

Récurtivité

Récurtivité

Fonction réursive = Fonction qui s'appelle elle même

```
def f(x):  
    print(x)  
    if x<=0:  
        return 0  
    else:  
        return 1+f(x-1)  
  
print(f(5))
```

$$\begin{aligned} f(5) &= 1 + \underbrace{f(4)} \\ &= 1 + \underbrace{1 + f(3)} \\ &= 1 + \underbrace{1 + f(2)} \\ &= 1 + \underbrace{1 + f(1)} \\ &= 1 + \underbrace{1 + f(0)} \\ &= 1 + 0 \end{aligned}$$

Il faut une condition d'arrêt, sinon recursion infinie

```
def g(x):  
    return g(x-1)
```

RuntimeError: maximum recursion depth exceeded

Factoriel

En itératif $n! = n (n-1) (n-2) \dots 1$

```
def factoriel(n):  
    valeur=1  
    for x in range(1,n+1):  
        valeur=valeur * x  
    return valeur  
  
print(factoriel(5))
```

Factoriel

En récursif $n! = n (n-1) (n-2) \dots 1$
 $= n (n-1)!$

\Rightarrow `factoriel(n) = n x factoriel(n-1)`

```
def factoriel(n):  
    if n<=1:  
        return 1  
    else:  
        return n*factoriel(n-1)
```

Combinaisons

$$C_p^n = n! / ((n-p)! p!)$$

↑ problèmes, chiffres très grands

$$C_p^n = C_p^{n-1} + C_{p-1}^{n-1}$$

$$C_0^n = 1$$

$$C_p^n = 1$$

```
def combi(n,p):  
    if n==p or p==0:  
        return 1  
    else:  
        return combi(n-1,p)+combi(n-1,p-1)
```

↑ ↑
Récursion double
recalculé intégralement
combi(n-1,p-1) pour combi(n-1,p)

Combinaisons

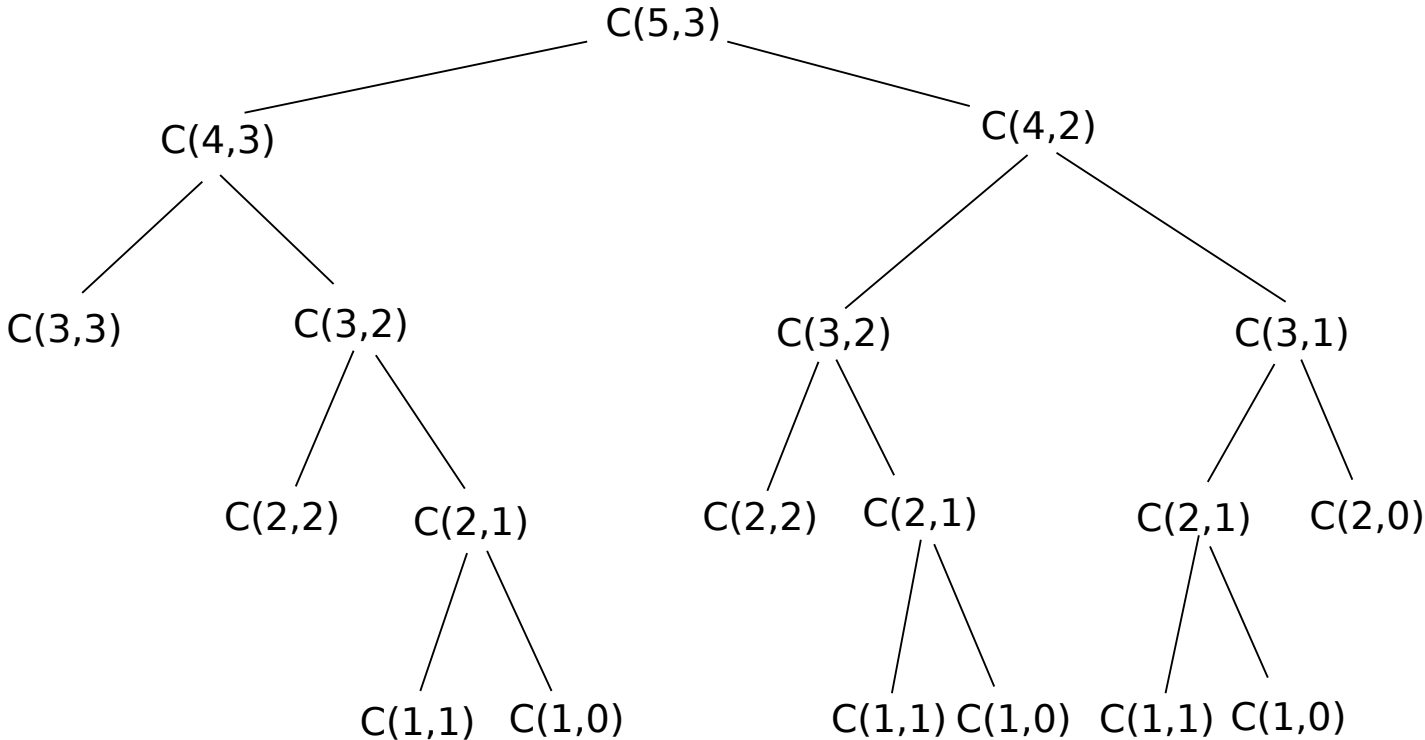
```
def combi(n,p):  
    if n==p or p==0:  
        return 1  
    else:  
        return combi(n-1,p)+combi(n-1,p-1)
```

Pour chaque appel à $\text{combi}(n,x)$

=> 2 appels à $\text{combi}(n-1,x)$ + On itère n fois

=> Complexité en $O(2^n)$

Combinaisons



$C(25,13)$ demande 10 millions d'appels de fonctions

Combinaisons

On aurait pu remarquer:

$$C_p^n = (n/p) C_{p-1}^{n-1}$$

```
def combi2(n,p):  
    if n==p or p==0:  
        return 1  
    else:  
        return n/p*combi2(n-1,p-1)
```

division
(erreur d'arrondis)

1 seule récursion

Complexité en $O(n)$

Combinaisons

Coder les deux fonctions:

Comparer les temps de calculs pour n variant

```
def combi2(n,p):  
    if n==p or p==0:  
        return 1  
    else:  
        return n/p*combi2(n-1,p-1)
```

```
def combi(n,p):  
    if n==p or p==0:  
        return 1  
    else:  
        return combi(n-1,p)+combi(n-1,p-1)
```

```
import time  
  
start=time.time()  
[...]  
end=time.time()  
  
print(end-start)
```

Recursion

Avantages

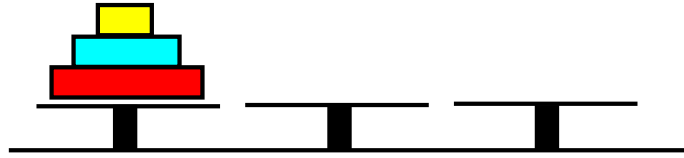
- Expression concise de certains problèmes

Inconvénients

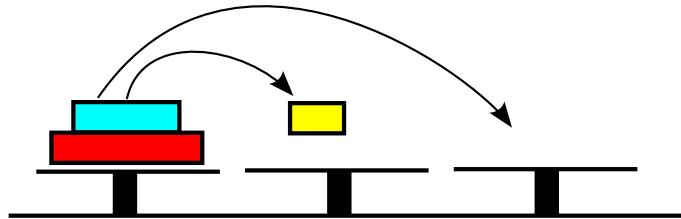
- Terminaison ?
- Complexité croit rapidement
- Utilisation de la pile

En développement:
A complexité égale, préférer une version itérative

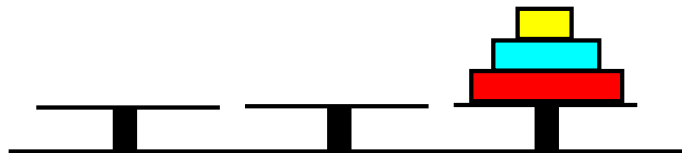
Tour de Hanoi



But:



...



Contraintes:


1 seul déplacement à la fois
(le jeton en haut de la pile)

le jeton doit se trouver sur un
jeton plus grand (ou sur le plateau)

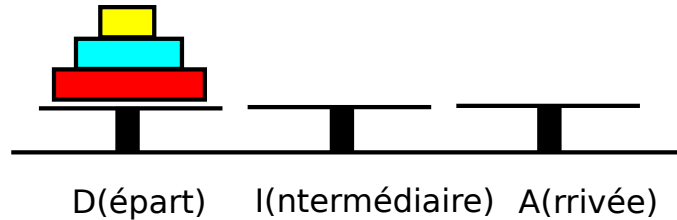
Tour de Hanoi

Modèle

1 

2 

3 



Algorithme:

```
Hanoi (n, D, A, I) :  
  Si n>0:  
    Hanoi (n-1, D, I, A)  
    D placer jeton de D vers A  
    Hanoi (n-1, I, A, D)
```

Implémenter un déplacement pour une tour de Hanoi de taille n

Combien de déplacements pour n=20 ?