Notes de cours d'algorithmique – L3
2-connexité

François Laroussinie
francoisl@irif.fr
Page web du cours : https://www.irif.fr/~francoisl/l3algo.html

7 octobre 2020

Cette courte note vient compléter les 4 vidéos sur le sujet.

On rappelle deux définitions :

Définition 1 *Soit $G = (S, A)$ un graphe non-orienté connexe.*

- *un sommet $p \in S$ est un point d'articulation de G si il existe $u, v \in S$ tels que u, v et p sont distincts et que tout chemin reliant u à v passe par p.*
- *G est 2-connexe si pour $\forall u, v, w \in S$, il existe un chemin reliant u à v qui ne passe pas par w.*

L'idée du point d'articulation, c'est que si on l'enlève, on casse la connexité : le graphe est alors en « plusieurs » morceaux. Avec ces définitions, on voit tout de suite que G est 2-connexe si et seulement si G n'a pas de point d'articulation.

Nous avons aussi la propriété suivante (prouvée en amphi) qui est à la base de l'algorithme pour trouver les points d'articulation :

Propriété 1 *Soit $G = (S, A)$ connexe et non-orienté. Soit $G_\Pi = (S, A_\Pi)$ un arbre de parcours issu de PP. Alors $p \in S$ est un point d'articulation ssi :*

1. *p est la racine de G_Π et p a au moins deux fils dans G_Π, ou*
2. *p n'est pas la racine de G_Π et pour un de ses fils s dans G_Π (*i.e. $\Pi[s] = p$*), aucun des descendants de s n'est relié par une arête retour à un ancêtre strict de p.*

Il reste à trouver une méthode pour savoir si il existe « un fils s dans G_Π (*i.e.* $\Pi[s] = p$), tel que aucun des descendants de s n'est relié par une arête retour à un ancêtre strict de p ». Cela va être fait en calculant un coefficient pour chaque sommet ($\mathsf{Low}[-]$) défini par :

$$\mathsf{Low}[v] \stackrel{\text{def}}{=} \min\Big(\{\mathsf{d}[v]\} \cup \{\mathsf{d}[w] \mid \exists v \rightarrow^*_{G_\Pi} x \rightarrow_{AR} w \text{ t.q. } w \rightarrow^+_{G_\Pi} v\}\Big)$$

Dès lors, nous pouvons reformuler la propriété précédente en utilisant les coefficients Low et obtenir un critère pour savoir si un sommet v est un point d'articulation : c'est le cas ssi (1) $\Pi[v] = \mathsf{nil}$ et v a plusieurs fils dans G_Π, **ou** (2) $\Pi[v] \neq \mathsf{nil}$ et il existe s tel que $\Pi[s] = v$ et $\mathsf{Low}[s] \geq \mathsf{d}[v]$.

A présent, on peut donner une définition alternative (mais équivalente !) des coefficients Low qui sera facilement calculable avec l'algorithme de parcours en profondeur. La définition alternative est la suivante :

$$\mathsf{Low}[v] \stackrel{\text{def}}{=} \min\Big(\{\mathsf{d}[v]\} \cup \{\mathsf{Low}[s] \mid v = \Pi[s]\} \cup \{\mathsf{d}[w] \mid v \rightarrow_{AR} w\}\Big)$$

Notons d'abord que cette seconde définition récursive est bien fondée : le coefficient $\mathsf{Low}[v]$ des feuilles de G_Π ne dépend que des valeurs d des ancêtres et des arcs retour, et ensuite, à chaque niveau la définition s'appuie sur les niveaux inférieurs et toujours les valeurs d sur le chemin de la racine de G_Π au noeud courant... On peut procéder ainsi (par induction sur la hauteur des noeuds dans G_Π) pour montrer l'équivalence. Intuitivement on peut distinguer deux cas :

1. soit $\mathsf{Low}[v] = \mathsf{d}[v]$ car aucun des ancêtres de v n'est accessible via un arc retour depuis un descendant de v dans G_Π : et dans ce cas aucun des fils de v dans G_Π n'aura un coefficient Low inférieur à $\mathsf{d}[v]$ et v n'aura pas non plus d'arc retour vers un de ces ancêtres, et la seconde définition fournira le bon résultat.
2. soit $\mathsf{Low}[v] < \mathsf{d}[v]$ et dans ce cas un descendant (non strict) de v dans G_Π est relié par un arc retour à un ancêtre w de v, ayant un d inférieur à $\mathsf{d}[v]$). Il y a donc soit un fils s de v dans G_Π avec $\mathsf{Low}[s] < \mathsf{d}[v]$, soit c'est directement depuis v avec un arc retour.

Il reste à intégrer le calcul des coefficients Low au parcours en profondeur. Tester $v \rightarrow_{AR} w$ peut s'écrire $(v, w) \in A \wedge \mathsf{d}[w] < \mathsf{d}[v]$, cf la propriété vue en amphi : pour les graphes non orientés, dans un parcours en profondeur, toute arête est un arc de G_Π ou une arête retour.

On reprend donc l'algorithme de parcours en profondeur [1] pour la procédure `PA2C` qui trouvent les points d'articulation de G. Le tableau booléen $\mathsf{PA}[-]$ est le résultat : $\mathsf{PA[u]}$ sera égal à vrai ssi u est un point d'articulation de G.

Procédure `PA2C`(G, v)
`//`$\mathsf{PA}[-]$ `(var. globale) stockera le résultat:` $\mathsf{PA}[s] = \top$ `ssi` s `est un PA.`

```
1  begin
2      Couleur[v] := gris;
3      temps++;
4      d[v] := temps;
5      Low[v] := d[v];
6      PA[v] := ⊥;
7      nbfils := 0 ; //var. locale
8      pour chaque (v, w) ∈ A faire
9          si Couleur[w] = blanc alors
10             Π[w] := v ;
11             nbfils++;
12             PA2C(G, w) ;
13             si (Π[v] != nil ∧ Low[w] ≥ d[v]) alors PA[v] := ⊤ ;      —> cas n°2 de la propriété 1.
               Low[v] := min(Low[v], Low[w]) ;
14         sinon si (w != Π[v]) alors Low[v] := min(Low[v], d[w]) ;   —> test pour ne pas prendre une arête de
                                                                          l'arborescence pour une arête retour....
15     si Π[v] = nil ∧ nbfils > 1 alors PA[v] := ⊤ ;                  —> cas n°1 de la propriété 1.
16 end
```

Algorithme 1 : procédure de base pour trouver les points d'articulation

1. on supprime les données qui ne servent pas ici : le coloriage en noir et les dates f. De plus, on ne donne ici que la procédure de base (équivalente à PP) dans la mesure où la procédure principale se limite à colorier les sommets en blanc et à initialiser la variable *temps* à 0. Notons aussi que puisque G est connexe, il n'y a qu'un seul appel à la procédure de base depuis la procédure principale.

On donne à présent l'algorithme de calcul des composantes 2-connexes : c'est l'algorithme `PABC` ci-dessous.

Procédure `PABC`(G, v)
`//res (var. globale) contiendra l'ensemble des comp. 2-connexes.`

```
1  begin
2      Couleur[v] := gris;
3      temps++;
4      d[v] := temps;
5      Low_A[v] := d[v];
6      pour chaque (v, w) ∈ A faire
7          si (Couleur[w] = blanc) ∨ ((Couleur[w] = gris) ∧ ¬((d[v] > d[w] ∧ w =
           Π[v]) ∨ (d[v] < d[w]))) alors        (pour ne pas remettre dans P une arete
                                                 qui y est deja !)
8              P.push((v, w));
9          si Couleur[w] = blanc alors
10             Π[w] := v;
11             PABC(G, w);
12             si (Low_A[w] ≥ d[v]) alors
13                 aux = ∅;                     On dépile les arêtes d'une C2C
14                 si P ≠ ∅ alors
15                     (x, y) = P.pop();
16                     tant que P ≠ ∅ ∧ (x, y) ≠ (v, w) ∧ (x, y) ≠ (w, v) faire
17                         aux+ = {(x, y)};
18                         (x, y) = P.pop();
19                     aux += {(x, y)};
20                     res += aux;
21             Low_A[v] := min(Low_A[v], Low_A[w]);
22         sinon si (w != Π[v]) alors Low[v] := min(Low_A[v], d[w]);
23 end
```

Algorithme 8 : Procédure de base pour trouver les composantes 2-connexes