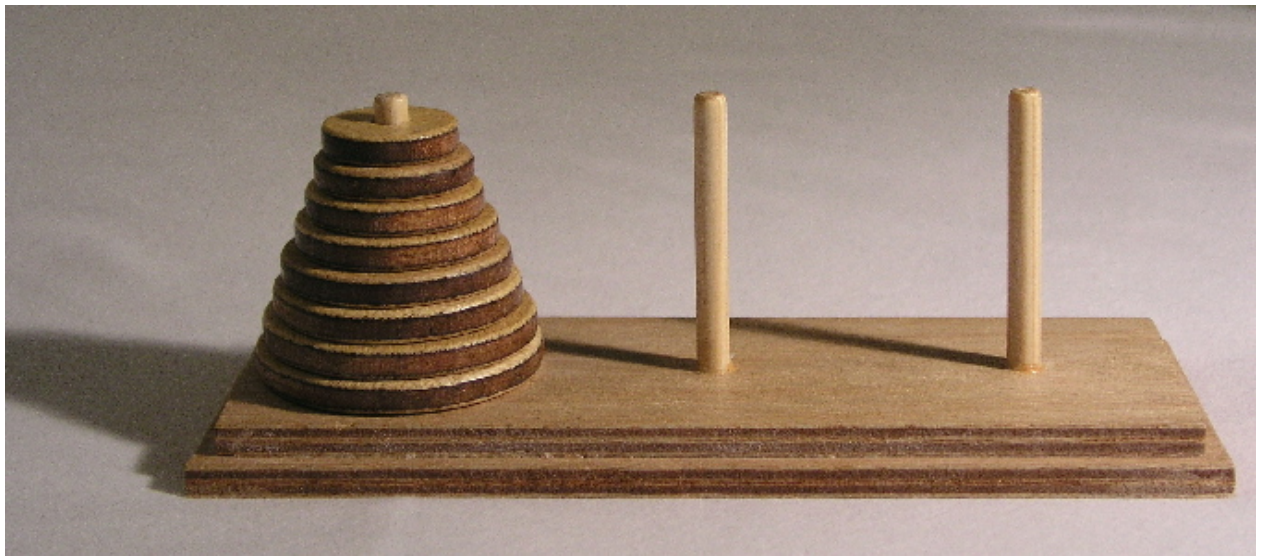
Semaine 10

Initiation à l'algorithmique et programmation

Revekka Kyriakoglou

Tours de Hanoi

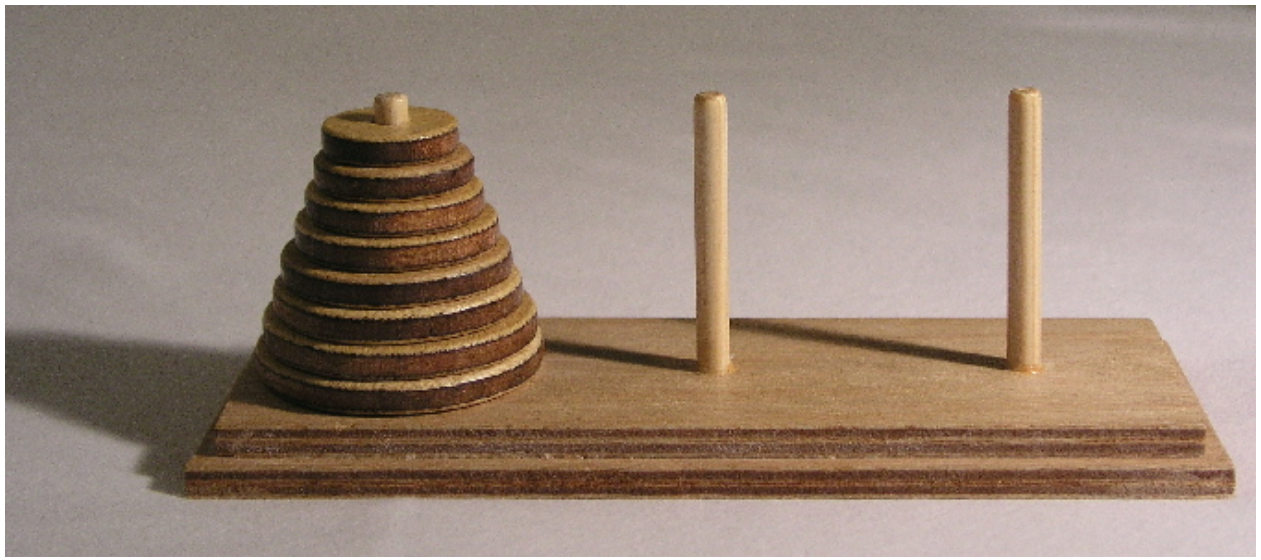


Les tours de Hanoï sont un jeu de réflexion consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

1. on ne peut déplacer plus d'un disque à la fois ;
2. on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ.

Tours de Hanoi



Ce problème peut être résolu par **récursivité** ! ! !

Tours de Hanoi

Récursivité

Une technique de résolution de problèmes dans laquelle les problèmes sont résolus en les réduisant à des problèmes plus petits de même forme.

La récursivité dans la vie réelle

? Combien d'élèves au total sont assis directement derrière vous dans votre "colonne" de la classe ?

Règles :

1. Vous ne pouvez voir que les personnes qui se trouvent directement devant et derrière vous. Vous ne pouvez donc pas regarder derrière vous et compter.
2. Vous avez le droit de poser des questions aux personnes qui se trouvent devant / derrière vous.

Comment pouvons-nous résoudre ce problème de manière récursive ?

La récursivité dans la vie réelle

Solution :

1. La première personne regarde derrière elle pour voir s'il y a une personne. Si ce n'est pas le cas, la personne répond "0".
2. S'il y a une personne, elle répète l'étape 1 et attend une réponse.
3. Lorsqu'une personne reçoit une réponse, elle ajoute "1" pour la personne derrière elle, et elle répond à la personne qui lui a posé la question.

La récursivité dans la vie réelle

```
def numStudentsBehind(student_curr):
    if (noOneBehind(student_curr)):
         return 0
    else:
        student_behind = getBehind(student_curr)
        return numStudentsBehind(student_behind) + 1
```

La récursivité dans la vie réelle

```
def numStudentsBehind(student_curr):
    if (noOneBehind(student_curr)):
         return 0
    else:
        student_behind = getBehind(student_curr)
        return numStudentsBehind(student_behind) + 1
```

- La partie `if` est le cas de base !
- La partie récursive est l'appel de la fonction elle-même !

Récursivité

- **Cas de Base :** C'est la condition sous laquelle la récursivité s'arrête. Le cas de base empêche la fonction de s'appeler indéfiniment, évitant ainsi une boucle infinie ou une erreur de dépassement de pile. Il gère généralement le problème le plus simple, le plus petit à résoudre directement.

- **Cas Récursif :** C'est la partie de la fonction où elle s'appelle elle-même avec un sous-ensemble du problème original ou un pas de plus vers le cas de base. Le cas récursif réduit le problème global en instances plus petites, se rapprochant progressivement du cas de base jusqu'à ce qu'il soit atteint.

Récursivité

- **Cas de Base :** C'est la condition sous laquelle la récursivité s'arrête. Le cas de base empêche la fonction de s'appeler indéfiniment, évitant ainsi une boucle infinie ou une erreur de dépassement de pile. Il gère généralement le problème le plus simple, le plus petit à résoudre directement.

- **Cas Récursif :** C'est la partie de la fonction où elle s'appelle elle-même avec un sous-ensemble du problème original ou un pas de plus vers le cas de base. Le cas récursif réduit le problème global en instances plus petites, se rapprochant progressivement du cas de base jusqu'à ce qu'il soit atteint.


Pour assurer qu'une fonction récursive se termine, il est crucial que chaque appel récursif progresse vers l'atteinte du cas de base.

Qu’est-ce qu’un nombre factoriel ?

Un nombre factoriel, noté avec un point d’exclamation (!), représente le produit de tous les nombres entiers positifs jusqu’à ce nombre. Le factoriel d’un nombre n, noté $n!$, est donc défini comme :

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

Qu'est-ce qu'un nombre factoriel ?

Un nombre factoriel, noté avec un point d'exclamation (!), représente le produit de tous les nombres entiers positifs jusqu'à ce nombre. Le factoriel d'un nombre n, noté $n!$, est donc défini comme :

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

Exemple

Le factoriel de 5 est calculé comme suit :

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Qu'est-ce qu'un nombre factoriel ?

Un nombre factoriel, noté avec un point d'exclamation (!), représente le produit de tous les nombres entiers positifs jusqu'à ce nombre. Le factoriel d'un nombre n, noté $n!$, est donc défini comme :

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

Exemple

Le factoriel de 5 est calculé comme suit :

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Cas spéciaux

- Le factoriel de 0 (0!) est défini comme étant égal à 1.
- Les factoriels ne sont définis que pour les nombres entiers non négatifs.

Calcul du Factoriel : Approche Itérative

Définition du Factoriel

Le factoriel d'un nombre entier non négatif n, noté $n!$, est le produit de tous les entiers positifs jusqu'à n. Mathématiquement, cela est exprimé comme :

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

? Créer une fonction `factorial_iterative(n)` qui calcule le factoriel d'un nombre entier `n`.

Calcul du Factoriel : Approche Itérative

Définition du Factoriel

Le factoriel d'un nombre entier non négatif n, noté $n!$, est le produit de tous les entiers positifs jusqu'à n. Mathématiquement, cela est exprimé comme :

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

? Créer une fonction `factorial_iterative(n)` qui calcule le factoriel d'un nombre entier `n`.

```
def factorial_iterative(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

Calcul du Factoriel avec la Récursivité

Définition du Factoriel

Le factoriel d'un nombre entier non négatif n, noté $n!$, est le produit de tous les entiers positifs jusqu'à n. Mathématiquement, cela est exprimé comme :

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

Cas de Base

Pour $n = 0$, le factoriel est défini comme $0! = 1$.

Cas Récursif

Pour $n > 0$, le factoriel peut être calculé comme $n! = n \times (n-1)!$, où $(n-1)!$ est le factoriel de $n-1$, calculé de manière récursive.

Calcul du Factoriel avec la Récursivité

? Créer une fonction `factorial_recursive(n)` qui calcule le factoriel d'un nombre entier `n`.

Calcul du Factoriel avec la Récursivité

? Créer une fonction `factorial_recursive(n)` qui calcule le factoriel d'un nombre entier `n`.

```
def factorial(n):
    # Cas de base : si n est 1, retourner 1
    if n == 1:
        return 1
    else:
        # Cas r cursif : n! = n * (n-1)!
        return n * factorial(n - 1)
```

Calcul de la Puissance d'un Entier : Approche Iterative

Définition de la Puissance d'un Nombre

La puissance d'un nombre x élevé à n, notée x^n, est le produit de x multiplié par lui-même n fois. Pour $n = 0$, $x^0 = 1$ (si $x \neq 0$).

? Ecrire une fonction récursive `power(x,n)` qui prend un nombre x et un exposant n et renvoie le résultat de x^n.

Calcul de la Puissance d'un Entier : Approche Récursive

Définition de la Puissance d'un Nombre

La puissance d'un nombre x élevé à n, notée x^n, est le produit de x multiplié par lui-même n fois. Pour $n = 0$, $x^0 = 1$ (si $x \neq 0$).

? Ecrire une fonction récursive `power(x,n)` qui prend un nombre x et un exposant n et renvoie le résultat de x^n.

Calcul de la Puissance d'un Entier : Approche Récursive

Définition de la Puissance d'un Nombre

La puissance d'un nombre x élevé à n, notée x^n, est le produit de x multiplié par lui-même n fois. Pour $n = 0$, $x^0 = 1$ (si $x \neq 0$).

? Ecrire une fonction récursive `power(x,n)` qui prend un nombre x et un exposant n et renvoie le résultat de x^n.

Cas de Base

Pour $n = 0$, $x^0 = 1$. C'est le cas de base qui arrête la récursivité.

Calcul de la Puissance d'un Entier : Approche Récursive

Définition de la Puissance d'un Nombre

La puissance d'un nombre x élevé à n, notée x^n, est le produit de x multiplié par lui-même n fois. Pour $n = 0$, $x^0 = 1$ (si $x \neq 0$).

? Ecrire une fonction récursive `power(x,n)` qui prend un nombre x et un exposant n et renvoie le résultat de x^n.

Cas de Base

Pour $n = 0$, $x^0 = 1$. C'est le cas de base qui arrête la récursivité.

Cas Récursif

Pour $n > 0$, la puissance peut être calculée comme $x^n = x \times x^{n-1}$, où x^{n-1} est calculé de manière récursive.

Calcul de la Puissance d'un Entier : Approche Récursive

```
def power(x, n):
    if n == 0:
        return 1
    else:
        return x * power(x, n - 1)
```

Calcul de la Puissance d'un Entier : Approche Récursive

```
def power(x, n):
    if n == 0:
        return 1
    else:
        return x * power(x, n - 1)
```

Optimisation

Pour réduire le nombre d'appels récursifs, on peut utiliser la méthode de "diviser pour régner" en traitant séparément les cas où n est pair ou impair.

Optimisation du Calcul Récursif de la Puissance

Méthode de "Diviser pour Régner"

Cette méthode réduit le nombre d'appels récursifs en traitant différemment les cas où l'exposant est pair ou impair. Elle permet de calculer x^n en utilisant moins d'opérations que l'approche linéaire.

Cas Pair et Impair

- Si n est pair, alors $x^n = (x^{n/2})^2$.
- Si n est impair, alors $x^n = x \times (x^{(n-1)/2})^2$.

Cela permet de réduire de moitié le nombre d'opérations nécessaires à chaque étape récursive.

Optimisation du Calcul Récursif de la Puissance

```
def power_optimized(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return power_optimized(x, n // 2) ** 2
    else:
        return x * power_optimized(x, (n - 1) // 2) ** 2
```

Optimisation du Calcul Récursif de la Puissance

```
def power_optimized(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return power_optimized(x, n // 2) ** 2
    else:
        return x * power_optimized(x, (n - 1) // 2) ** 2
```

Avantages

Cette optimisation réduit considérablement le temps d'exécution, surtout pour les grands nombres, en diminuant le nombre total d'appels récursifs.

Qu’est-ce qu’un Palindrome ?

Definition

Un palindrome est un mot, une phrase, un nombre ou toute autre séquence de caractères qui se lit de la même manière de l’avant vers l’arrière et de l’arrière vers l’avant, en ignorant les espaces, la ponctuation et les majuscules.

Example

Des exemples de palindromes incluent "radar", "kayak", et "Madam, in Eden, I’m Adam".

La Fonction isPalindrome

Signature de la Fonction

La fonction ‘isPalindrome‘ accepte une chaîne de caractères et retourne vrai (‘true‘) si la chaîne se lit de la même manière dans les deux sens.

Approche Récurisive

Pour vérifier si une chaîne est un palindrome, nous comparons le premier et le dernier caractère, puis appliquons la même logique récursivement sur la sous-chaîne restante, en excluant les deux caractères.

? Créer une fonction récursive `isPalindrome(s)`, avec paramétre qui prend en paramètre une chaîne de caractères `s`.

Implémentation de isPalindrome

```
def isPalindrome_1(s):
    # base case
    if (len(s) < 2):
        return True
    # recursive case
    elif (s[0] != s[len(s) - 1]):
        return False
    new_string = s[1,len(s)-2]
    return isPalindrome_1(new_string)
 }
}
```

Implémentation de isPalindrome

```
def isPalindrome_2(s):
    # Base case:
    # empty string or single character string
    if len(s) <= 1:
        return True
    s = s.lower()
    # Check if the first and last characters are the same
    # and recurse on the substring excluding these characters
    return s[0] == s[-1] and isPalindrome_2(s[1:-1])
```

Explication

Cette fonction élimine les caractères non alphanumériques et les différences de casse avant de vérifier la propriété de palindrome, ce qui la rend robuste pour des chaînes complexes.

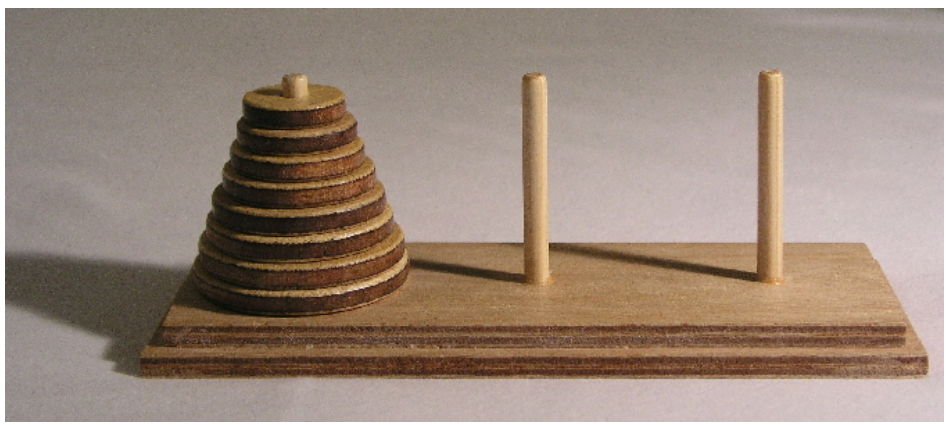
Les trois principes de la recirculation

1. Votre code doit comporter un cas pour toutes les entrées valides.
2. Vous devez avoir un cas de base qui ne fait aucun appels récursifs.
3. Lorsque vous faites un appel récursif, il doit s'agir d'un appel à une instance plus simple.

Les Tours de Hanoï : Introduction au Problème

Un puzzle mathématique où l'objectif est de déplacer une pile de disques de différentes tailles d'une tige à une autre, en suivant trois règles simples :

1. Un seul disque peut être déplacé à la fois.
2. Chaque déplacement consiste à prendre le disque supérieur d'une des piles et à le placer sur le dessus d'une autre pile.
3. Aucun disque ne peut être placé sur un disque plus petit.



Les Tours de Hanoï

- Nous devons trouver un cas très simple que nous pouvons résoudre directement pour que la récursion fonctionne.

Les Tours de Hanoï

- Nous devons trouver un cas très simple que nous pouvons résoudre directement pour que la récursion fonctionne.
- Si la tour est de taille 1, nous pouvons simplement déplacer ce disque unique de la source à la destination.

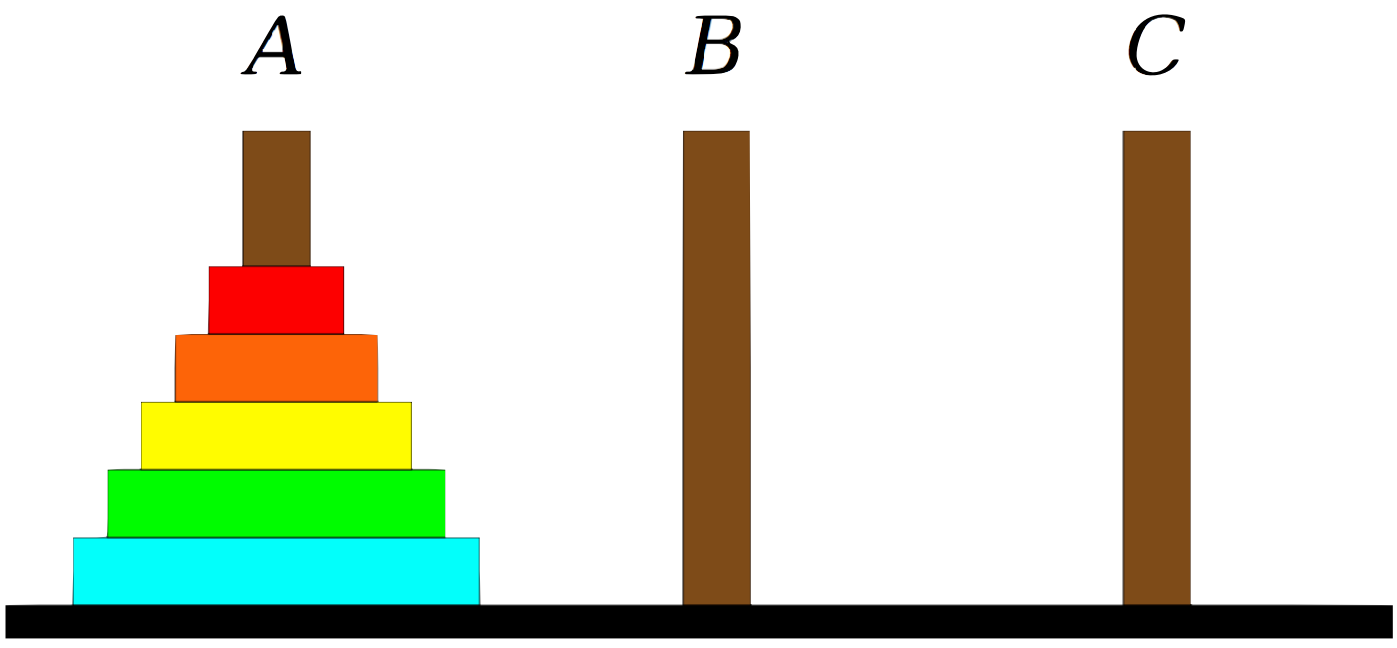
Les Tours de Hanoï

- Nous devons trouver un cas très simple que nous pouvons résoudre directement pour que la récursion fonctionne.
- Si la tour est de taille 1, nous pouvons simplement déplacer ce disque unique de la source à la destination.
- Si la tour a plus d'une taille, nous devons utiliser l'auxiliaire.

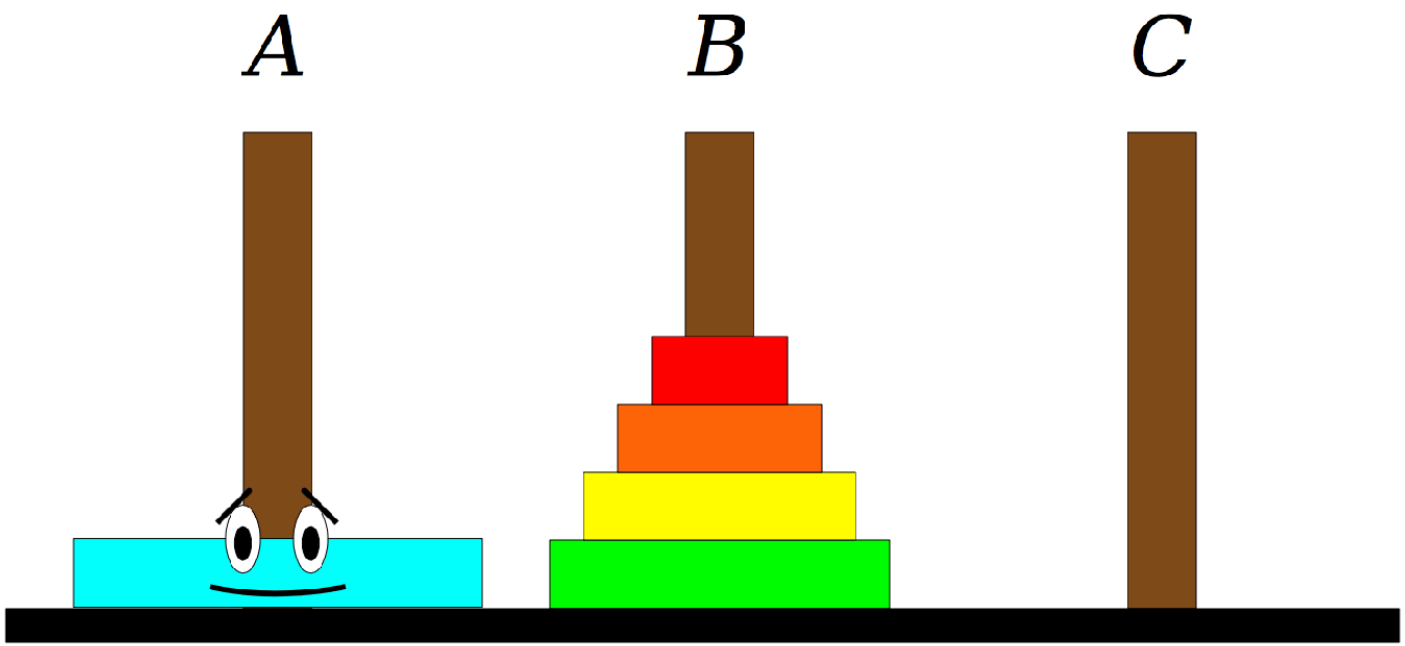
Référence du PDF Inclus

Source : web.stanford.edu,
Récupéré de : `https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1178/lectures/7-IntroToRecursion/7-IntroToRecursion.pdf`

Back to Towers of Hanoi

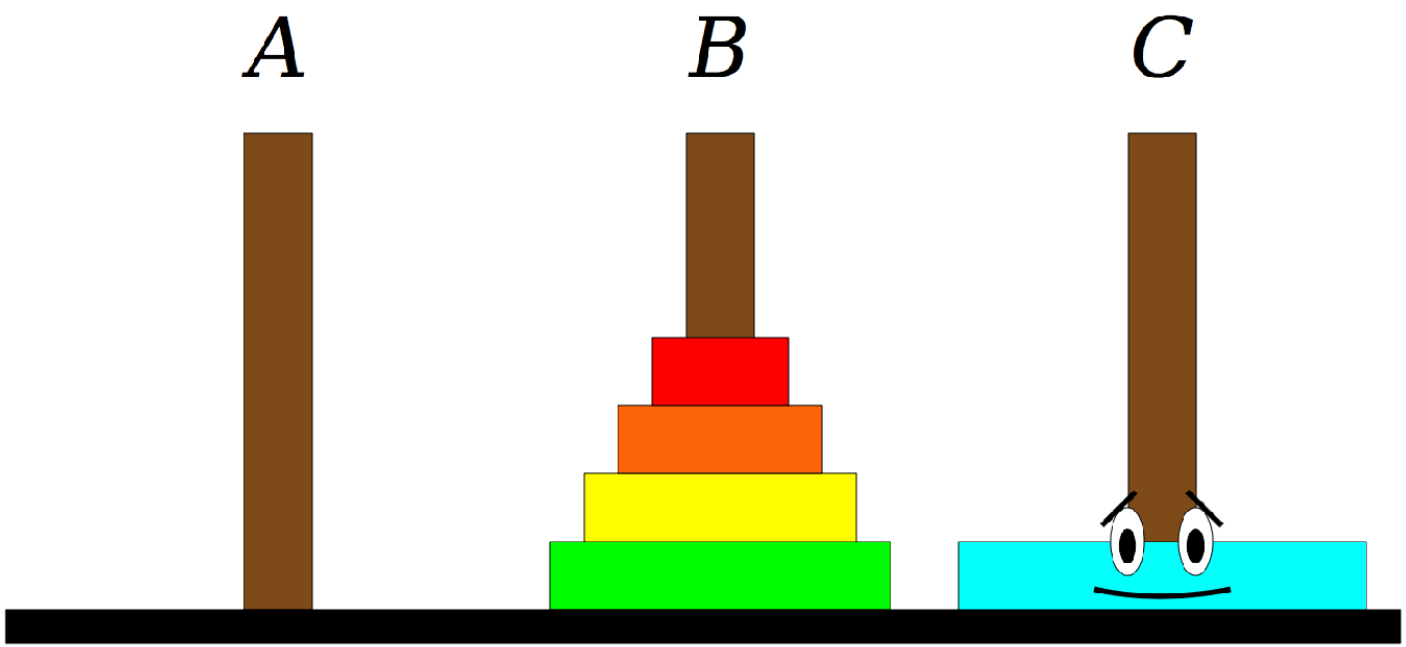


Back to Towers of Hanoi



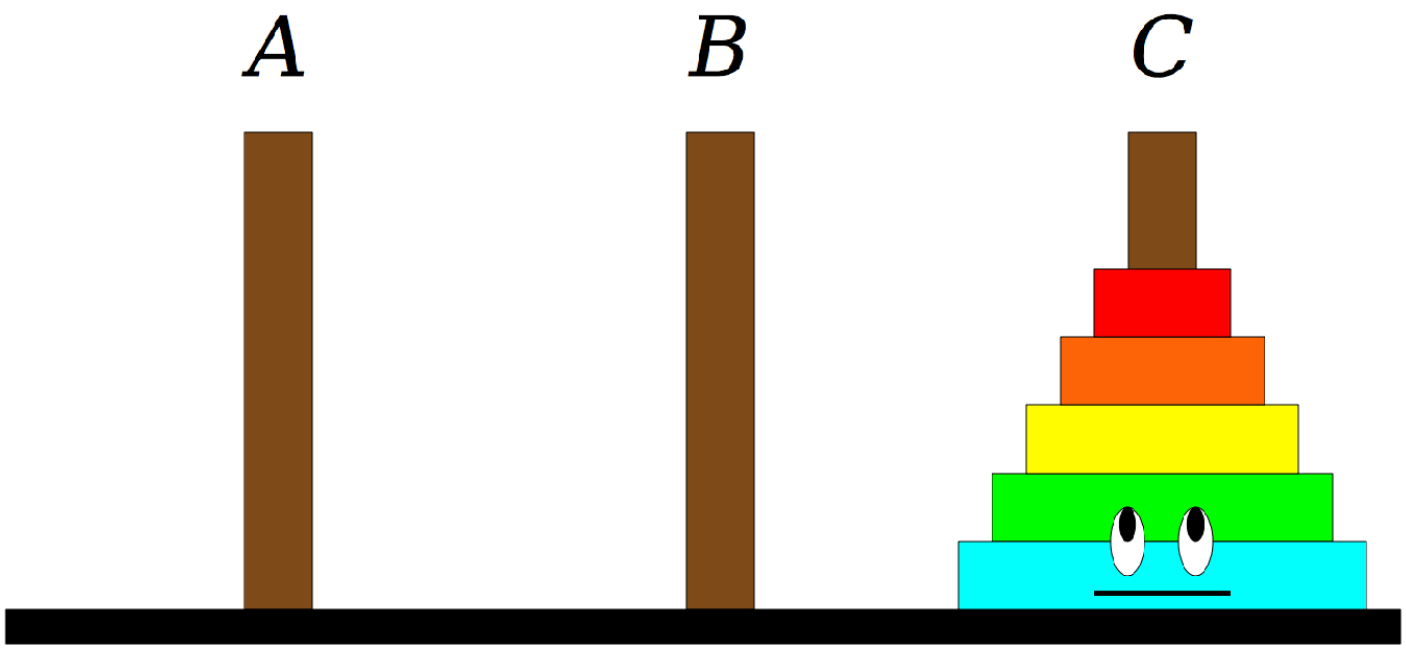
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.

Back to Towers of Hanoi



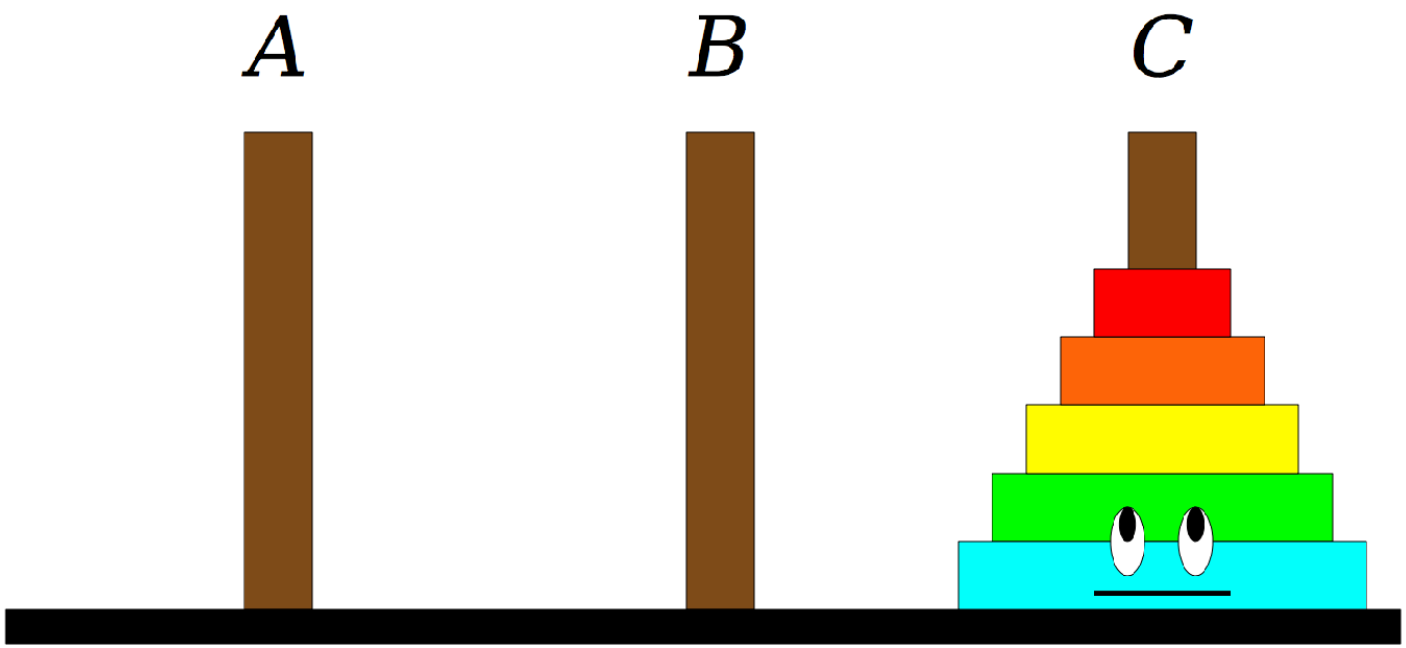
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.
Step Two: Move the blue disk from Spindle *A* to Spindle *C*.

Back to Towers of Hanoi



Step One: Move the four smaller disks from Spindle A to Spindle B.
Step Two: Move the blue disk from Spindle A to Spindle C.
Step Three: Move the four smaller disks from Spindle B to Spindle C.

Back to Towers of Hanoi



Repeat these steps at each stage!

Step One: Move the four smaller disks from Spindle A to Spindle B.
Step Two: Move the blue disk from Spindle A to Spindle C.
Step Three: Move the four smaller disks from Spindle B to Spindle C.

Solution Récurisve

Approche Récurisve

La solution au problème des Tours de Hanoï peut être abordée récursivement, en décomposant le problème en sous-problèmes plus petits.

Idée Clé

Pour déplacer n disques de la tige A à la tige C en utilisant la tige B comme auxiliaire :

1. Déplacez $n - 1$ disques de A vers B, en utilisant C comme auxiliaire.
2. Déplacez le disque restant de A vers C.
3. Déplacez les $n - 1$ disques de B vers C, en utilisant A comme auxiliaire.

Fonction Récurisve pour les Tours de Hanoï

```
def ToH(n, A, B, C):
    if n == 1:
        print("Disk␣1␣from", A, "to", B)
        return
    ToH(n - 1, A, C, B)
    print("Disk", n, "from", A, "to", B)
    ToH(n - 1, C, B, A)
```

ToH(3, "A", "B", "C")