Graphes

Eric Goubault

CEA/Saclay

Eric.Goubault@cea.fr

http://www.di.ens.fr/~goubault

Remarque sur les arborescences

$G = (X, A)$ graphe, arborescence (par exemple de Trémaux) (Y, T, x). Les arcs de G qui ne sont pas dans Y sont de quatre nature:

- Les arcs issus d'un sommet qui ne sont pas atteignable depuis x,
- Les arcs de descente: (y, z) avec z descendant de y dans (Y, T, x) mais n'est pas un des successeurs dans cette arborescence,
- Les arcs de retour: (y, z) où z est un ancêtre de y dans (Y, T, x)
- Les arcs transverse: (y, z) où z n'est pas ancêtre ni descendant de y dans (Y, T, x)

Bi-connexité

- Point d'articulation: point qui sépare le graphe en parties non-connexes si on l'enlève
- Un graphe sans point d'articulation est un graphe bi-connexe
- "Robustesse" d'un réseau

Une composante biconnexe est un sous-graphe biconnexe maximal (arête ou plus). Les composantes biconnexes sont "séparées" par les points d'articulation.

Principe du calcul des points d'articulation

Point d'attache:

- $G = (X, A)$ graphe, x sommet et (Y, T, x) arborescence de Trémaux induisant une numérotation préfixe num des sommets (ou parcours dfs)
- Le point d'attache $at(y)$ d'un sommet y de Y est le sommet de plus petit numéro, extrémité d'un chemin de G d'origine y et contenant au plus un arc (u, v) tel que $num[u] > num[v]$

Principe du calcul des points d'articulation

Un sommet x est un point d'articulation si:

- si c'est la racine de l'arbre de recouvrement, alors oui si la racine contient plusieurs fils dans l'arbre de recouvrement
- si ce n'est pas la racine de l'arbre de recouvrement, alors oui si il possède un fils y tel que $num[x] \leq num[at(y)]$

Principe du calcul des points d'attache

Point d'attache $at(y)$ du sommet y est le sommet de plus petit numéro parmi:

- le sommet y
- les points d'attaches des fils de y dans (Y, T, x)
- les extrémités des arcs transverses ou de retour dont l'origine est y

Code

```
static int numOrdre;
static int[] num;

static boolean[] trouverArticulations(Graphe g) {
  int n = g.succ.length;
  id = -1;
  num = new int[n];
  boolean[] articulation = new boolean[n];
  for (int x = 0; x< n; ++x) num[x] = -1;
```

Code

```
for (int x = 0; x < n; ++x)
  if (num[x] == -1) {
    num[x] = ++numOrdre;
    int nfils = 0;
    for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant
      int y = ls.val;
      if (num[y] == -1) {
        ++nfils; int m = attache(g, y, articulation); } }
    articulation[x] = nfils > 1; }
return articulation; }
```

Code

```
static int attache(Graphe g, int x, boolean[] artic-
ulation) {
  int min = num[x] = ++numOrdre;
  for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
    int y = ls.val; int m;
    if (num[y] == -1) {
      m = attache(g, y, articulation);
      if (m >= num[x])
        articulation[x] = true; }
      else m = num[y];
    min = Math.min(min,m); } return min; }
```

Forte connexité

Un graphe G est fortement connexe si pour tout x et y de G il existe:

- un chemin (orienté) de x vers y et
- un chemin (orienté) de y vers x

Principe de calcul

Soit $G = (X, A)$ graphe, (Y, T, x) une arborescence de Trémaux de G à partir de x. Si u est un sommet de Y satisfaisant,

- $u = at(u)$,
- Pour tout descendant v de u dans (Y, T, x) on a $at(v) < v$

Alors, l'ensemble $desc(u)$ des descendants de u dans (Y, T, x) forme une coposante fortement connexe de G.

Code

```
static int id = -1;

static void imprimerCompFConnexes(Graphe g) {
  int n = g.succ.length;
  int [] num = new int[n];
  for (int x = 0; x < n; ++x) num[x] = -1;
  for (int x = 0; x < n; ++x) {
    if (num[y] == -1)
      imprimerCompF(g, x, num); } }
```

Code

```
static int imprimercompF(Graphe g, int x, int[] num) {
  num[x] = ++numOrdre; Pile.ajouter(x,p);
  int min = id;
  for (Liste ls = g.succ[x]; ls != null; ls = ls.suivant) {
    int y = ls.val; int m;
    if (num[y] == -1)
      m = imprimerCompF(g,y,num);
    else m = num[y];
    min = Math.min(min, m);
  }
```

Code

```
if (min == num[x]) {
  int y;
  do {
    y = Pile.supprimer(p);
    System.out.print(y+'' ``);
    num[y] = g.succ.length;
  } while (y != x);
  System.out.println();
} return min; }
```