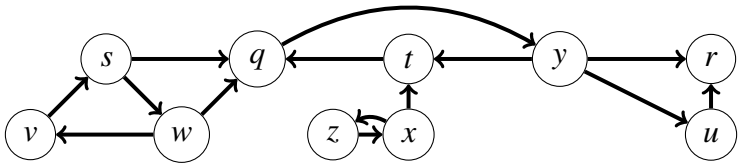
6.2.4 Composantes fortement connexes

Dans cette sous-section, nous allons nous intéresser au calcul des composantes fortement connexes.Commençons par définir les composantes fortement connexes.

Définitions

Soit $G = (S, A)$ un graphe orienté (pas forcément acyclique). On définit la relation $\leftrightarrows$ sur S par pour tout $u, v \in S$, $u \leftrightarrows v$ ssi il existe un chemin de u à v et de v à u.

Exemple 25. *Par exemple, dans le graphe suivant*



nous avons $s \leftrightarrows w$ car il existe un chemin de s à w et un chemin de w à s. Par contre, $s \not\leftrightarrows y$ car il n'existe pas de chemin de y à s.

Proposition. $\leftrightarrows$ est une relation d'équivalence.

Définition. Une composante fortement connexe est une classe d'équivalence de la relation $\leftrightarrows$.

Définition. On appelle graphe quotient de G le graphe $G' = (S', A')$ avec S' les composantes fortement connexes de G et A' l'ensemble des arêtes définies pas $C \xrightarrow{G'} D$ ssi $C \neq D$ et il existe $x, y \in C \times D$ tel que $x \xrightarrow{G} y$.

Proposition.
G' est acyclique.

Démonstration.
Absurde. ■

Exemple 26. *Le graphe quotient du graphe ci-dessus est :*

$$\{s, v, w\} \longrightarrow \{q, t, y\} \longrightarrow \{r\}$$
$$\uparrow \qquad\qquad \uparrow$$
$$\{x, z\} \qquad\qquad \{u\}$$

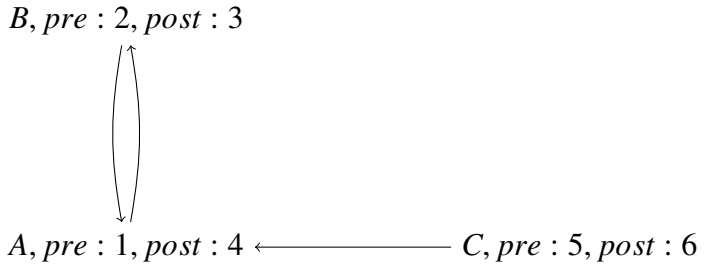
Morale : un graphe est un graphe acyclique de ses composantes fortement connexes.

Algorithme de Kosaraju

Le problème est le suivant :
— Entrée : un graphe orienté $G = (S, A)$;
— Sortie : l'ensemble des composantes fortement connexes de G.
Nous allons donner un algorithme pour résoudre ce problème : l'algorithme de Kosaraju.

Voici l'intuition de l'algorithme. Si on utilise un parcours en profondeur depuis n'importe quel sommet, l'arbre n'est pas forcément une composante fortement connexe. Par contre, si on commence un parcours en profondeur depuis un sommet qui est dans une composante finale (composante puits), alors l'arbre produit est une composante fortement connexe. En utilisant alors le tri topologique, il semblerait qu'il suffit de commencer par un sommet avec $post(u)$ minimal. Malheureusement, cette idée ne fonctionne pas.

Exemple 27. *Voici un exemple de graphe et un parcours en profondeur où $post(B)$ est minimal mais pourtant la composante $\{A, B\}$ n'est pas finale.*



Par contre, la valeur $post(t)$ maximale est atteinte pour un sommet dans une composante initiale. Par exemple, $post(C)$ est maximale et $\{C\}$ est une composante initiale. Mais, bonne nouvelle, une composante initiale est une composante finale dans le graphe inverse.

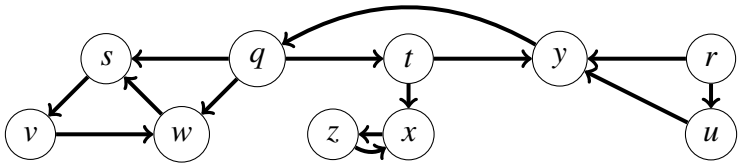
Exemple 28. *Dans le graphe inverse, la composante $\{C\}$ est finale :*

Donc on réalise d'abord un premier parcours profondeur sur le graphe G afin d'obtenir les valeurs $post_1(.)$. Puis en attaquant un parcours en profondeur dans le graphe inverse avec le sommet t de valeur $post_1(t)$ maximale, t est bien dans une composante finale et le premier arbre produit par le deuxième parcours est un composante fortement connexe. On continue le processus.

Définissons d'abord formellement la notion de graphe inverse dans lequel on inverse le sens des flèches.

Définition. Soit $G = (S, A)$ un graphe. On appelle *graphe inverse* de G, le graphe $G^t = (S^t, A^t)$ défini par $S^t = S$ et $A^t = \{(y, x), (x, y) \in A\}$.

Exemple 29. *Par exemple, le graphe inverse du graphe précédent est*

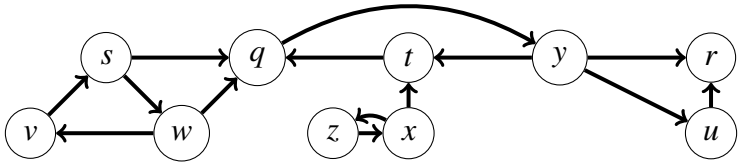


De la discussion précédente, on propose l'algorithme de Kosaraju qui fonctionne en deux étapes :

1. On réalise un premier parcours en profondeur sur G pour obtenir la liste L des sommets s de G triée par ordre décroissant des valeurs $post_1[s]$;
2. On réalise un second parcours en profondeur sur G^t avec la boucle principe qui parcourt les sommets dans l'ordre donné par L.

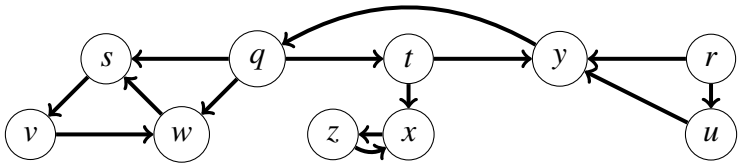
Le lecteur se convaincra que l'on peut implémenter cet algorithme avec une complexité en $O(|S| + |A|)$.

Exemple 30. *Sur le graphe G suivant*



la liste L pourrait être : $[s, v, w, x, z, q, t, y, u, r]$ *(et correspond à un tri topologique si le graphe n'avait pas de cycles).*

Puis on réalise un second parcours sur le graphe inverse



en utilisant la liste L.

Les arbres de ce deuxième parcours sont : $\{s, v, w\}, \{z, x\}, \{q, t, y\}, \{u\}, \{r\}$.

Théorème. *Les conposantes fortement connexes sont les arbres du deuxième parcours de l'algorithme de Kosaraju.*

Démonstration.

Etape 1 : le premier parcours est semblable à un tri topologique sur le graphe quotient de G.

Soit $U \subseteq S$ non vide. On pose :

— $post_1(U) = \max_{s \in U} post_1[s]$, ie l'instant à partir duquel on a fini l'exploration de U.

Lemme.

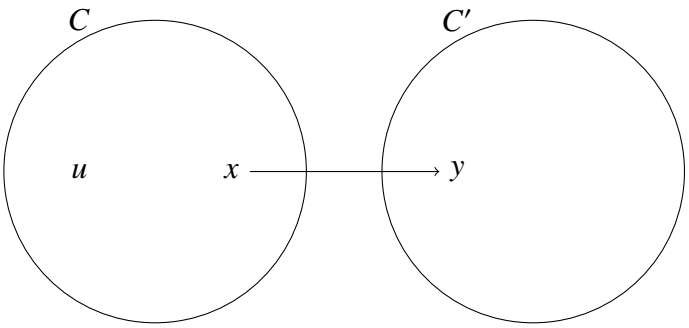
Soit C et C' deux composantes fortement connexes distinctes telles que $C \to C'$ dans le graphe quotient de G (c'est à dire il existe $x \in C$ et $y \in C'$ tels que $x \to y$).

Alors $post_1(C) > post_1(C')$.

Démonstration.

Soit u le premier sommet visité de $C \cup C'$.

- Si $u \in C$, $post_1(C) = post_1[u]$.



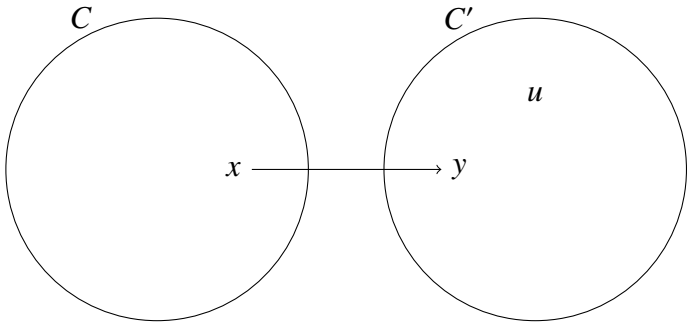
En effet :

— $post_1(C) \geq post_1[u]$ par définition ;

— Pour tout $v \in C$, il existe un chemin de u à v et lors de l'exploration de u, tous les sommets du chemins sont non vus. Donc *explorer*(G, v) a lieu lors de l'appel de *explorer*(G, u). Donc $post_1(v) \leq post_1[u]$. En passant au maximum, $post_1(C) \leq post_1[u]$.

Pour tout $s \in C'$, $post_1[s] < post_1[u]$ car de la même façon, il existe un chemin de u à s avec des sommets non vus, et donc *explorer*(G, s) a lieu lors de l'appel de *explorer*(G, u). Donc $post_1(C') < post_1[u] = post_1(C)$.

- Si par contre $u \in C'$, alors on a $post_1(C') = post_1[u]$.



u n'appelle pas de sommets de C car il n'y a pas d'arcs qui relie C' à C. Ainsi pour tout $s \in C$, $post_1[s] > post_1[u]$. Donc $post_1(C) > post_1(C')$.

■

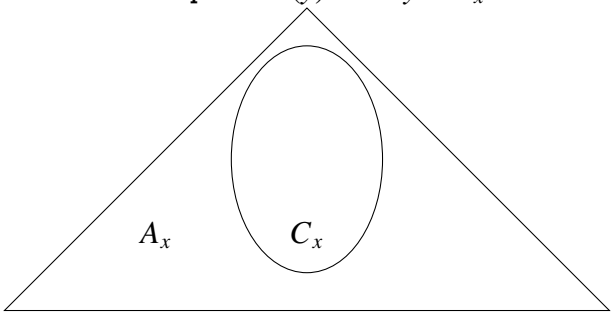
On va montrer par récurrence que pour tout k, la propriété P_k qui dit

> 'les k premiers arbres obtenus pas le deuxième parcours sont des composante fortement connexes de G'.

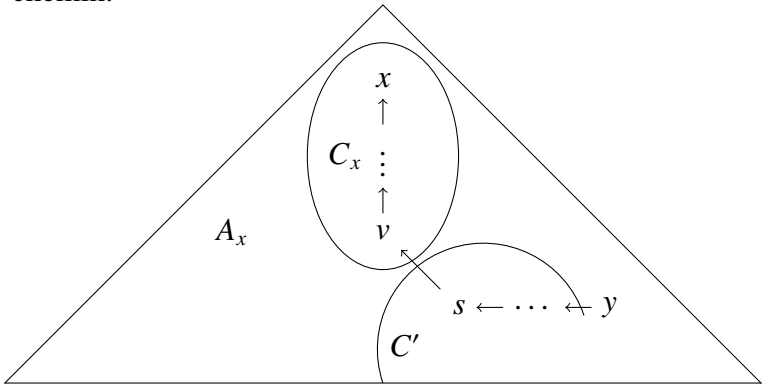
— P_0 est vraie : il n'y a pas d'arbres !

— Soit k strictement plus petit que le nombre de composantes fortement connexes. Supposons P_k et montrons P_{k+1}. Le deuxième parcours choisit le sommet x non vus et tel que $post_1[x]$ maximal. Notons C_x la composante fortement connexe qui contient x et A_x l'arbre issu de x dans le deuxième parcours.

- $\boxed{C_x \subseteq A_x}$ Quand on débute l'exploration de x, tous les sommets de C_x ne sont pas vus. (sinon, par P_k, l'un des sommets serait l'un des arbres déjà calculés, et donc l'arbre serait tout C_x).
  Ainsi, si $y \in C_x$, il y a un chemin de sommets non vus de x à y. La fonction `explorer`(x) appelle tous les `explorer`(y) donc $y \in A_x$.



- $\boxed{A_x \subseteq C_x}$ Par l'absurde. Supposons qu'il existe $y \in A_x \setminus C_x$. Alors `explorer`(x) appelle indirectement `explorer`(y) donc il existe un chemin de x à y dans G'. Dans ce chemin, on note s le premier sommet qui sort de C_x. Soit $v \in C_x$ le prédécesseur de s dans ce chemin.



Dans G^t on a $v \to^{G^t} s$ donc dans G, on a $s \to v$ donc $post_1(C') > post_1(C_x)$ avec C' la composante fortement connexe de s. Comme s est non vu, par P_k, la classe C' est intégralement non vu. Soit z le noeud (non vu) tel que $post_1(z) = post_1(C')$. On a, $post_1(z) > post_1(C_x) = post_1(x)$. Contradiction avec le fait que x est non vu tel que $post_1(x)$ soit maximal. Donc $A_x \subseteq C_x$.

Donc $A_x = C_x$ et P_{k+1} est vraie.

Par récurrence, on a montré P_k pour tout k entre 0 et le nombre de composantes fortement connexes. En particulier, si k est égal au nombre de composantes fortement connexes, on a P_k.

■