M2201 TP 4

Quelques algorithmes classiques

12 mai 2019 – B. Colombel

Objectifs : *Implémenter une version simplifier de l'algorithme de Belleman-Ford et l'utiliser pour résoudre un problème d'ordonnancement.*

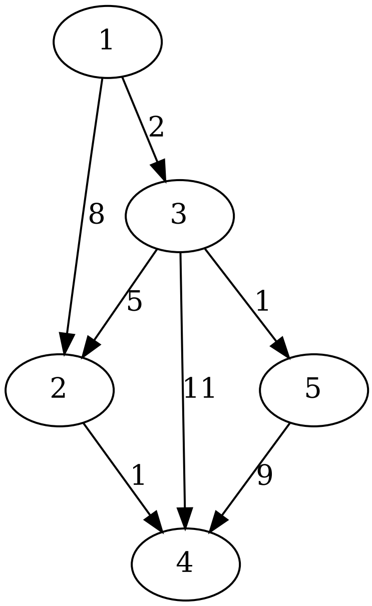
Sources : *Étienne coutant*
IUT de Reims
(département informatique)

Dans le TP n° 2, nous avons étudié différents formats de représentation d'un graphe orienté et valué. Dans ce TP, nous considérerons qu'un graphe orienté valué est défini par sa matrice d'adjacence et sa matrice de valuation.

Par exemple,

```
--> MV = [%inf, 8, 2, %inf, %inf ;
  > %inf, %inf, %inf, 1, %inf ;
  > %inf, 5, %inf, 11, 1 ;
  > %inf, %inf, %inf, %inf, %inf ;
  > %inf, %inf, %inf, 9, %inf]
 MV  =

   Inf   8.    2.    Inf   Inf
   Inf   Inf   Inf   1.    Inf
   Inf   5.    Inf   11.   1.
   Inf   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   9.    Inf
```



est la matrice de poids du graphe ci-contre.

Questions préliminaires : On considère les matrices $M = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$ et $N = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$.

1. Quelle réponse obtient-on suite à la commande `M == N` ?
2. Quelle réponse obtient-on suite à la commande `isequal(M, [1 0 ; 1 2])` ?
3. Quelle commande doit-on utiliser pour savoir si deux matrices sont égales ?

1 Recherche de plus court (long) chemin

1.1 Algorithme de Bellman-Ford (simplifié)

Le but de ce TP est d'implanter sous scilab l'algorithme de Bellman-Ford qui permet de déterminer, sous conditions, un plus court chemin entre un sommet choisi et tous les autres sommets du graphe.

On rappelle l'algorithme de Belleman-Ford dans une version simplifiée (qui détermine les plus courtes distances sans fournir le chemin) :

Algorithme de Bellman-Ford (simplifié)

Entrées : La matrice de poids M d'un graphe G
Initialisation : $Dist$ = liste des distances depuis le sommet 1, initialisées à l'infini ;
$Dist(1) \leftarrow 0$;

début
  tant que *Dist change* **faire**
    pour *tout sommet i* **faire**
      pour *tout sommet j* **faire**
        si $Dist(i) + M(i,j) < Dist(j)$ **alors**
          $Dist(j) \leftarrow Dist(i) + M(i,j)$;
        fin
      fin
    fin
  fin
fin

Sorties : retourner *Dist*

Exercice 1 :

1. Construire une fonction `Dist = bellman_ford(M)` , qui renvoie les distances du plus court chemin du sommet 1 à tous les autres, dans un graphe orienté de matrice de poids M, à l'aide de l'algorithme de Bellman-Ford.

   On pourra tester avec la matrice MV :

```
--> bellman_ford(MV)
 ans  =

   0.   7.   2.   8.   3.
```

2. Que doit-on modifier dans les algorithmes et matrices précédents pour traiter un problème de plus long chemin ? [1]
   Écrire alors une fonction `Dist = bf_long(M)` qui renvoie les distances du plus long chemin du sommet 1 à tous les autres.

   Par exemple :

```
--> bf_long(MV)
 ans  =

   0.   8.   2.   13.   3.
```

1. N'hésitez pas à sortir vos TD et/ou le cours !

1.2 Application : problème d'ordonnancement

Exercice 2 : Utiliser les fontions de la partie précédente pour retrouver les dates au plus tôt et les dates au plus tard du problème d'ordonnancement étudié lors du TP précédent :

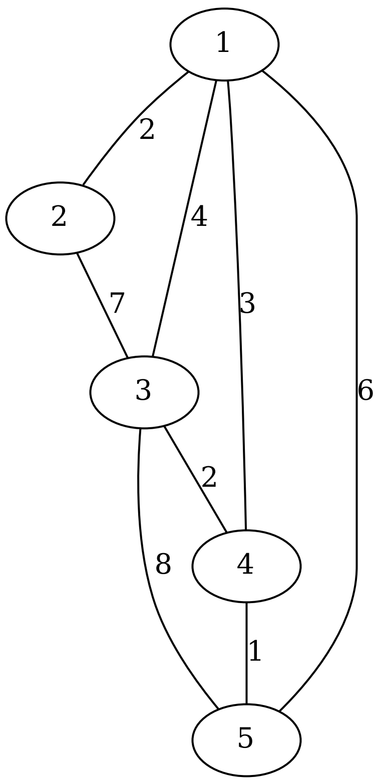
Tâches	**Durée**	Tâches antérieures
A	7	–
B	3	A
C	1	B
D	8	A
E	2	D, C
F	1	D, C
G	1	D, C
H	3	F
I	2	H
J	1	E, G, I

$$M = \begin{pmatrix} 0 & 0 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 7 & \infty & 7 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 3 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & 1 & 1 & 1 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & 8 & 8 & 8 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty & 1 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 3 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

2 Arbres de recouvrement

2.1 Énoncé du problème

Les villes Cityone, Deuxville, Troisbourg, Saint-Quatre et Cinco-Ciudad souhaitent être reliées par un réseau de trains. Pour cela, une étude a été menée sur le coût de projet, aboutissant au graphe suivant :



Par exemple, cela couterait 3 millions de relier directement Cityone à Saint-Quatre, et 6 millions de relier directement Cityone à Cinco-Ciudad.

Le regroupement de communes décide d'autoriser les changements de lignes, et de ne construire que les lignes nécessaires, afin d'assurer un coût total minimal.

Mathématiquement, il s'agit de construire un arbre couvrant de poids minimal, c'est à dire un arbre passant par tous les sommets du graphe, dont la somme des poids des arêtes est minimale.

Pour cela, nous allons utiliser un des deux algorithmes étudiés en cours ; l'algorithme de Prim.

2.2 Algorithme de Prim

On rappelle l'algorithme de Prim :

Algorithme de Prim

Entrées : La matrice de poids M d'un graphe G
Initialisation : Marquer le sommet 1 ;
arbre = arbre résultat ne contenant initialement pas d'arêtes ;
début
  tant que *tous les sommets ne sont pas marqués* **faire**
    $\{x; y\}$ = arête de poids minimal joignant un sommet marqué x à un sommet non marqué y ;
    marquer y ;
    ajouter l'arête $\{x,y\}$ à *arbre* ;
  fin
fin
Sorties : retourner *arbre*

Exercice 3 : Écrire une fonction en scilab `arbre = prim(M)` qui retourne la matrice de l'arbre couvrant de poids minimum d'une matrice de poids `M`.

Remarque. On pourra par exemple créer une liste `marquage`, avec `marquage(i) = 1` si le sommet `i` est marqué, `0` sinon.

On pourra tester avec la matrice $MV2$ du graphe du problème 2.1 :

```
--> prim(MV2)
 ans  =

   Inf   2.    Inf   3.    Inf
   2.    Inf   Inf   Inf   Inf
   Inf   Inf   Inf   2.    Inf
   3.    Inf   2.    Inf   1.
   Inf   Inf   Inf   1.    Inf
```