Parcours en profondeur et recherche de circuits.

On s'intéresse dans ce cours au problème suivant: étant donné un graphe G orienté, dont on suppose que l'on connait un sommet s duquel tous les autres sont accessibles, on cherche à savoir si G comporte un circuit.

NB: *A quoi ça sert?Par exemple, si vous prenez un graphe dont les sommets représentent des tâches et les arcs des contraintes de précédence: un arc (x,y) signifie que la tâche x doit être terminée avant que y ne puisse commencer, détecter un circuit permet de détecter une contradiction dans les contraintes.*

Détection de circuit: un algorithme naif

On a vu qu'un parcours à partir d'un sommet x permettait de répondre à la question suivante:

Quels sont les sommets accessibles à partir de x par un chemin?

Pour savoir si un graphe comporte un circuit, on peut se poser la question suivante pour chaque arc (x,y) de G: *existe-t-il un circuit passant par cet arc?*

Or un circuit passant par un arc (x,y) est composé de cet arc et d'un chemin de y à x. autrement dit, la question : *existe-t-il un circuit passant par (x,y)?* peut-être ramenée à la question: *x est-il accessible à partir de y?*

On voit ici apparaître l'ébauche d'un algorithme qui pourrait s'écrire ainsi:

pour tout arc (x,y) de G

    Effectuer un parcours à partir du sommet y.

    Si x est visité lors du parcours, alors G comporte un circuit.

Analysons la complexité dans le pire des cas de cette détection de circuit: la boucle pour comporte au maximum m (nombre d'arcs) itérations. Pour chacune on effectue un parcours à partir d'un sommet (complexité O(m+n)).

Par conséquent, la complexité de cet algorithme est en $O(m^2+mn)$

Nous allons voir par la suite que l'on peut faire beaucoup mieux.

Parcours en profondeur récursif

Considérons l'algorithme de parcours en profondeur récursif suivant, où G est un graphe orienté, statut est un tableau de sommet qui conserve l'état des sommets (-1 pour libre, 0 pour ouvert, 1 pour fermé)

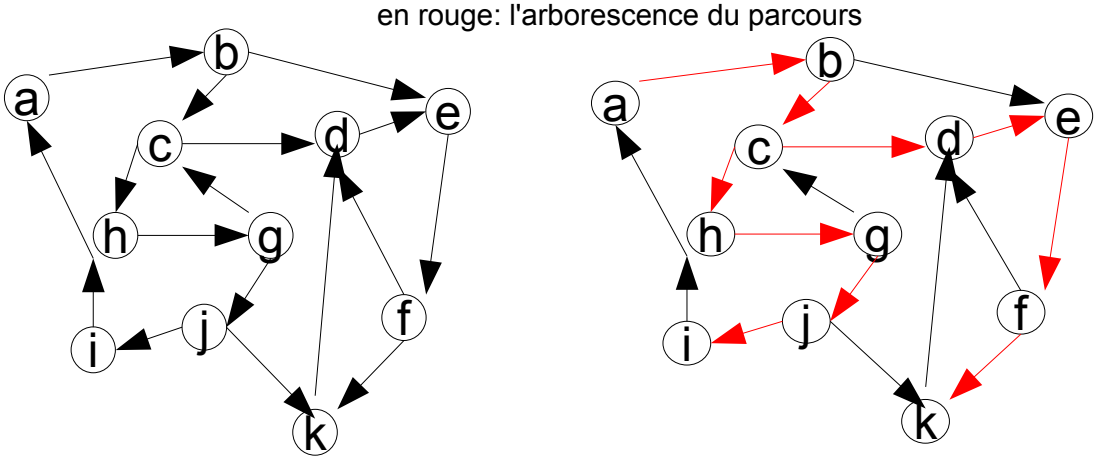
Initialement, tous les sommets sont libres, sauf s sommet de départ du parcours.

```
profondeur (graphe G, tableau statut, sommet x)
{pour tout successeur y de x
    {si y est libre statut[y]=0 profondeur (G,statut,y)}
statut[x]=1
}
```

Cours du 14 novembre 2007

Application à un graphe :

Parcours à partir de a:
abcdefkhgji
en rouge: l'arborescence du parcours



Analyse de la complexité

Supposons que le graphe est représenté à l'aide de listes d'adjacence.

On veut analyser la complexité de l'appel à profondeur (G, statut,s). Pour cela, on remarque tout d'abord que pour chaque sommet x, il ne peut y avoir qu'un seul appel récursif à profondeur(G,statut,x). En effet, lorsqu'un appel récursif a lieu, le sommet x passe de libre à ouvert, et il n'y a jamais dans l'algorithme de changement d'état d'ouvert ou de fermé vers libre.

A l'intérieur de chaque appel récursif, il y a une boucle qui parcourt les successeurs de x et qui réalise un test à chaque fois et éventuellement une affectation.

La complexité de ces opérations pour un appel à profondeur(G, statut,x) est donc proportionnelle au degré extérieur de x d+(x).

La complexité totale s'exprime donc comme le coût de appels de fonction (O(n)) plus le coût des autres opérations qui est proportionnel à la somme pour tous les sommets des degrés extérieurs, donc au nombre d'arcs m du graphe. D'où une complexité en O(n+m).

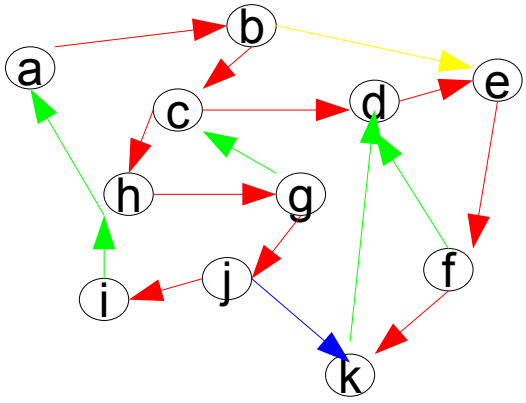
Typologie des arcs.

Supposons que l'on a effectué un parcours en profondeur d'un graphe orienté G.

Ce parcours va induire une partition des arcs de G en plusieurs types:

1) les arcs de l'arborescence du parcours A : les arcs (x,y) tels que profondeur(G,statut,y) est appelé dans la boucle de profondeur(G,statut,x).
2) les arcs dits « avant », qui sont de la forme (x,y) avec y descendant de x dans A
3) les arcs dits « arrière » qui sont de la forme (x,y) avec x descendant de y dans A
4) les autres arcs de G sont appelés arcs transverses.

La figure suivante indique la typologie des arcs sur l'exemple précédent:



Typologie des arcs: en rouge arcs de l'arborescence du parcours
en jaune arcs avant, en bleu arcs transverses et en vert arcs arrière

Propriétés du parcours en profondeur

On suppose ici qu'un parcours en profondeur du graphe a été effectué, et qu'il a conduit à définir une partition des arcs de G selon la typologie précédent.

Propriété 1:

Si (x,y) est un arc avant ou un arc de l'arborescence, alors l'appel à profondeur(G,statut,y) a commencé après l'appel à profondeur(G,statut,x) et s'est terminé avant la fin de cet appel. On dit que l'appel à profondeur (G,statut,y) est imbriqué dans l'appel à profondeur(G,statut,y).

Réciproquement, si (x,y) est un arc de G et que l'appel à profondeur(G,statut,y) est imbriqué dans l'appel à profondeur(G,statut,x), alors (x,y) est un arc avant ou un arc de l'arborescence.

Preuve.

Supposons que (x,y) est un arc avant. Cela implique qu'il existe un chemin dans l'arborescence du parcours de x à y. Soit x_0=x,x_1,...x_k=y ce chemin. Par définition de l'arborescence du parcours, pour tout i, x_i a été ouvert à partir de x_{i-1}, c'est à dire que l'appel à profondeur(G,statut,x_i) a été fait pendant l'appel à profondeur(G,statut,x_{i-1}). Or, les appels récursifs étant nécessairement imbriqués, on voit que dans ce cas profondeur(G,statut,x_i) se termine avant profondeur(G,statut,x_{i-1}). Par conséquent profondeur(G,statut,y) commence après et se termine avant profondeur(G,statut,x).

Réciproquement, Supposons l'appel à profondeur(G,statut,y) est imbriqué dans profondeur(G,statut,x). profondeur(G,statut,y) a été appelé à partir d'une exécution de profondeur(G,statut,y_1), qui a lui même été appelé par une exécution profondeur(G,statut,y_2)....jusqu'à l'appel initial profondeur(G,statut,y_k) avec y_k=x. Par définition de l'arborescence du parcours, yi est un successeur de yi+1 dans cette arborescence. On a donc bien un chemin de y_k=x à y dans A.

Propriété 2

Si (x,y) est un arc arrière, alors profondeur(G,statut,x) est imbriqué dans profondeur(G,statut,y) . Réciproquement, si (x,y) est un arc et que profondeur(G,statut,x) est imbriqué dans profondeur(G,statut,y) alors (x,y) est un arc arrière.

Preuve: le raisonnement est identique.

Cours du 14 novembre 2007

Propriété 3

Si (x,y) est un arc transverse alors profondeur(G,statut,y) s'est terminé avant le début de l'appel à profondeur(G,statut,x).

Preuve :

En vertu des deux propriétés précédentes, comme (x,y) n'est ni avant ni arrière ni un arc de l'arborescence, il est impossible que les deux appels à profondeur(G,statut,y) et profondeur(G,statut,x) soient imbriqués.

Soit z le sommet predecesseur de y dans l'arborescence (c'est à dire le sommet tel que profondeur(G,statut,y) a été appelé dans la boucle de profondeur(G,statut,z)).

Si z n'existe pas, c'est que y=s. Or tous les appels récursifs sont nécessairement imbriqués dans profondeur(G,statut,s) qui est le premier. D'où une contradiction.

Donc z existe. Et par construction, pendant la boucle de profondeur(G,statut,z), on a un appel récursif à profondeur(G,statut,y) parce que y est libre.

Lorsque profondeur(G,statut,x) se termine, par définition de l'algorithme, aucun de ses successeurs ne peut plus être libre.

Par conséquent, comme y est un successeur de x, il est impossible que profondeur(G,statut,x) soit terminé lorsque profondeur(G,statut,y) est lancé. Comme les deux appels ne sont pas imbriqués, c'est que profondeur(G,statut,x) commence après la fin de profondeur(G,statut,y).

Circuits et typologie des arcs.

Des propriétés précédentes on déduit une propriété fondamentale:

Propriété 4

Le graphe G comporte un circuit si et seulement si, après un parcours en profondeur, G possède au moins un arc arrière.

Preuve :

Il est assez évident que si G possède un arc arrière (x,y) alors il comporte un circuit. En effet, comme x est un descendant de y dans l'arborescence du parcours, on a un chemin de y à x dans G, qui forme un circuit lorsqu'on le complète par l'arc (x,y).

Réciproquement, supposons que G n'a aucun arc arrière lors d'un parcours en profondeur. Cela implique que tous ses arcs sont soit des arcs de l'arborescence, soit avant soit transverse.

D'après les propriétés 1 et 3, un arc (x,y) qui n'est pas arrière vérifie que lors de l'algorithme, profondeur(G,statut,y) est terminé avant que profondeur(G,statut,x) ne se termine. Dans le premier cas les deux appels sont imbriqués dans le second (transverse) il ne le sont pas.

Supposons maintenant que G possède un circuit $x_1,...x_k$. On sait donc que :

profondeur(G,statut,x_k) se termine avant profondeur(G,statut,x_{k-1}),... qui se termine avant profondeur(G,statut, x_1). Or le circuit comporte un arc (x_k,x_1), qui implique que profondeur(G,statut,x_1) se termine avant profondeur(G,statut,x_k). D'où une contradiction.

Algorithme de détection de circuit

Cette propriété essentielle permet d'utiliser le parcours en profondeur pour détecter la présence d'un circuit dans un graphe. En effet, pour détecter un circuit, il suffit de détecter la présence d'un arc

arrière (X,Y). La propriété 2 nous dit que lors du parcours, l'appel à profondeur(G,statut,X) est imbriqué dans l'appel à profondeur(G,statut,Y). Or au cours de l'appel à profondeur(G,statut,X), on va parcourir tous les successeurs de X. le sommet Y s'y trouvera nécessairement. On doit donc trouver un moyen de savoir, à ce stade, si l'appel à profondeur(G,statut,Y) est commencé et pas encore terminé.

Or si l'on observe le tableau statut, pour tout sommet x, statut[x] vaut -1 lorsque l'appel à profondeur(G,statut,x) n'est pas commençé, 0 lorsqu'il commence et 1 lorsqu'il se termine. Par conséquent il contient toute l'information nécessaire.

On en déduit l'algorithme suivant, sous forme d'une fonction qui retourne 1 s'il y a un circuit dans le sous graphe des sommets accessibles à partir de s 0 sinon. L'initialisation est la même que précédemment (tous sommets libres sauf s ouvert).

Entier circuit (graphe G, tableau statut, sommet x)

{entier r

pour tout successeur y de x

{si y est libre statut[y]=0 r=circuit (G,statut,y) si r=1 retourner 1

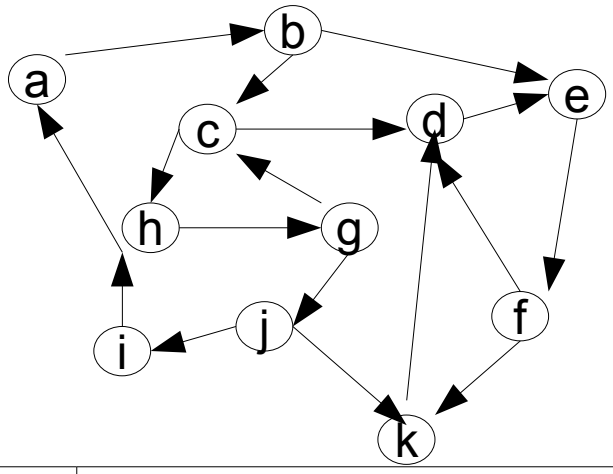
sinon si statut[y]=0 retourner 1}

statut[x]=1

retourner 0

}

Pour l'exemple:



	statut[a]=0
Circuit(G,statut,a)	b libre: statut[b]=0
circuit(G,statut,b)	c libre: statut[c]=0
circuit(G,statut,c)	d libre: statut[d]=0
circuit(G,statut,d)	e libre: statut[e]=0
circuit(G,statut,e)	f libre: statut[f]=0
circuit(G,statut,f)	d succ de f pas libre et statut[d]=0 retourne 1
circuit(G,statut,e)=1	
circuit(G,statut,d)=1	
circuit(G,statut,c)=1	
circuit(G,statut,b)=1	
Circuit(G,statut,a)=1	

Notez que la complexité de cet algorithme est inchangée par rapport au parcours précédent: on ajoute juste un test et une affectation par successeur au maximum. O(n+m). On remarque que c'st considérablement mieux que l'algorithme naif.

Cours du 14 novembre 2007

Graphes sans circuit

Nous allons voir que cet algorithme de détection de circuit peut être également utilisé pour travailler sur les graphes sans circuit.

Soit G un graphe sans circuit. On suppose que ses sommets représentent des tâches à faire et ses arcs des relations de précédence. On suppose également qu'une personne doit faire toutes ces tâches, et qu'elle ne peut pas en faire plusieurs à la fois. Elle se demande donc *dans quel ordre* effectuer les tâches sans entrer en contradiction avec les contraintes de précédence.

Numérotation topologique

Définition:

*On appelle **numérotation topologique** d'un graphe de n sommets une bijection qui à chaque sommet x associe un numéro N(x) dans {1,..n} de sorte que tous les numéros soient distincts et que:*

pour tout arc (x,y) de G, N(x)<N(y)

On peut montrer qu'un graphe sans circuit possède toujours une numérotation topologique. En effet:

Propriété 4

Soit G un graphe sans circuit. Alors G possède un sommet sans successeur.

Preuve

Si chaque sommet avait un successeur, alors en partant d'un sommet quelquonque x1, on pourrait trouver un successeur x2, qui lui-même aurait un sucesseur x3,....

Comme le graphe ne comporte pas de circuit, on ne pourrait pas avoir xi =xj. Comme il n'y a qu'un nombre fini de sommets, c'est impossible.

Propriété 5

Soit G un graphe sans circuit. Alors G possède une numérotation topologique.

Preuve

Raisonnons par récurrence sur le nombre de sommets de G. lorsque n=1, le graphe a un sommet s, pas d'arcs, et une numérotation topologique: N(s)=1.

Supposons que la propriété est vraie pour les graphes de n-1 sommets. Soit G un graphe sans circuit de n sommets, et soit z un sommet sans successeur de G (qui existe d'après la propriété 4). Le graphe G' construit en retirant le sommet x de G ainsi que tous ses arcs entrants est un graphe de n-1 sommets sans circuit. Par hypothèse de récurrence, il possède une numérotation topologique N, qui associe à chaque sommet un numéro entre 1 et n-1.

Etendons cette numérotation à G en posant N(z)=n.

Considérons maintenant un arc (x,y) de G. Si cet arc est dans G', la numérotation N vérifie N(x)<N(y). Si cet arc n'est pas dans G', c'est que y=z. Et comme N(x)<n=N(z), la propriété est vérifiée.

Algorithme naif pour la numérotation topologique.

La propriété précedente conduit naturellement à définir un algorithme pour numéroter un graphe sans circuit, qui peut s'exprimer ainsi schématiquement:

num=n

tant que le graphe G n'est pas vide

choisir un sommet x sans successeur de G

Poser N(x)=num

num=num-1

retirer x de G.

Supposons que le graphe soit représenté à l'aide de listes d'adjacence. Chercher un sommet sans successeur consiste à parcourir la liste des sommets jusqu'à en trouver un dont la liste d'adjacence est vide. Retirer un sommet du graphe consiste à parcourir le graphe en retirant systématiquement x de la liste d'adjacence des sommets dont il est le successeur.

Autrement dit, une itération de la boucle tant que a une complexité O(n+m). A chaque itération on a un sommet de moins et quelques arcs en moins également dans G. notons $x_1,..x_n$ les sommets successivement retirés dans les n itérations de la boucle.

La première itération a une complexité n+m

la seconde une complexité $n-1+m-d^-(x_1)$

la troisième $n-2+m-d^-(x_1)-d^-(x_2)$...

la kième $n-k+1+m-d^-(x_1)...-d^-(x_{k-1})$.

D'où une complexité, somme de précédentes de:

$n(n-1)/2+nm-(n-1)d^-(x_1)-(n-2)d^-(x_2)-...-d(x_{n-1})=(n-1)/2+d^-(x_1)+2d^-(x_2)+..+(n-1)d(x_{n-1})$

On peut bien sur majorer cette expression par n(n-1)/2+nm.

En raffinant un peu, pour évaluer le pire des cas on doit regarder quand l'expression

$d^-(x_1)+2d^-(x_2)+..+(n-1)d(x_{n-1})$ est la plus grande possible, sachant que $d^-(x_k)\leq n-k$ et que la somme de ces degrés est égale à m. On voit qu'on obtient une somme sur k de k(n-k)

(n(n-1)(2n-1))/6 = somme des carrés des entiers jusqu'à n-1,

n(n-1)/2= somme des entiers jusqu'à n-1

Donc somme des k(n-k) jusqu'à n-1= $n(n-1)^2/2-(n(n-1)(2n-1))/6 =n(n-1)(n-2)/6$

En remplaçant dans la première expression, on voit que l'ordre de grandeur de la complexité est $O(n^3)$

Parcours en profondeur et numérotation topologique

Revenons sur le parcours en profondeur étudié précédemment, en supposant que G est sans circuit. Observons que lorsque l'on a un arc (x,y), il est soit dans l'arborescence, soit avant soit transverse, et que dans ces trois cas l'appel à parcours(G,statut,y) se termine avant la fin de l'appel à parcours(G,statut,x).

Cours du 14 novembre 2007

Propriété 6:

Considérons les sommets x dans l'ordre inverse de terminaison de parcours(G,statut,x): si parcours(G,statut,x) est la kième terminaison de l'algorithme, posons N(x)=n-k+1.

Alors N(x) est une numérotation topologique du graphe.

Preuve:

Soit (x,y) un arc, avec cette définition de la numérotation, N(x)>N(y) équivaut à parcours(G,statut,x) se termine avant parcours(G,statut,y), et nous avons vu que c'était impossible pour un arc du graphe. Par conséquent N(x)<N(y).

Algorithme de calcul de la numérotation topologique d'un graphe sans circuit.

On définit ici un algorithme qui calcule un tableau N en utilisant une variable globale ter, initialisée à n avant le premier appel:

Initialement, tous les sommets sont libres, sauf s sommet de départ du parcours.

topologique (graphe G, tableau statut,N,sommet x)

{pour tout successeur y de x

{si y est libre statut[y]=0 topologique (G,statut,N,y)}

statut[x]=1

N[x]=ter--

}

A nouveau, on observe que la complexité de cet algorithme est la même que celle du parcours en profondeur: on a juste ajouté deux affectations par appel. D'où un O(n+m). L'exemple suivant montre la numérotation topologique calculée par l'algorithme. En rouge l'arborescence du parcours.