M2201 TP 2

Stockage de graphes sur machine

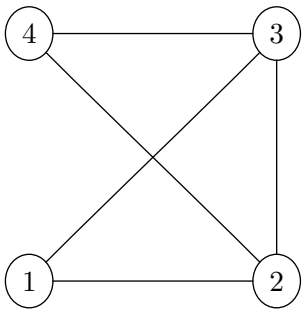
26 mars 2019 – B. Colombel

Objectifs : *Se familiariser avec les différentes représentations d'un graphe en machine*

Sources : *Sébastien Martin*
IUT Paris Descartes
(département informatique)

1 Description d'un graphe non orienté

Pour stocker un graphe en machine, il est nécessaire de connaître le nombre de sommets et la liste des arêtes (sous une forme donnée). Considérons le graphe associé à cette représentation :



Nous allons décrire plusieurs façons de décrire ce graphe de référence sous scilab (dans la suite, le suffixe `_ref` renvoie explicitement à ce graphe de référence).

Matrice d'adjacence Le graphe peut être défini par la matrice d'adjacence

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Sous scilab, cette matrice est définie par la commande :

```
M_ref = [0,1,1,0; 1,0,1,1; 1,1,0,1; 0,1,1,0]
```

Liste d'arêtes Le graphe peut être défini par la donnée du nombre de ses sommets, `N_ref=4`, et de la liste d'arêtes [(1,2),(1,3),(2,3),(2,4),(3,4)]. Sous scilab, cette liste pourra être stockée sous la forme matricielle suivante (d'autres choix sont possibles !) :

```
A_ref = [1,2; 1,3; 2,3; 2,4; 3,4]
```

Liste d'adjacence Le graphe peut être défini par la donnée du nombre de ses sommets, `N_ref=4`, et de la liste d'adjacence [[2,3],[1,3,4],[1,2,4],[2,3]]. Cette *liste de listes* contient l'information suivante : la liste n° i de la liste d'adjacence décrit l'ensemble des voisins du sommet i. Sous scilab, cette liste peut être stockée sous la forme matricielle

```
L_ref = [2,3,0; 1,3,4; 1,2,4; 2,3,0]
```

où, afin de conférer à `L_ref` une structure matricielle, nous avons ajouté des 0 en l'absence de voisins supplémentaires (**Note :** les sommets sont numérotés de 1 à `N_ref` ; 0 n'est donc pas un sommet et aucune confusion n'est donc possible dans la lecture de `L_ref`).

Remarque. Rigoureusement, le nombre de sommets n'est pas nécessaire si l'on définit les arêtes par la matrice d'adjacence. Mais l'information est nécessaire si on définit les arêtes par liste d'arêtes ou liste d'adjacence : en effet, le(s) dernier(s) sommet(s) pourrai(en)t être isolé(s).

2 Stockages sur machine d'un graphe non orienté

Nous allons travailler avec des graphes non orientés représentés par liste d'adjacence, matrice d'adjacence ou liste d'arêtes. Le but de ce TP est de se familiariser avec ces représentations en définissant des procédures de *conversion* d'un format à un autre.

Exercice 1 :

1. Dans un script `TP1.sce`, définir le graphe de référence dans les différents formats présentés :

```
clear
N_ref=4;                                    // nombre de sommets
M_ref=[0,1,1,0; 1,0,1,1; 1,1,0,1; 0,1,1,0]; // matrice d'adjacence
A_ref=[1,2;1,3;2,3; 2,4; 3,4];              // liste d'aretes
L_ref=[2,3,0;1,3,4; 1,2,4; 2,3,0];          // liste d'adjacence
```

   Afficher ces formats dans scilab en lançant le script (appuyer sur la touche F5).

2. Construire une fonction scilab, `listadjTOmatadj`, qui permet de renvoyer la matrice d'adjacence associée à une liste d'adjacence donnée. Pour ce faire, on intégrera la fonction dans le script puis on testera la fonction sur le graphe de référence :

```
function M =listadjTOmatadj(N,L)
      ...
endfunction
M=listadjTOmatadj(N_ref,L_ref)
```

   Pour valider la fonction, on comparera `M` et `M_ref` (qui doivent être identiques !). De façon analogue, construire la fonction réciproque `matadjTOlistadj.sci` qui permet de renvoyer la liste d'adjacence associée à une matrice d'adjacence donnée.

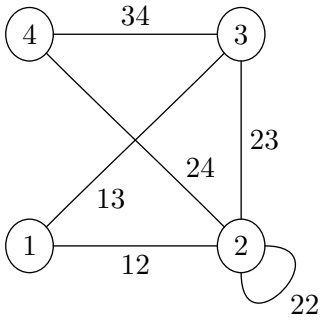
3. Construire une fonction scilab, `listadjTOlistaretes.sci`, qui permet de renvoyer la liste des arêtes associée à une liste d'adjacence donnée. Construire la fonction réciproque `listaretesTOlistadj.sci`.

4. Construire une fonction scilab, `matadjTOlistaretes.sci`, qui permet de renvoyer la liste des arêtes associée à une matrice d'adjacence donnée. Construire également la fonction `listaretesTOmatadj.sci`.

Chaque fonction sera validée à l'aide du graphe de référence.

3 Stockages sur machine des graphes valués

Pour souhaite stocker en machine le graphe non-orienté et valué ci-contre :



Poue cela, nous adaptons les conventions de la partie précédente de la manière suivante :

```
// Nombre de sommets :
Ng=4;
// Definition des aretes par liste des aretes
// Convention : l'arete est decrite par ij avec i<j
arete_Ag=[1,2; 1,3; 2,2; 2,3; 2,4; 3,4];
value_Ag=[ 12;  13;  22;  23;  24;  34];
// Definition des aretes par la matrice d'adjacence
arete_Mg=[0   , 1, 1,   0;  1, 1, 1, 1;  1, 1,   0, 1;    0, 1, 1,   0];
value_Mg=[%inf,12,13,%inf; 12,22,23,24; 13,23,%inf,34; %inf,24,34,%inf];
// Definition des aretes par liste d'adjacence (les voisins)
arete_Lg=[ 2, 3,   0,   0;  1, 2, 3, 4;  1, 2, 4,   0;  2, 3,   0,   0];
value_Lg=[12,13,%inf,%inf; 12,22,23,24; 13,23,34,%inf; 24,34,%inf,%inf];
```

Exercice 2 :

1. En utilisant l'exemple ci-dessus, expliquer comment on peut définir, de manière simple, un graphe non orienté *valué* en conservant les structures définies précédemment ?
2. Adapter les fonctions précédentes afin de traiter des graphes non orientés *valués*.

Exercice 3 : Reprendre les définitions et procédures précédentes afin de traiter le cas des graphes *orientés valués*. On pourra par exemple utiliser le graphe :