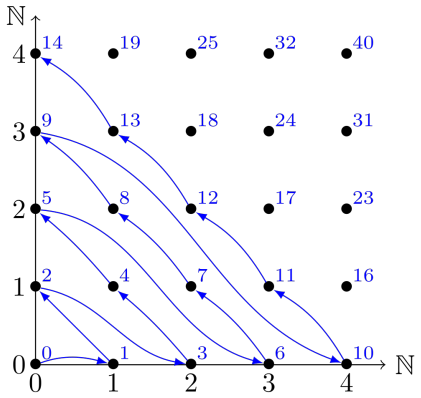
TP 6 : Corrigé

Exercice 1. *Itératif et récursif*. Version itérative et récursive. Pour l'écrire en récursif, on utilise le fait que $u_n = 3u_{n-1} - 2(n-1) + 4$ pour $n \geq 1$.

```
def terme(a,n):
    u=a
    for i in range(n):
        u=3*u-2*i+4
    return u
```

```
def terme2(a,n):
    if n==0:
        return a
    else:
        return 3*terme(a,n-1)-2*(n-1)+4
```

Exercice 2. *Fonction de Cantor*.




```
def enum(x,y):
    if x==0 and y==0:
        return 0
    elif y==0:
        return 1+enum(0,x-1)
    else:
        return 1+enum(x+1,y-1)

def couple(n):
    if n==0:
        return 0,0
    else:
        a,b=couple(n-1)
        if a==0:
            return b+1,0
        else:
            return a-1,b+1
```

Exercice 3. *Toutes les listes binaires*.

```
def listes_bin(n):
    if n==0:
        return [[]] # Cas de base : liste contenant uniquement la liste vide.
    else:
        X=listes_bin(n-1)
        T=[]
        for x in X:
            T.append(x+[0])
            T.append(x+[1])
        return T
```

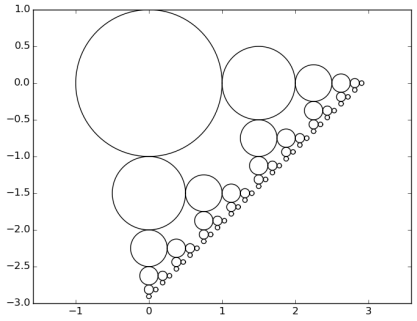
Exercice 4. *Tri fusion*.

```
def fusion(L1, L2):
    i1, i2 = 0, 0
    n1, n2 = len(L1), len(L2)
    L = []
    for i in range(n1+n2):
        if i2==n2 or (i1<n1 and L1[i1]<=L2[i2]):
            L.append(L1[i1])
            i1+=1
        else:
            L.append(L2[i2])
            i2+=1
    return L

def tri_fusion(L):
    if len(L)<=1:
        return L[:]
    else:
        m=len(L)//2
        return fusion(tri_fusion(L[:m]), tri_fusion(L[m:]))
```

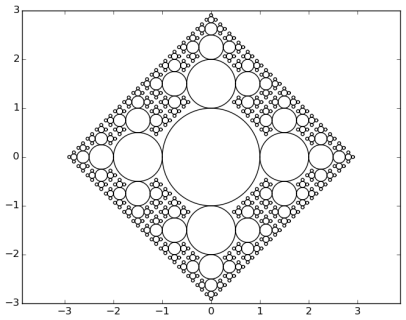
Exercice 5. *Des fractales*.

1. Pour `bulles1`, si $n > 0$ on fait un appel récursif vers la gauche, et un appel récursif vers la droite. Pour qu'un cercle de rayon r soit tangent à un cercle de rayon $r/2$, il faut que le centre du petit cercle soit situé à distance $\frac{3}{2}r$ du cercle initial. D'où le code.



```
def bulles1(n):
    plt.figure()
    plt.axis("equal")
    def aux(x,y,n,r):
        cercle(x,y,r)
        if n>0:
            aux(x+1.5*r,y,n-1,r/2)
            aux(x,y-1.5*r,n-1,r/2)
    aux(0,0,n,1)
    plt.show()
```

2. `bulles2` suit la même idée, sauf qu'à chaque étape on fait trois appels récursifs, dans les directions qui dépendent d'où l'on vient (sauf pour l'appel initial où on part dans les 4 directions) : il ne faut pas faire d'appel récursif dans la direction opposée à celle d'où l'on vient. On intègre cela dans un paramètre supplémentaire qui peut valoir 0, 1, 2 ou 3. Vu comment est rédigé la fonction suivante, pour l'appel initial, toute valeur différente permettra un appel récursif dans les 4 directions.



```
def bulles2(n):
    plt.figure()
    plt.axis("equal")
    def aux(x,y,n,r,d):
        #d=0,1,2,3 si on vient de droite, haut, gauche, bas
        cercle(x,y,r)
        if n==0:
            return #on ne fait rien
        if d!=0:
            #on ne vient pas de la droite : on peut y aller
            aux(x+1.5*r,y,n-1,r/2, 2)
        if d!=2:
            #on ne vient pas de la gauche.
            aux(x-1.5*r,y,n-1,r/2, 0)
        if d!=1:
            #on ne vient pas du haut.
            aux(x,y+1.5*r,n-1,r/2, 3)
        if d!=3:
            #on ne vient pas du bas.
            aux(x,y-1.5*r,n-1,r/2, 1)
    aux(0,0,n,1,"tout sauf 0, 1, 2, 3")
    plt.show()
```

Exercice 6. *Indices de sous-séquences.*

```
def indices_seq(s,t):
    L=[] #liste des indices dans s où apparait la dernière lettre de t
    for i in range(len(s)):
        if s[i]==t[-1]:
            L.append(i)
    if len(t)==1:
        return [[x] for x in L]
    else:
        T=[]
        for x in L:
            Tx = indices_seq(s[:x],t[:-1])
            for u in Tx:
                T.append(u+[x])
        return T
```