DST Informatique pour tous
PSI et MP
Durée : 2 heures

15 février

La clarté de la présentation et la qualité de la rédaction font partie de l'évaluation. On veillera à utiliser des noms explicites pour les variables et les fonctions introduites. Le candidat pourra clarifier les programmes par l'ajout de commentaires judicieux si nécéssaire.

Dans tout le sujet on supposera avoir importé la bibliothèque `numpy`, sous la forme :

```
import numpy as np
```

Le graphe simplifié du réseau Ultranet de ce pays est représenté à la figure 1. Le nom des villes correspondant aux identifiants des sommets est enregistré dans la base de données (Table1). Une arête relie deux villes lorsqu'un câblage physique direct est établi entre elles et la bande passante permise par ce câblage est indiquée comme poids. Il y a au plus une arête entre deux villes. La structure du réseau Ultranet a été intégralement terminée lors du grand plan « Réseau pour Tous » mené par le premier ministre Petro Sank-Ærixh et il s'agit désormais d'en confier l'exploitation aux deux fournisseurs d'accès concurrents. Dans ce sujet, on s'intéresse à une procédure de partage de la gestion du réseau entre les deux opé- rateurs. On suppose qu'une liaison entre deux villes ne peut être exploitée que par l'un ou par l'autre des deux fournisseurs d'accès qui en aura alors l'usage exclusif. Le gouvernement du Listenbourg ne souhaite pas s'embarrasser avec une procédure complexe d'appel d'offre et impose la procédure simple suivante pour répartir l'exploitation exclusive des liaisons physiques du réseau : • tant qu'il reste des liaisons à attribuer, chaque opérateur, à tour de rôle et en commençant par MaxDébit, choisit parmi les liaisons restantes une liaison qu'il souhaite exploiter exclusivement. Une fois la procédure de répartition terminée, c'est-à-dire une fois que toutes les liaisons ont été attribuées, chaque fournisseur d'accès dispose de son propre sous-réseau exclusif, dont la gestion lui revient entièrement. L'objectif d'un fournisseur d'accès est d'obtenir un sous-réseau permettant de proposer la meilleure bande passante possible entre deux villes quelconques du pays.

Les cinq parties sont essentiellement indépendantes dans leur traitement mais des notions utiles à la résolution d'une partie peuvent être introduites dans les parties précédentes. La partie I est indépendante des quatre autres. La partie II introduit des notions utiles à la résolution des parties III et IV. La partie IV introduit des concepts utilisés dans la partie V.

Dans tout le sujet, il est toujours admis d'utiliser un résultat ou un programme correspondant à une question précédente, même si cette question n'a pas été résolue.

Partie I- Base de données du réseau

Le ministère des affaires environnementales, des ressources, de l'agriculture et des forêts du Listenbourg met à disposition des deux fournisseurs d'accès une base de données relationnelle. Celle-ci comporte deux tables dont le schéma est le suivant :

— villes(id : entier, nom : texte, pop : flottant)

— liaisons(id1 : entier, id2 : entier, bp : flottant)

Un enregistrement de la table villes comporte l'identifiant unique d'une ville (id), le nom de cette ville (nom) et sa population en millions d'habitants (pop). A priori, plusieurs villes du Listenbourg peuvent porter le même nom. Un enregistrement de la table liaisons correspond à une liaison physique directe entre deux villes données par leur identifiant (id1, id2) ainsi que la bande passante en $Tb.s^{-1}$ correspondant à ce câblage (bp). On garantit que dans la représentation d'une liaison, l'identifiant de la première ville est toujours strictement inférieur à celui de la deuxième ville. Rappelons qu'il ne peut exister qu'au plus une liaison entre deux villes.

Un exemple simplifié du contenu de cette base de données, correspondant au graphe simplifié du réseau du Listenbourg de la figure 1, est donné Table 1.

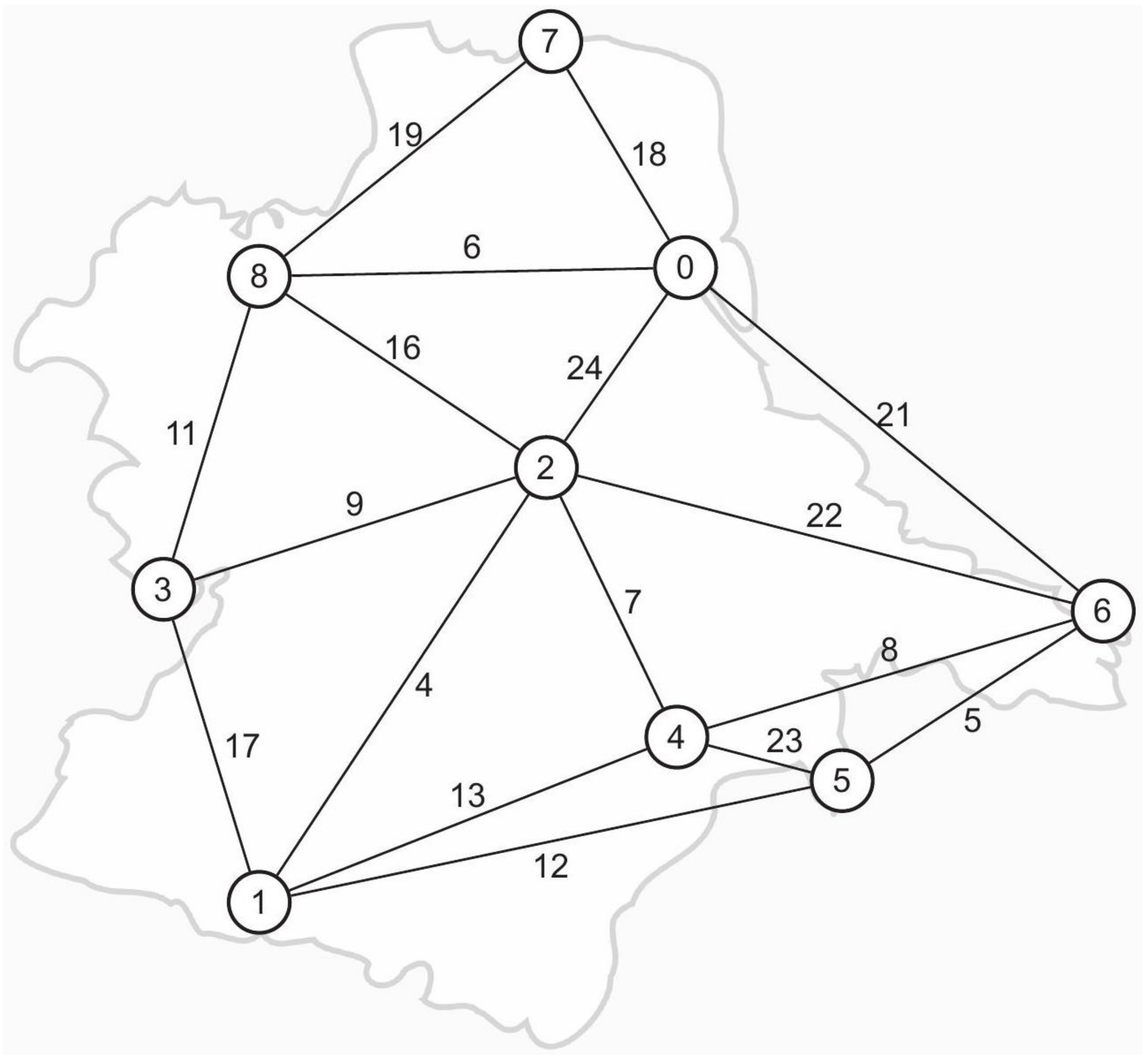

FIGURE 1 – - Réseau simplifié du Listenbourg

villes		
id	nom	pop
0	Lurenberg	12.0
1	Veroja	10.0
2	Aschlöss	8.0
3	Stratord	5.2
4	Gossard	5.1
5	La Galinera	5.0
6	Sainte Marie	5.0
7	Atlanitkischer	4.7
8	Gasparländ	3.5

1. Pour la table villes id est une clé primaire et pour la table liason bp.
2. Pour chacune des questions suivantes on demande d'écrire une requête SQL portant sur le schéma relationnel proposé qui fonctionnerait quel que soit le contenu de la base de données et pas uniquement sur l'exemple de contenu proposé à la figure 3.

liaisons		
id1	id2	bp
0	2	24.0
0	6	21.0
0	7	18.0
0	8	6.0
1	2	4.0
1	3	17.0
1	4	13.0
1	5	12.0
2	3	9.0
2	4	7.0
2	6	22.0
2	8	16.0
3	8	11.0
4	5	23.0
4	6	8.0
5	6	5.0
7	8	19.0

Table 1 – - Contenu de la base de données correspondant au réseau simplifié de la figure 2

(a) Écrire une requête SQL qui donne les noms des villes comportant strictement plus de cinq millons d'habitants ;
SELECT nom FROM villes WHERE pop>5

(b) Écrire une requête SQL qui donne la bande passante moyenne des liaisons incidentes à la ville d'identifiant 2 ;
SELECT AVG(bp) FROM liaisons WHERE id1=2

(c) Écrire une requête SQL qui donne les noms des deux villes reliées par la liaison de bande passante maximale. On suppose que toutes les liaisons ont une bande passante différente.
SELECT nom FROM villes WHERE id IN (SELECT id1,id2 FROM liaison WHERE bp=(SELECT max(bp) FROM liaison))

Ou aussi

SELECT nom FROM villes UNION (SELECT id1 AS x,id2 AS y FROM liaison) UNION (SELEcT MAX(bp) AS b FROM liason) HAVING bp=b AND (id=x OR id =y)

3. Déterminer le résultat de la requête SQL suivante sur l'exemple de contenu de la base de données représentée à la Table 1.

```
SELECT nom, COUNT(*) AS d
    FROM villes
        JOIN (SELECT id1 AS x, id2 FROM liaisons UNION SELECT id2 AS x, id1 FROM liaisons)
        ON id = x
    GROUP BY id
    HAVING d >= 4
    ORDER BY id ASC

    Renvoie le nom des villes qui sont relier à plus de 4 autres villes classée couplé à ce nombre de
```

```
import numpy as np

D=np.array([[ 0.,  0., 24.,  0.,  0.,  0., 21., 18.,  6.],
        [ 0.,  0.,  4., 17., 13., 12.,  0.,  0.,  0.],
        [24.,  4.,  0.,  9.,  7.,  0., 22.,  0., 16.],
        [ 0., 17.,  9.,  0.,  0.,  0.,  0.,  0., 11.],
        [ 0., 13.,  7.,  0.,  0., 23.,  8.,  0.,  0.],
        [ 0., 12.,  0.,  0., 23.,  0.,  5.,  0.,  0.],
        [21.,  0., 22.,  0.,  8.,  5.,  0.,  0.,  0.],
        [18.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 19.],
        [ 6.,  0., 16., 11.,  0.,  0.,  0., 19.,  0.]])
#4-a
def Ladj(A):
```

```
    n=len(A)
    L=[[] for k in range(n)]
    for i in range(n):
        for j in range(n):
            if A[i,j]!=0:
                L[i].append((j,A[i,j]))
    return L
#4-b
def arrete(A):
    L=[]
    n=len(A)
    for i in range(n):
        for j in range(i,n):
          if A[i,j]!=0:
            L.append((A[i,j],i,j))
    return L
#4-c
def noeud(S):
    m={}
    for u in S:
        m[u[1]]=True
        m[u[2]]=True
    return list(m.keys())
#4-d
def matrice_adj(L):
    n=len(noeud(L))
    A=np.zeros((n,n))

    for u in L:
        i,j=u[1],u[2]
        A[i,j]=u[0]
        A[j,i]=u[0]
    return A
#5)
def P(G,s,marque):
     v=[]
     m={}
     attente=[]
     attente.append(s)
     while len(attente)>0:
           u=attente.pop()
           if u not in marque:
               v.append(u)
               marque[u]=True
               for s in G[u]:
                   if s[0] not in marque:
                       attente.append(s[0])
     return v
# On remarque que G doit etre liste d'ajacence.
# C'est un programme de recherche en profondeur adapter
# pour obtenir la liste des noeuds relies au "premier noeud".
# Remarquon que G peut etre aussi bien la liste d'adjacence ou un dictionnaire.
#5-a
# AIE ici il faut la liste d'adjacence pour utiliser le programme,
# alors que L est la liste des aretes.
# Puis verifier que tous les noeuds de G sont relies ensemble.
def connex(L):
    A=matrice_adj(L)
    S=noeud(L)
    G=Ladj(A)
    n=len(S)
    return n==len(P(G,G[0],{}))
```

```
 #5-b
def Comp_connex(G):
    A=matrice_adj(G)
    S=noeud(G)
    L=Ladj(A)
    marque={}
    Connexe=[]
    for u in S:
        if marque[u]==False:
            visite=P(L,u,marque)
            Connexe.append(visite)
    return Connexe
 #6
def sur(A,S):
    n=len(A)

    p=0
    j,s=0,0
    for i in S:
        for k in range(n):
            if k not in S:
                if A[i,k]>p:
                    p=A[i,k]
                    if i>k:
                     j,s=k,i
                    else:
                       s,j=k,i
    return(p,j,s)
 #7 et 8
def initialisation(A):
    n=len(A)
    H=[]
    for i in range(n):
        p=0
        for j in range(n):
            if A[i,j]>p:
                p=A[i,j]
                k=j
        H.append((p,np.min([i,k]),np.max([i,k])))
    return H
 #Erreur denonce je voulais demander :
S= initialisation(array([[0., 6., 2., 7.],
       [6., 0., 5., 8.],
       [2., 5., 0., 3.],
       [7., 8., 3., 0.]]))
       renvoie
initialisation(A)
Out[6]: [(7.0, 0, 3), (8.0, 1, 3), (5.0, 1, 2)]
#A=matrice_adj(S)
def Boruvka(A):
    H=initialisation(A)
    while connex(H)==False:
        L=Comp_connex(H)
        for S in L:
            u=sur(A,S)
            if u not in H:
              H.append(sur(A,S))
    return H
 # 9
def configuration_initiale(g):
   n =len( g)
   etat = (0,) * n
```

```
    return (1, etat)

# 10
def finale(c):
    for u in c[1]:
        if u==0:
            return False
    return True
# 11
def configurations_suivantes(c):
    L=[]
    j,b=c

    for i in range(len(b)):
        if b[i]==0:
            v=list(b)
            v[i]=j
            L.append((j*(-1),(tuple(v)))
    return L
#12
# renvoie 1 si la configuration est gagnante pour 1 et respectivement
# soit 1 a gagner soit toutes les issus sont gagante pour 1.

#13

# parmi toutes les configuration_suivantes pour t=1 on choisit celle de score max
# ( respctivement pour -1 de score min)

 #14
cache={}
def score_memo(g,c):

    if finale(c):
        return gagnant(g, c)
    t, etat = c
    C=configurations_suivantes(c)
    score=0
    if t==1:
        for suiv in C:
            if suiv in cache:
                score=max ([score,cache[suiv]])
            else:
                score_suiv=score_memo(g,suiv)
                cache[suiv]=score_suiv
                score=max ([score,cache[suiv]])
        return score
    else:
        for suiv in C:
            if suiv in cache:
                score=min ([score,cache[suiv]])
            else:
                score_suiv=score_memo(g,suiv)
                cache[suiv]=score_suiv
                score=min ([score,cache[suiv]])
        return score

 #15
def score_minmax(g,c,p):
    if est_finale(c)or p==0:
        return gagnant(g, c)
    t, etat = c
    score_fils = [score(g, suiv,p-1) for suiv in configurations_suivantes(c)]
```

```
    if t == 1:
        return max(score_fils)
    else:
        return min(score_fils)
```