Interrogation d'informatique n°7. Corrigé

Exercice A

```
def creer_file_vide() :  return ([],[])

def creer_file_vide(f) :  return (f==[] and g==[])

def enfiler(x,f) :  f[0].append(x)

def defiler(x,f) :
    p,q = f
    if q == [] :
        while p :  q.append(p.pop())
    return q.pop()
```

Exercice B

1) Lors de la copie , on crée de nouvelles cases mémoires, mais elle pointent vers les mêmes valeurs, qui sont ici des listes. Le problème arrive lorsqu'on utilise une modification interne à l'une de ces listes (c'est se qui se produit lorsqu'on fait `dico[0].append(0)`) ; en revanche, il n'y a pas de problème lors de l'affectation `dicoBis[1] = [3]` après laquelle `dicoBis[1]` pointe vers une liste distincte de `dico[1]` : si on fait alors `dicoBis[1].append(0)`, on ne modifie plus la valeur de `dico[1]`.

2) On considère les instructions suivantes :

```
dicoBis = {}
for x in dico :  dicoBis[x] = dico[x].copy()
```

Exercice C

1)
```
def verif(G) :
    c = {}
    for s in G : c[s] = 0
    for s in G :
        for t in G[s] :
            c[s] = c[s] + 1 ; c[t] = c[t] - 1
    for s in G :
        if c[s] != 0 :  return False
    return True
```

2) a) (i) Supposons que s_k n'appartient pas à $\{s_0, ..., s_{k-1}\}$.

Alors s_k admet au moins une arête entrante.

Donc admet au moins une arête sortante, d'où l'existence de s_{k+1}. Donc l'algorithme ne bloque pas.

(ii) termine par le principe des tiroirs, car à chaque étape k augemente et qu'il y a un nombre fini de sommets.

b)

```
def cycle(G,s0) :
    marquage = {} ; k = 0 ; s = s0 ; L = []
    while not(s in marquage) :
        # on utilise un compteur k pour mémoriser la position d'un sommet dans L
        marquage[s] = k ; L.append(s)
        s = G[s].pop() ; k = k+1
    j = marquage[s]
    # il faut remettre dans G les arêtes qui ne sont pas dans le cycle
    for i in range(j) :  G[L[i]].append(L[i+1])
    return [L[j] for j in range(j,k)]
```

3) a) Lors du passage à G à G', les valeurs de $d^+(s)$ et $d^-(s)$ sont diminuées de 1 pour tout sommet $s \in C$, et sont inchangées pour les autres sommets. Donc la propriété $(\mathcal{P})$ est conservée.

b) Par récurrence forte sur le nombre d'arêtes, il résulte de a) que G est réunion de cycles.
Or, un cycle et un circuit peuvent être fusionnés en un seul circuit s'ils ont au moins un sommet commun. On aboutit donc finalement par fusions successives à écrire G comme réunion de circuits disjoints. Comme le graphe est connexe, il y a un et un seul circuit, qui est donc eulérien.

Exercice D

1)
```
def ind_min(L) :
    i = 0 ; n = len(L)
    for j in range(n) :
        if L[j]<L[i] :  i = j
    return i
```

2) Si A, B, C sont respectivement de types $(2, p)$, $(p, 2)$ et $(2, q)$, le produit $(AB)C$ nécessite $(4p^2 + 4q)$ alors que le produit $A(BC)$ nécessite $4pq$.
Comme $q = p^2$ est grand, les ordres de grandeur sont trsè différents (p^2 vs p^3).

3) Pour $1 \leq i \leq j \leq n$, on notre $d(i, j)$ le nombre d'opérations minimum pour effectuer le produit $A_i...A_j$.
On pose en particulier $d(i, i) = 0$. On a pour $i < j$,

$$d(i,j) = \min_{i \leq k \leq j-1} (d(i,k) + d(k+1,j) + p_{i-1}p_kp_j)$$

On considère k de sorte que dernier produit effectué est entre $A_i...A_k$ et $A_{k+1}...A_j$, où $i \leq k \leq j-1$, est

$$(A_i...A_k)(A_{k+1}...A_j)$$

puis on est ramené à optimiser le calcul des produits matriciels $(A_i...A_k)$ et $(A_{k+1}...A_j)$. D'où :

```
def minimum(P) :
    n = len(P)-1 ; dico = {}
    def aux(i,j) :      ### définit la valeur de la clé (i,j) dans dico si elle n'y est pas déjà
        if i == j ; dico[(i,j)] = 0
        elif not (i,j) in dico :
            L = [dico[(i,k)]+dico[(k+1,j)]+P[i-1]*P[k]*P[j] for k in range(i,j)]
            dico[(i,j)] = L[ind_min(L)]
    aux(1,n) ; return dico[(1,n)]
```

4) La complexité est $O(n^3)$ pour le calcul du tableau des $d(i,j)$.

En effet, le calcul de chaque $d(i,j)$ demande $O(n)$ opérations.

5) On attribue ici à `dico[(i,j)]` un couple `(m,s)`, où $m = d(i,j)$ et s une solution optimale associée.

```
def solution(P) :
    n = len(P)-1 ; dico = {}
    def aux(i,j) :
        if i == j ; dico[(i,j)] = (0,[])
        elif not (i,j) in dico :
            L = [dico[(i,k)][0]+dico[(k+1,j)][0]+P[i-1]*P[k]*P[j] for k in range(i,j)]
            m = L[ind_min(L)]
            sol = [1]+dico[(i,k)][1]+dico[(k+1,j)][1]+[-1] ; dico[(i,j)] = (m,sol)
    aux(1,n) ; return dico[(1,n)][1]
```

6) On considère un graphe dont les sommets sont les listes $[i_1, ..., i_p]$, où $0 < i_1 < ... < i_p < n$.

On prend par convention $i_0 = 0$ et $i_{p+1} = n$.

La liste signifie que les $p+1$ produits $(A_1...A_{i_1})$, $(A_{i_1+1}...A_{i_1})$, ... , $(A_{i_p+1}...A_n)$ ont été calculés.

L'état initial est la liste $[1, 2, ..., n-1]$ associés aux matrices initiales A_1, A_2, ... , A_n.

L'état final est la liste vide [ ] associée à la matrice $(A_1...A_n)$ qu'on souhaite calculer.

Chaque liste $[i_1, ..., i_p]$ admet p successeurs qui sont les listes obtenus en supprimant un des p éléments de la liste. Le poids de l'arête $[i_1, ..., i_p] \to [i_1, ..., i_{k-1}, i_{k+1}, ..., i_p]$ est $p[i_{k-1}]p[i_k]p[i_{k+1}]$, car il correspond au nombre d'opérations nécessaires pour effectuer le produit des deux matrices $(A_{i_{k-1}+1}...A_{i_k})$ et $(A_{i_k+1}...A_{i_{k+1}})$.