Interrogation d'informatique n°10. Corrigé

Exercice A

1) SELECT DISTINCT auteur FROM *Corpus*

2) SELECT auteur , SUM(nombre) AS somme

FROM Corpus JOIN Frequences ON Corpus.id = Frequences.Id

GROUP BY auteur

ORDER BY somme DESC

3) SELECT symbole

FROM Corpus

JOIN Frequences ON Corpus.id = Frequences.Id

JOIN Caracteres ON Frequences.code = Caracteres.code

WHERE auteur = 'Vian'

GROUP BY code

ORDER BY SUM(nombre) DESC

LIMIT 2

Exercice B. Arbres couvrants aléatoires

1)
```
def permut(L) :
    L = L.copy() ; n = len(L)
    for i in range(n) :
        j = numpy.random.randint(0,i+1)
        L[i] , L[j] = L[j] , L[i]
    return L
```

2) $T[2] = 0$ ssi 1 et 2 sont voisins de 0 et qu'il n'existe pas de chemin reliant 1 à 2 sans passer par 0.

3) Version récursive :

```
def parcours_alea(G) :
    n = len(G) ; T = [-1]*n ; T[0] = 0 ; visites = [False]*n
    def traite(s) :
        if not visites[s] :
            visites[s] = True
            for t in permut(G[s]) :
               if T[t] == -1 :
                  T[t] = s ; traite(t)
    traite(0) ; return T
```

Version itérative :

```
def parcours_alea(G) :
```

```
    n = len(G) ; T = [-1]*n ; T[0] = 0 ; visites = [False]*n ; pile = [0]
    while pile :
         s = pile.pop()
         if not visites[s] :
              visites[s] = True
              for t in permut(G[s]) :
                if T[t] == -1 :
                    T[t] = s ; pile.append(t)
    return T
```

4) a)
```
def code(c,N) : (i,j) = c ; return i+j*N
```

b)
```
def decode(k,N) : return (k%N,k//N)
```

5) a)
```
def voisins(c,N) :
        V = [] ; c = (i,j)
        for k in [-1,1] :
             if i+k>=0 and i+k<= N : M.append((i+k,j))
             if j+k>=0 and j+k<= N : M.append((i,j+k))
        return V
```

b)
```
def graphe_lab(N) :
    G = []
    for k in range(N**2) :
         c = decode(k,N) ; V = voisins(c)
         G[k] = [code(d,N) for d  in voisins(c,N)]
    return G
```

6) a)
```
def murs_lab(N) :
    G = graphe_lab(N) ; T = parcours_alea(G)
    # initialisation de D
    D = {}
    for i in range(N) :
         for j in range(N) :
              D[(i,j)] = []
    # construction de D
    for k in range(1,N**2) :
         D[decode(k,N)].append(decode(T[k],N))
         D[decode(T[k],N)].append(decode(k,N))

    return D
```

b)
```
def dessin_lab(N) :
```

```
    D = murs_lab(N)
    # on commence par les murs horizontaux
    # le mur reliant (i,j) à (i+1,j) sépare les cellules (i,j) et (i,j-1)
    for i in range(N) :
        for j in range(N+1) :
            if j == 0 or j == N or not((i,j-1) in D[(i,j)]) :
                plt.plot([i,i+1],[j,j])
    # on finit les murs verticaux en inversant les rôles de i et j
    for j in range(N) :
        for i in range(N+1) :
            if i == 0 or i == N or not((i-1,j) in D[(i,j)]) :
                plt.plot([i,i],[j,j+1])
    # on affiche la fenêtre graphique
    plt.show()
```

7)
```
def centre(D) :
    # initialisation de M et de L
    M = {} ; L = []
    for s in D :
        M[s] = len(D[s])
        if M[s] == 1 :  L.append(s)
    # suppression des extrémités tant que card M >= 3
    while len(M) >= 3 :
        Lnew = []
        for s in L :
            for t in D[s] :
                if (t in M) : M[t] = M[t] - 1
                if M[t] == 1 :  Lnew.append(t)
            del M[t]
        L = Lnew
    return M.keys()
```

Remarque : La complexité est linéaire chaque clé s est supprimée au plus une fois dans M, donc chaque arête $(s \to t)$ du graphe D est parcourue au plus une fois. Autrement dit, la complexité est en $O(n+m)$, où $m = n-1$ est le nombre d'arêtes de D. D'où une complexité en $O(n)$.

4) Dans un arbre, la profondeur d'un sommet est la longueur du chemin reliant la racine à ce sommet (c'est-à-dire le nombre d'arêtes du chelin). En particulier, la profondeur de la racine est 0.

a)
```
def profondeurs(T) :
    n = len(T) ; P = [-1]*n
```

```
    def aux(k) :
        if k == 0 :  P[0] = 0
        if T[k] == -1 :  aux(T[k])
        P[k] = 1 + P[T[k]]
    for k in range(n) :  aux(k)
    return P
```

b)
```
def prof_max(T)(T) :
    n = len(T) ; P = profondeurs(T)
    k = 0
    for j in range(n) :
        if P[j] > P[k] :  k = j
    chemin = [k]
    while k != 0 :
        k = T[k] ; chemin.append(k)
    return chemin
```