Elèves : KALOUGINE Anne, TRAN Benoît, LAVENANT Hugo en seconde en 2007-2008

Tolérance aux ruptures sur les canaux de communication d'un réseau

année 2008

I Généralités relativement aux réseaux étudiés

On symbolisera ici un réseau d'ordinateurs par un *graphe* : chaque ordinateur sera représenté par un petit cercle qu'on nommera *nœud*. Ces nœuds s'envoient des informations par des canaux de communications à sens unique représentés par des segments fléchés (appelés *arêtes*).

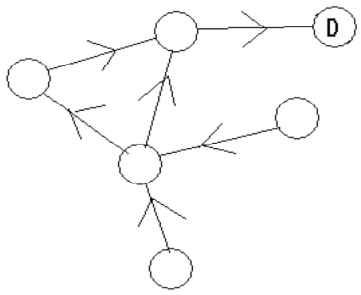
Voici quatre **hypothèses** que nous ferons relativement aux réseaux étudiés :

H0 On ne considèrera que des réseaux où il y ait un *nombre fini de nœuds*.
H1 Les graphes seront *connexes*, c'est à dire en un seul morceau ; autrement dit, il n'y aura pas d'ordinateurs ou de groupes d'ordinateurs isolés.
H2 Les graphes ne possèderont *pas de cycles orientés* : autrement dit, l'information provenant d'un nœud ne pourra lui revenir.
H3 Un nœud D joue, dans nos réseaux, un rôle particulier (ce sera, par exemple, l'ordinateur « tête de réseau » ou « serveur ») : le nœud *D ne fait que recevoir des informations* (toutes les arêtes auxquelles il est lié pointent vers lui).

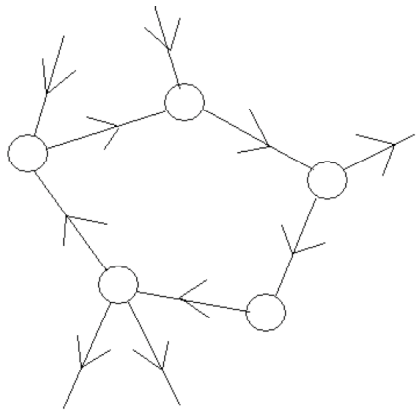
Mettons en place un **vocabulaire** qui nous sera utile :

- Un graphe est dit *D-ciblé* si chacun des nœuds peut envoyer des informations au nœud D.
- Un nœud pouvant envoyer des informations à D est dit *connecté à D*.

Voici un exemple de graphe D-ciblé :

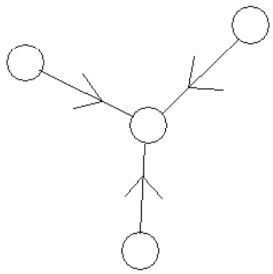


Voici un exemple d'une partie d'un graphe comportant un cycle orienté :



- Un *puits* est un nœud qui ne fait que recevoir des informations (toutes les arêtes qui y mènent sont orientées vers lui).

Voici un exemple de puits :



Démontrons d'abord un résultat important, essentiel pour la suite.

Théorème 1 :
Soit G un graphe connexe et sans cycle orienté.
Le graphe G est D-ciblé si et seulement si G ne possède pas d'autres puits que D.

Démonstration :

1) Prouvons que, si G est D-ciblé, alors G ne possède pas d'autre puits que D :
Si G possède un puits autre que D, alors ce nœud ne pourra pas envoyer d'informations (en particulier à D), donc le graphe ne sera pas D-ciblé.

2) Prouvons que, si G ne possède pas d'autres puits que D, alors G est D-ciblé :
On considère G sans puits autre que D
Soit N_1 un nœud de G : N_1 n'est pas un puits donc N_1 est connecté à au moins un nœud N_2.
Puisque le nœud N_2 n'est pas un puits donc il est connecté à au moins un nœud N_3, différent de N_1, G ne possédant pas de cycle orienté (si $N_1=N_3$ alors N_1 reçoit des informations de lui-même, donc le graphe possède un cycle orienté, ce qui est impossible).
Ce nœud N_3 n'est pas un puits donc il est connecté à au moins un nœud N_4, différent de N_2 et de N_1 (pour ma même raison que la fois précédente : G ne possédant pas de cycle orienté).

Le nombre de nœuds étant fini, l'itération du procédé s'achèvera nécessairement sur D.
Donc G ne possède pas d'autre puits que D, ce qui implique G est D-ciblé.

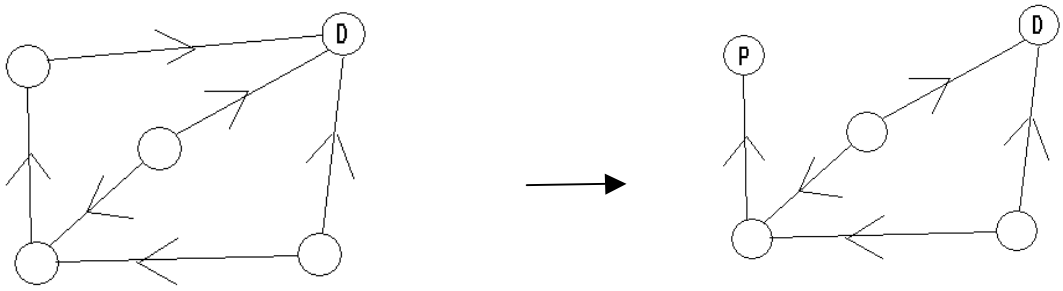
II Le Problème

Considérons un graphe respectant les hypothèses énoncées précédemment et supprimons une de ses connexions (nous nommerons cette opération « cassure »).

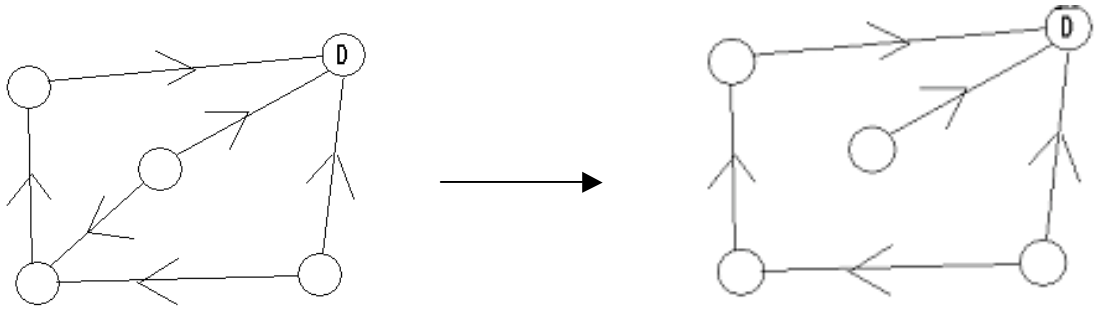
H5 : nous ne traiterons que les cas où la cassure n'invalide pas l'hypothèse H1 de connectivité.

Le graphe obtenu n'est plus nécessairement D-ciblé : par exemple, dans le cas ci-dessous, on observe la création d'un puits autre que D (sur ce dessin, P désigne un puits). Cependant, ce n'est pas toujours le cas, et le graphe peut rester D-ciblé.

- Voici un exemple de graphe qui ne reste pas D-ciblé après cassure :



- Voici un exemple de graphe restant D-ciblé après cassure :

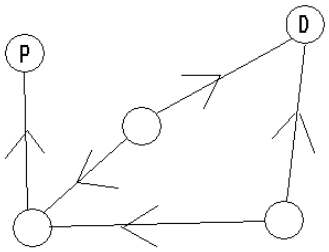


Le problème étudié dans ce texte consiste à savoir s'il existe un algorithme qui permette, après une cassure, de rendre le graphe de nouveau D-ciblé (s'il ne l'est plus) avec les règles suivantes : dans l'algorithme, à chaque étape, chaque nœud est acteur, la seule opération qui lui soit permise étant le changement de sens d'une ou plusieurs des arêtes dont il est un sommet. De plus, pour agir, un nœud n'aura qu'une visibilité limitée, et ne connaîtra que l'état des arêtes dont il est un sommet.

Nous allons démontrer que l'algorithme suivant répond à la question :

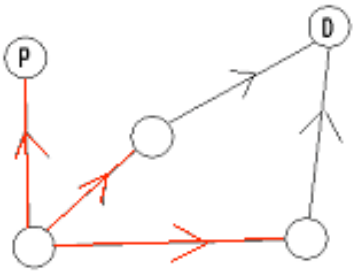
À chaque étape, tous les puits (sauf D) retournent le sens des connexions qui y aboutissent ; on itère l'algorithme tant qu'il reste des puits autres que D. On ne retourne jamais le sens des connexions de D.

Voici un exemple de mise en œuvre de cet algorithme :

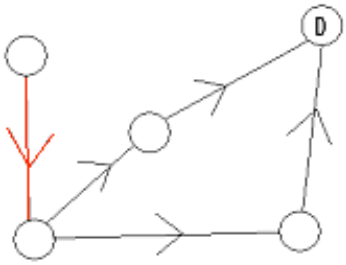


Etape 1

Etape 2



Etape 3



Etape 4

Remarque : nous avons figuré en rouge les arêtes dont le sens de la connexion a changé lors de la précédente itération, parce qu'elles étaient connectées à un puits.

III Propriétés préalables :

Propriété 1 : puits après cassure
Après cassure, le graphe ne peut contenir plus d'un puits.

Démonstration
La cassure n'affecte les connexions que de 2 nœuds seulement. Par l'absurde, si deux puits existaient après cassure, ce ne pourrait donc être que ces deux nœuds. Or, la connexion supprimée lors de la cassure était dirigée vers l'un de ces nœuds, qui était donc déjà un puits avant cassure, ce qui est contraire aux hypothèses : le graphe initial étant D-ciblé, il ne possède pas de puits.

Propriété 2 : cycles orientés pendant l'algorithme
En faisant fonctionner l'algorithme, aucun cycle orienté ne peut être créé dans les graphes générés.

Démonstration
Raisonnons par récurrence sur les étapes de l'algorithme.
- Initialisation
Prouvons que l'algorithme ne peut pas créer de cycle orienté lors de la cassure.
Raisonnons par l'absurde. Supposons qu'un cycle orienté soit apparu dans le graphe obtenu après la cassure. Pour revenir au graphe initial, il faut ajouter une arête à ce graphe. Or, rajouter une arête à un graphe ne peut couper un cycle orienté. Le graphe initial aurait donc déjà possédé un cycle orienté, ce qui est contraire à nos hypothèses.

- Hérédité
Prouvons que, si à l'étape n de l'algorithme, le graphe ne possède pas de cycle orienté, il n'en possèdera pas non plus à l'étape $n+1$.
Raisonnons par l'absurde. Si le graphe à l'étape $n+1$ possède un cycle orienté, et que celui à l'étape n n'en possède pas, c'est que l'une des arêtes du cycle orienté a été retournée par l'algorithme. Donc, c'est que l'un des nœuds du cycle était un puits dans le graphe de l'étape n. Etant un puits, toutes ses arêtes pointaient vers lui, et se sont vues retournées par l'algorithme : à l'étape $n+1$, les arêtes pointent hors de lui, et il ne peut donc contribuer à un cycle orienté, ce qui infirme notre hypothèse de raisonnement.

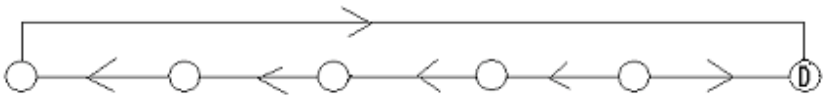
Ainsi, en appliquant l'algorithme, les graphes obtenus restent connexes (hypothèse H5 et définition de l'algorithme) et, comme nous venons de le voir, sans cycle orienté, d'après le théorème 1, il suffit que l'un de ces graphes n'ait plus de puits autre que D pour qu'il soit de nouveau D-ciblé.
Nous pouvons donc énoncer le :

Théorème 2
Par application de l'algorithme, tout graphe obtenu sera D-ciblé dès qu'il n'aura plus de puits autre que D.

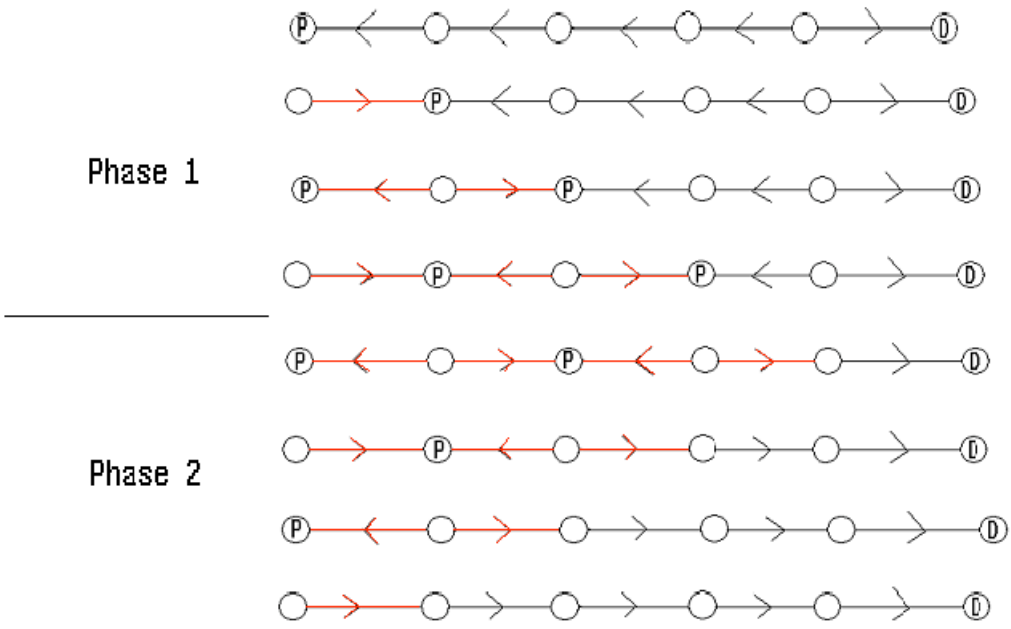
III L'algorithme fonctionne !

a) Un exemple de mise en œuvre de l'algorithme : le graphe en ligne

Pour simplifier le problème, nous nous sommes intéressés aux graphes où chaque nœud n'est, au plus, connecté qu'à deux autres nœuds (nous avons appelés ces graphes des *graphes en ligne*). Les remarques suivantes ne s'appliquent donc qu'aux graphes en ligne.
Voici un exemple de graphe en ligne (ils peuvent bien sûr être plus longs) :



Par cassure, rompons l'arête supérieure : le graphe obtenu possède un puits à l'une de ses extrémités.
Appliquons l'algorithme (les arêtes dont le sens de la connexion a été changé lors de la dernière itération sont figurées en rouge) :



Le dernier graphe est D-ciblé : chaque nœud peut envoyer des informations à D.

Remarquons que l'on distingue deux phases : si l'on note U le nœud connecté à D, la distance entre le puits le plus proche de D et U diminue (phase 1) puis, lorsqu'elle est égale à 1, le nombre de nœuds connectés à D augmente (phase 2) lorsqu'on applique à nouveau l'algorithme.
Nous avons rédigé une démonstration valide pour les graphes en ligne s'appuyant sur cette remarque, mais elle s'est révélée impossible à généraliser (pour les graphes non en ligne) puisque, lorsque les nœuds possédaient plus de 3 connexions, les puits ne se rapprochaient pas forcément de D (on n'y observe pas toujours la phase 1).

IV Démonstration générale

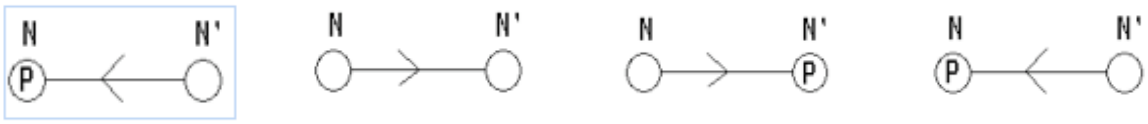
Nous avons observé, dans les graphes en ligne, que les points situés loin de D sont plus souvent des puits que les autres. Nous avons donc conjecturé le théorème suivant :

Théorème 3
Au fil des itérations successives de l'algorithme, un nœud à distance *d* de D ne peut devenir un puits qu'au maximum (*d*-1) fois.

Démonstration
Nous allons démontrer ce résultat par récurrence sur *d* :

Initialisation :
Soit N_1 un nœud à distance 1 de D (donc voisin de D).
La connexion entre D et N_1 est toujours orientée vers D, puisque D est le seul puits dont on ne change pas les connexions.
N_1 ne peut donc pas être un puits : il peut l'être au plus 0 fois, donc *d*-1 fois avec *d*=1.

Hérédité :
- *Lemme* : démontrons d'abord que, si N et N' sont deux nœuds voisins, entre 2 états du graphe où N est un puits, N' l'est une fois : en effet (cf. schéma), si N est un puits, à l'étape suivante, la connexion N-N' est dirigée vers N'. Pour que N redevienne un puits, il faut que cette connexion soit à nouveau dirigée vers N. Pour changer le sens de cette connexion il faut que N' devienne un puits pour que l'algorithme agisse sur N'.



Donc, si N devient un puits *d* fois au fil des itérations, N' l'aura été *d*-1 fois.

- Soit maintenant N_d un nœud à distance *d* de D et N_{d+1} un nœud à distance *d*+1 de D, voisin de N_d.
Supposons que N_d devienne un puits *d*-1 fois au maximum (c'est l'hypothèse de récurrence).
Raisonnons par l'absurde : si N_{d+1} devient un puits *d*+1 fois ou plus, alors, d'après le lemme, N_d est un puits *d* fois ou plus. Ceci infirme l'hypothèse de récurrence, donc N_{d+1} est un puits au maximum *d* fois.

Conclusion
À chaque fois que l'on applique l'algorithme, on retire au moins 1 au nombre de fois où un nœud peut devenir un puits.
Puisqu'il y a un nombre fini de nœuds, et qu'ils sont tous à distance finie de D, au bout d'un nombre fini d'itérations, aucun des nœuds ne pourra plus devenir un puits.
Le graphe obtenu sera donc, d'après le théorème 2, un graphe D-ciblé.

Nous avons donc prouvé que l'algorithme fonctionne.

Références :

Elèves : KALOUGINE Anne, TRAN Benoît, LAVENANT Hugo (en seconde en 2007-2008)
Etablissement : Lycée Blaise Pascal à Orsay (91)
Atelier Math en Jeans, session 2007-2008
Chercheurs : Jean-Benoît Bost, Nicolas Burq
Enseignants : Denis Julliot, Didier Missenard

Sujet (contextualisé…) :

Le réseau Carnaval

On est en 2010. Suite à des débordements, le proviseur a interdit de célébrer mardi-gras. Néanmoins, un groupe d'élèves est bien décidé à préparer un carnaval suivant la tradition. Par mesure de sécurité, la composition de l'équipe doit rester secrète. En fait, chaque membre ne doit connaître que ses "récepteurs", i.e. ceux parmi les autres membres à qui il est chargé de transmettre les consignes, et ses "émetteurs", i.e. ceux qui lui transmettent les consignes. Malheureusement, le réseau est fréquemment désorganisé par la défection de membres intimidés par la proviseure adjointe. Pour y remédier rapidement, il faut imaginer un procédé automatique de reconstitution du réseau après défection. Un bon réseau doit avoir les propriétés suivantes :

1. Il y a un seul chef.
2. Les consignes émises par le chef doivent pouvoir atteindre, de proche en proche, tous les membres.
3. Les consignes ne tournent pas en rond, i.e. les messages voyageant d'un membre à l'autre ne reviennent jamais à leur point de départ. En particulier, on n'est pas simultanément émetteur et récepteur d'une même personne.

En cas de défection, le réseau se "répare" en renversant certains couples émetteur-récepteur : on décide par exemple que y n'est plus récepteur de x, mais c'est x qui devient récepteur de y.

Le problème, c'est d'imaginer un procédé de réparation automatique.
En voici un. Dans le réseau qui vient de perdre un membre, on repère un "puit", i.e. un membre qui n'est le récepteur de personne. On renverse tout : il devient récepteur de tous ceux dont il était émetteur. On répète l'opération tant qu'on trouve des puits.
Sous quelle condition cet algorithme fonctionne-t-il ?
Comment construire des algorithmes plus simples ou plus efficaces ?