

# TP - Récursivité - Les tours de Hanoï

Nom :

---

## Introduction

---

L'un des problèmes qui se résout particulièrement simplement avec un algorithme récursif est celui des tours de Hanoï inventé par le mathématicien français Édouard Lucas (1842-1891).

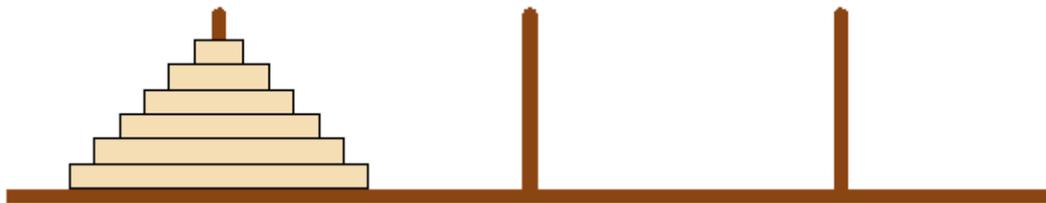
Pour en savoir plus (wikipedia)



Ce jeu mathématique est constitué de trois tiges sur lesquelles sont enfilés  $n$  disques de diamètres différents.

Au début du jeu, ces disques sont tous positionnés sur la première tige (du plus grand au plus petit) et l'objectif est de déplacer tous ces disques sur la troisième tige, en respectant les règles suivantes :

- un seul disque peut être déplacé à la fois ;
- on ne peut jamais poser un disque sur un disque de diamètre inférieur.



### Commençons par une situation simple : 2 disques

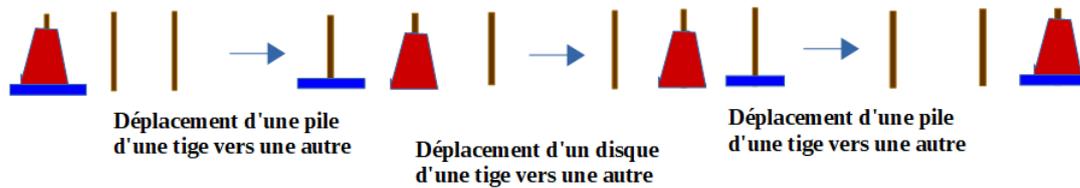


Imaginons qu'à la place du disque rouge nous ayons une pile de  $n - 1$  disques. Notre problème revient à :

- Déplacer cette pile sur la tige du milieu
- Déplacer le disque bleu sur la 3ème tige
- Déplacer la pile sur la 3ème tige



En y regardant de plus près, le déplacement d'une pile de disques d'une tige vers une autre se décompose en 3 phases dont deux sont exactement le problème de départ mais pour une pile diminuée d'un disque.



On peut aborder le problème de la façon suivante : pour envoyer une tour de taille  $n$  de la place de gauche à la place de droite :

- on envoie les  $n - 1$  premiers disques sur la place du milieu, ce qui revient à résoudre le jeu pour  $n - 1$  disques avec le départ à gauche et l'arrivée au milieu ;
- on envoie le disque  $n$  à droite ;
- on envoie les  $n - 1$  premiers disques sur la place de droite, ce qui revient à résoudre le jeu pour  $n - 1$  disques avec le départ au milieu et l'arrivée à droite.

Ainsi la résolution pour  $n$  disques fait appel à la résolution pour  $n - 1$  disques, deux fois. Construire un algorithme récursif est alors très simple, mais il nous manque notre condition d'arrêt : résoudre le problème à 0 disque consiste à ne rien faire.

On peut donc proposer l'algorithme suivant :

**Données :**  $n$ : entier indiquant le nombre de disques à déplacer. dep, inter, arr : chaînes de caractères pour indiquer les tiges de départ, intermédiaire et d'arrivée

fonction Hanoi( $n$ ,dep,inter,arr): **Si**  $n = 0$  **alors**  
 | renvoyer Rien

**Sinon**

| Hanoi( $n - 1$ ,dep,arr,inter)

| afficher "déplacement d'un disque de" dep "vers" arr

| Hanoi( $n - 1$ ,inter,dep,arr)

### À FAIRE 1:

Implémenter cette fonction en Python.

Pour l'affichage des déplacements, on pourra utiliser l'instruction suivante:

```
print("Déplacer le disque {} de la tige {} vers la tige {}".format(n, dep, arr))
```

## À FAIRE 2:

Mesurer le temps d'exécution de cette fonction pour  $n$  valant 2,3,4,5,6,7,8,9,10,11,12,13,14,15

*Remarque: pour  $n$  disques il faut faire  $2^n - 1$  déplacements, donc pour de grande valeurs de  $n$ , il est très probable que le programme soit très lent et "plante" au vue des limitations de la pile d'exécution.*

### Un plus...

Pour visualiser les déplacements : compléter le code suivant:

```
def affichage(tours) :
    d = len(tours['départ'])
    i = len(tours['intermédiaire'])
    a = len(tours['arrivée'])
    hauteur = max(d, i, a)

    for h in range(hauteur, 0, -1) :
        if d >= h : print( tours['départ'][h-1], end=' ')
        else : print(' ', end=' ')
        if i >= h : print( tours['intermédiaire'][h-1], end=' ')
        else : print(' ', end=' ')
        if a >= h : print( tours['arrivée'][h-1])
        else : print(' ')
    print('\u2AEE \u2AEE \u2AEE')
    print('-----')

def Hanoi(n,dep='départ',inter='intermédiaire',arr='arrivée'):
    """ spécification à faire """
    if n==0:
        # à compléter
    else:
        # à compléter
        # à compléter
        anneau_deplace = tours[dep].pop()
        tours[arr].append(anneau_deplace)
        affichage(tours)
        # à compléter

n=4
tours = dict()
tours['départ'] = [ j for j in range(n,0,-1) ]
tours['intermédiaire'] = []
tours['arrivée'] = []
Hanoi(n)
```