Algorithmes sur les graphes orientés

1er décembre 2014

Résumé

Il s'agit d'implanter deux algorithmes du cours concernant les graphes orientés pour déterminer les composantes fortement connexes (Tarjan) et détecter la présence de circuits absorbants (Bellman-Ford).

1 Terminologie

Un graphe orienté est un ensemble de sommets et d'arcs (arêtes orientées). Si V désigne l'ensemble des sommets (vertex) du graphe, les arêtes (edge) sont représentées par une partie A de $V \times V$. Un chemin de longeur n est une suite de sommets x_0, x_1, …, x_n tel que :

$$\forall i, \quad 0 \leq i < n \Longrightarrow (x_i, x_{i+1}) \in A \tag{1}$$

Un chemin fermé $x_0 = x_n$ est un circuit. On définit une relation d'équivalence sur le graphe en disant que deux sommets sont connectés quand il existe un circuit passant par ces deux sommets. Les classes d'équivalences pour cette relation sont des composantes fortement connexes. Une fonction f à valeurs réelles sur les arêtes se prolonge aux chemins par addition des coûts, le coût du chemin [1] vaut :

$$\sum_{i=0}^{n-1} f\big((x_i, x_{i+1})\big)$$

Un circuit de coût strictement négatif est dit absorbant.

2 Représentation

Pour cette séance de travaux pratiques, vous utiliserez les trois représentations usuelles des graphes à sommets entiers : liste d'arcs, listes et matrice d'adjacences.

```
typedef  struct _list_ {
  int    num;
  struct _list_ * svt;
} enrliste , *liste;
```

```
typedef  struct {
  int     card;
  liste* adj;
} digraph;
```

L'objectif de cet exercice est de coder les représentations matricielles et par listes d'adjacence pour un graphe défini dans un fichier texte par une suite d'arcs. Plus précisément, toutes les chaines de caractères du fichier correspondant à l'expression régulière

$$[0-9]+[[:\mathit{space}:]]*->[[:\mathit{space}:]]*[0-9]+$$

seront interprétés comme des arcs du graphe. Le programme `arc.c` filtre les arcs d'un fichier de nom passé par l'option $-f$.

Par exemple, en appliquant la commande `grep` au fichier `comp.dot` (utilisé pour obtenir le dessin [2]), on obtient :

```
tarjan> grep -E '[0-9]+[[:space:]]*->[[:space:]]*[0-9]+'
    compo.dot
  1 -> 6-> 9 -> 1;
  2 -> 3 -> 7 -> 0 -> 2;
  4-> 5->8 -> 10 -> 4;
        5 -> 0;
        6 -> 4;
        1 -> 7;
tarjan> ./arc.exe -f compo.dot
 (1,6) (6,9) (9,1) (2,3) (3,7) (7,0) (0,2) (4,5)
 (5,8) (8,10) (10,4) (5,0) (6,4) (1,7)
```

1. Modifier la source de `arc.c` pour gérer une option $-l$ qui contruit une représentation du graphe par liste d'adjacence.
2. Ajouter une option $-d$ pour produire un fichier `.dot` utilisable par la commande `dot` ou `neato` du projet graphviz.
3. Gérer une option $-m$ pour construire la matrice d'adjacence du graphe.
4. Ajouter une option $-c$ pour dénombrer tous les chemins de petite longueur.

3 Parcours de graphe

Quand un sommet x est visité lors d'un parcours de graphe en profondeur (ou largeur), on distingue deux types d'arcs (x, y). Les arcs de rebroussement qui correspondent aux sommets y déjà visités. Les arcs de poursuite du parcours qui forment une forêt.

1. Implanter une commande pour réaliser un parcours en profondeur récursif du cours.

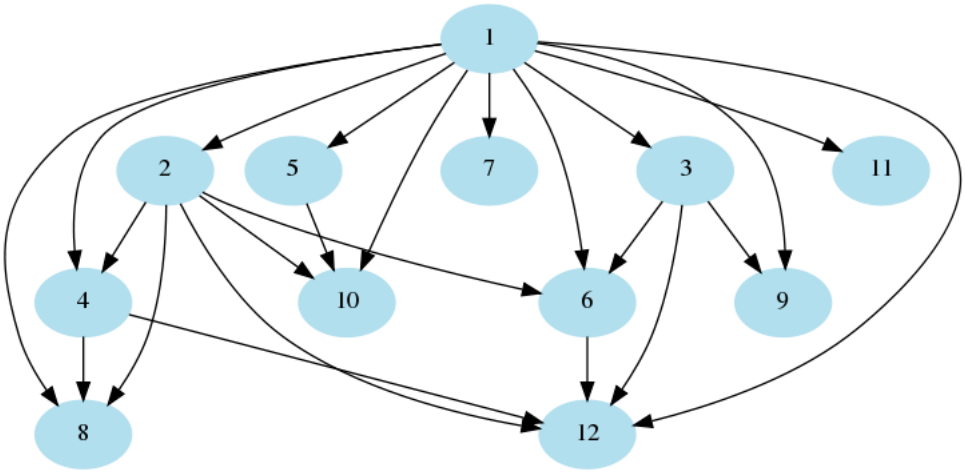

FIGURE 1 – L'arbre de divisibilité dessiné avec la commande `dot`

2. Modifier la commande pour afficher l'ordre de visite des sommets et les arcs de la forêt associée au parcours.
3. Dessiner les graphes avec `dot` pour mettre en évidence : l'ordre de visite des sommets, les arcs de reboussement et la forêt.

4 Algorithme de Tarjan

1. Implanter l'algorithme de Tarjan.
2. Donner la distribution des composantes fortement connexes des graphes : niho-1, niho-2, niho-4.

5 Algorithme de Bellman-Ford

```
digraph G {

 subgraph cluster_0 {
  node [style=filled,color=lightgrey];
  1 -> 6-> 9 -> 1;
  label = "#1";
  color=blue
 }

 subgraph cluster_1 {
  node [style=filled, color=lightgrey];
  2 -> 3 -> 7 -> 0 -> 2;
  label = "#2";
  color=blue
 }
 subgraph cluster_2 {
  node [style=filled, color=lightgrey];
  4-> 5->8 -> 10 -> 4;
  label = "#3";
  color=blue
 }
        5 -> 0;
        6 -> 4;
        1 -> 7;
}
```

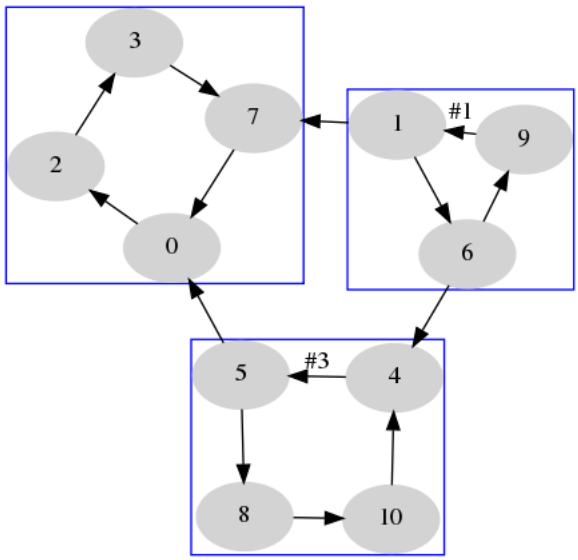

FIGURE 2 – Un graphe orienté avec 3 composantes fortement connexes, dessiné avec la commande `neato`.