Ch 12. Fonctions Récursives.

1 Définitions

Définition :

Une fonction récursive est une fonction qui s'appelle elle-même.
On appelle récursion l'appel de la fonction à elle-même.

La programmation récursive est un paradigme de programmation au même titre que la programmation itérative. Un programme écrit de manière récursive peut être traduit de manière itérative, même si dans certain cas, cela peut s'avérer délicat.

Méthode :

- Une fonction récursive doit posséder une condition d'arrêt (ou cas de base).
- Une fonction récursive doit s'appeler elle-même (récursion).
- L'argument de l'étape de récursion doit évoluer de manière à se ramener à la condition d'arrêt.

2 Suites définies par récurrence

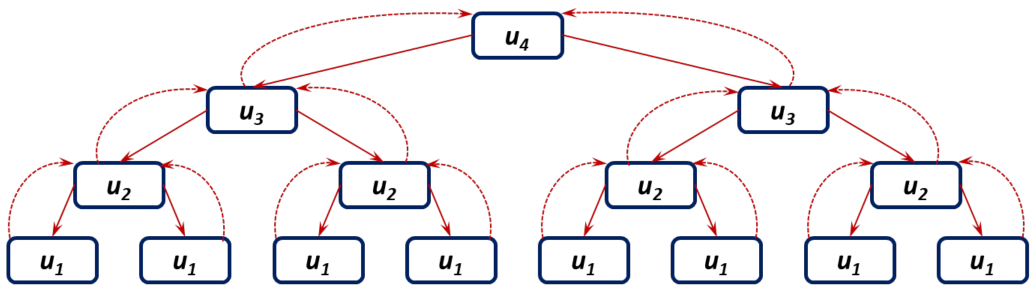
Les suite définies par récurrence pour les quelles $u_n = f(u_{n-1}, u_{n-2}, ...)$ sont des cas d'application directs des fonctions récursives.

Par exemple, soit la suite u_n définie par récurrence pour tout $n \in \mathbb{N}^*$ par $\begin{cases} u_1 = 1 \\ u_{n+1} = \dfrac{u_n + 6}{u_n + 2} \end{cases}$. Il est possible de calculer le n^{e}terme par un algorithme itératif ou un algorithme récursif.

```python
def un_it (n : int) -> float :
    if n == 1 :
        return 1
    else :
        u = 1
        for i in range(2,n+1):
            u = (u+6)/(u+2)
        return u
```

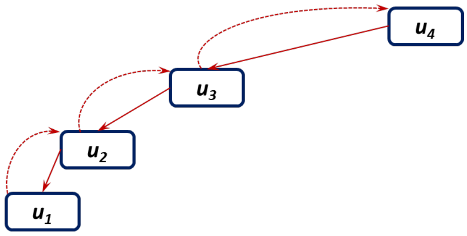
```python
def un_rec (n : int) -> float :
    if n == 1 :
        return 1
    else :
        return (un_rec(n-1)+6)/(un_rec(n-1)+2)
```

La figure suivante montre que dans le cas de l'algorithme récursif proposé plusieurs termes sont calculés à plusieurs reprises ce qui constitue une perte de temps et d'espace mémoire.



Une autre formulation de l'algorithme récursif permet très simplement de diminuer le nombre de termes calculés.

```python
def un_rec (n : int) -> float :
    if n == 1 :
        return 1
    else :
        v = un_rec(n-1)
        return (v+6)/(v+2)
```



3 Algorithmes dichotomiques – Divisier pour régner

Les algorithmes dichotomiques se prètent aussi à des formulations récursives. Prenons comme exemple la recherche d'un élément dans une liste triée. L'algorithme de gauche propose une version itérative. L'algorithme de droite une version récursive.

```python
def appartient_dicho(e : int , t : list) -> bool:
    """Renvoie un booléen indiquant si e est
    dans t. Préconditions : t est un tableau
    de nombres trié par ordre croissant e est
    un nombre"""
    # Limite gauche de la tranche où l'on
        recherche e
    g = 0
    # Limite droite de la tranche où l'on
        recherche e
    d = len(t)-1
    # La tranche où l'on cherche e n'est pas vide
    while g <= d:
        # Milieu de la tranche où l'on recherche e
        m = (g+d)//2
        pivot = t[m]
        if e == pivot: # On a trouvé e
            return True
        elif e < pivot:
            # On recherche e dans la partie gauche
                de la tranche
            d = m-1
        else:
            # On recherche e dans la partie droite
                de la tranche
            g = m+1
    return False
```

```python
def appartient_dicho_rec(e : int , t : list) ->
    bool:
    """Renvoie un booléen indiquant si e est dans
        t. Préconditions : t est un tableau de
        nombres trié par ordre croissant e est un
        nombre"""
    # Limite gauche de la tranche où l'on
        recherche e
    g = 0
    # Limite droite de la tranche où l'on
        recherche e
    d = len(t)-1
    # La tranche où l'on cherche e n'est pas vide
    while g <= d:
        # Milieu de la tranche où l'on recherche e
        m = (g+d)//2
        pivot = t[m]
        if e == pivot: # On a trouvé e
            return True
        elif e < pivot:
            # On recherche e dans la partie gauche
                de la tranche
            d = m-1
            appartient_dicho_rec(e,t[g:d])
        else :
            # On recherche e dans la partie droite
                de la tranche
            g = m+1
            appartient_dicho_rec(e,t[g:d])
    return False
```

4 Tracer de figures définies par récursivite

Un grand nombre de figures peuvent être tracés en utilisant des algorithmes récursifs (flocon de Koch, courbe de Peano, courbe du dragon *etc.*).

```python
import matplotlib.pyplot as plt
import numpy as np
def cercle(x,y,r):
    theta = np.linspace(0, 2*np.pi, 100) #des points régulièrement espacés dans l'intervalle [0,2pi]
    X = [x+r*np.cos(t) for t in theta] #abscisses de points du cercle C((x,y),r)
    Y = [y+r*np.sin(t) for t in theta] #ordonnées de points du cercle C((x,y),r)
```

```
    plt.plot(X,Y,"b") #tracé sans affichage
```

```
def bubble(n, x, y, r, d):
    cercle(x, y, r)
    if n > 1:
        if d != 's':
            bubble(n-1,x,y+3*r/2,r/2,"n")
        if d != 'w':
            bubble(n-1,x+3*r/2,y,r/2,"e")
        if d != 'n':
            bubble(n-1,x,y-3*r/2,r/2,"s")
        if d != 'e':
            bubble(n-1,x-3*r/2,y,r/2,"w")
bubble(4,0,0,8,"")
plt.axis("equal")
plt.show()
```