ISM MPSI/PCSI, Informatique
Année 2023/2024

Corrigé du TP Informatique 24

Exercice 1

1. On saisit :

```
def triangle1(n):
    if n==1:
        return "*"
    else:
        return triangle1(n-1)+"\n"+"*"*n
```

2. On saisit :

```
def triangle2(n):
    if n==1:
        return "*"
    else:
        return "*"*n+"\n"+triangle2(n-1)
```

3. On saisit :

```
def triangle3(n):
    if n==1:
        return "*"
    elif n==2:
        return "**"
    else:
        return triangle3(n-2)+"\n"+"*"*n
```

Exercice 2

On saisit :

```
def parties(ens):
    """parties(ens:list)->list
    Renvoie l'ensemble des sous-listes de ens"""
    if ens==[]:
        return [[]]
    else:
        p_aux=parties(ens[1:])
        return [[ens[0]]+p for in p_aux]+p_aux
```

Exercice 3

On saisit :

```
def deepcopy(L):
    if type(L)==list:
        return [deepcopy(x) for x in L]
    elif type(L)==tuple:
        return tuple(deepcopy(x) for x in L)
    else:
        return L
```

Exercice 4

L'idée récursive consiste à chercher les anagrammes du mot privé de sa première lettre puis d'insérer celle-ci à toutes les positions possibles sur les anagrammes fournis par la récursion. On saisit :

```
def anagramme(mot):
    """anagramme(mot:str)->list
    Renvoie la liste des anagrammes de mot"""
    n=len(mot)
    if n==0:
        return [mot]
    else:
        res=[]
        liste=anagramme(mot[1:])
        for m in liste:
            for k in range(n):
                res.append(m[:k]+mot[0]+m[k:])
        return res
```

Exercice 5

2. On saisit :

```
def hanoi(n,dep,arr,etape):
    if n==1:
        return dep+"-->"+arr+"\n"
    else:
        return hanoi(n-1,dep,etape,arr)+dep+"-->"+
               arr+"\n"+hanoi(n-1,etape,arr,dep)
```

Pour n entier non nul, on note a_n le nombre de déplacements effectués pour résoudre le problème à n disques. Pour résoudre le problème au rang n avec $n \geqslant 2$, on effectue a_{n-1} déplacements, puis un déplacement pour le grand disque, puis à nouveau a_{n-1} déplacements d'où la relation

$$\forall n \geqslant 2 \qquad a_n = 2a_{n-1} + 1$$

C'est une suite arithmético-géométrique de premier terme $a_1 = 1$ et on conclut

$$\boxed{\forall n \in \mathbb{N}^* \qquad a_n = 2^n - 1}$$

Exercice 6

1. Notons $[a_1, \dots, a_p]$ les valeurs entières des jetons. On veut calculer

$$\mathrm{S}([a_1, \dots, a_p], n) = \mathrm{Card}\left\{(k_1, \dots, k_p) \in \mathbb{N}^p \mid \sum_{i=1}^{p} a_i k_i = n\right\}$$

On a

$$\begin{aligned}\mathrm{S}([a_1, \dots, a_p], n) &= \mathrm{Card} \bigsqcup_{k_p \in \mathbb{N} \mid a_k k_p \leqslant n} \left\{(k_1, \dots, k_{p-1}) \in \mathbb{N}^{p-1} \mid \sum_{i=1}^{p-1} a_i k_i = n - a_p k_p\right\} \\ &= \sum_{k_p \in \mathbb{N} \mid a_p k_p \leqslant n} \mathrm{S}([a_1, \dots, a_{p-1}], n - a_p k_p)\end{aligned}$$

En particulier, pour $p = 1$, on a $\mathrm{S}([a_1], n) = \mathbb{1}_{a_1 | n}$

Les deux relations précédentes fournissent respectivement le cas de propagation et le cas de base pour une programmation récursive. On saisit donc :

```
def P(V,n):
    """P(V:list,n:int)->int
    Renvoie le nombre de façons de payer n avec le système de jetons V"""
    if len(V)==1:
        return int(n%V[0]==0)
    else:
        s=0
        for k in range(0,n+1,V[-1]):
            s+=P(V[:-1],n-k)
        return s
```

2. Dans les deux cas, on trouve :

```
>>> P([1,2,3,4,5,6],100)
189509
>>> P([6,5,4,3,2,1],100)
189509
```

Mais le deuxième appel est beaucoup plus lent. Dans la boucle `for` de la fonction `P(V,n)`, plus l'argument `V[-1]` est grand, plus le nombre de passages est petit. Ainsi, en ordonnant préalablement `V`, les boucles successives sont minimales ce qui améliore l'efficacité de la fonction. On saisit :

```
def P2(V,n):
    return P(sorted(V),n)
```