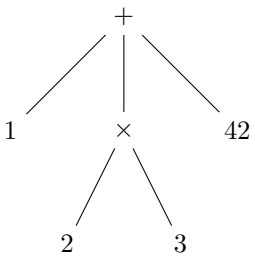
ALGO1 – Parcours en profondeur

François Schwarzentruber

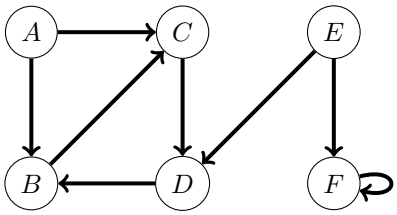
February 7, 2021

1 Parcours en profondeur générique dans un arbre

```
pre : un sommet s d'un arbre
procédure explorer(s)
    prévisite(s)
    pour t successeur de s faire
        explorer(t)
    postvisite(s)
explorer(racine)
```



2 Représentations d'un graphe explicite



Matrice d'adjacence

	$A\downarrow$	$B\downarrow$	$C\downarrow$	$D\downarrow$	$E\downarrow$	$F\downarrow$
$A\rightarrow$	0	0	0	0	0	0
$B\rightarrow$	0	0	0	0	0	0
$C\rightarrow$	0	0	0	0	0	0
$D\rightarrow$	0	0	0	0	0	0
$E\rightarrow$	0	0	0	0	0	0
$F\rightarrow$	0	0	0	0	0	0

Listes d'adjacence

$A\rightarrow$ $B\rightarrow$ $C\rightarrow$ $D\rightarrow$ $E\rightarrow$ $F\rightarrow$

3 Algorithme générique

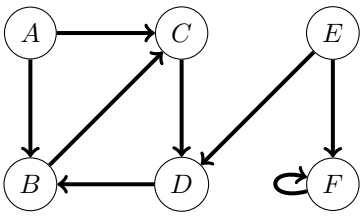
```
pre : un graphe orienté G
procédure parcoursProfondeur(G)
    pour s ∈ S faire
        vu[s] :=faux
    pour s ∈ S faire
        si vu[s] =faux alors
            explorer(s)
```

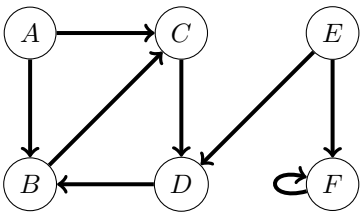
```
pre : un graphe G et un sommet s non visité
procédure explorer(G, s)
    vu[s] :=vrai
    prévisite(s)
    pour t successeur de s faire
    si vu[t] =faux alors
        explorer(G, t)
    postvisite(s)
```

```
pre : un sommet s
procédure prévisite(s)
    pre[s] :=temps
    temps:=temps+1
```

```
pre : un sommet s
procédure postvisite(s)
    post[s] :=temps
    temps:=temps+1
```

Définition 1 On appelle parcours en profondeur de G l'un des parcours de parcoursProfondeur(G).





Théorème 2 Soit $G = (S, A)$ un graphe. Un parcours en profondeur sur G termine.

Théorème 3

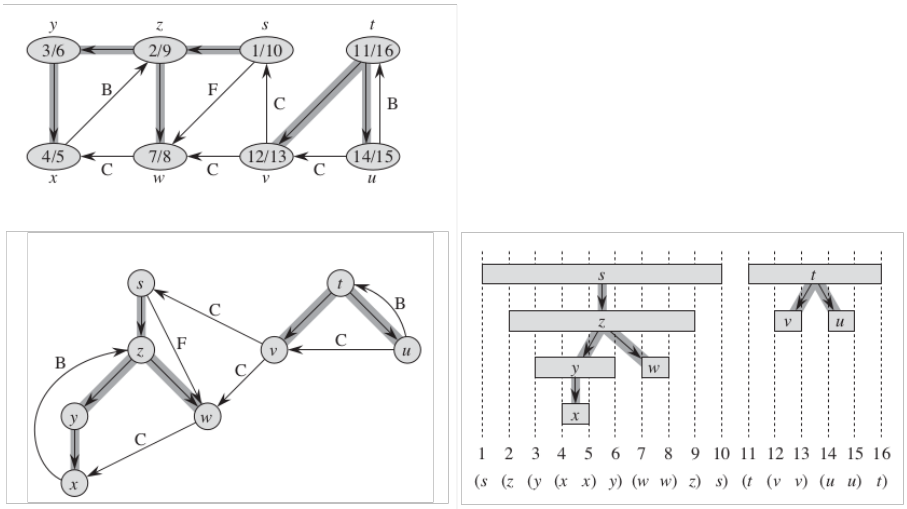
- Si G est représenté par une matrice d'adjacence, un parcours en profondeur sur G coûte $O(|S|^2)$ opérations.
- Si G est représenté par des listes d'adjacence. un parcours en profondeur sur G coûte $O(|S|+|A|)$ opérations.

Démonstration. Regardons la complexité lorsque G est représenté par des listes d'adjacence. La boucle principale du parcours s'exécute $O(|S|)$ fois. Pour tout sommet s, l'appel `explorer`(s) n'est effectivement exécuté qu'une fois. Un appel `explorer`(s) requiert $|\underbrace{\{t \in S, s \to t\}}_{E_s}|$ opérations, propre à l'appel. On a donc une complexité de $O\left(\sum s \in S|E_s|\right) = O(|A|)$ pour tout le calcul dans les appels `explorer`(...).

D'où une complexité totale en $O(|S| + |A|)$. ■

4 Classification des arcs

Notation 4 Pour tout sommet s, on note $[_s]_s$ l'intervalle $[pre[s], post[s]]$.



Proposition 5 (théorème des parenthèses)
Soient s et t deux sommets. Les intervalles $[_s]_s$ et $[_t]_t$ sont soit disjoints, ou alors l'un est inclus dans l'autre.

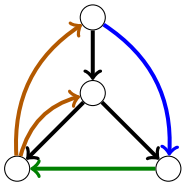
Démonstration. Soit $s \neq t$ tel que $pre[s] < pre[t]$. L'appel explorer(G, s) se fait donc avant explorer(G, t) :

- Soit l'appel de explorer(G, t) se fait lors de l'appel de explorer(G, s) : $[_s[_t]_t]_s$
- Soit l'appel de explorer (G, t) se fait après la fin de l'appel de explorer(G, s) : $[_s]_s[_t]_t$

■

Définition 6 Considérons un parcours en profondeur d'un graphe G.

Un arc $s \to t$ de G est...	quand dans la forêt d'un parcours en profondeur...	intervalles
appelant	`explorer`(s) appelle directement `explorer`(t)	$[_s\ [_t\ ]_t\ ]_s$
avant	t est un ancêtre non père de s	$[_s\ [_t\ ]_t\ ]_s$
arrière	s est un descendant de t	$[_t\ [_s\ ]_s\ ]_t$
transverse	tous les autres cas	$[_t\ ]_t\ [_s\ ]_s$



Lemme 7 Soit $u \neq v$ deux sommets.
`explorer`(v) est appelé alors que l'appel de `explorer`(u) est en cours ssi, au moment où u est découvert, il existe un chemin de u à v composé uniquement de sommets non vus.

Démonstration.

($\Rightarrow$) `explorer`(u) appelle `explorer`(u_1) qui appelle ... qui appelle `explorer`(v) donc pour tout i, $vu[u_i]$ =faux et le chemin $u \to u_1 \to \cdots \to v$ convient.

($\Leftarrow$) Supposons qu'il existe un chemin $u \to u_1 \to \cdots \to v$ composé de sommets non vus au moment où u est découvert.

Soit i le plus petit indice tel que `explorer`(u_i) soit non appelé. Pendant l'appel `explorer`(u_{i-1}), $vu[u_i]$ =faux et $u_{i-1} \to u_i$ existe donc `explorer`(u_i) est appelé. Contradiction.

■

5 Détection de cycles

Définition 8 Un cycle est un chemin de longueur non nulle $v_0 \to v_1 \to \dots v_k \to v_0$, avec $k \geq 1$.

Définition 9 (Problème de détection de cycles)

- Entrée : un graphe G ;
- Sortie : oui, ssi le graphe G possède un cycle.

Proposition 10 Un graphe orienté contient un cycle ssi un parcours en profondeur possède un arc arrière.

Démonstration.

($\Leftarrow$) Si $u \to v$ est un arc arrière, alors explorer(G, v) appelle explorer(G, v_1), qui appelle explorer(G, v_2), ... qui a appelé explorer(G, u), qui a testé $vu[v] = true$ car $u \to v$. Donc le graphe contient le cycle suivant $v \to v_1 \to v_2 \to \dots \to u \to v$.

($\Rightarrow$) Réciproquement, supposons qu'il y a un cycle $v_0 \to \dots \to v_k \to v_0$. Considérons un parcours en profondeur et montrons qu'il y a un arc arrière. Soit i l'indice tel que v_i soit le premier sommet exploré parmi les sommets du cycle. Il existe un chemin de v_i à v_{i-1} composé de sommets non vus.

D'après le lemme 7, `explorer`(v_{i-1}) est appelé alors que l'appel `explorer`(v_i) est encore en cours. Quand v_{i-1} est exploré, on a $vu[v_i]$ =vrai donc $v_{i-1} \to v_i$ est un arc arrière.

■

6 Tri topologique

Exemple 11 Le savant cosinus.

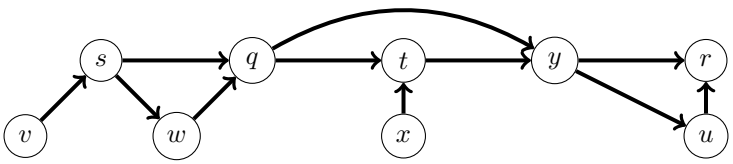
Définition 12 (problème du tri topologique)

- Entrée : un graphe G acyclique ;
- Sortie : une extension linéaire $\leqslant_\ell$, i.e. $\leqslant_\ell$ est un ordre total tel que si $x \to y$ dans G implique $x \leqslant_\ell y$.

Supposons $(s \to t)$. Dans un parcours, il n'y a pas d'arcs arrières, car G acyclique, voici les possibilités :

- $s \to t$ est un arc appelant/avant : $[_s\ [_t\ ]_t\ ]_s$
- $s \to t$ est un arc transverse : $[_t\ ]_t\ [_s\ ]_s$

Dans tous les cas, $post[s] > post[t]$.



valeurs $post[.]$ élevées — valeurs $post[.]$ faibles

Algorithme du tri topologique

- réaliser un parcours en profondeur.
- lire l'extension linéaire $\leqslant_\ell$ comme l'ordre décroissant des valeurs de $post[s]$: $x \leqslant_\ell y$ ssi $post[x] \geq post[y]$

Théorème 13 L'algorithme du tri topologique est correct.

Démonstration. Car il n'y a pas d'arcs arrière. ■

7 Composantes fortement connexes

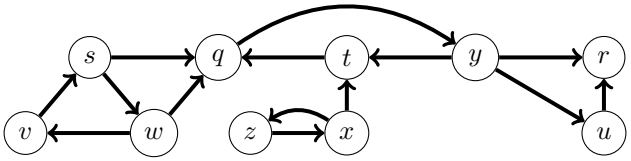
7.1 Définitions

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.

Proposition 14 $\leftrightarrows$ est une relation d'équivalence.

Définition 15 (composante fortement connexe) Une composante fortement connexe est une classe d'équivalence de $\leftrightarrows$.

Exemple 16



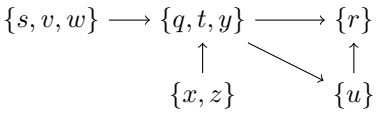
Définition 17 (graphe quotient) On appelle graphe quotient de G le graphe $G_{|\leftrightarrows} = (S', A')$ où

- S' est l'ensemble des composantes fortement connexes de G et
- A' l'ensemble des arêtes (C, D) tel que $C \neq D$ et il existe $x, y \in CD$ avec $x \overset{G}{\to} y$.

Proposition 18 $G_{|\leftrightarrows}$ est acyclique.

Démonstration. Absurde. ■

Exemple 1 Le graphe quotient du graphe ci-dessus est :



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

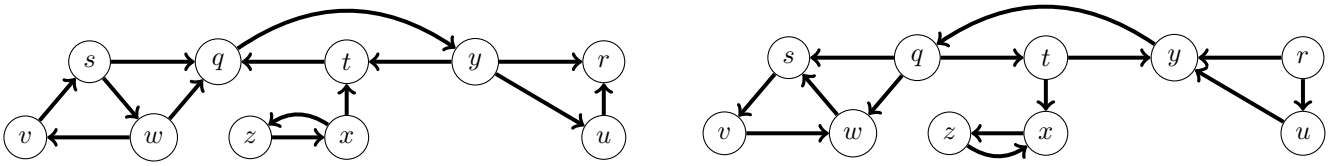
7.2 Algorithme de Kosaraju

Définition 19 (Problème du calcul des composantes fortement connexes)

- Entrée : un graphe orienté $G = (S, A)$;
- Sortie : l'ensemble des composantes fortement connexes de G.

Définition 20 (graphe inverse) 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 21



Algorithme de Kosaraju

1. réaliser un premier parcours en profondeur sur G

   $\Rightarrow$ obtenir la liste L les sommets de G triée par ordre décroissant des valeurs $post_1[.]$;
2. réaliser un second parcours en profondeur sur G^t avec la boucle principal parcourant les sommets dans l'ordre donné par L.

Théorème 22 On peut implémenter l'algorithme de Kosaraju en $O(|S| + |A|)$.

Théorème 23 Les conposantes fortement connexes sont les arbres du deuxième parcours de l'algorithme de Kosaraju.

Démonstration.
Étape 1: le 1^{er} parcours ressemble à un tri topologique sur le graphe quotient de G.
Soit $U \subseteq S$ non vide. On pose :

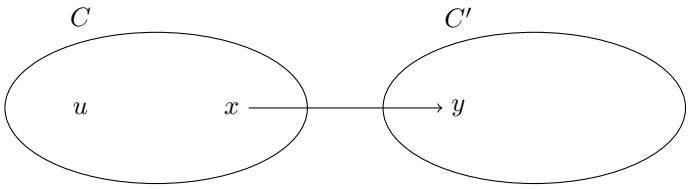
- $post_1(U) = \max_{s \in U} post_1[s]$, ie l'instant à partir duquel on a fini l'exploration de U.

Lemme 24 Soit C et C' deux composantes fortement connexes.

$$C \to^{G} C' \qquad \text{implique} \qquad post_1(C) > post_1(C').$$

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

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

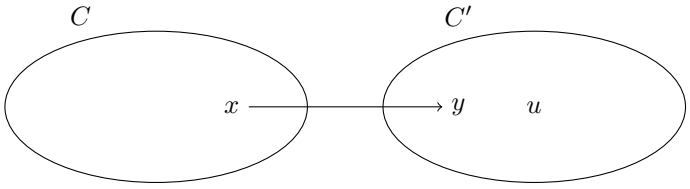


  En effet :

  - $post_1(C) \geq post_1[u]$ 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 $post_1(C) > post_1(C')$.

■

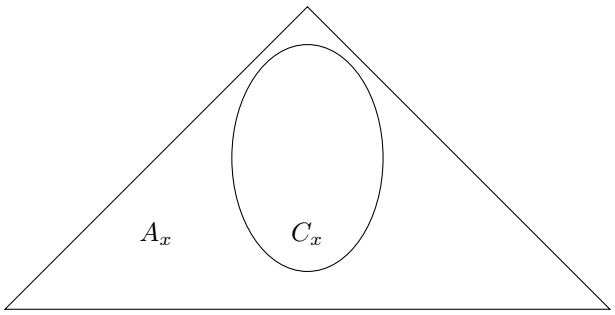
Étape 2: les arbres du second parcours sont les CFC.
Montrons par récurrence que pour tout k, la propriété P_k qui dit

> les k premiers arbres obtenus pas le $2^{\text{ème}}$ parcours sont des composantes fortement connexes de G.

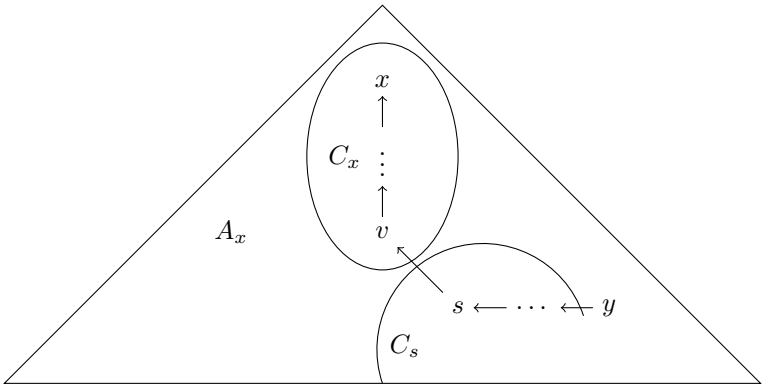
- P_0 est vraie : il n'y a pas d'arbres !

- Soit k strictement plus petit que le nombre de composantes fortement connexes. Supposons P_k et montrons P_{k+1}. Le second parcours choisit le sommet x non vus avec $post_1[x]$ maximal. Notons C_x la CFC contenant x et A_x l'arbre issu de x dans le second parcours.

  - $\boxed{C_x \subseteq A_x}$ Quand l'exploration de x commence, aucun sommet de C_x n'est vu (sinon, par P_k, la CFC C_x aurait déjà été traité).
    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^t. 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, on a $s \to v$ donc $post_1(C_s) > post_1(C_x)$ avec C_s la CFC de s. Comme s est non vu, par P_k, la classe C_s est intégralement non vu. Soit z le nœud (non vu) tel que $post_1(z) = post_1(C_s)$. 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 $A_x \subseteq C_x$.

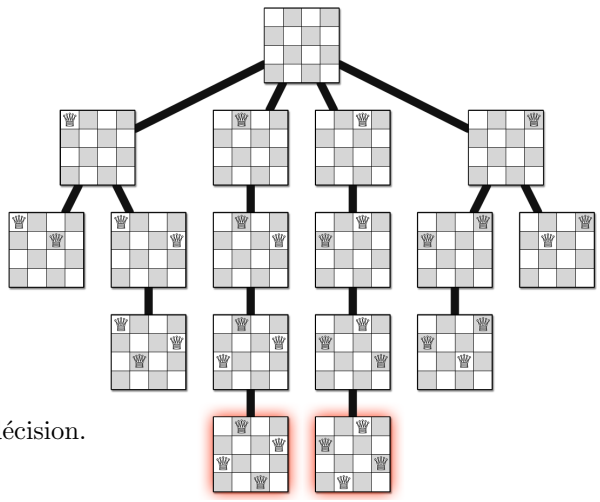
Donc $A_x = C_x$ et P_{k+1} est vraie.

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. ■

8 Backtracking : force brute

8.1 Recherche d'une solution

```
pre : une séquence de choix s dans un arbre de décision
post : une solution qui étend s s'il en existe une,
       ou impossible
procédure solution(s)
    si s est une solution retourner s
    pour tout choix c pour étendre s faire
        s' := solution(s.c)
        si s' ≠ impossible alors s'
    retourner impossible
solution(ϵ)
```



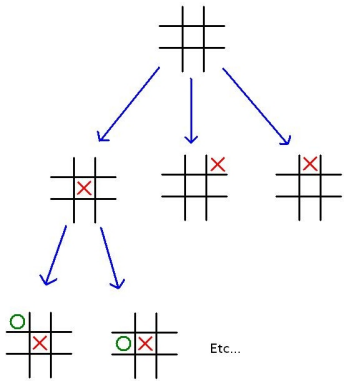
Théorème 25 Soit a est l'arité et h la hauteur de l'arbre de décision.

- La complexité temporelle est en $O(a^h)$.
- La complexité spatiale est en $O(h)$.

8.2 Jeu à deux joueurs

pre : une position s d'un arbre de jeu, un joueur i
post : vrai si le joueur i qui joue en s a une stratégie gagnante
procédure strategieGagnante?(s, i)
    si s est une position gagnante pour le joueur i **retourner** vrai
    si s est une position perdante pour le joueur i **retourner** faux
    pour tout coup de $s \to t$ **faire**
    si strategieGagnante?(t, $\neg i$) est faux **alors**
        retourner vrai
    retourner faux

strategieGagnante?( , $\times$)



Notes bibliographiques

Selon [CLRS09], on utilise le parcours en profondeur depuis 1950 en intelligence artificielle. C'est dans [HT73] que Hopcroft et Tarjan font l'éloge du parcours en profondeur et l'utilisation de la représentation d'un graphe avec des listes d'adjacence. Knuth est le premier à donner un algorithme linéaire pour le tri topologique [Knu68]. Tarjan [Tar72] donne un algorithme en temps linéaire pour le calcul des composantes fortement connexe, mais son algorithme est difficile à expliquer. Une légende raconte que Kosaraju a inventé son algorithme présenté ici, car plus simple à expliquer et enseigner, tout en ayant une implémentation en temps linéaire. L'algorithme de Kosaraju est présenté dans [CLRS09] et [DPV08].

References

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[DPV08] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani. *Algorithms*. McGraw-Hill, 2008.

[HT73] John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation [H] (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973.

[Knu68] Donald Ervin Knuth. *The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley, 1997 (first edition in 1968).

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.