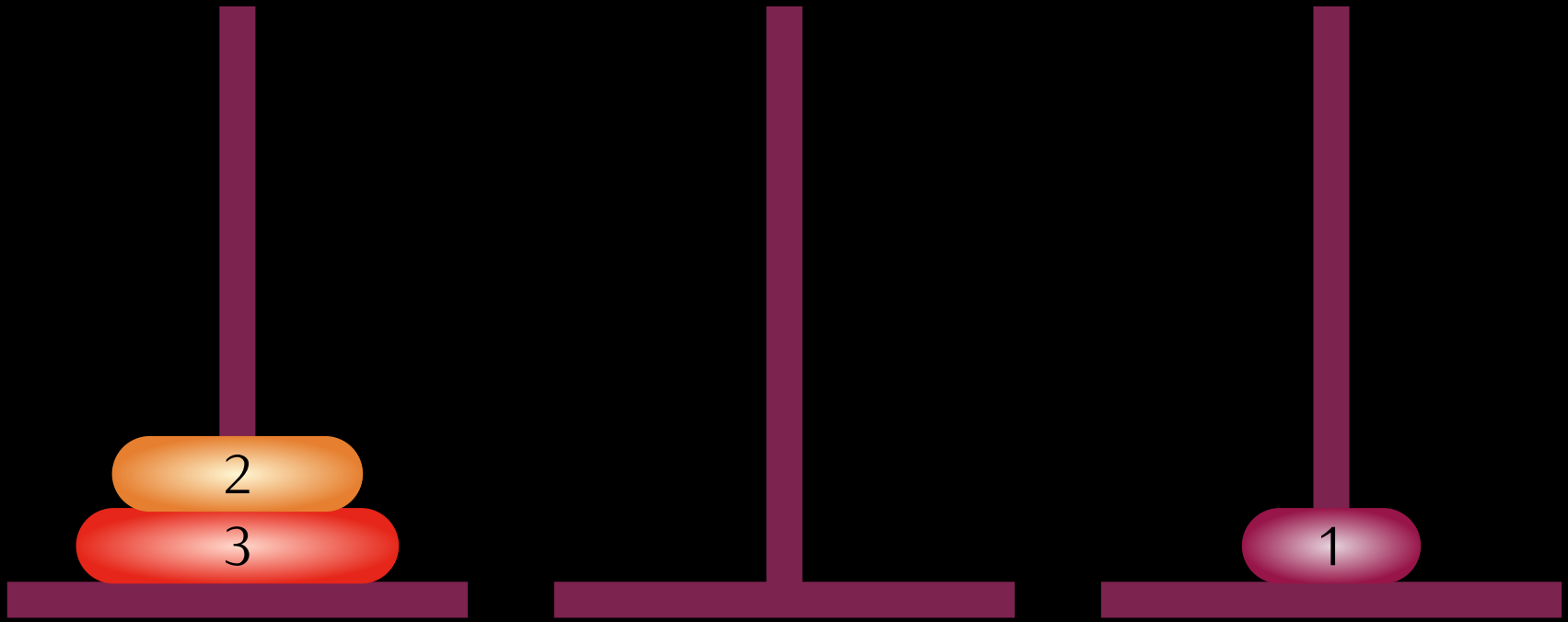
Tours de Hanoi – 2 Disques



Tours de Hanoi – 3 Disques

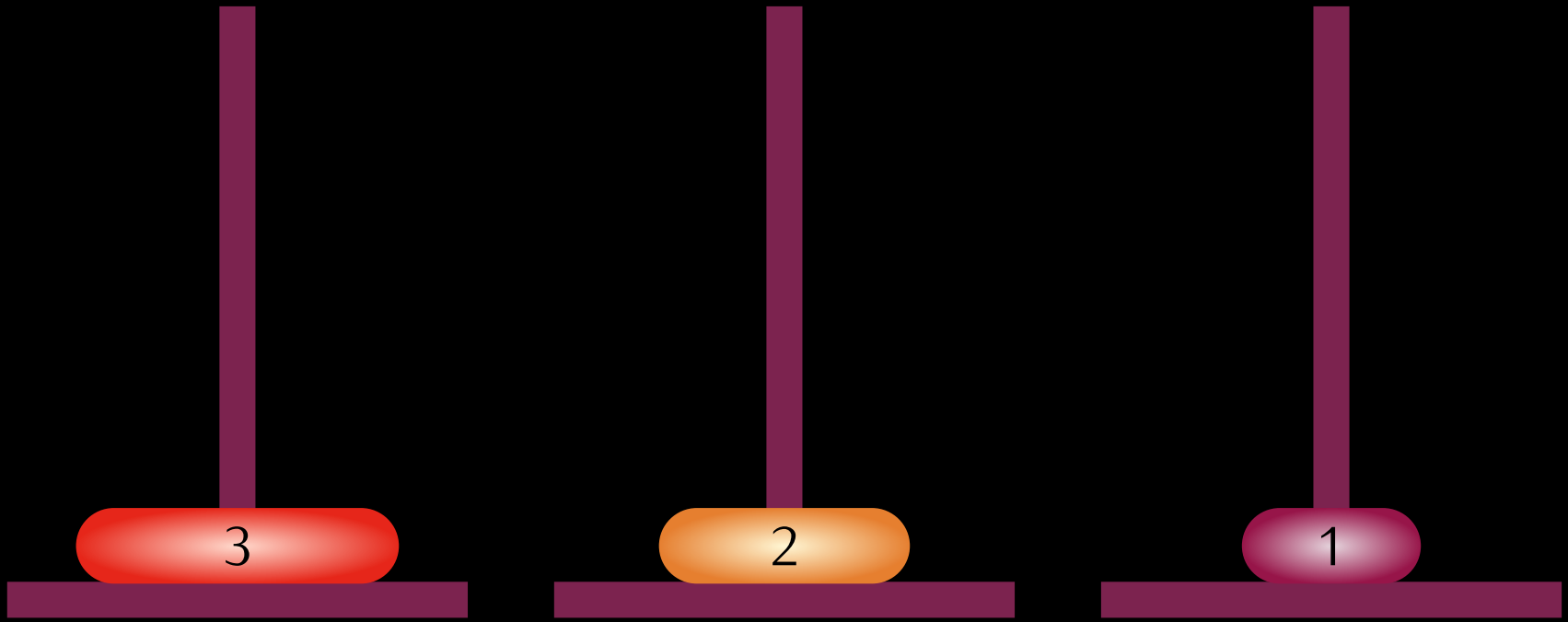


Tours de Hanoi – 3 Disques



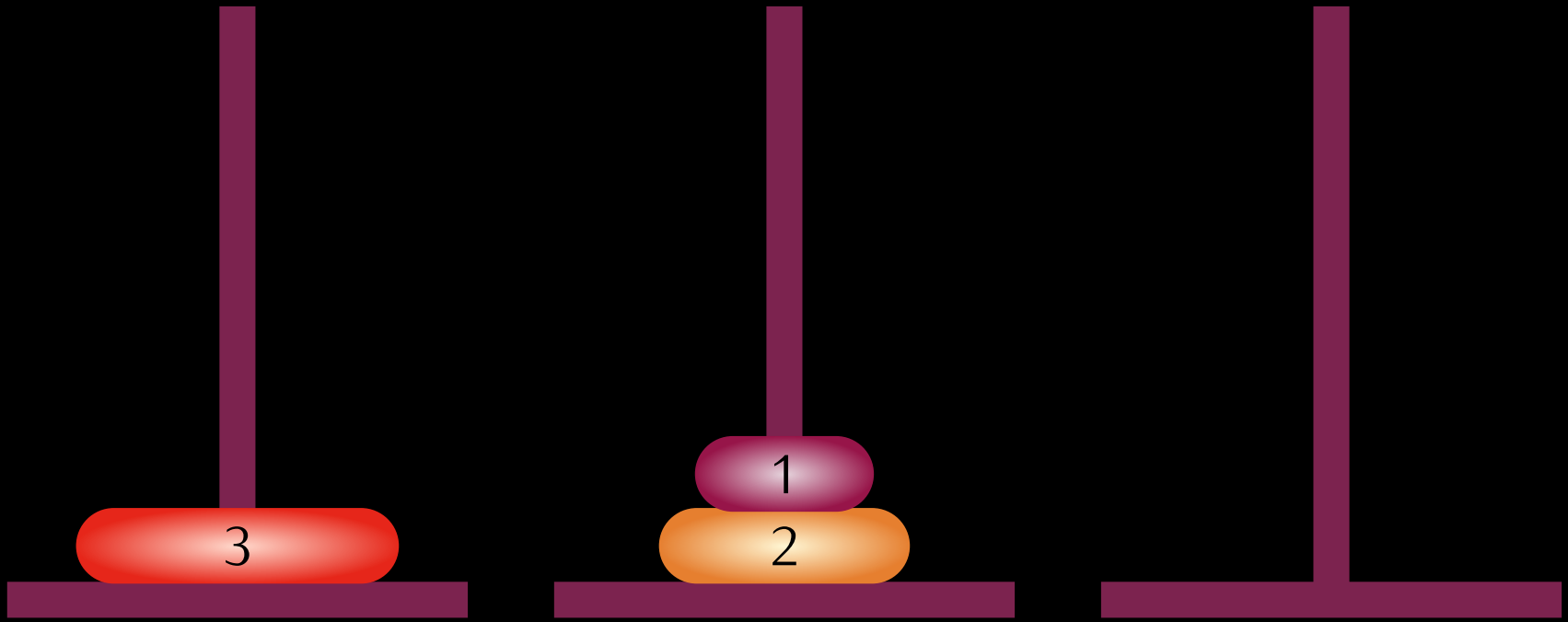
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 3 Disques



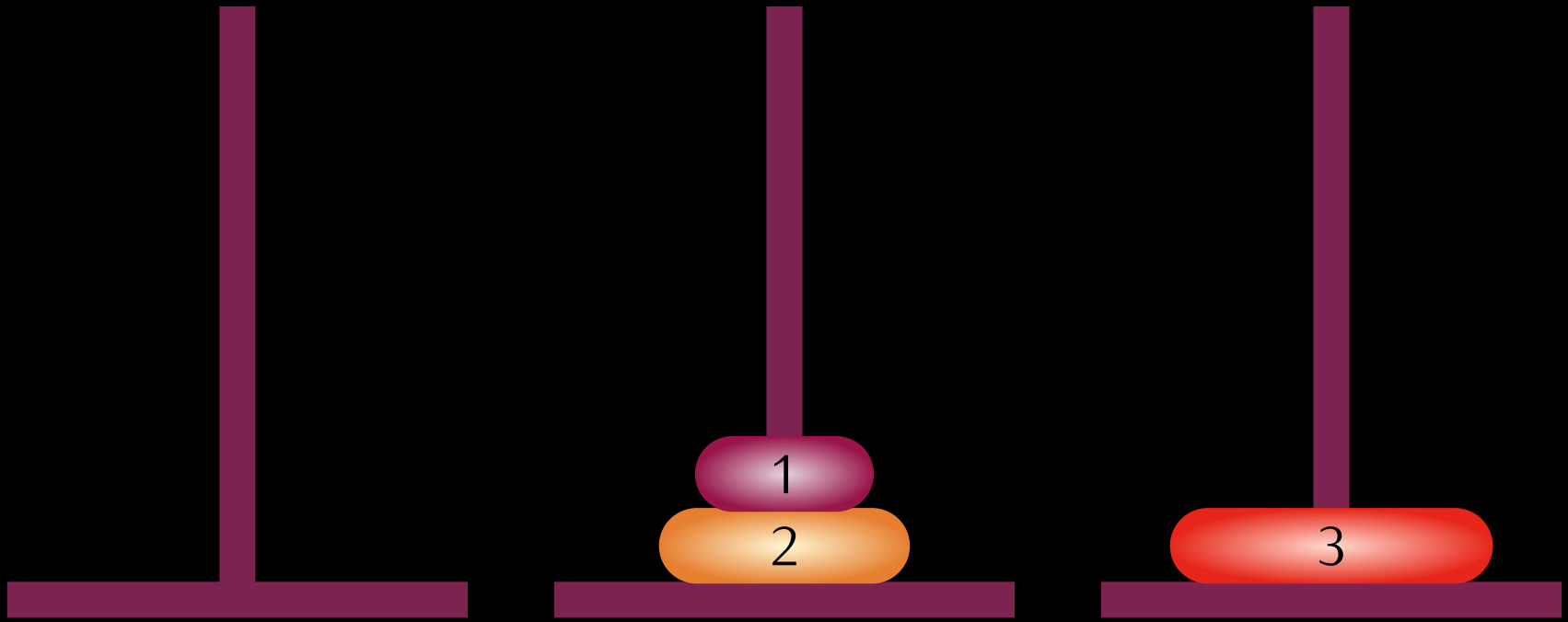
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 3 Disques



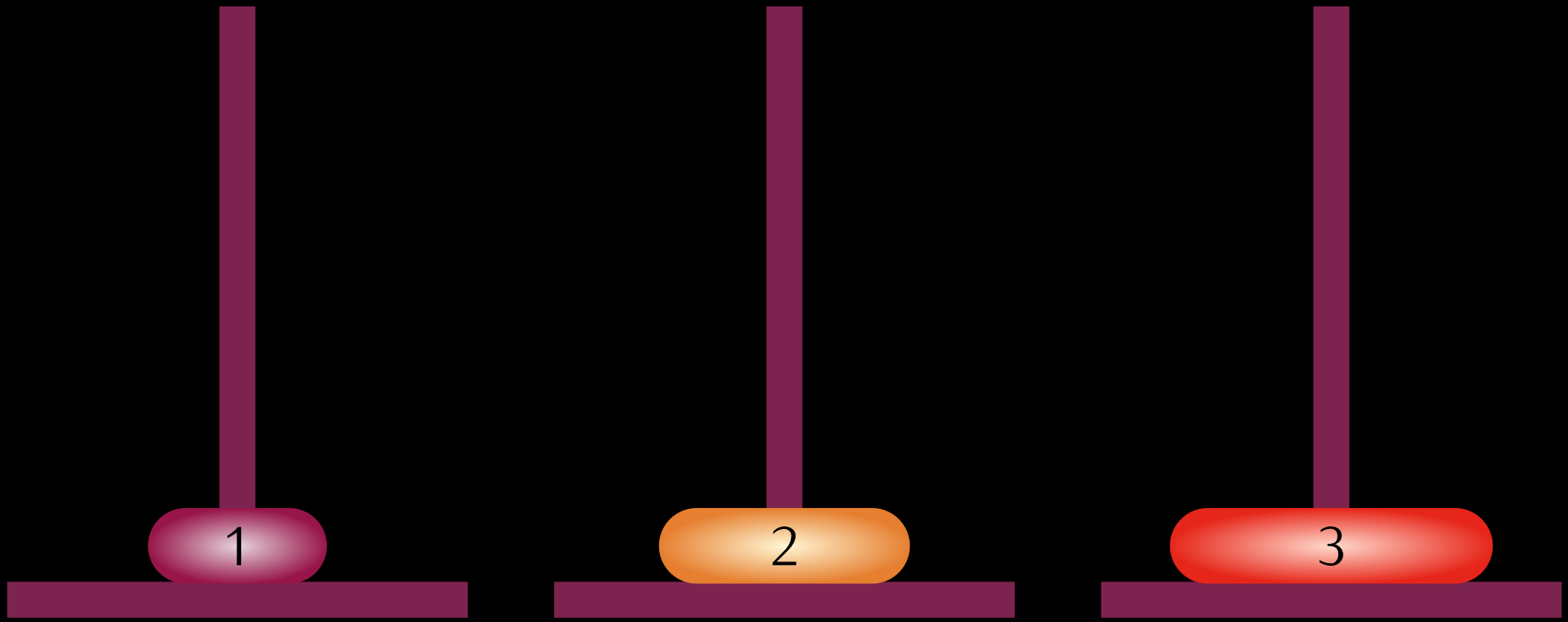
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 3 Disques



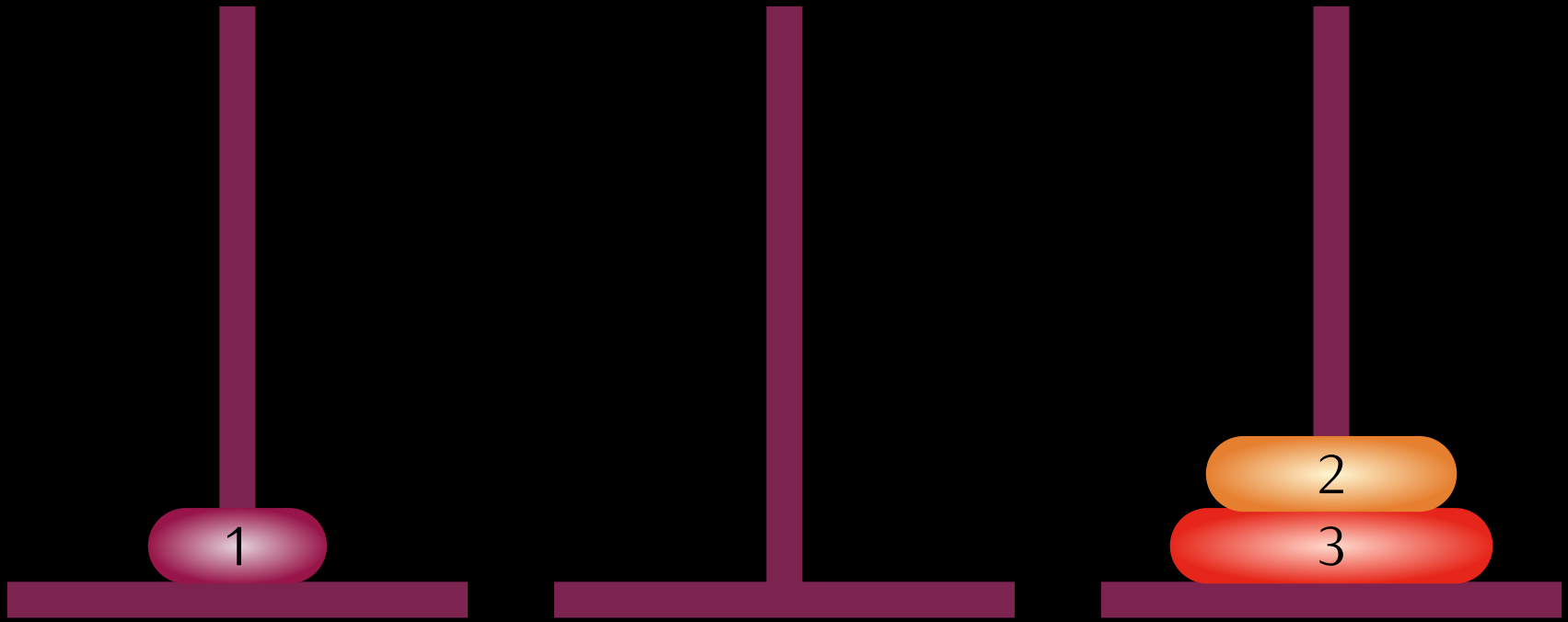
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 3 Disques



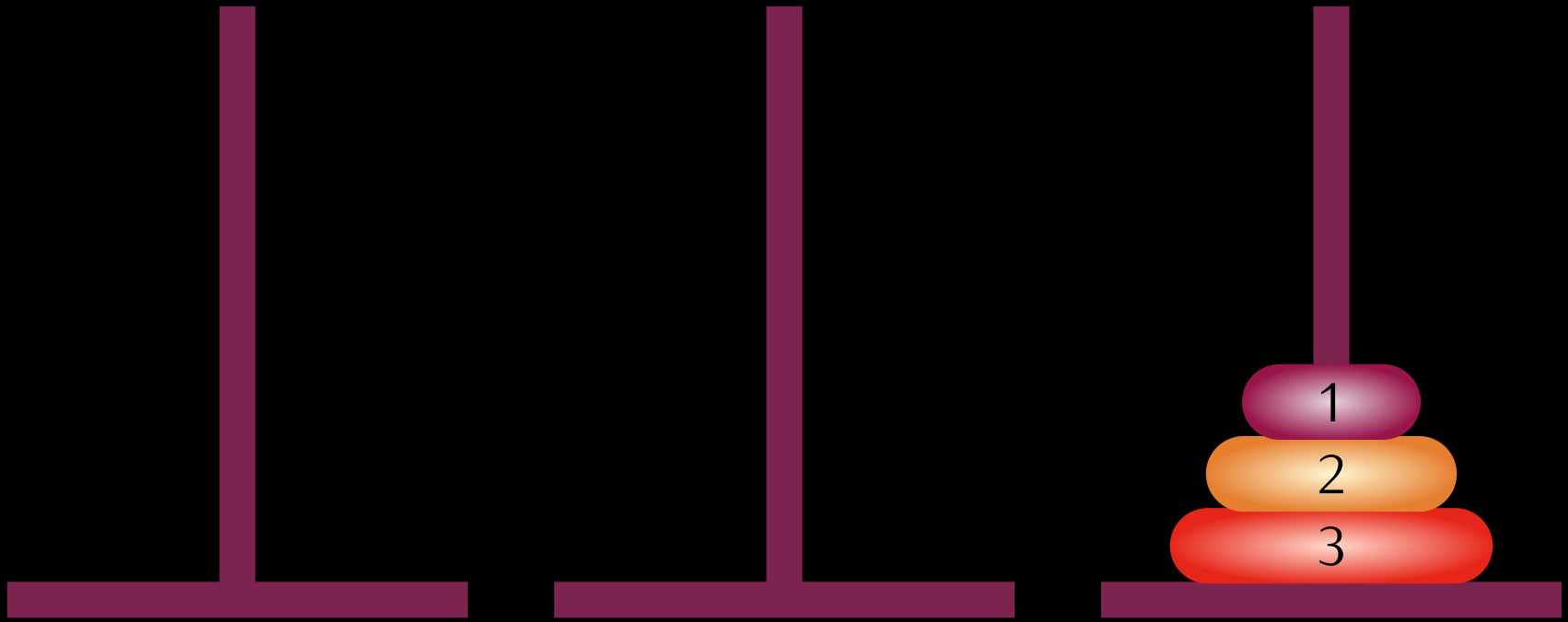
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 3 Disques



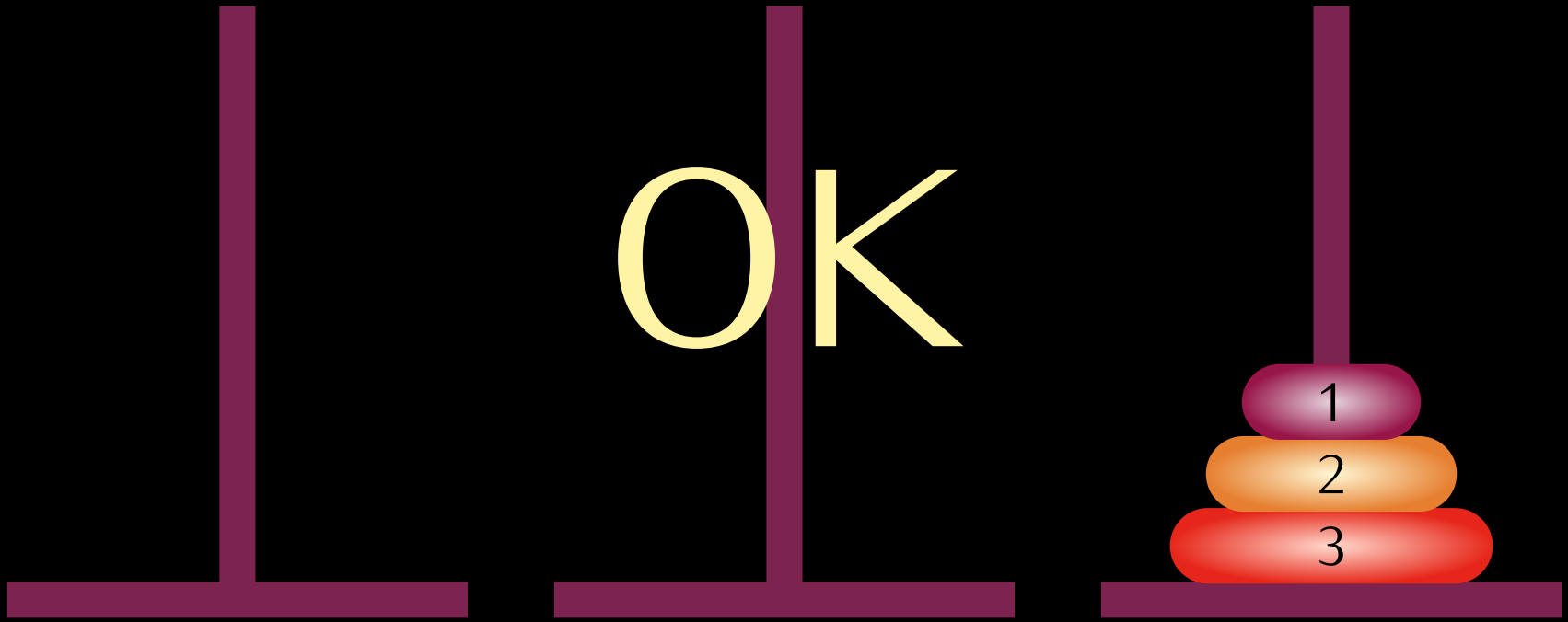
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 3 Disques

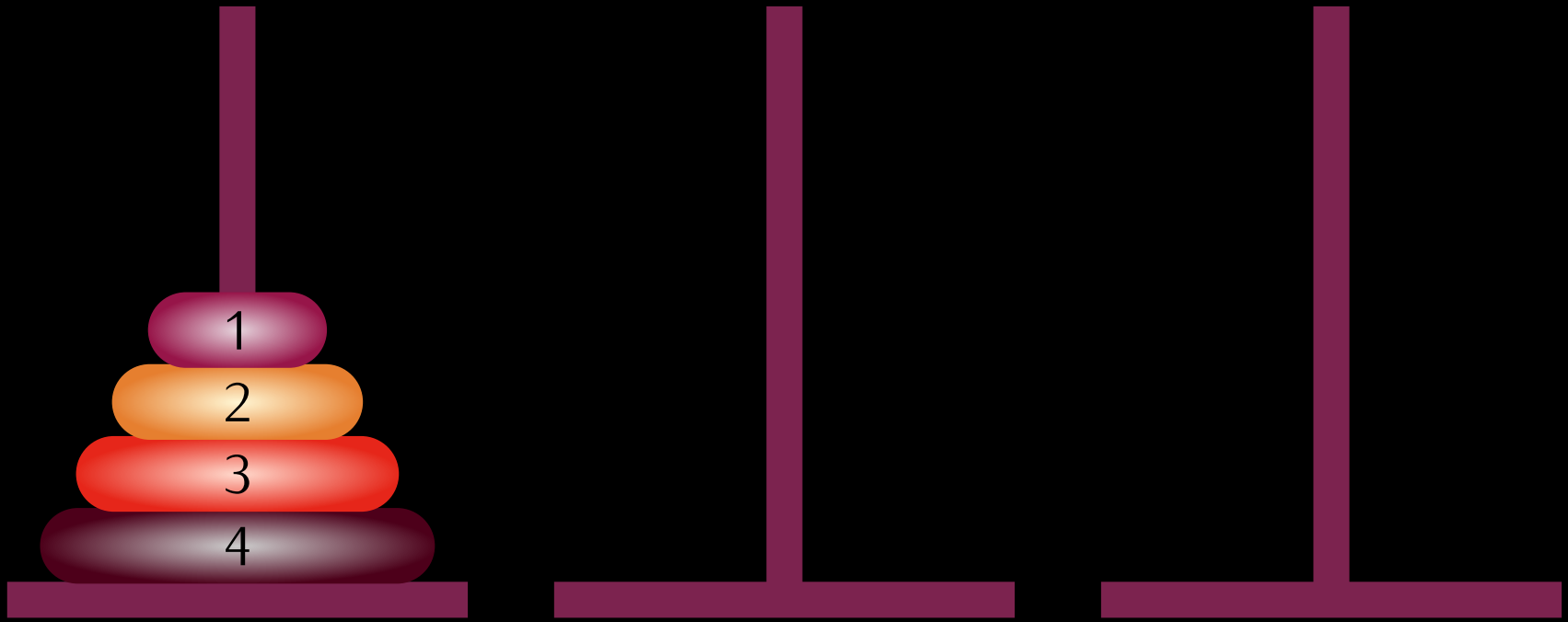


On déplace le disque de la tige 1 vers la tige 3.

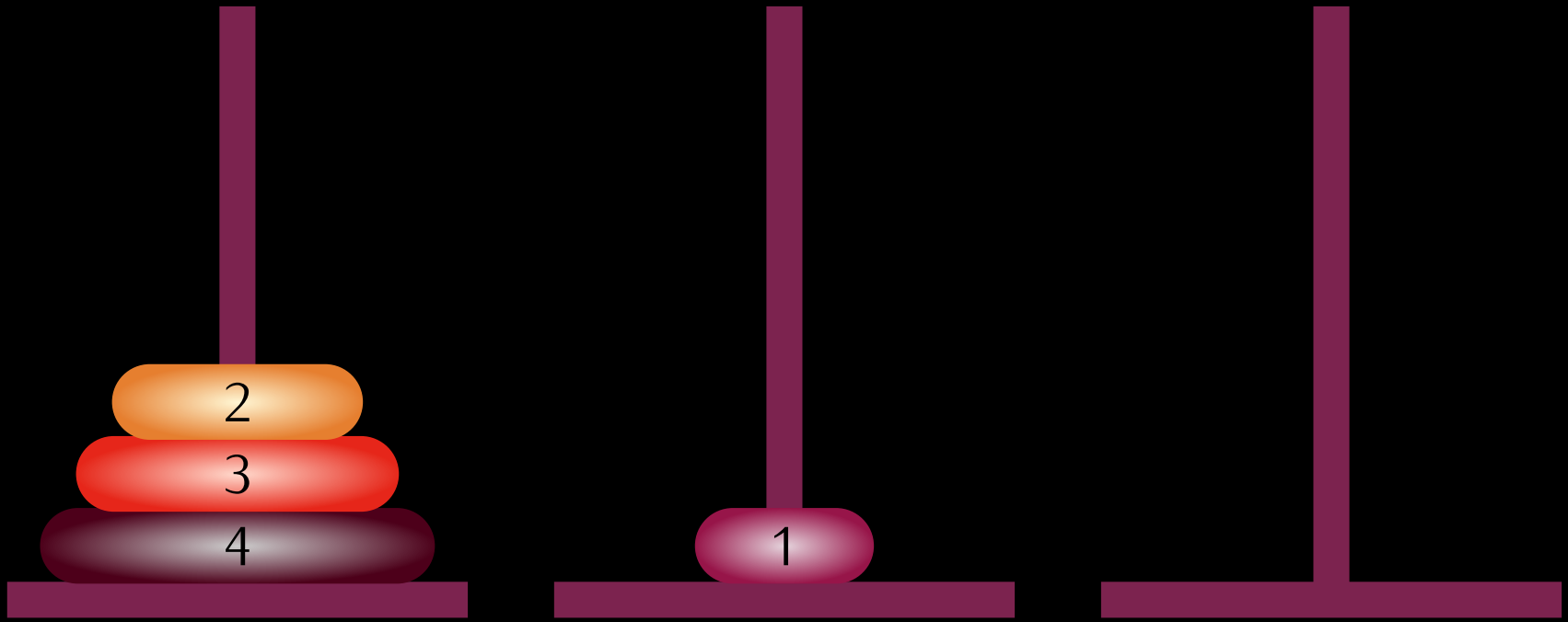
Tours de Hanoi – 3 Disques



Tours de Hanoi – 4 Disques

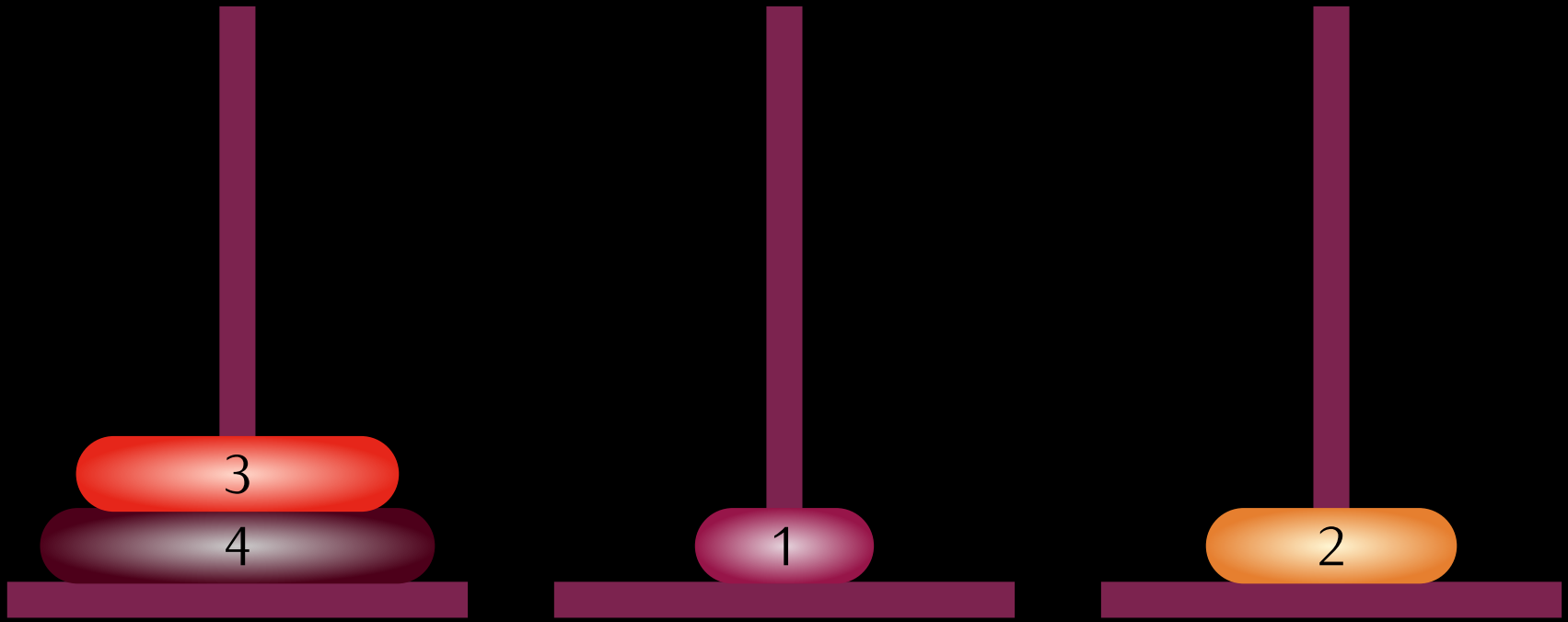


Tours de Hanoi – 4 Disques



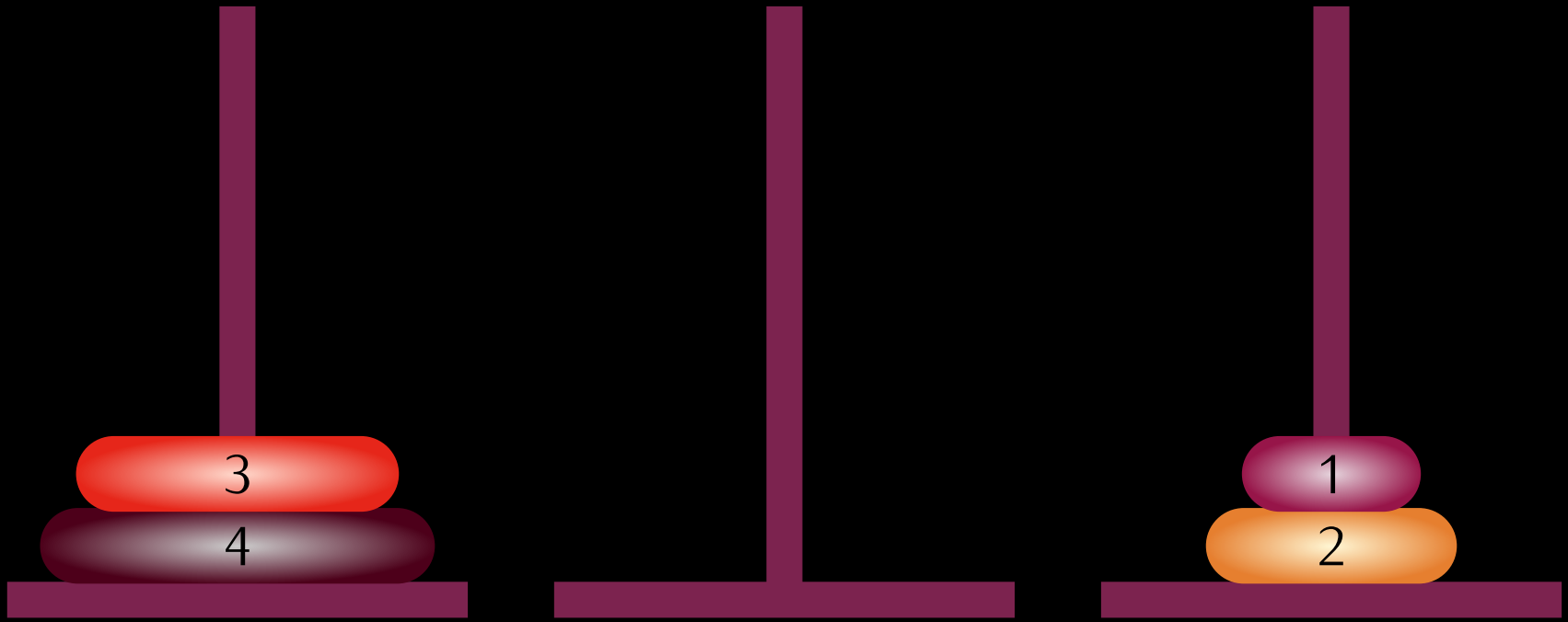
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



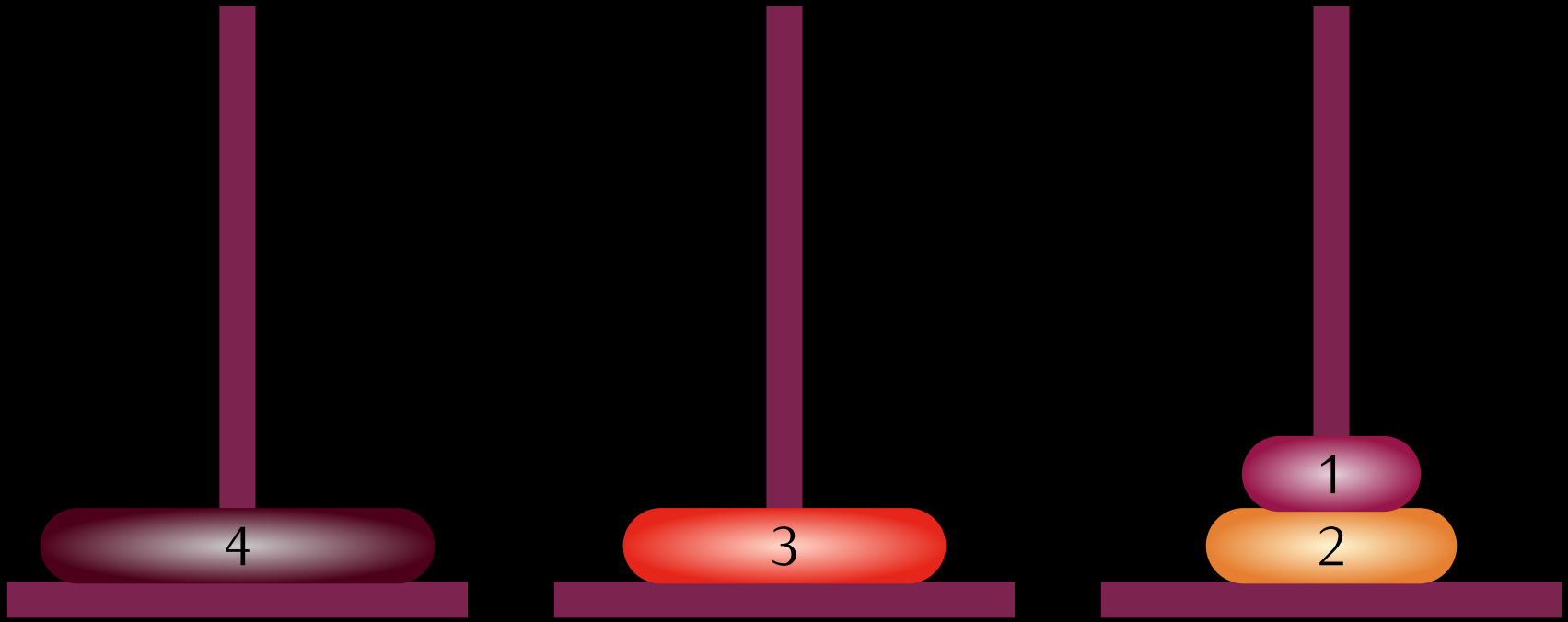
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques



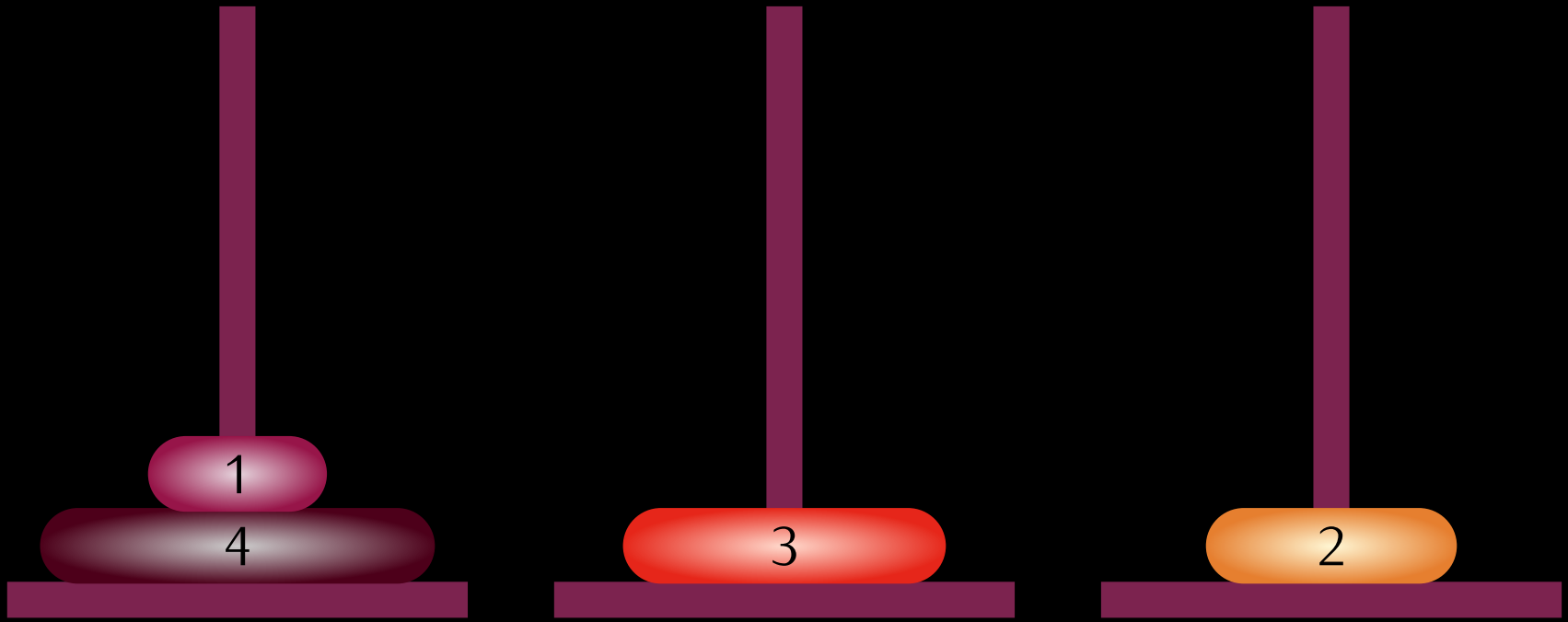
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



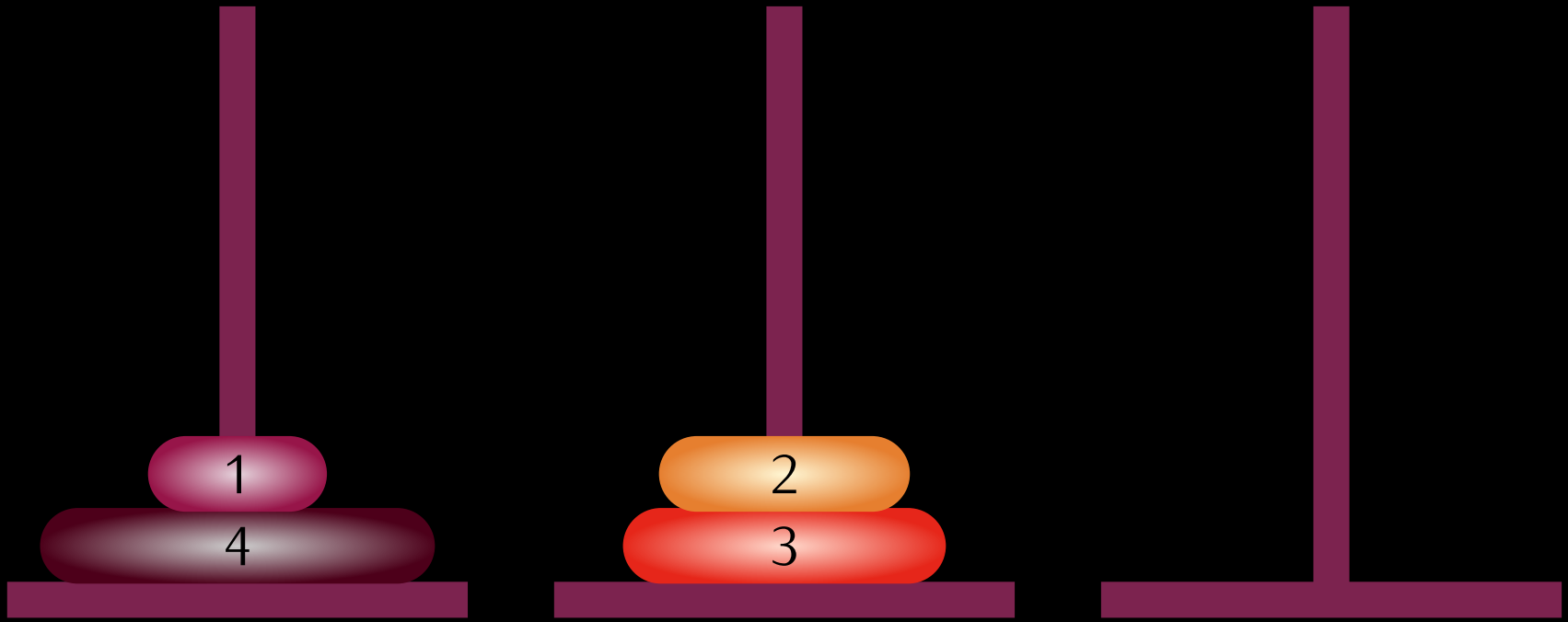
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



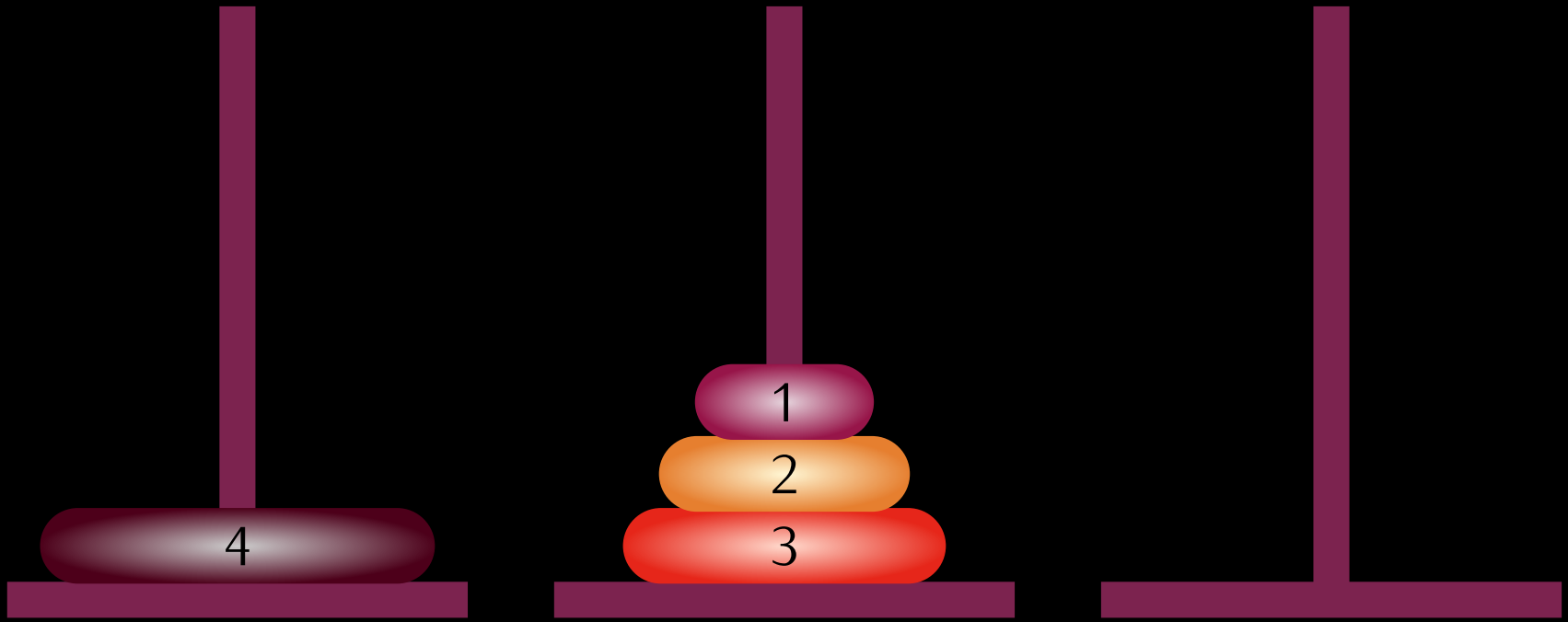
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 4 Disques



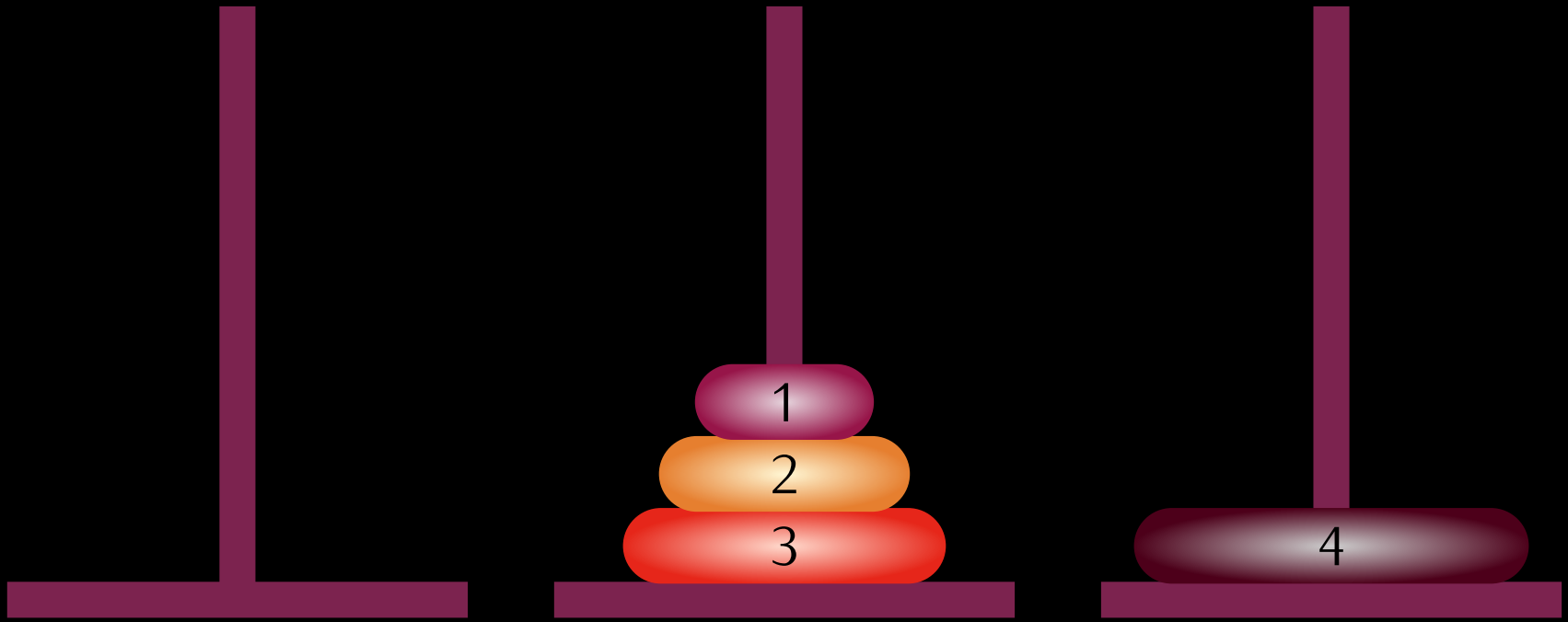
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 4 Disques



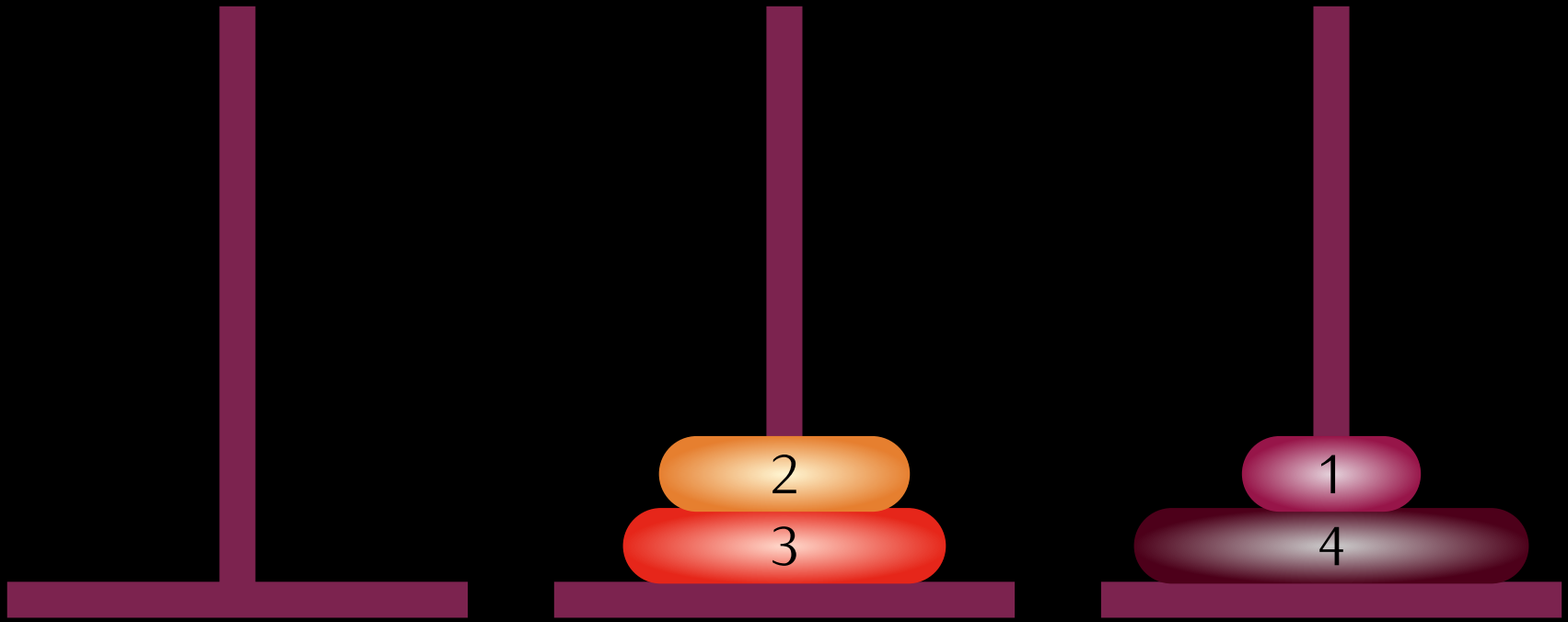
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



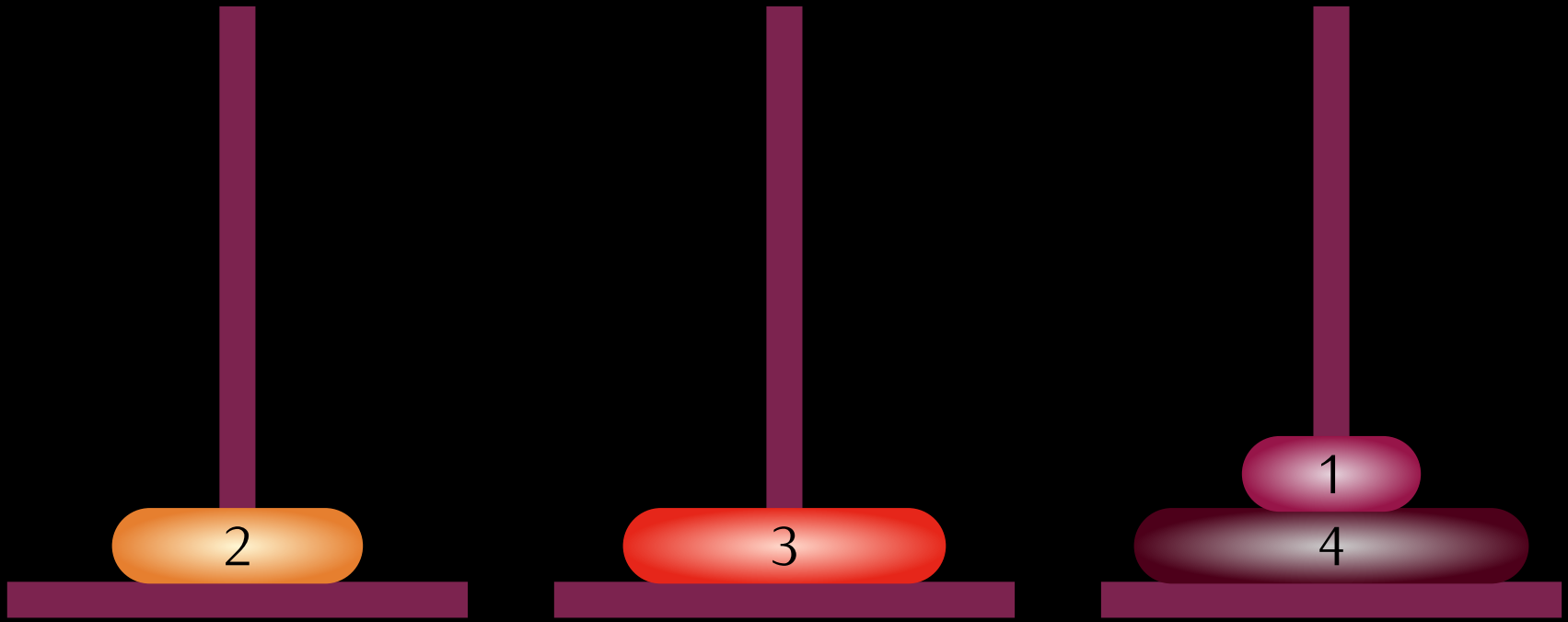
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques



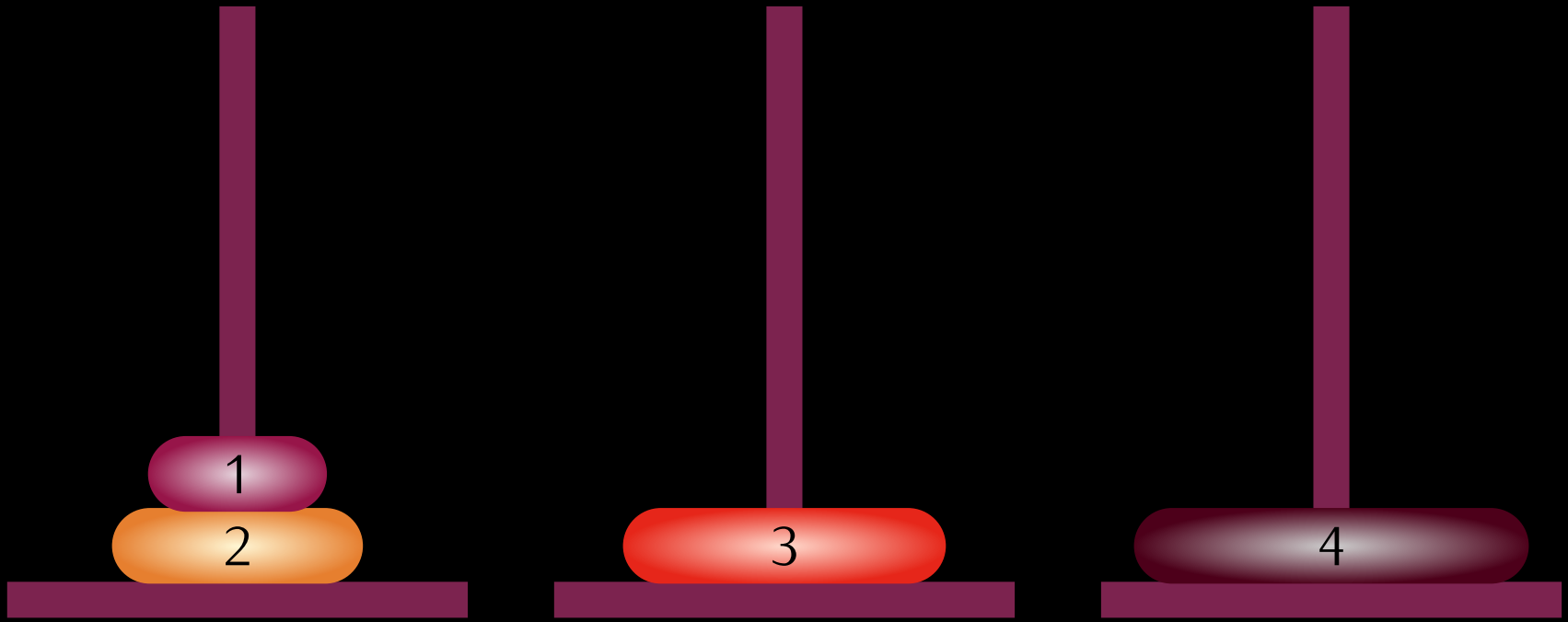
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



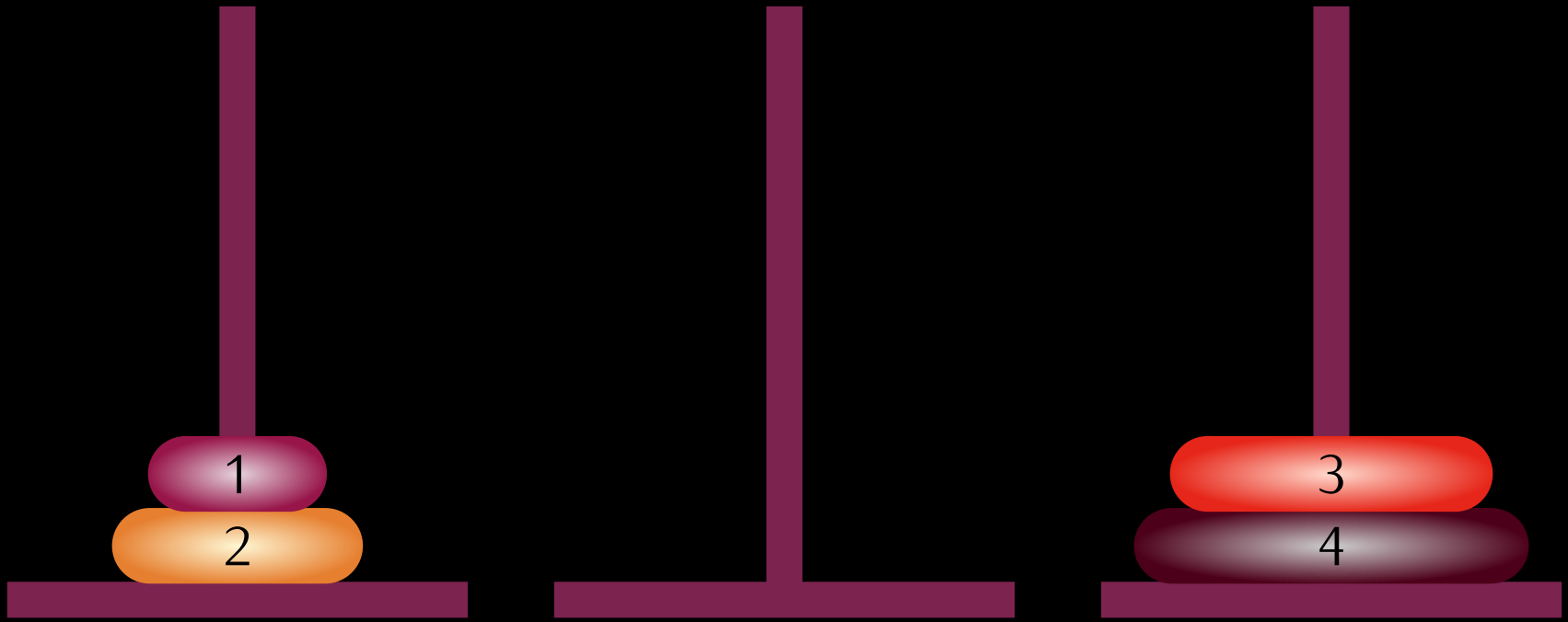
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 4 Disques



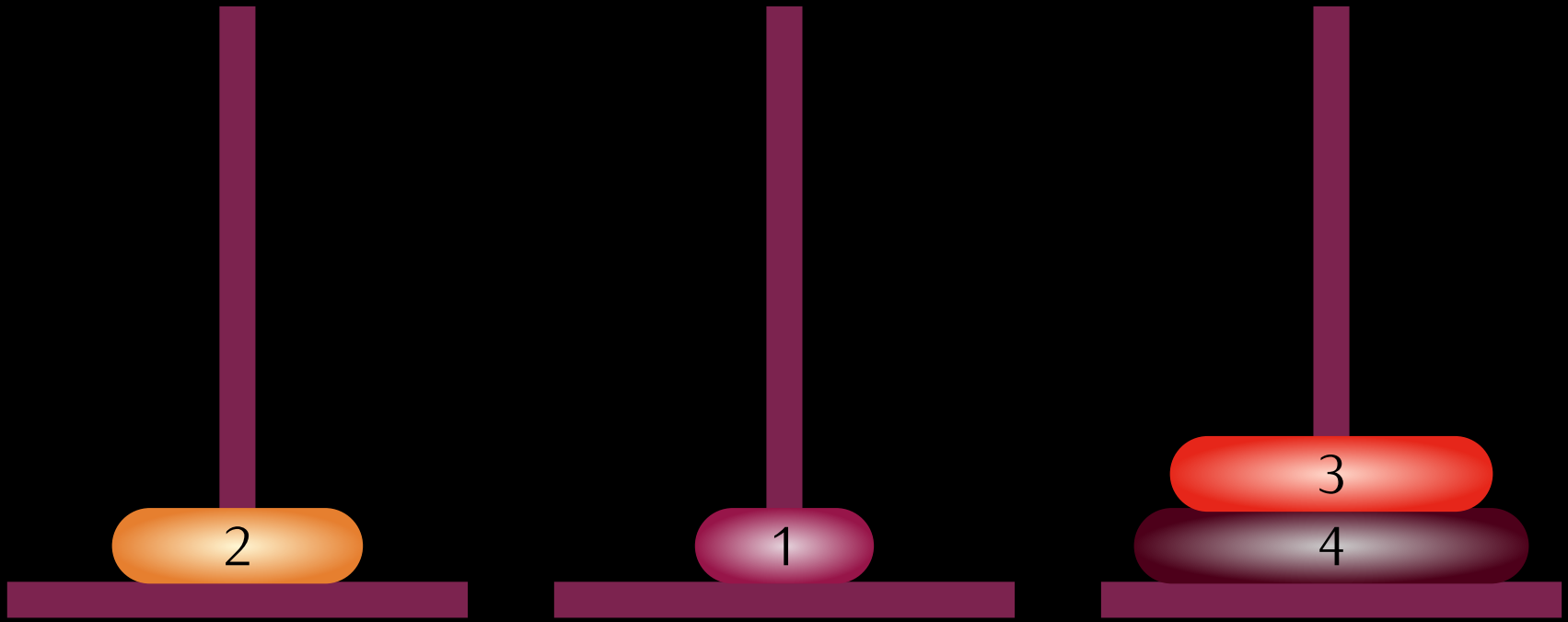
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 4 Disques



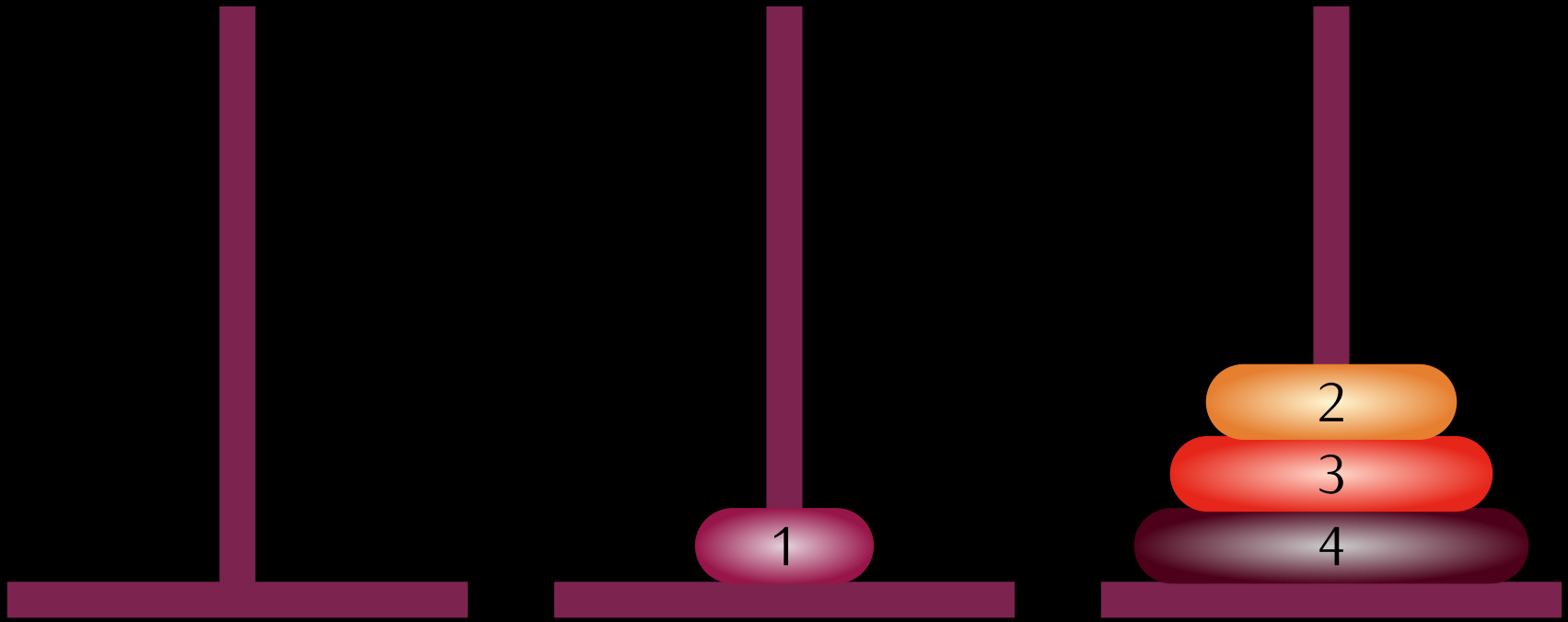
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



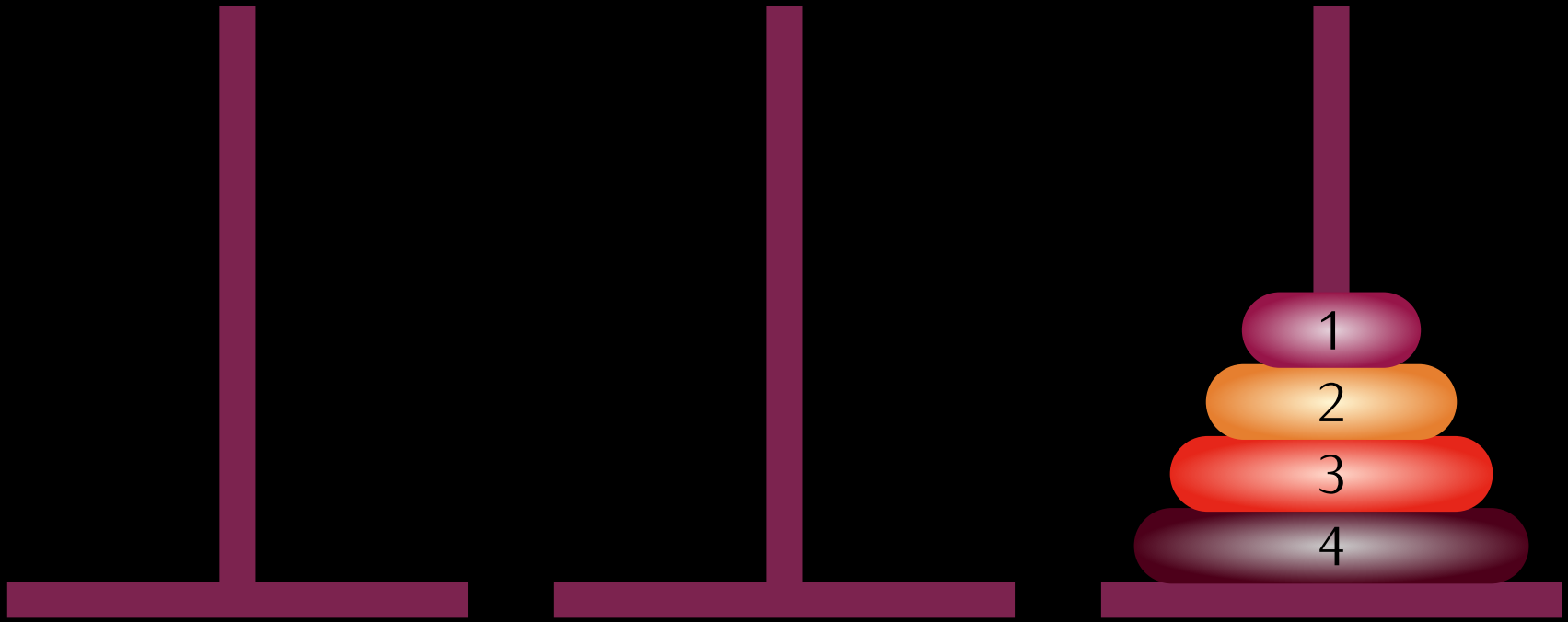
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



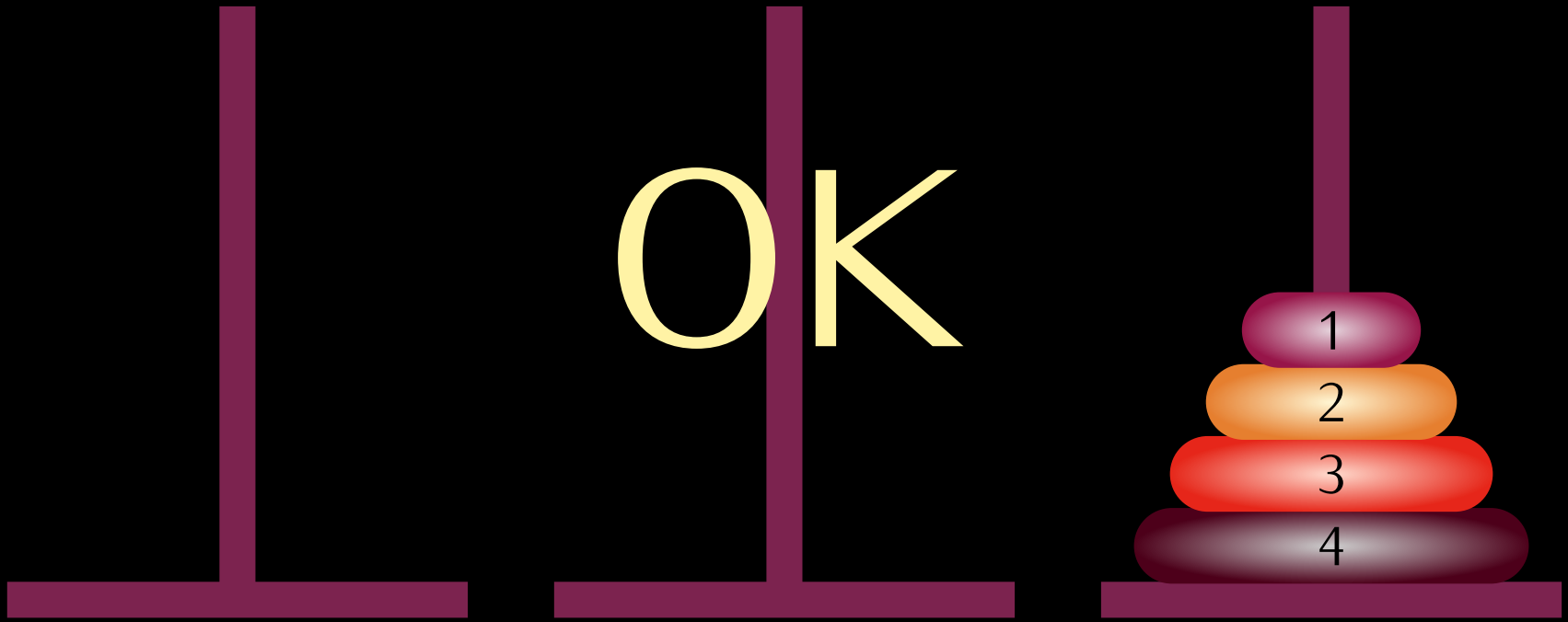
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques

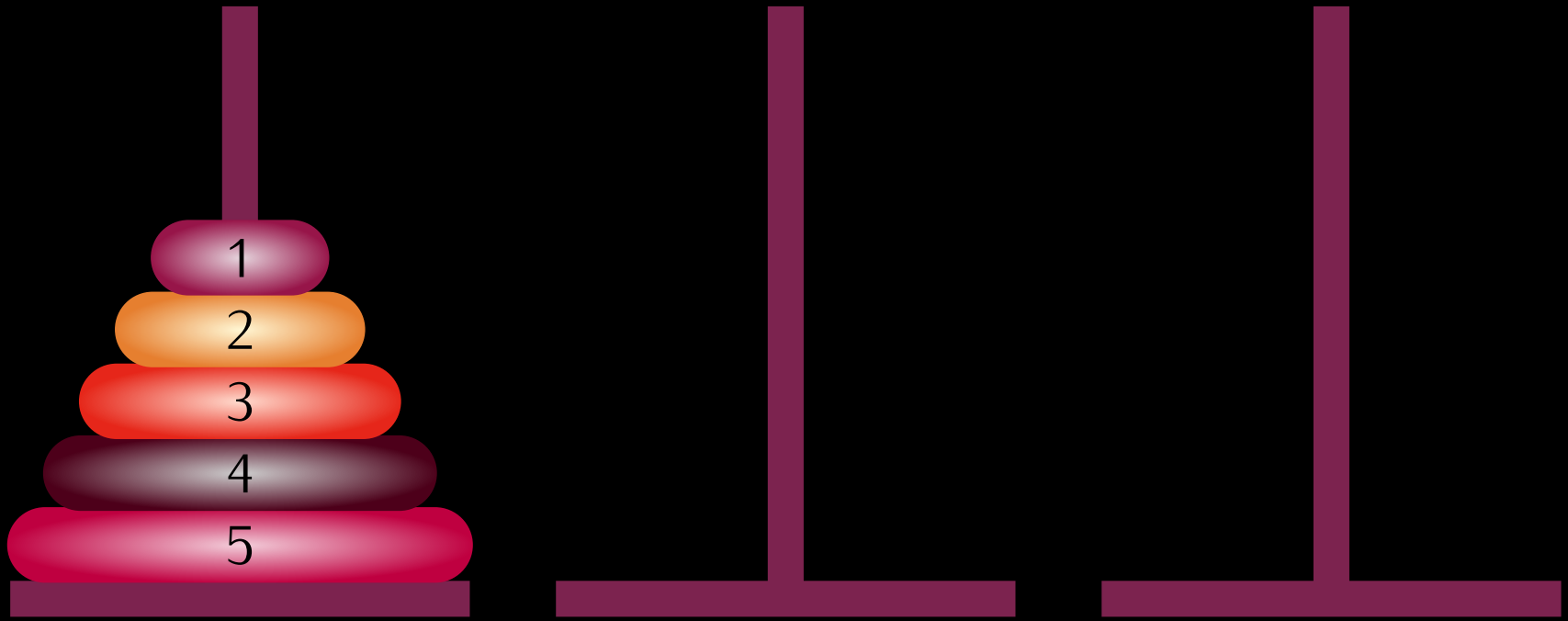


On déplace le disque de la tige 2 vers la tige 3.

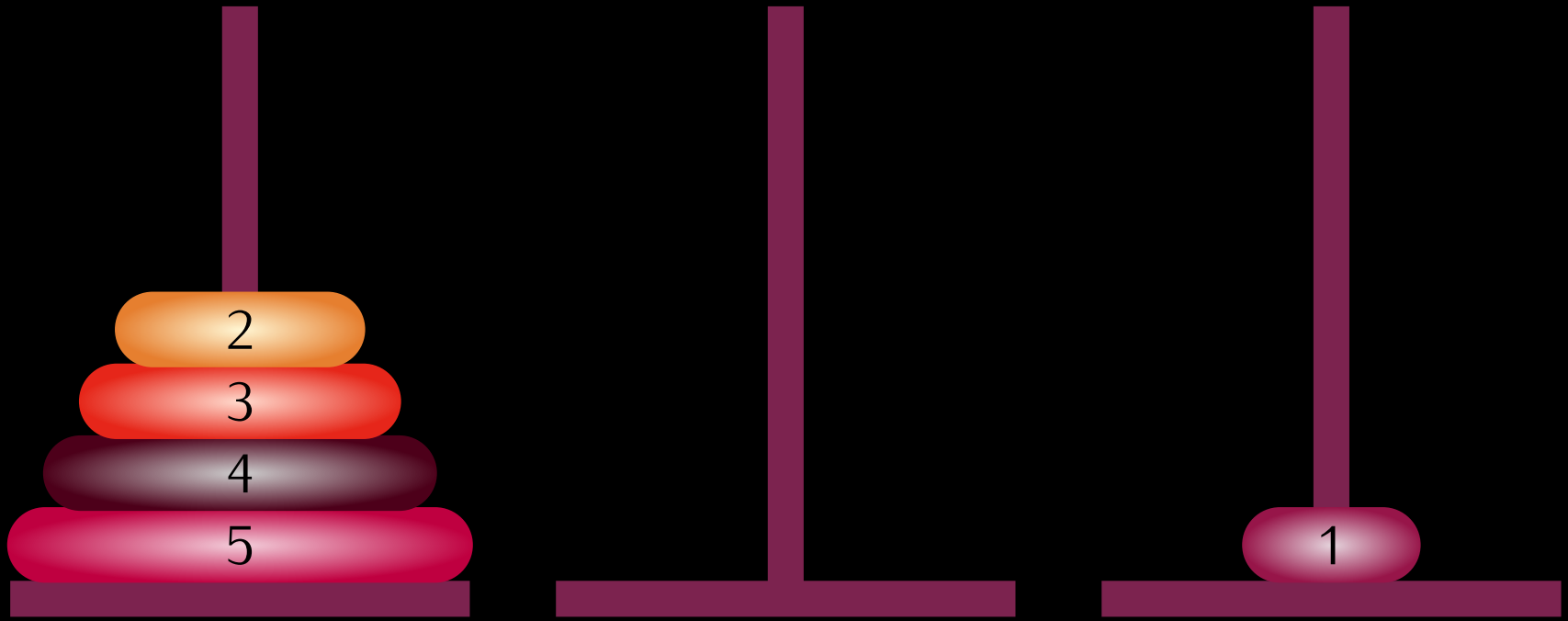
Tours de Hanoi – 4 Disques



Tours de Hanoi – 5 Disques

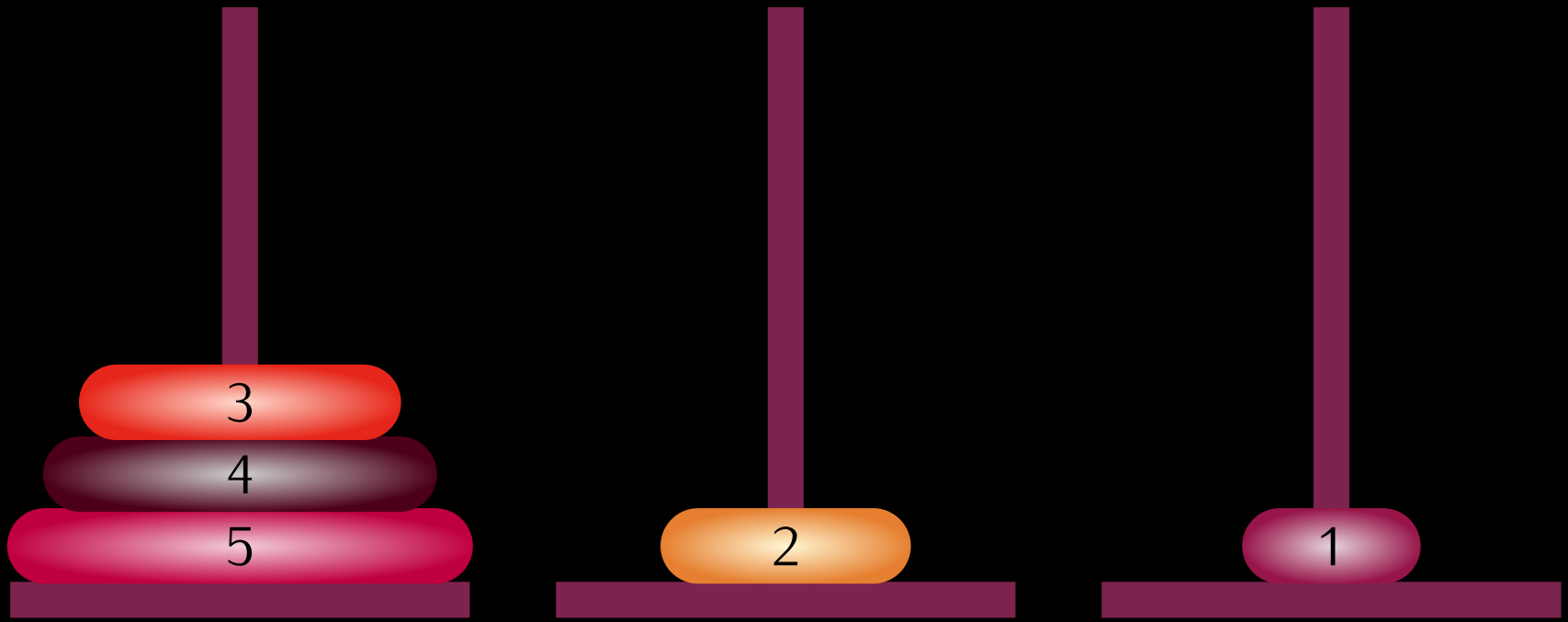


Tours de Hanoi – 5 Disques



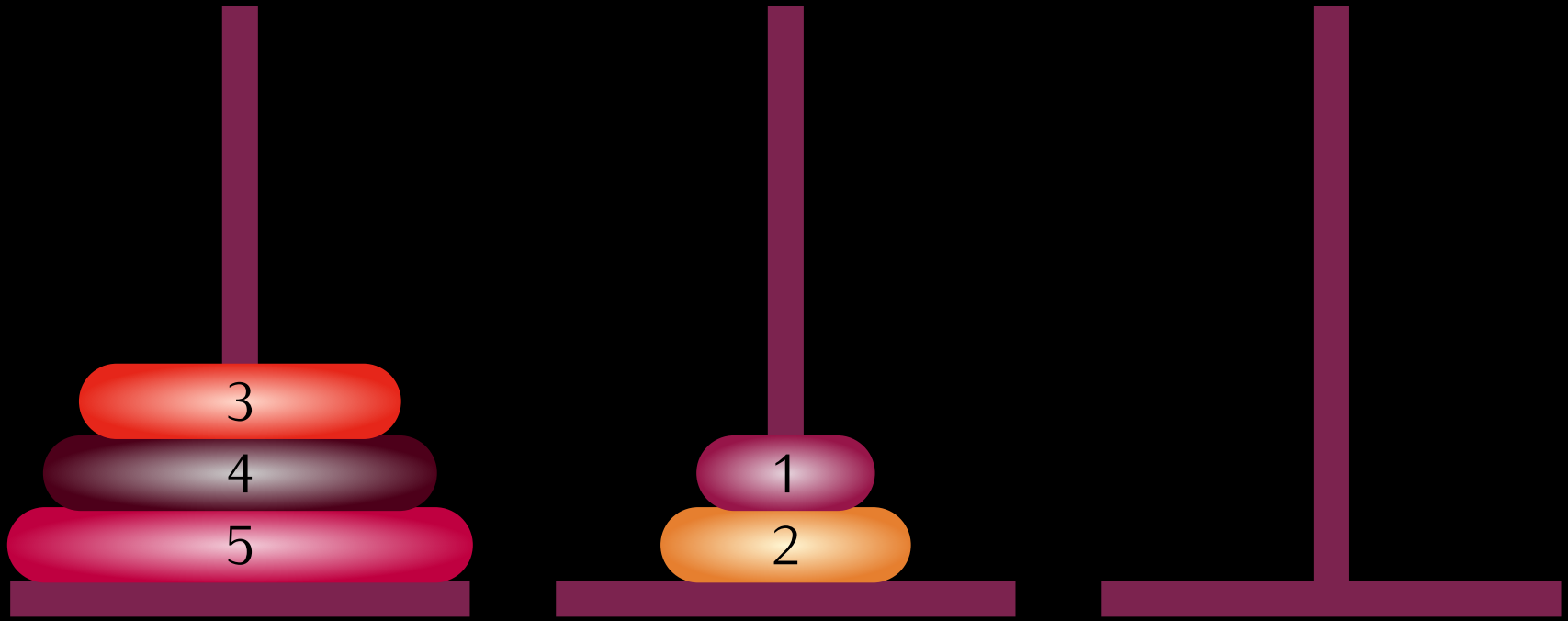
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



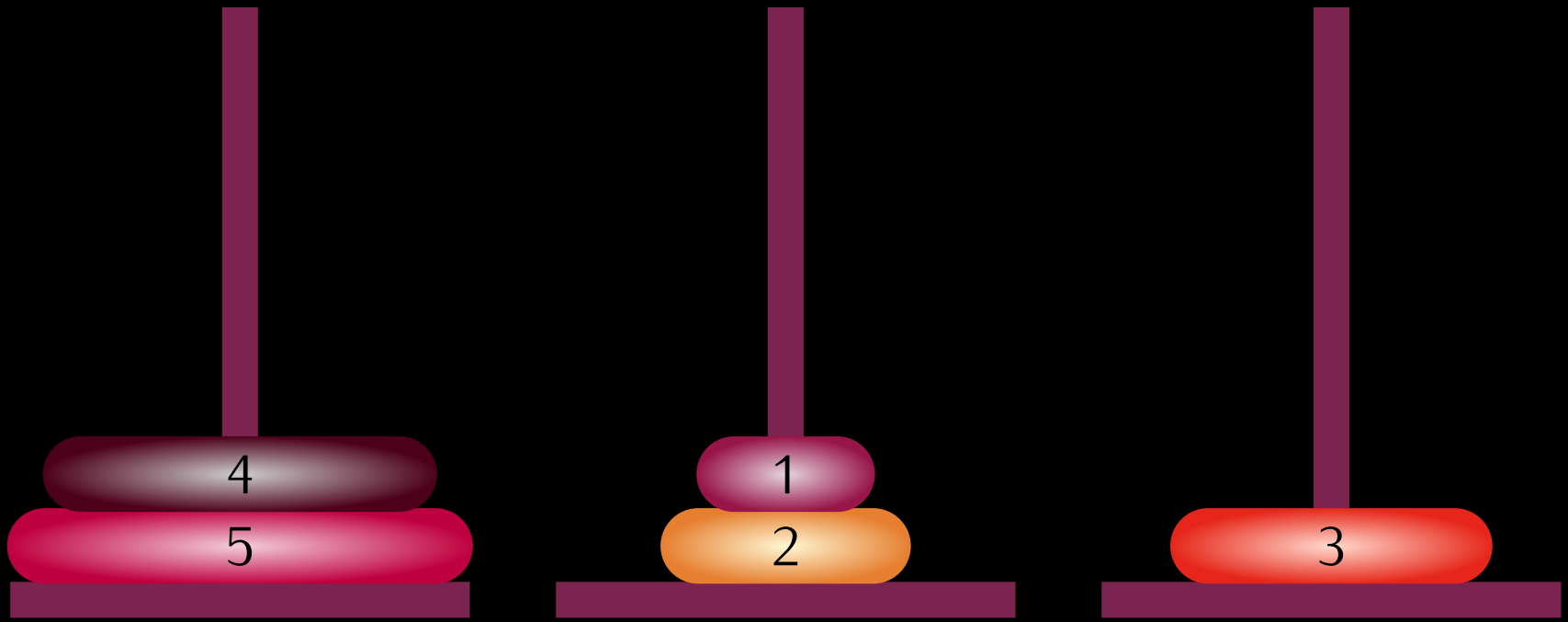
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



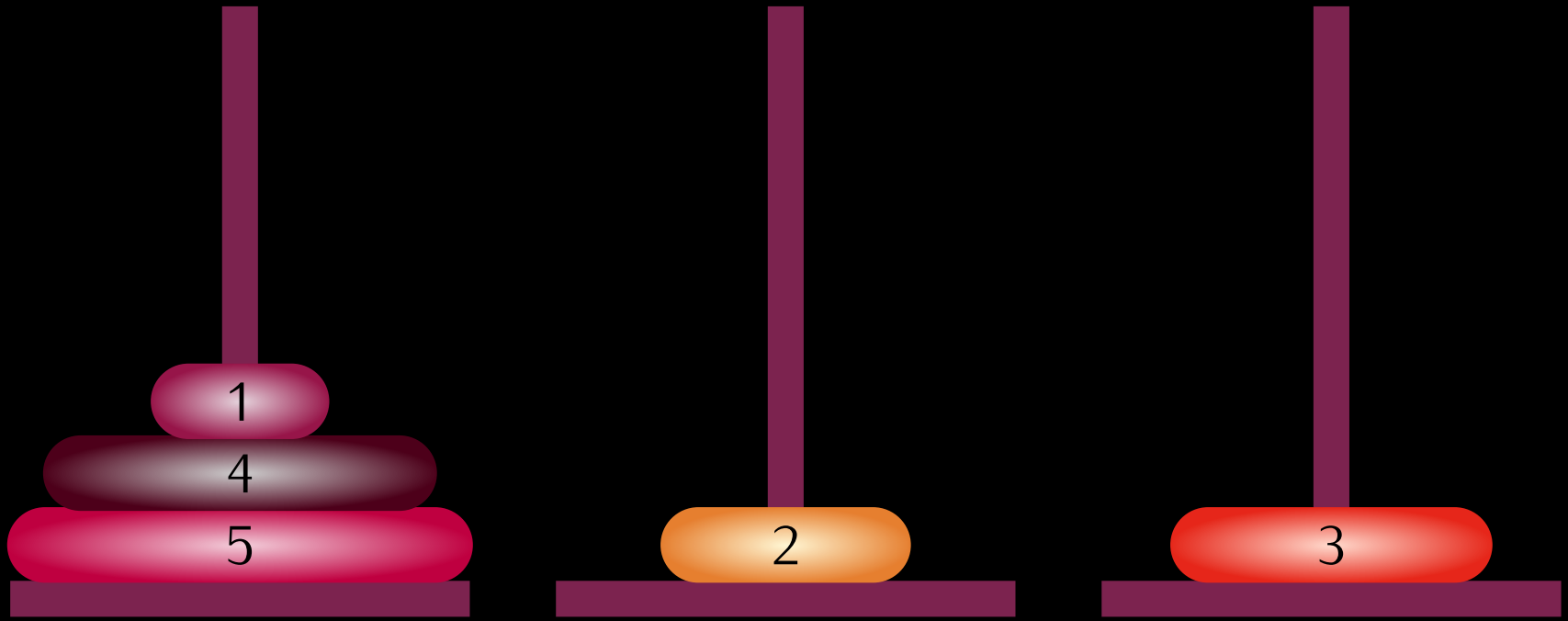
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



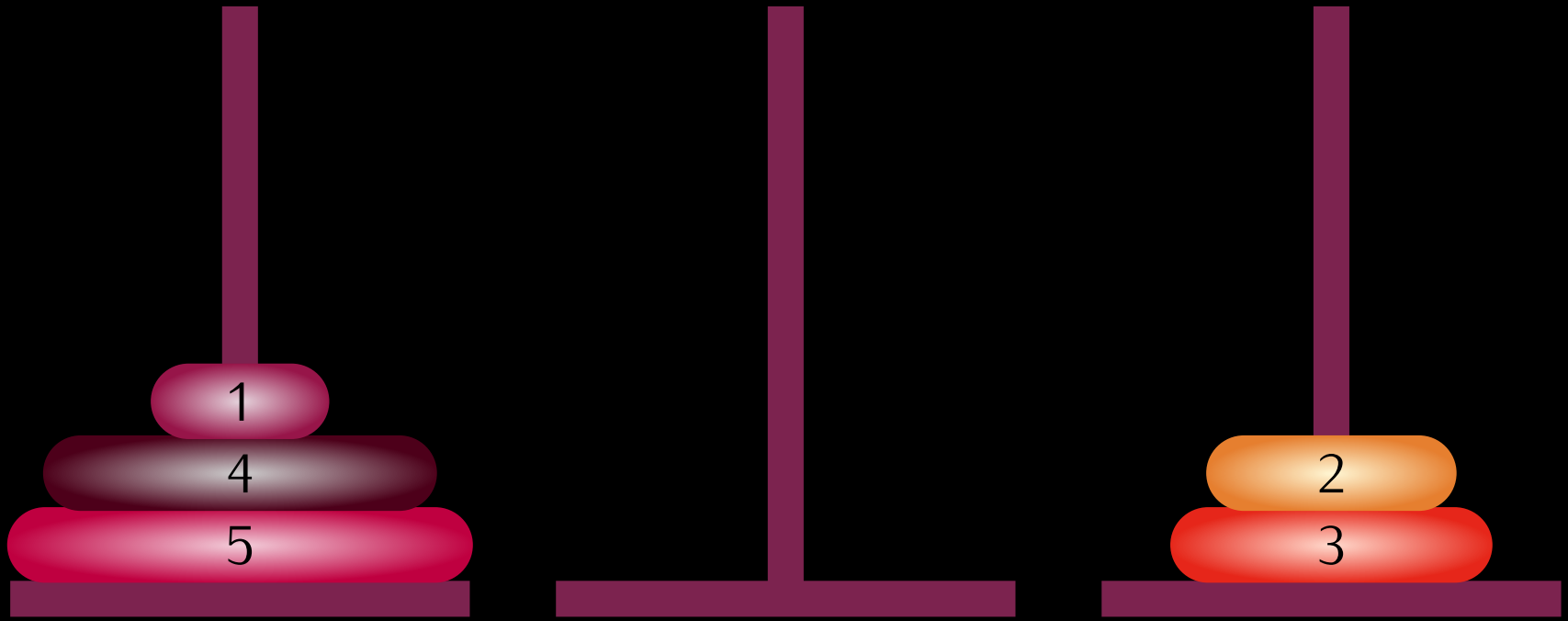
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



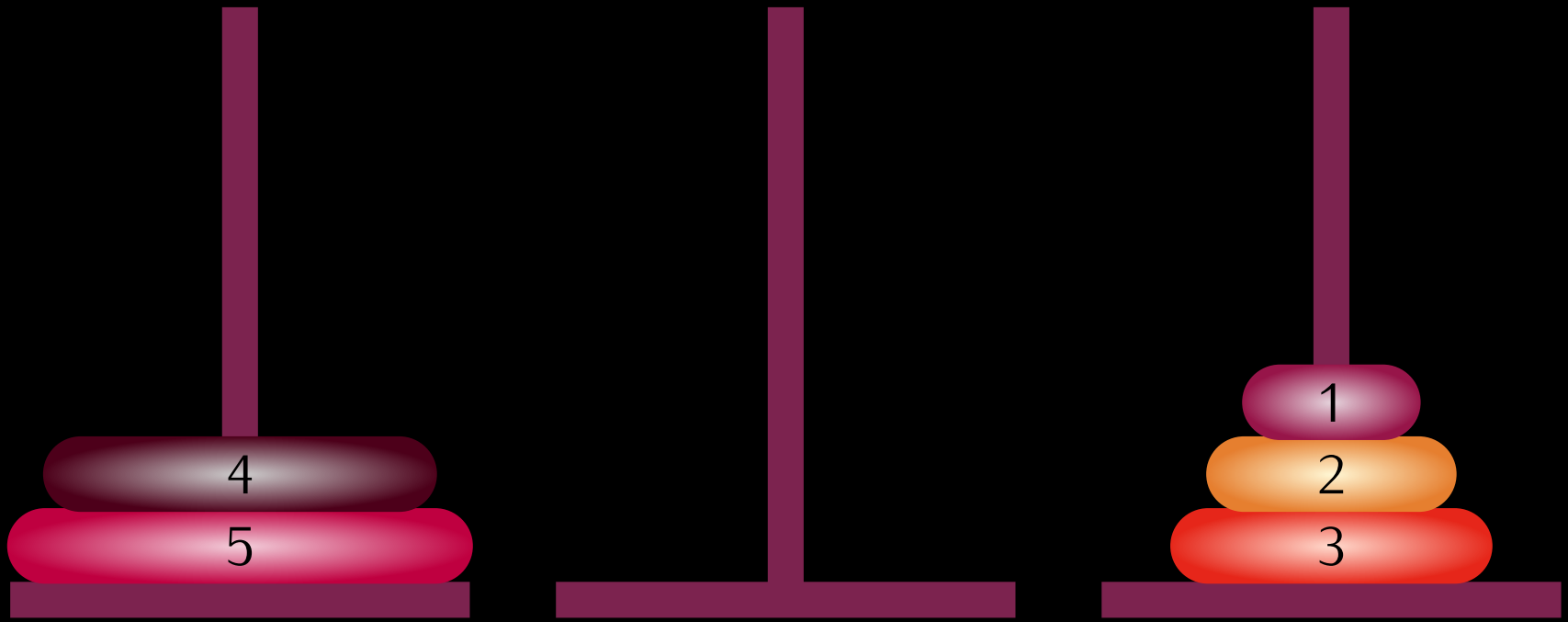
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



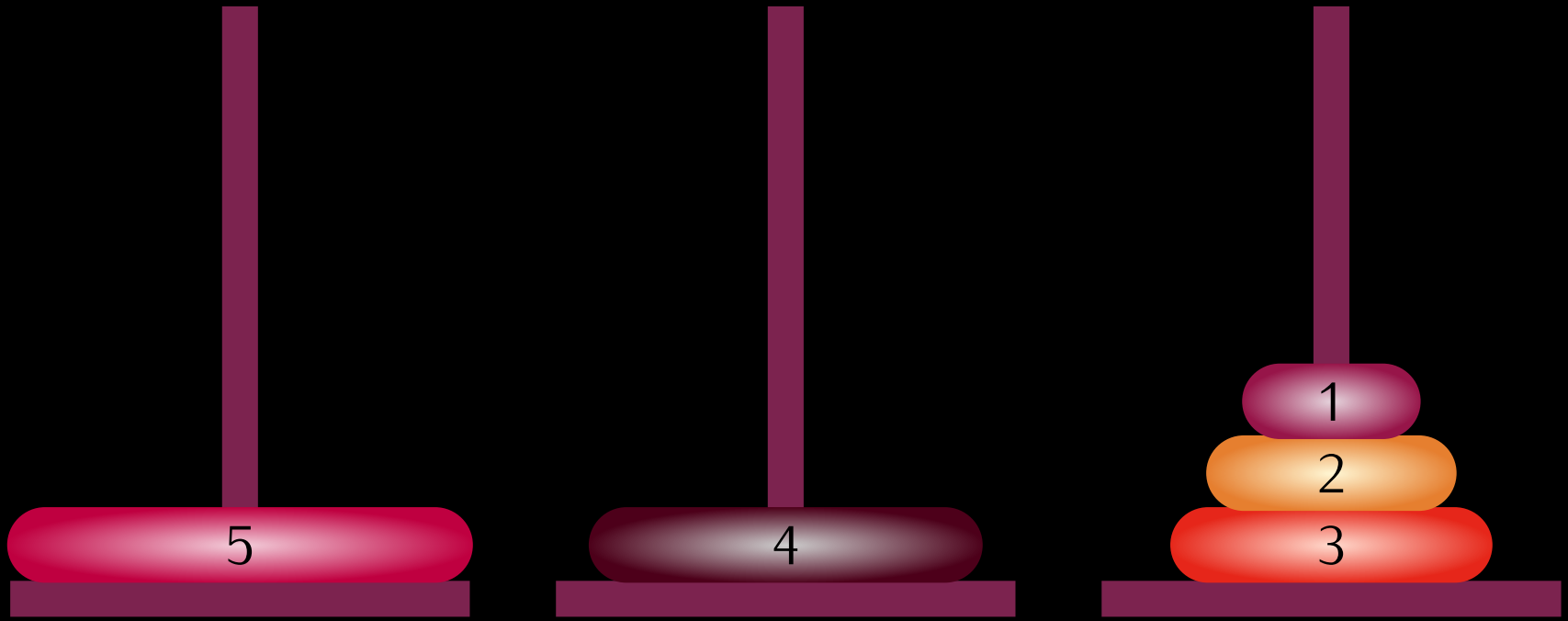
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



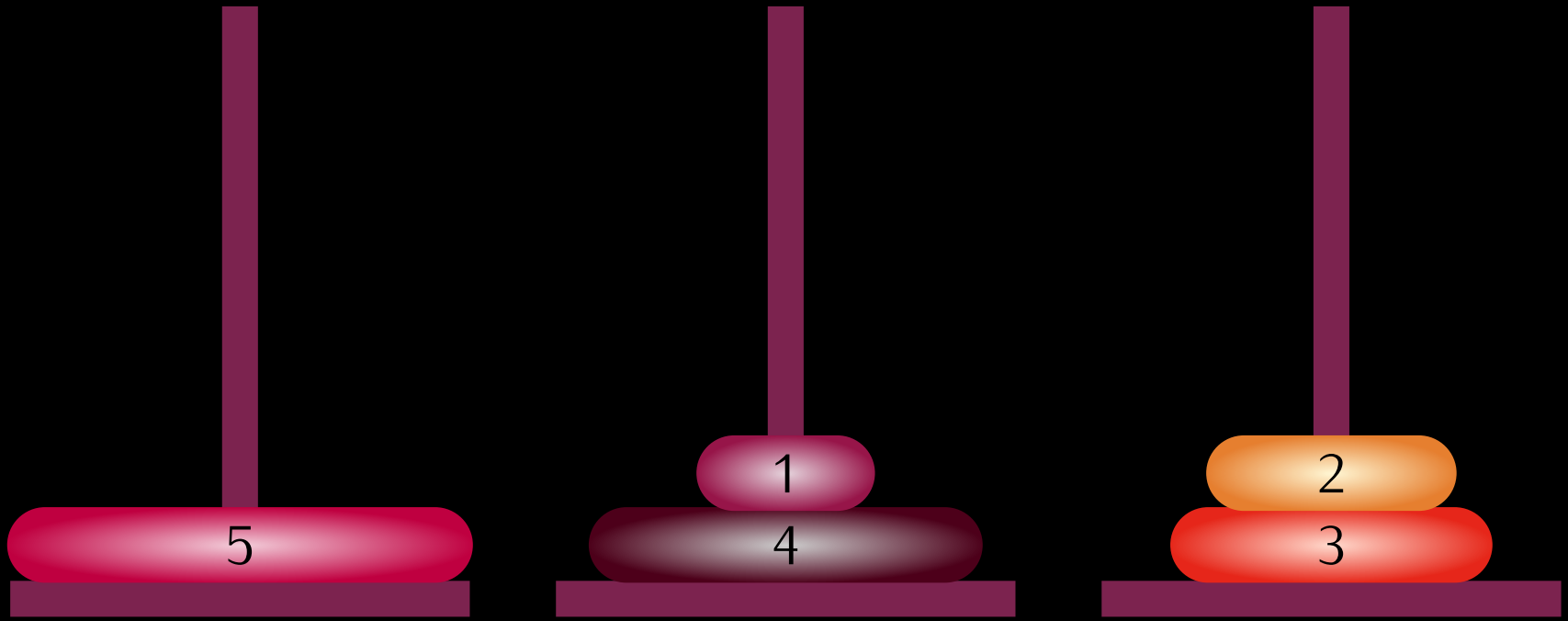
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



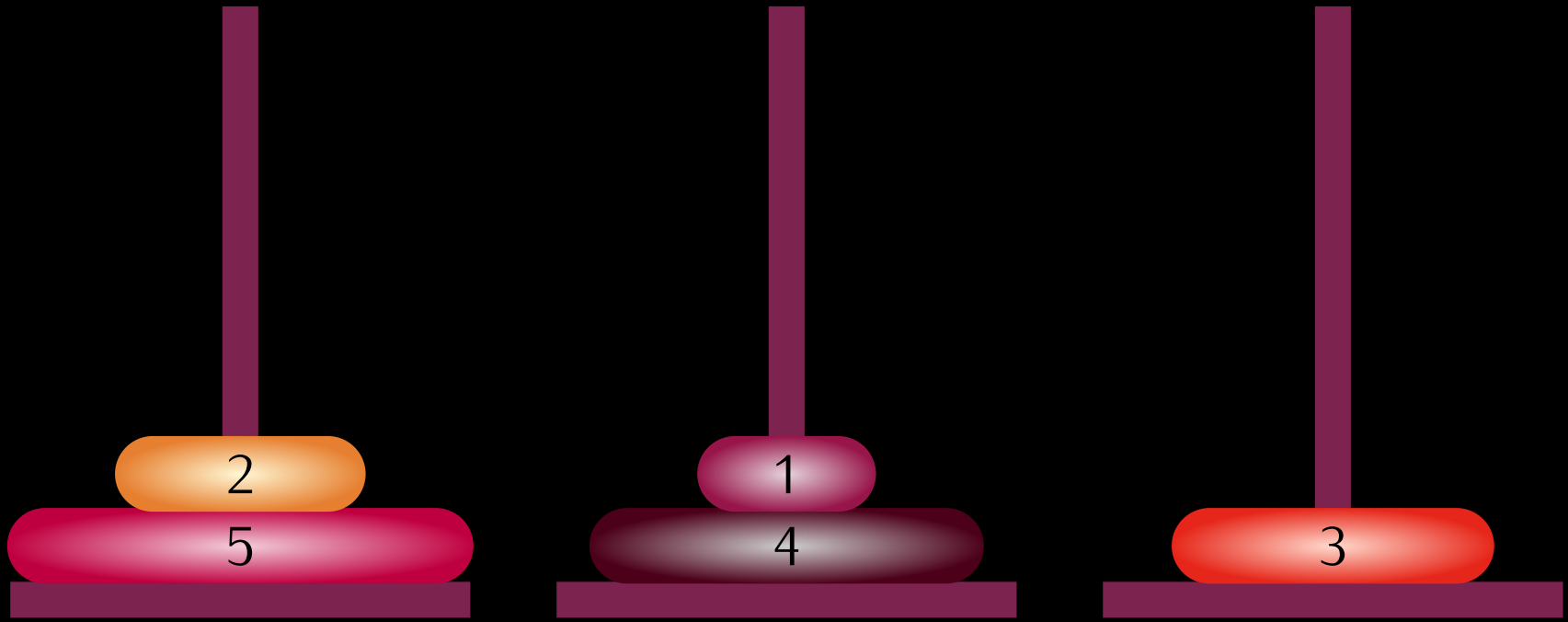
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



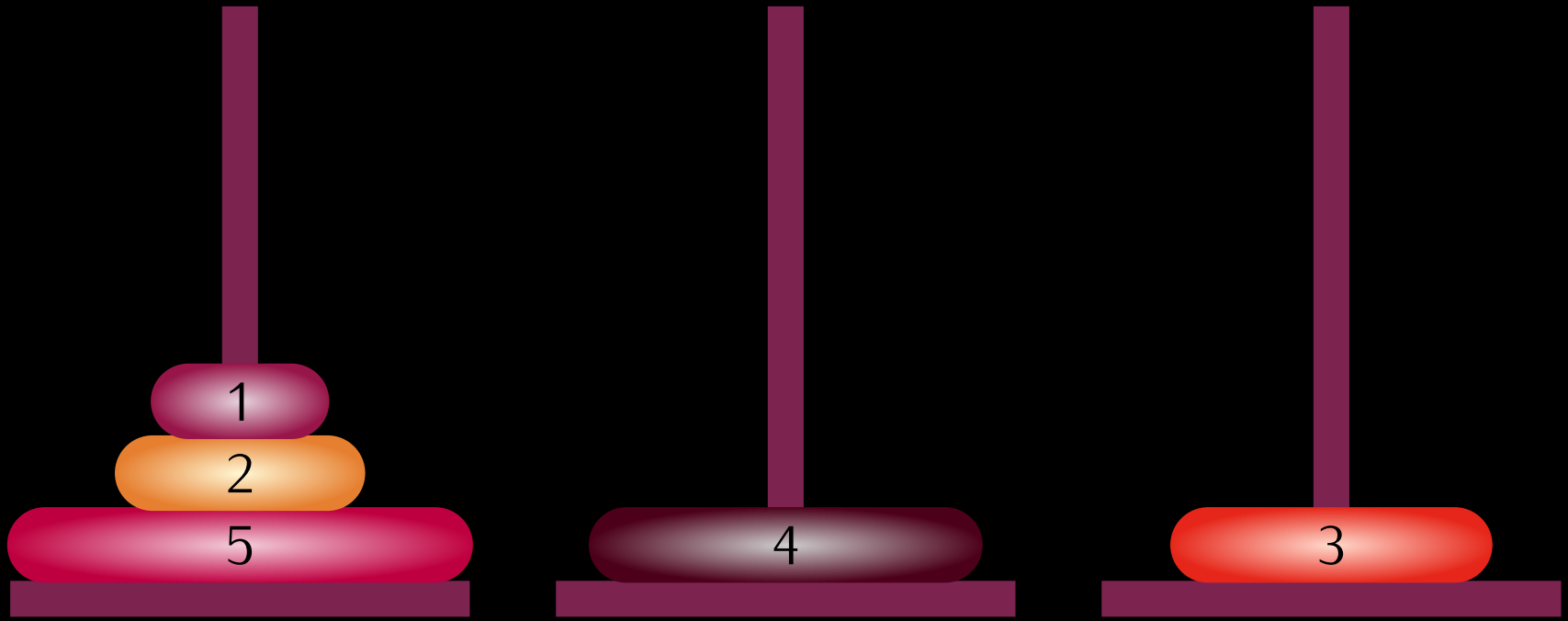
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



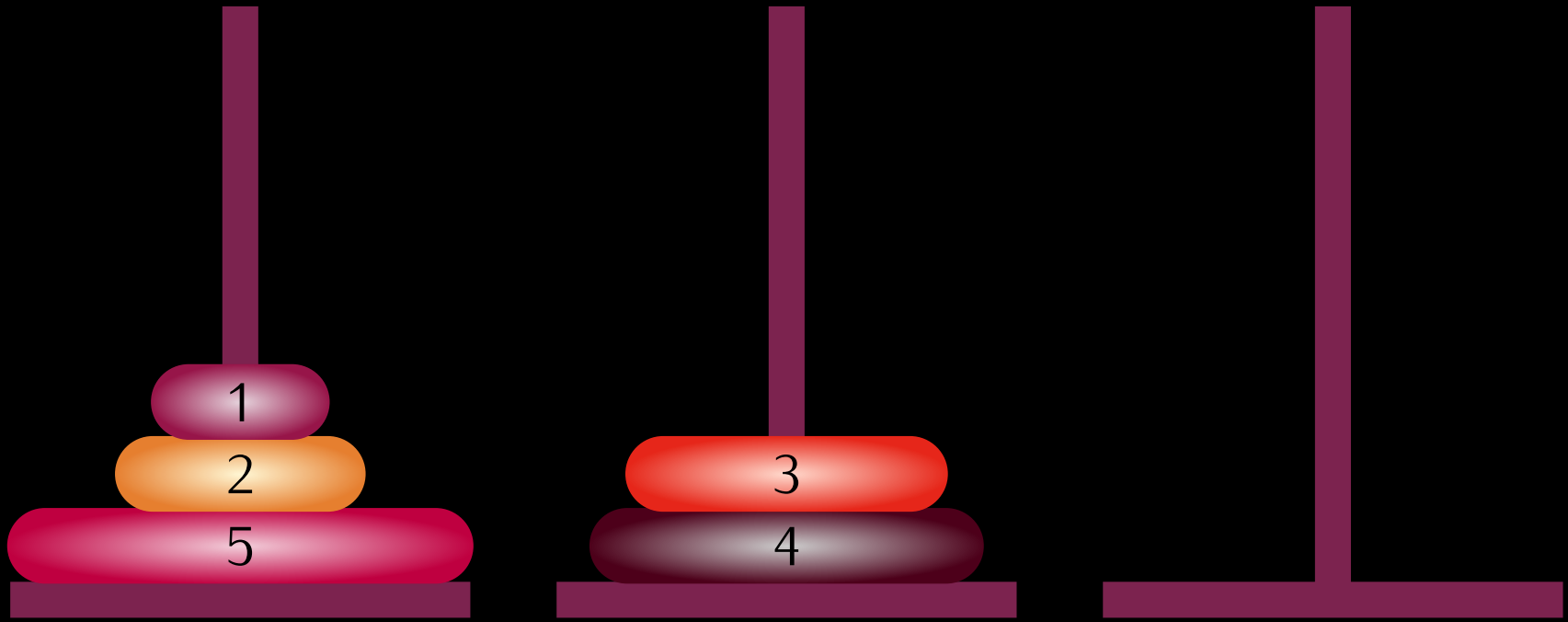
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 5 Disques



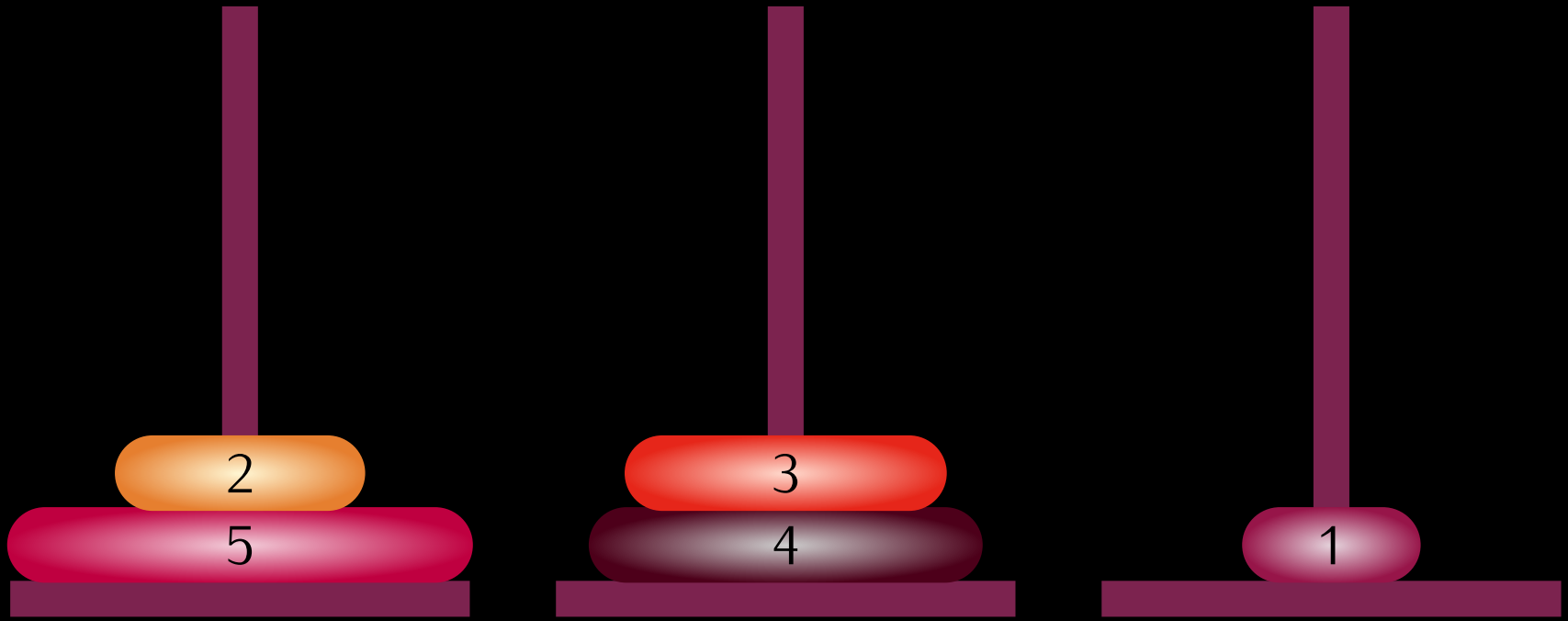
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



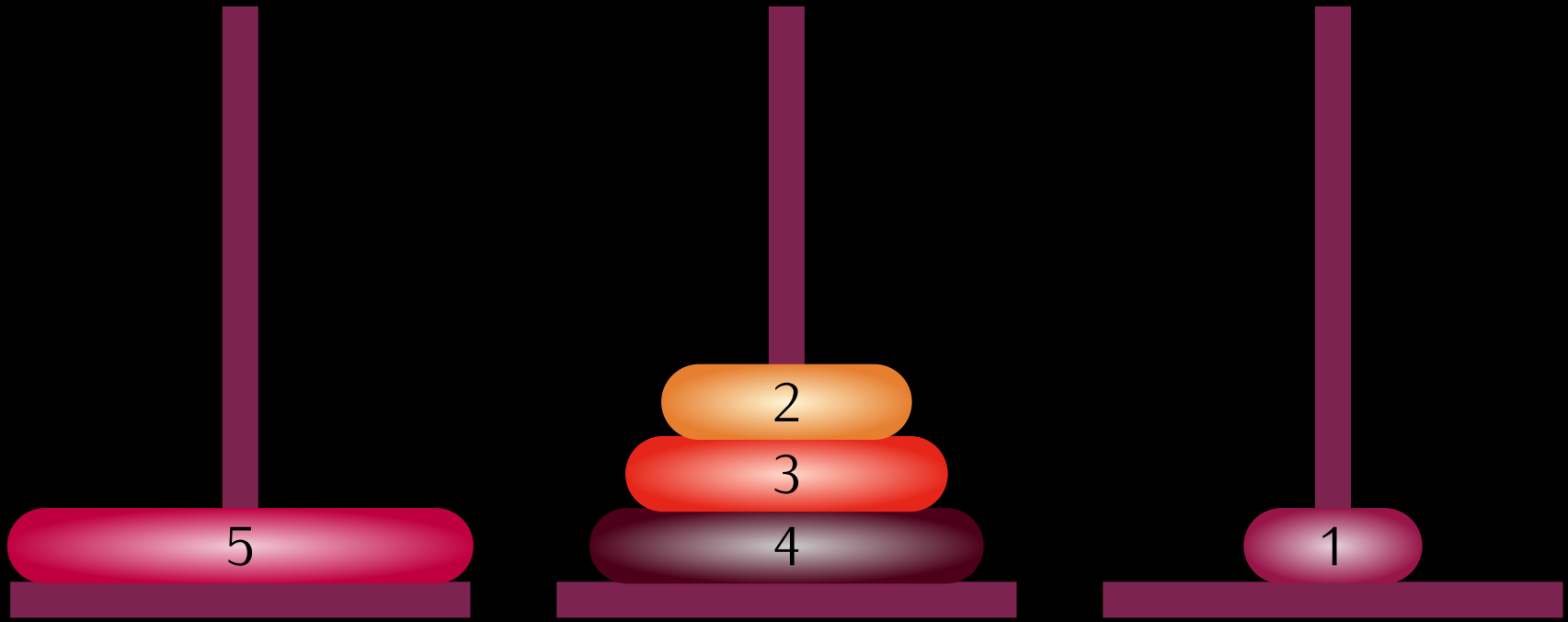
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



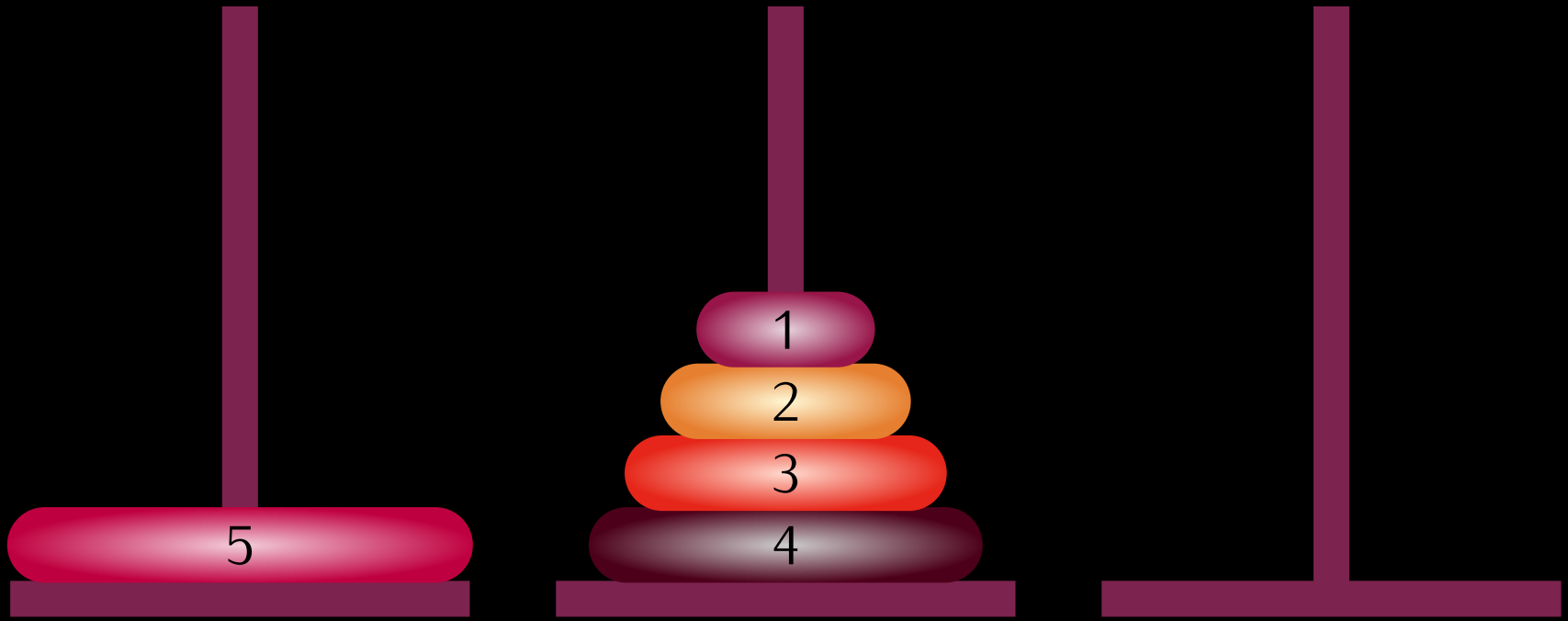
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



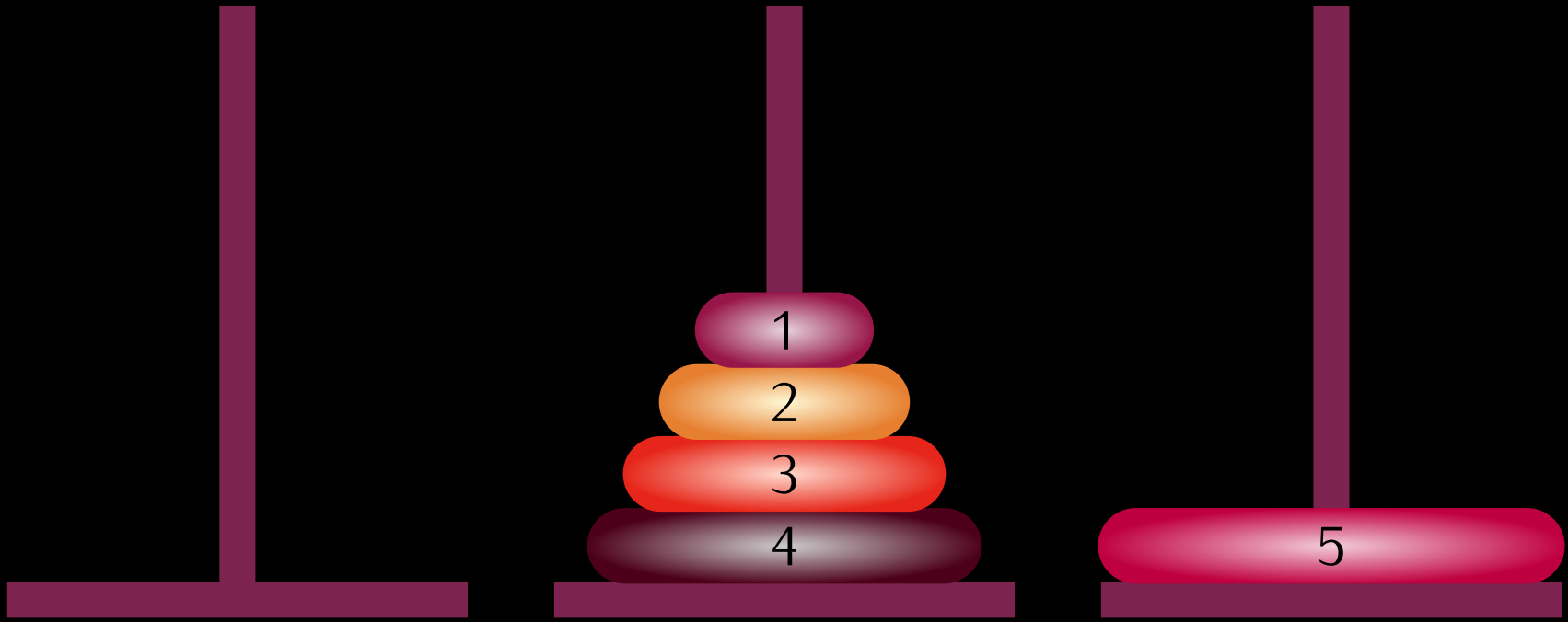
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



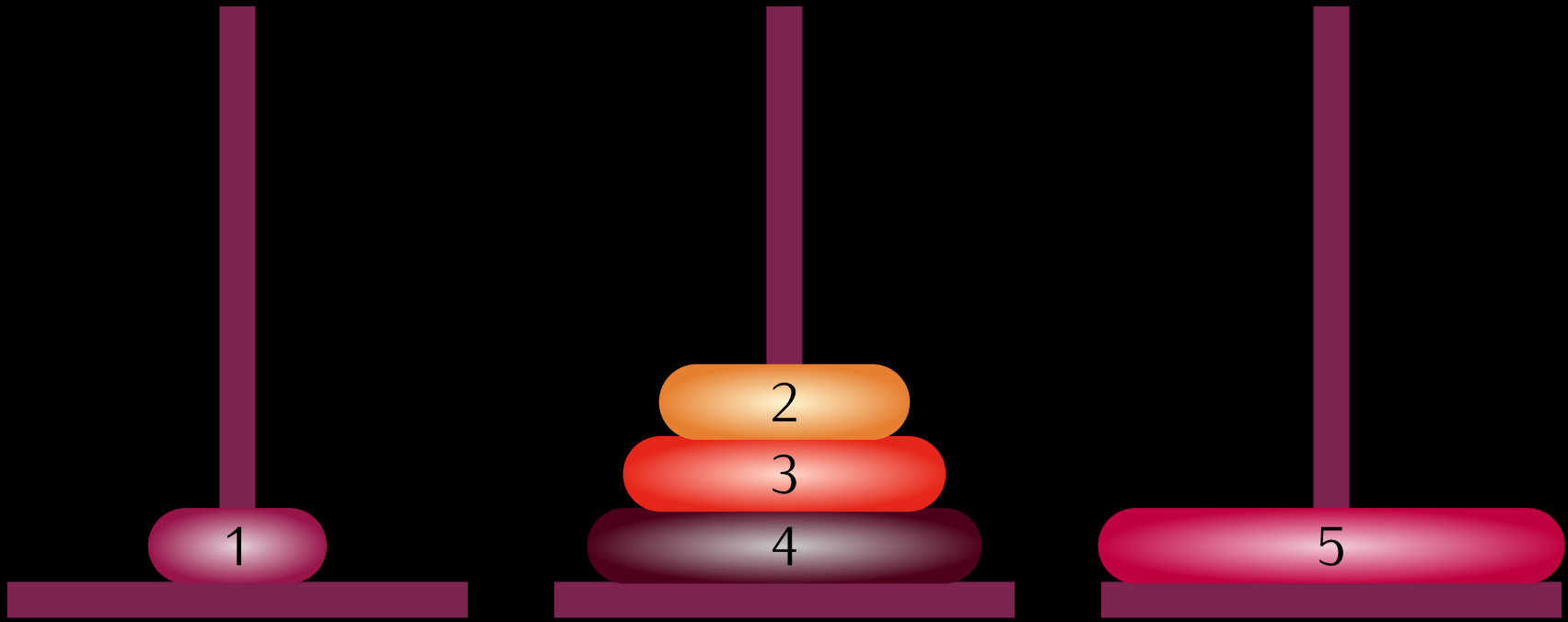
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



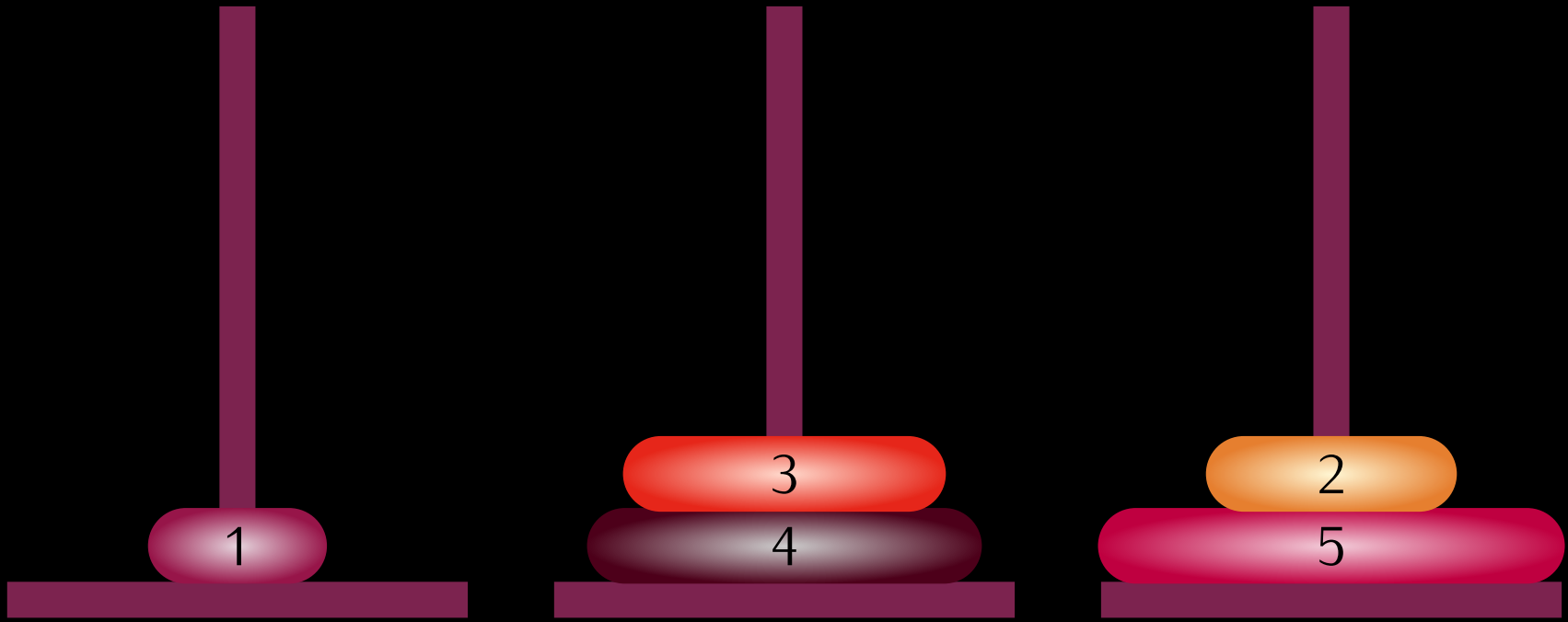
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



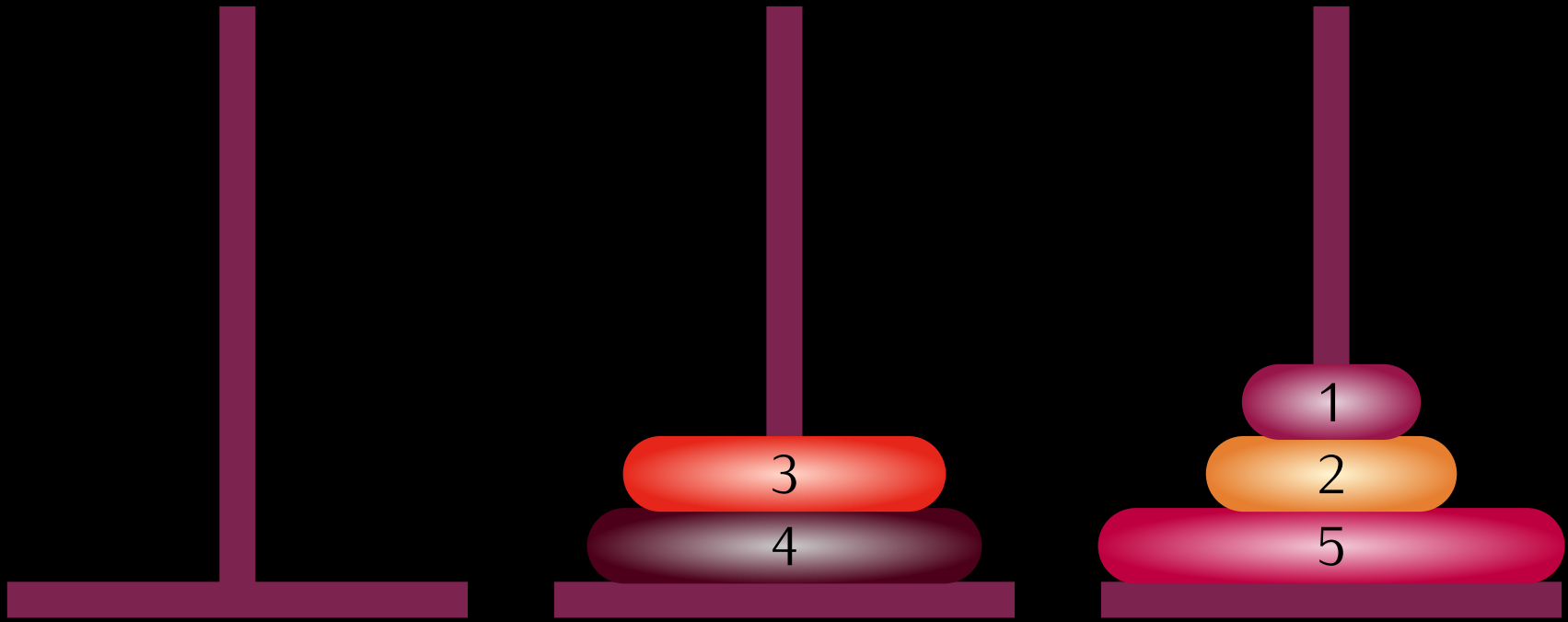
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



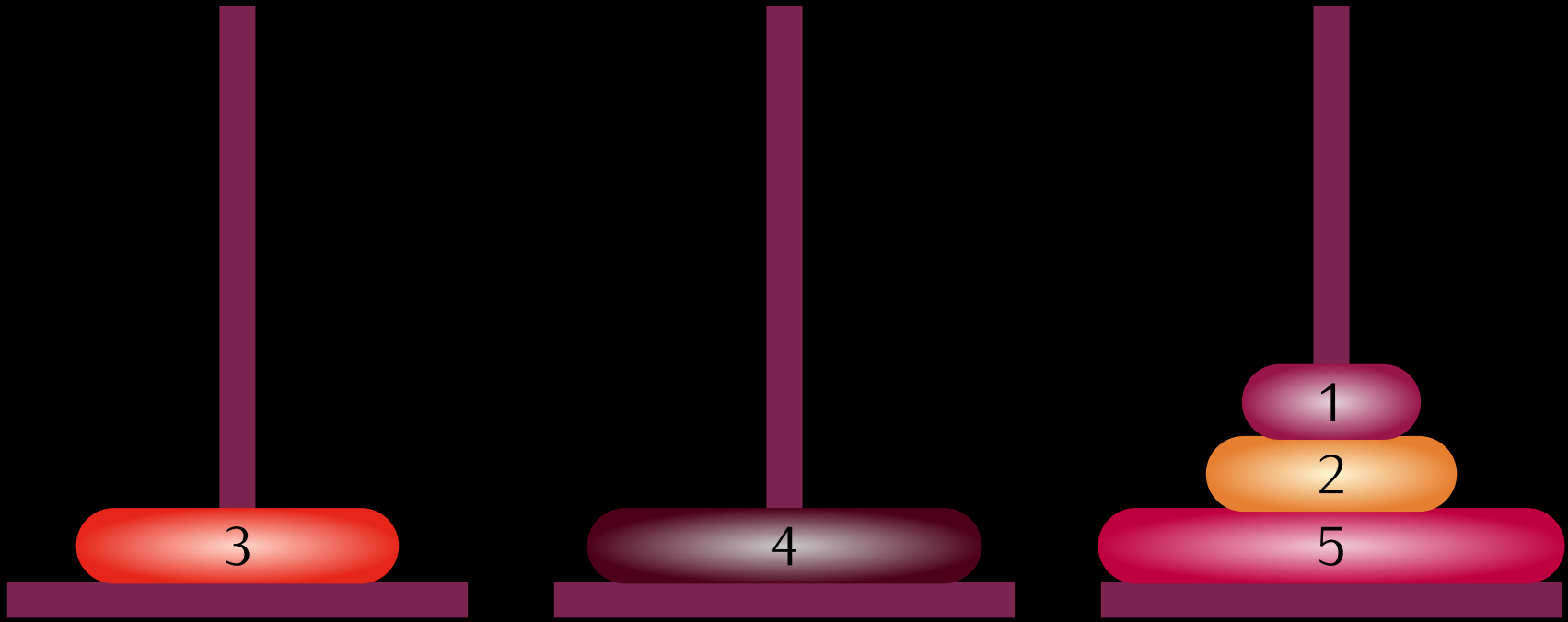
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



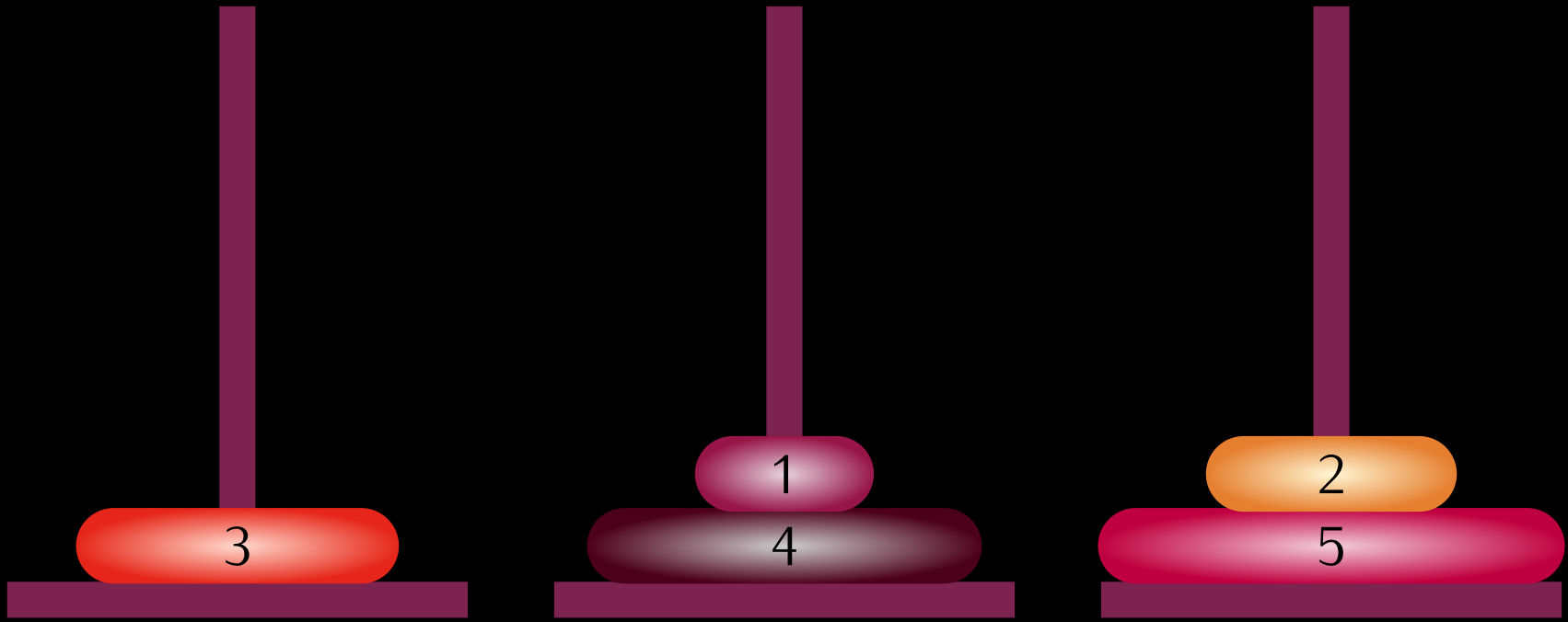
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



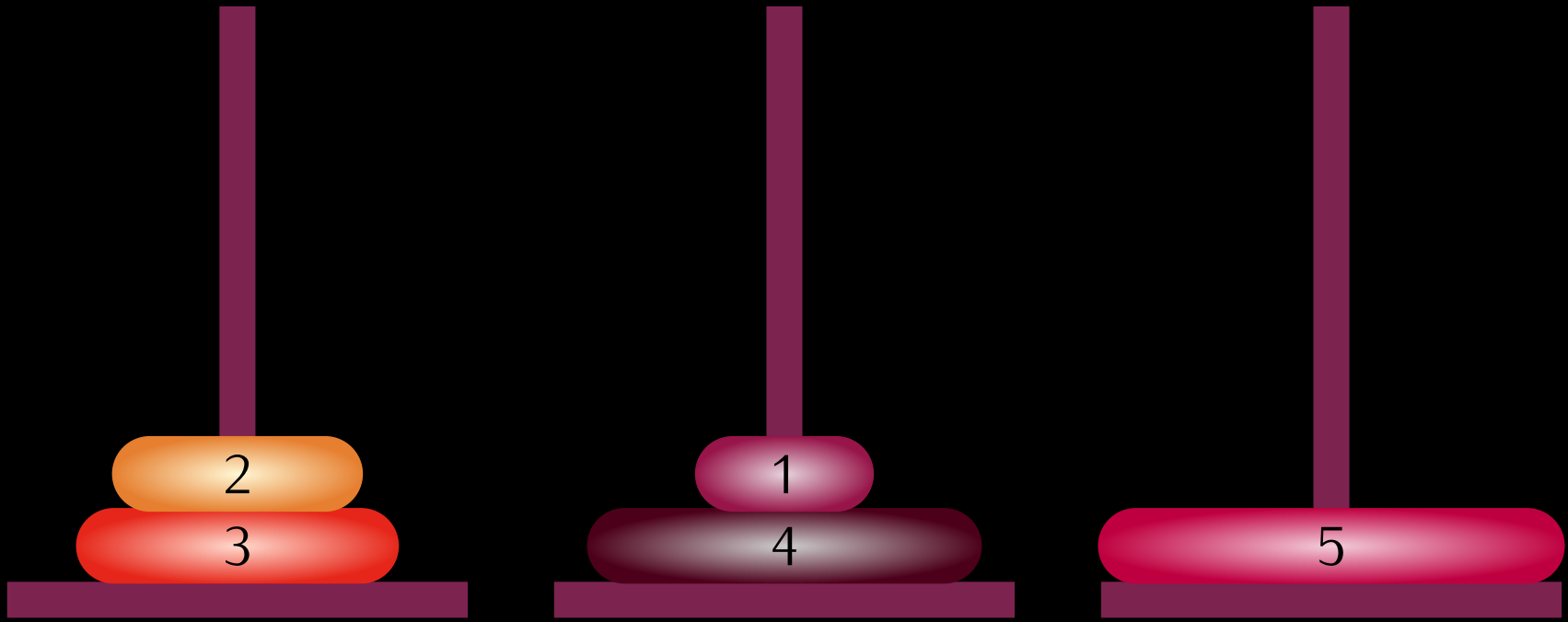
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



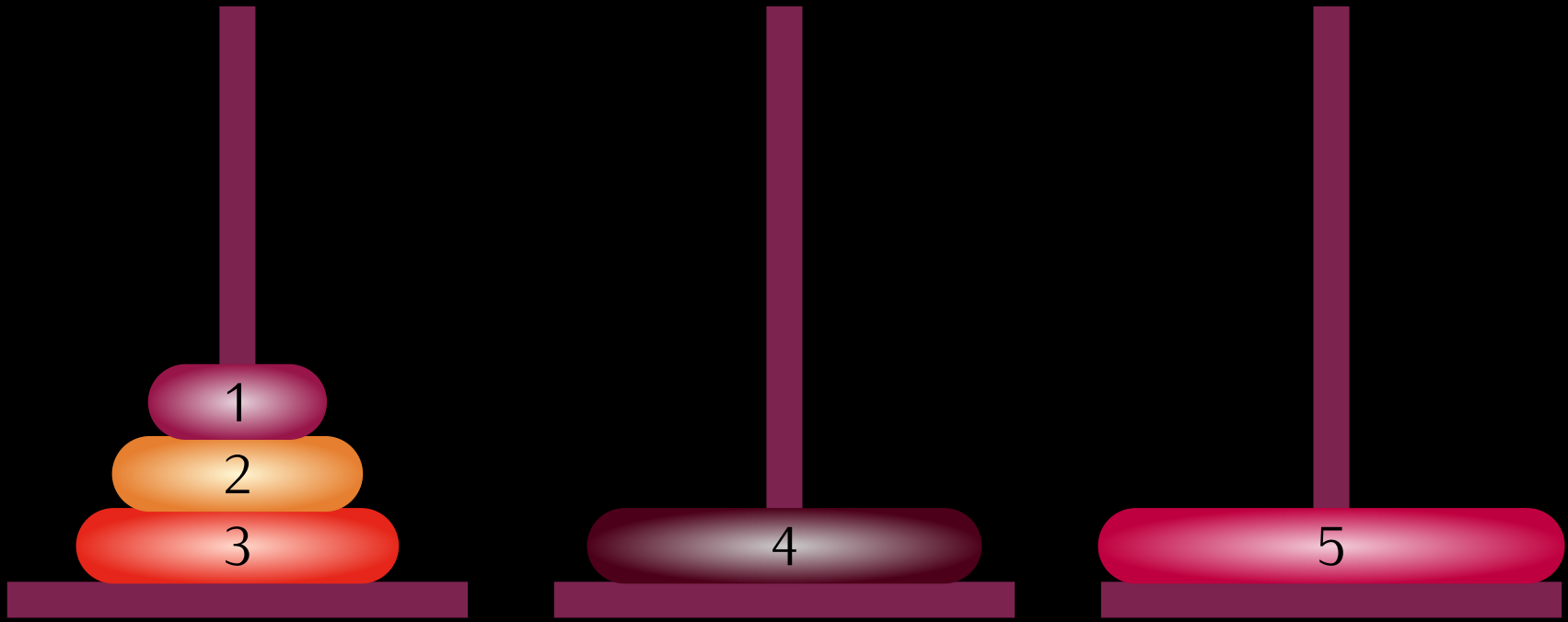
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



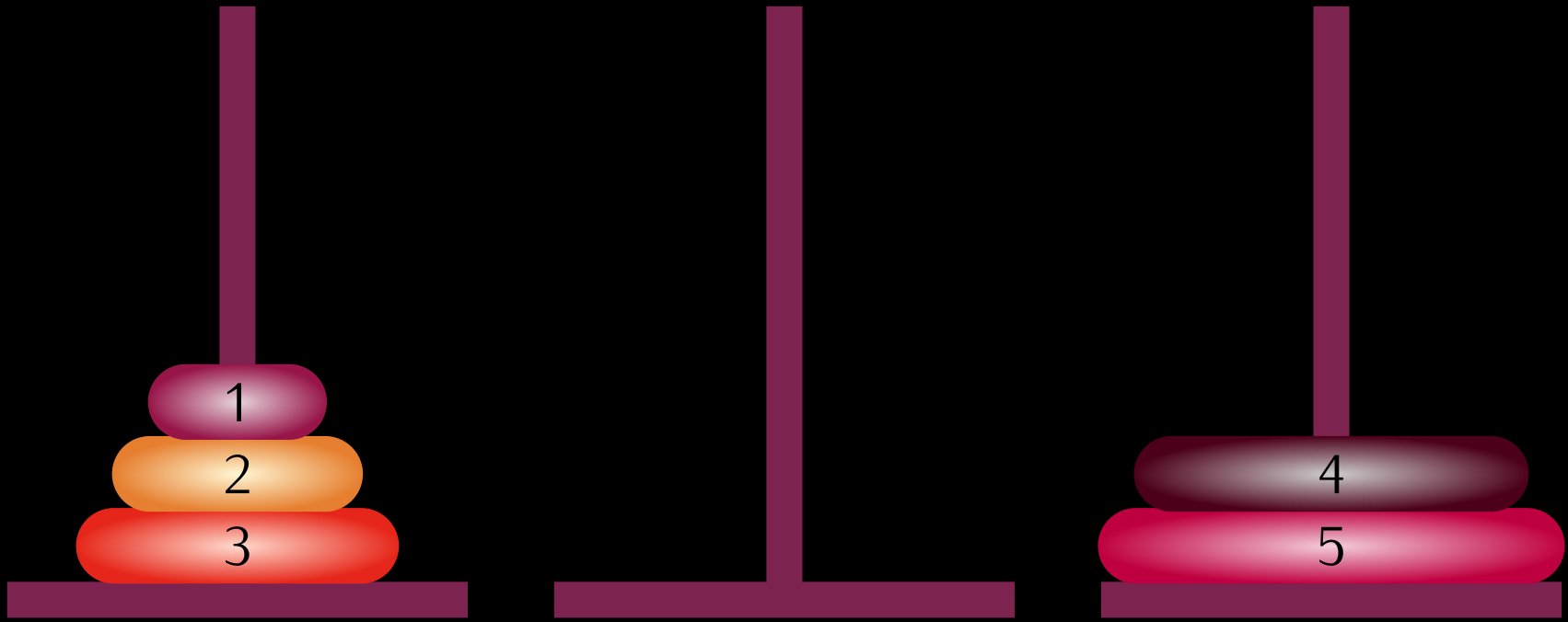
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 5 Disques



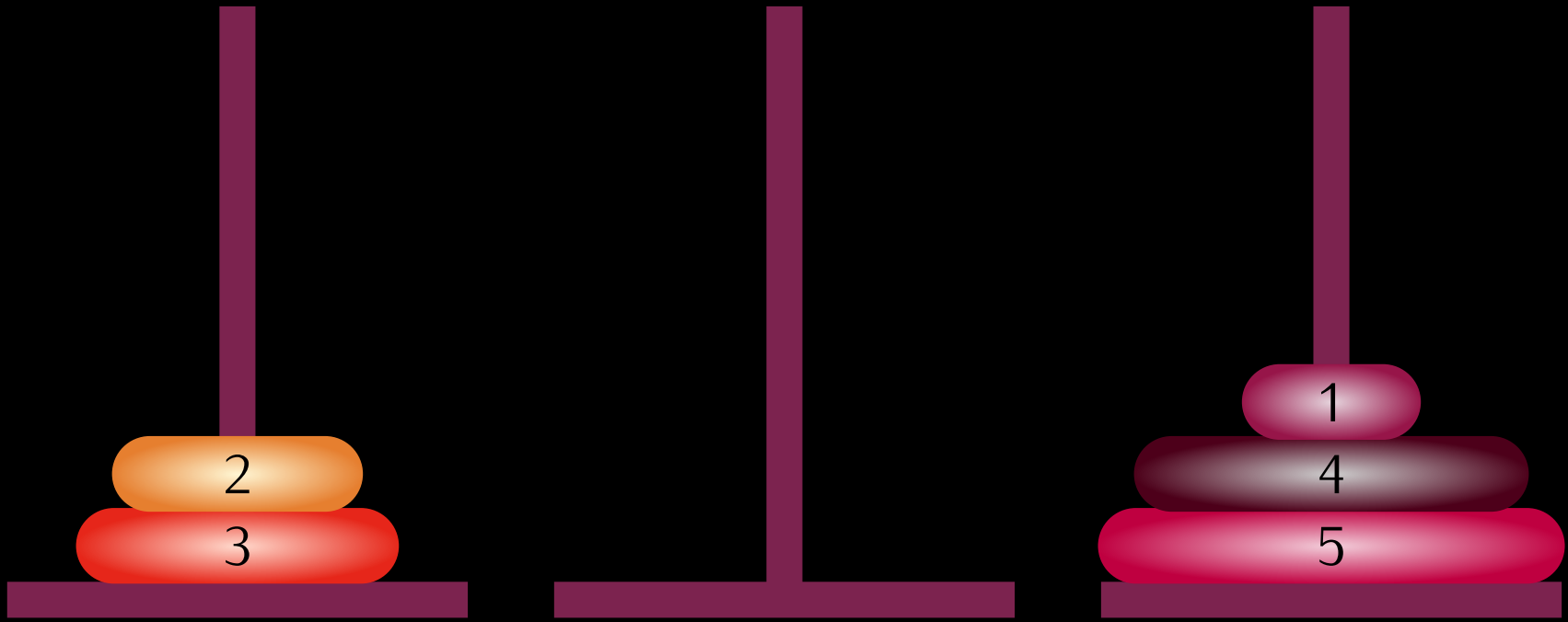
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



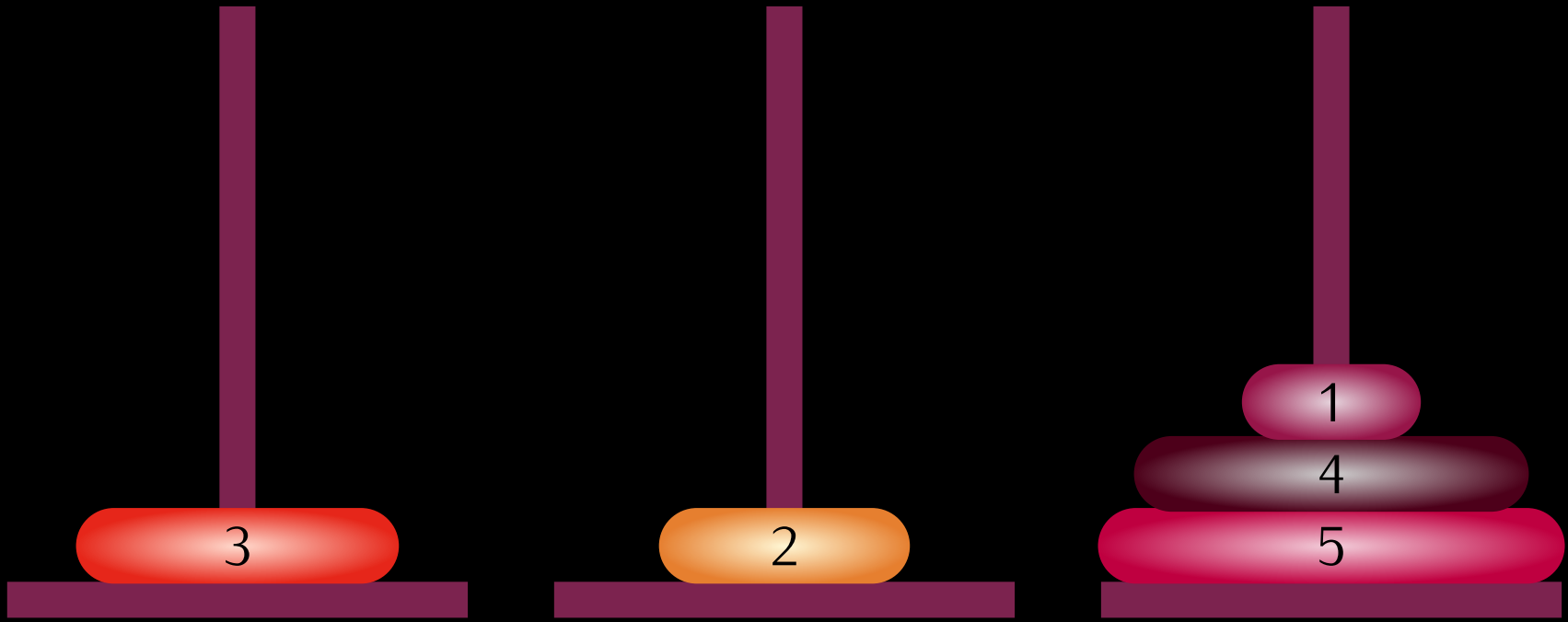
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



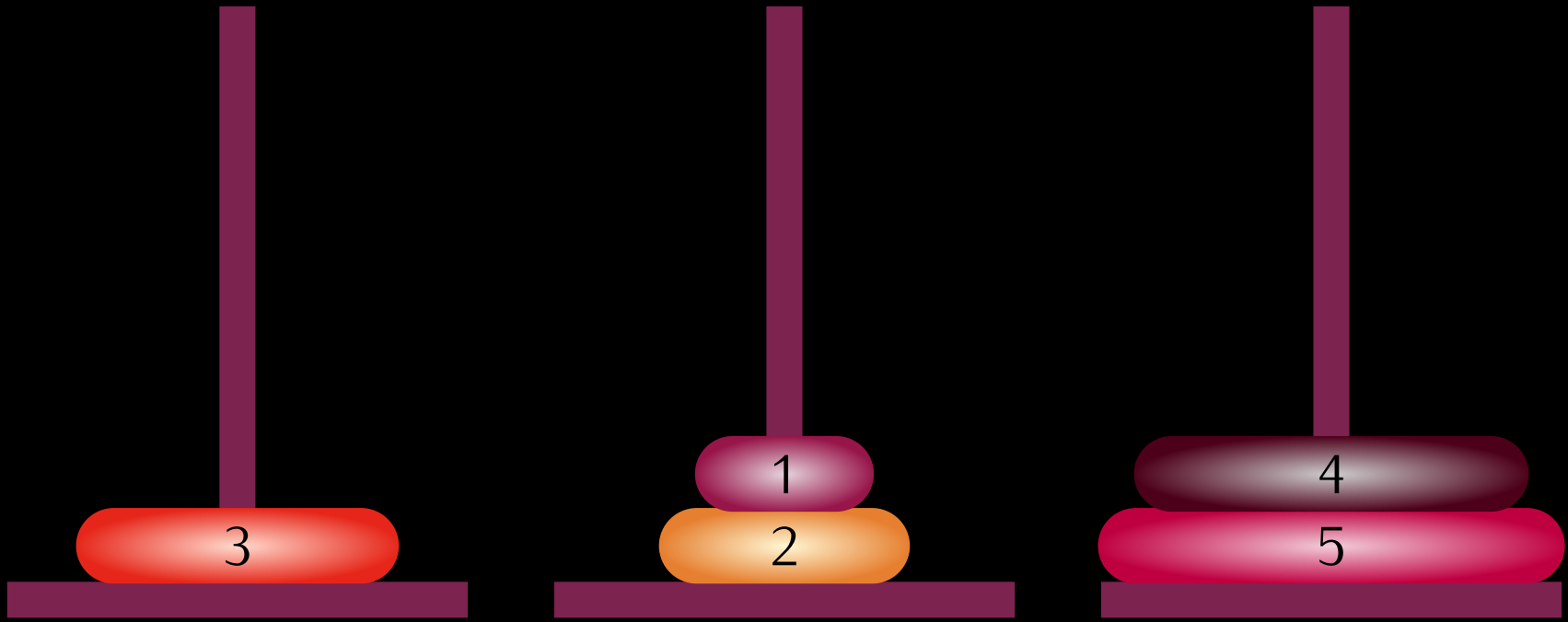
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



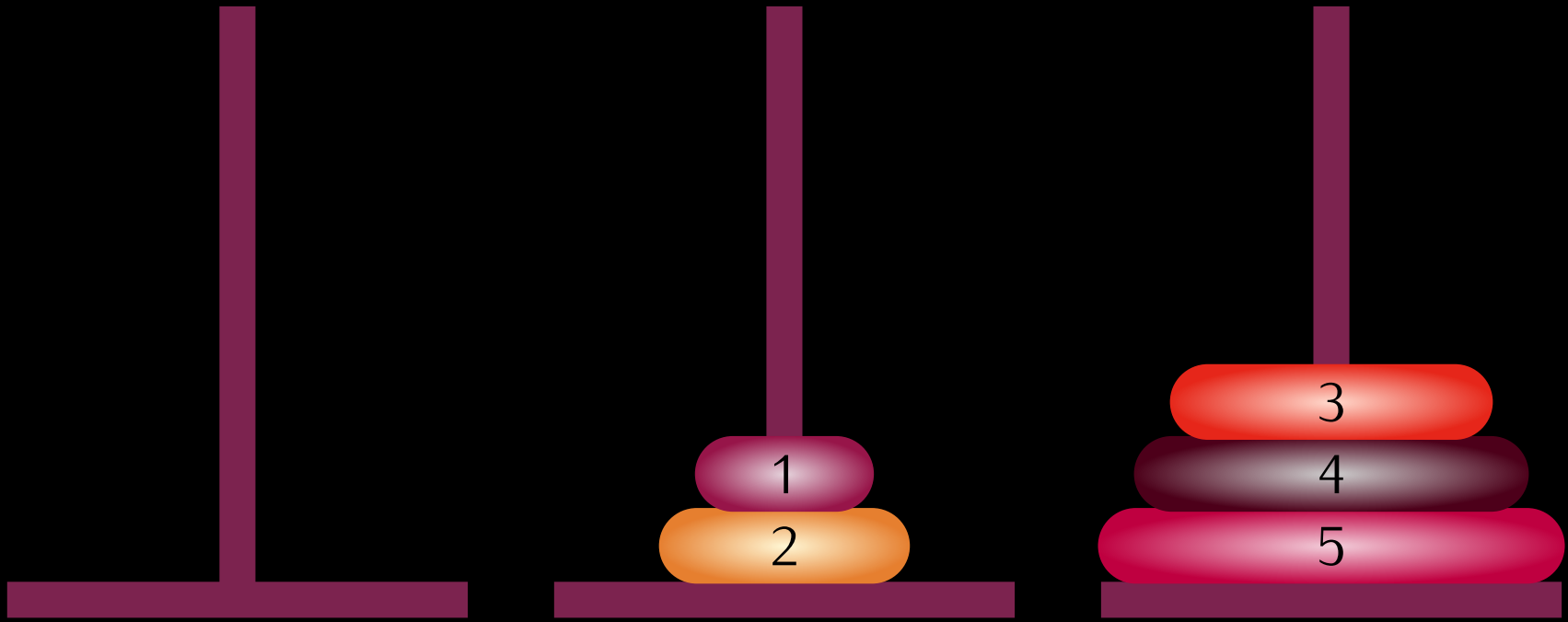
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



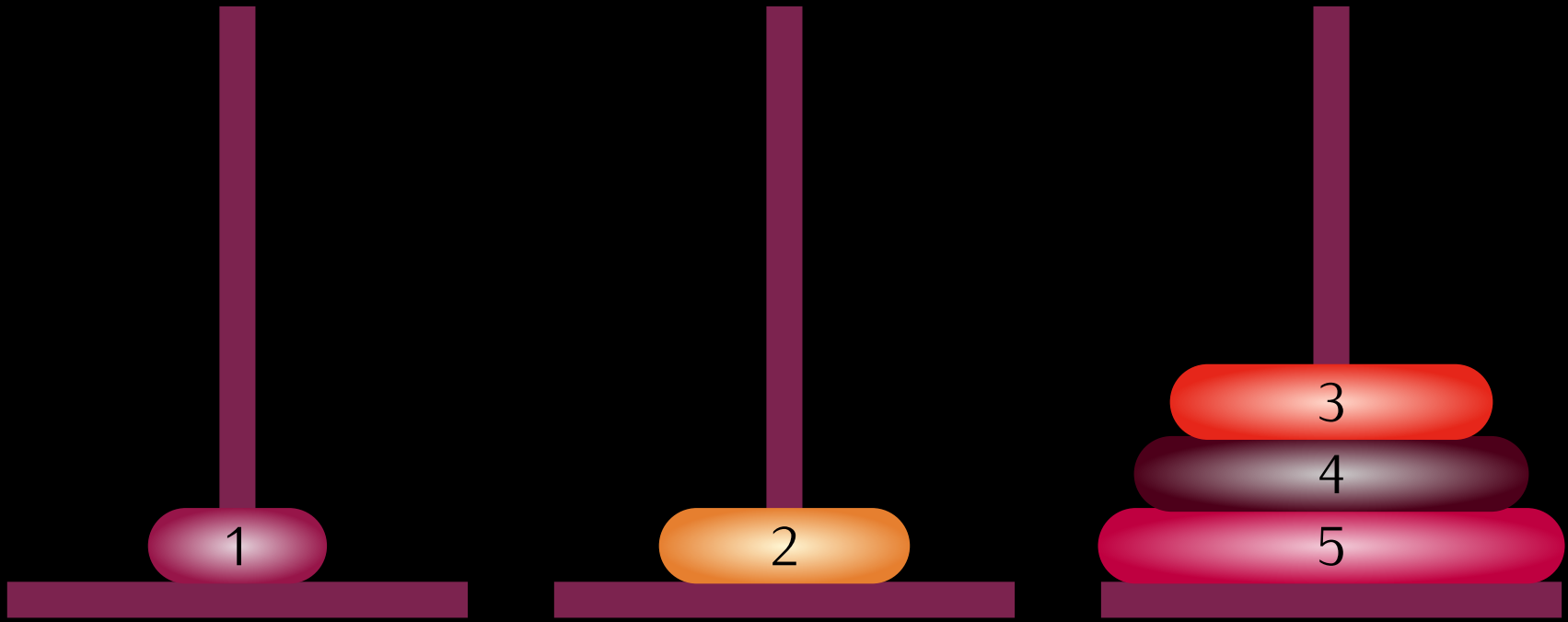
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



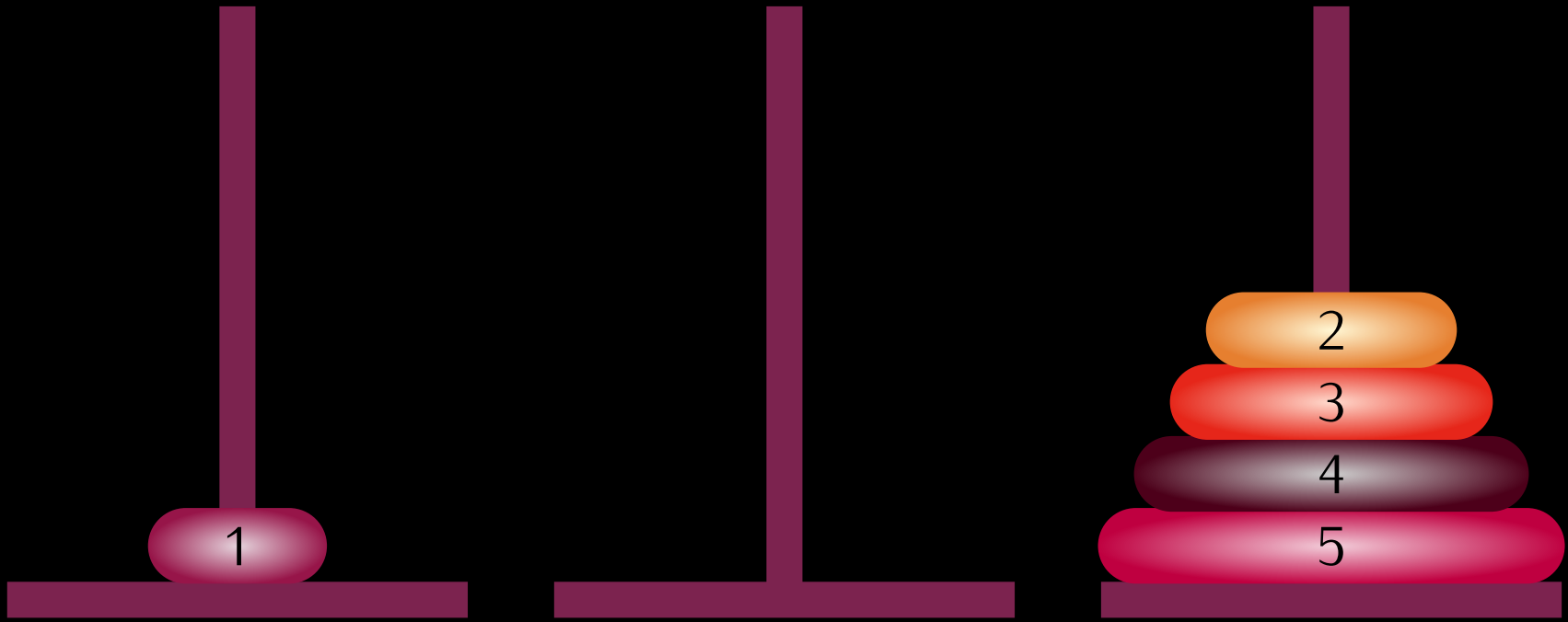
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



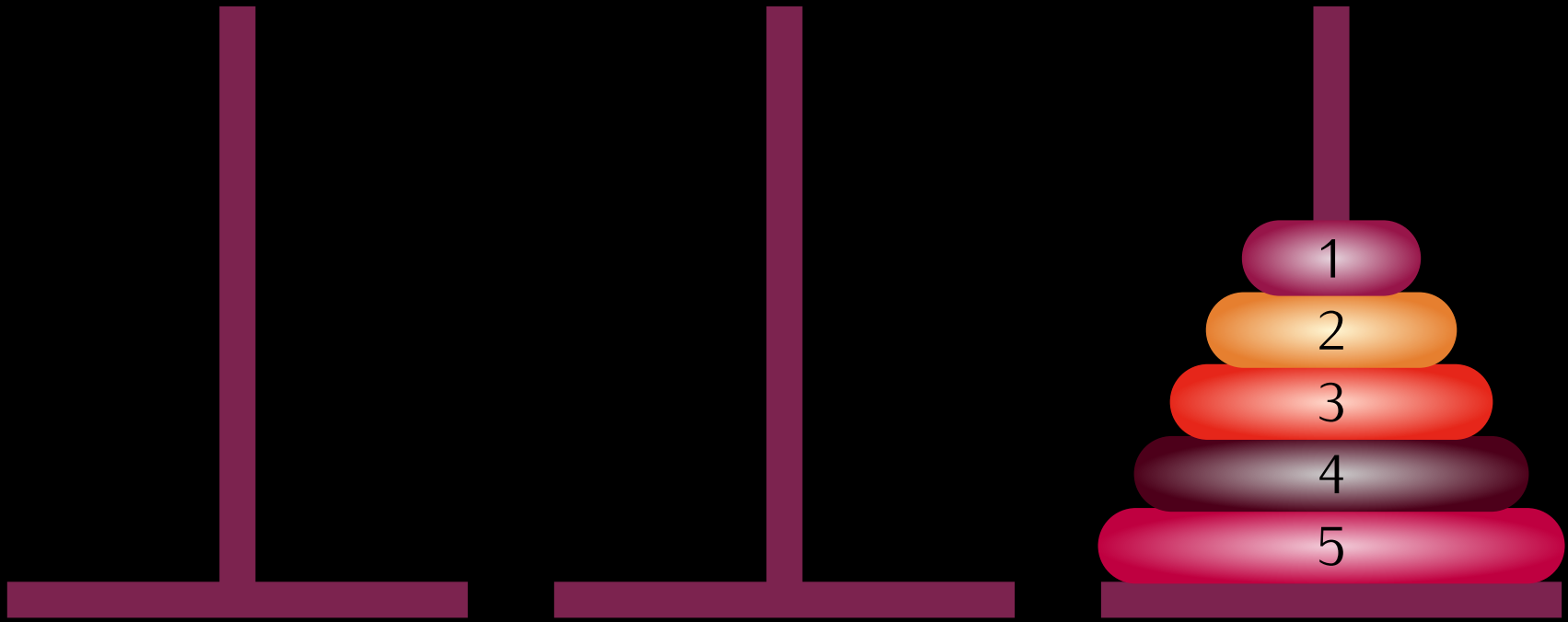
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 2 vers la tige 3.

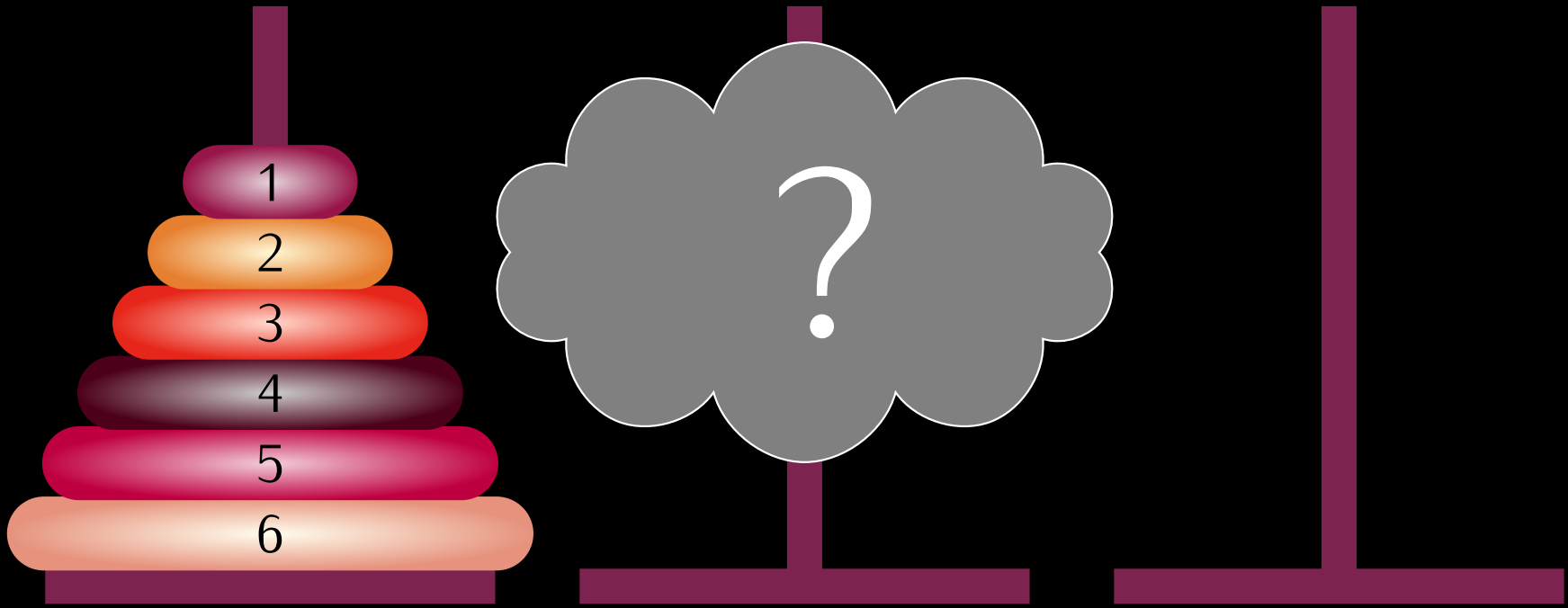
Tours de Hanoi – 5 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques

Tour de Hanoi – 6 Disques



Défi

Déterminez le nombre de mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

Créez une fonction qui affiche les mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

Déterminez le nombre de mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

Créez une fonction qui affiche les mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

Déterminez le nombre de mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

Créez une fonction qui affiche les mouvements effectués pour déplacer n disques des Tours de Hanoï.

```
In [52]: hanoi(3)
On déplace le disque de la tige 1 vers la tige 3
On déplace le disque de la tige 1 vers la tige 2
On déplace le disque de la tige 3 vers la tige 2
On déplace le disque de la tige 1 vers la tige 3
On déplace le disque de la tige 2 vers la tige 1
On déplace le disque de la tige 2 vers la tige 3
On déplace le disque de la tige 1 vers la tige 3
```