

Interrogation d'informatique n°4. Corrigé

Exercice A

1) Comme il n'y a pas de boucle de poids négatif, on **peut se restreindre aux chemins réduits** en supprimant les boucles. Les chemins injectifs de x à y sont en nombre fini, donc il en existe un de poids minimal (s'il existe au moins un chemin). Donc la distance est bien définie (et vaut $+\infty$ s'il n'existe pas de chemin).

2) Notons $\Gamma_{i,j}^{(k)}$ l'ensemble des chemins réduits dont les sommets intermédiaires appartiennent à $\{0, 1, \dots, k-1\}$ reliant le sommet i au sommet j .

Un chemin (réduit) de $\Gamma_{i,j}^{(k+1)}$ soit appartient à $\Gamma_{i,j}^{(k)}$ soit est le concaténé d'un chemin de $\Gamma_{i,k}^{(k)}$ et d'un chemin de $\Gamma_{k,j}^{(k)}$ (en considérant la position **unique** où le chemin passe par le sommet k). On a donc :

$$D_{k+1}[i][j] = \min(D_k[i][j] , D_k[i][k] + D_k[k][j])$$

3) On obtient les distances souhaitées en calculant $D_n[i, j]$

```
def distances(G) :  
    n = len(G) ; D = [ [ float("inf") for i in range(n)] for j in range(n)]  
    for i in G :  
        # initialisation  
        for (j,w) in G[i] : D[i,j] = w        # initialisation  
    for k in range(1,n+1) :  
        Dex = D.copy()        # bien faire une copie, sinon problème de mise à jour  
        for i in G :  
            for j in G :  
                D[i,j] = min(Dex[i,j],Dex[i,k]+Dex[k,j])  
    return D
```

4) La complexité est en $O(n^3)$ compte tenu des trois boucles **for** imbriquées.

5) L'idée est de trouver, lorsque $s \neq t$, le successeur v de s dans le chemin optimal, puis d'itérer le procédé. Le sommet v se caractérise par la propriété $D[s, t] = \omega(s, v) + D[v, t]$.

```
def chemin(D,G,i,j) :  
    if i == j : return [i]  
    for (k,w) in G[i] :  
        if D[i,k] == w + D[k,j] : return [i] + chemin(D,G,k,j)  
    assert False    # normalement, cette instruction n'est jamais effectuée
```

Exercice B

```
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 très différents (p^2 vs p^3).

3) Pour $1 \leq i \leq j \leq n$, on note $d(i, j)$ le nombre d'opérations minimum pour effectuer le produit $A_i \dots A_j$.

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 $A_i \dots A_k$ et $A_{k+1} \dots A_j$, où $i \leq k \leq j-1$, est

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

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

```

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

```

4) La complexité est au pire $O(n^3)$. En effet, le calcul de chaque $d(i, j)$ demande $O(n)$ opérations.

5) On attribue ici à $\text{dico}[(i, j)]$ 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 not (i,j) in dico : return
        if i == j ; dico[(i,j)] = (0, [])
        elif not (i,j) in dico :
            L = [aux[(i,k)][0]+aux[(k+1,j)][0]+P[i-1]*P[k]*P[j] for k in range(i,j)]
            k = ind_min(L) ; m = L[k]
            sol = [1]+aux[(i,k)][1]+aux[(k+1,j)][1]+[-1]

```

```

        dico[(i,j)] = (m,sol)

    dico[(i,j)]

    return aux(1,n)[1]

```

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

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

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

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

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

Chaque liste $[i_1, \dots, i_p]$ admet p successeurs qui sont les listes obtenus en supprimant un des p éléments de la liste. Le poids de l'arête $[i_1, \dots, i_p] \rightarrow [i_1, \dots, i_{k-1}, i_{k+1}, \dots, i_p]$ 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 $(A_{i_{k-1}+1} \dots A_{i_k})$ et $(A_{i_k+1} \dots A_{i_{k+1}})$.

7) On a $c_0 = 1$ et $\forall n \in \mathbb{N}^*$, $c_n = \sum_{k=0}^{n-1} c_k c_{n-1-k}$ (qui sont appelés nombres de Catalan)

Il faut mieux définir $c(n)$ en construisant à l'aide d'une boucle **for** la liste des c_k , pour $0 \leq k \leq n$.