# Algorithmes, Réseaux, Langages (INF431)
# Contrôle de classement CC2
# partie Réseau et concurrence

29 juin 2011

## Avertissement

Le sujet de ce contrôle de classement est constitué de deux parties, intitulées "Réseaux et concurrence" et "Langages". **Les deux parties doivent être traitées**. Elles sont indépendantes.

Ceci est la partie "Réseaux et concurrence". **Elle doit être traitée sur des feuilles jaunes**. Elle est constituée de deux problèmes indépendants.

## 1 Problème 1 : Labyrinthes et concurrence

On se propose d'étudier un graphe non orienté $G = G(S, A)$ dont l'ensemble des sommets est $S = \{1, \ldots, n\}$ et l'ensemble des arêtes est $A$. Le cardinal de $S$ est $n \geq 2$. On suppose que le graphe $G(S, A)$ est connexe. Dans l'ensemble $S$ on distingue deux sommets :

– un sommet appelé *entrée* ;

– un sommet appelé *sortie*.

Pour tout sommet $s \in S$ on note `G.V[s]` l'ensemble de ses voisins. Il s'agit de la liste des entiers identifiant les sommets voisins du sommet $s$.

### 1.1 La recherche simple

On considére le graphe $G$ comme un labyrinthe avec une *entrée* et une *sortie*, l'objectif est de trouver un chemin entre ces deux sommets.

Pour cela on utilise une DFS à partir du sommet d'entrée et jusqu'au sommet de sortie. On appelle $S_k$ l'ensemble atteint jusqu'au $k$-ième sommet. On a $S_1 = \{\textit{entrée}\}$. On suppose que le sommet *sortie* est sur un sommet aléatoirement choisi sur $S - \{\textit{entrée}\}$. Donc la probabilité que $\textit{sortie} \in S_k$ est égale à $\frac{k-1}{n-1}$.

**Question 1** Donner le nombre moyen de sommets explorés (y compris les sommets d'entrée et de sortie) au moment où la recherche trouve la sortie. ◇

### 1.2 La recherche sur une grille

On regarde le comportement de la recherche sur un réseau en forme de grille $2m \times 2m$ où $m$ est un entier non nul. La figure 1 donne l'exemple de la grille pour $m = 3$.

On a donc $n = 4m^2$. On fixe les sommets *entrée* et *sortie* comme les deux sommets les plus à gauche sur la ligne du haut. On suppose que pour tout sommet $s \in S$, ses voisins sont donnés dans `G.V[s]` dans l'ordre Sud-Est-Nord-Ouest. De cette manière, la succession des sommets explorés se présente comme indiqué dans la figure 2.

**Question 2** On inverse maintenant le rôle des sommets *entrée* et *sortie* dans l'exemple de la figure 1. Dessiner la succession des sommets explorés. ◇

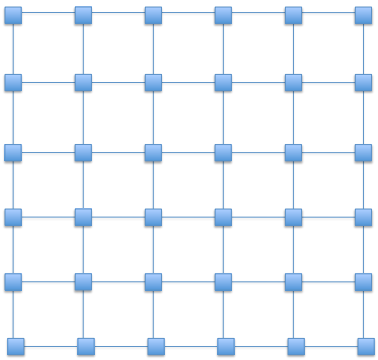

FIGURE 1 – Exemple de grille pour $m = 3$

## 1.3 La recherche double

Pour explorer moins de sommets, on se propose d'effectuer deux recherches en parallèle. On donne pour cela le pseudocode java :

```
public static void DFSDouble(int s, int c1, int c2) {
    for( v : G.V[s] ) {
        if ( G.trouve )
                return;
        if ( G.couleur[v] == c2 )
                G.trouve = true;
        if ( G.couleur[v] == blanc ) {
                G.couleur[v] = c1;
                G.parent[v] = s;
                DFSDouble(v, c1, c2);
        }
    }
}
```

Le paramètre `s` est un sommet et les paramètres `c1` et `c2` sont deux codes de couleurs. On initialise par :
$\forall s \in [1..n]$ : `parent[s]=-1` et `couleur[s]=blanc`, sauf
`couleur[entree]=bleu` et `couleur[sortie]=rouge`.

Ensuite on lance deux threads, chacun effectuant une des deux recherches :

- `DFSDouble(entree,bleu,rouge)`, appelée la recherche *bleue*, qui commence sur le sommet `entree`, qui marque les sommets explorés en `bleu` et qui s'arrête quand elle rencontre un sommet marqué `rouge` ;
- `DFSDouble(entree,rouge,bleu)`, appelée la recherche *rouge*, qui commence sur le sommet `sortie`, qui marque les sommets explorés en `rouge` et qui s'arrête quand elle rencontre un sommet marqué `bleu` ;

L'algorithme termine donc quand l'une des deux recherches atteint un sommet exploré par l'autre. Les deux parties du chemin sont alors mémorisées dans le tableau `parent` et on ne se pose pas la question de la reconstitution du chemin. On s'attend alors à ce que la double recherche termine en $O(\sqrt{n})$ au lieu de $O(n)$ sur une grande classe de grands graphes.

**Question 3** Indiquer des problèmes potentiels avec ce code. ◇

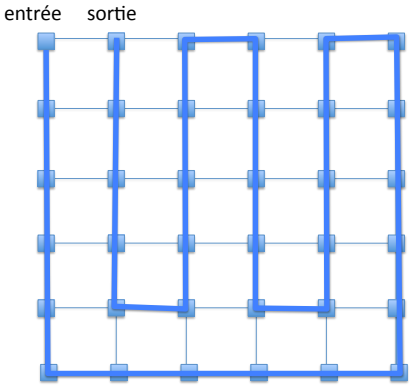


FIGURE 2 – Un parcours DFS dans la grille

Avant de chercher à corriger ce code, on va regarder quel est le fonctionnement que l'on attend pour une accélération de la recherche. Pour simplifier, nous supposons que la recherche *bleue* passe la main à la recherche *rouge* chaque fois qu'un nouveau sommet est coloré en bleu et que la recherche *rouge* passe la main à la recherche *bleue* chaque fois qu'un nouveau sommet est coloré en rouge.

**Question 4** En supposant que la recherche *rouge* prend la main la première, dessiner les successions des sommets explorés par les recherche *rouge* et *bleue*, jusqu'à que le chemin soit trouvé. Combien de sommets sont alors explorés, en fonction de $m$ et $n$. ◇

## 1.4 Synchronisation

On utilise un objet `mutex` simple (avec des méthodes `lock()` et `unlock()` comme vu en cours) pour adapter le code ainsi :

```
public static void DFSDouble(int s, int c1, int c2) {
    for( v : G.V[s] ) {
        mutex.lock();
        if ( G.trouve )
                return;
        if ( G.couleur[v] == c2 )
                G.trouve = true;
        if ( G.couleur[v] == blanc ) {
                G.couleur[v] = c1;
                G.parent[v] = s;
                DFSDouble(v, c1, c2);
        }
        mutex.unlock();
    }
}
```

**Question 5** Expliquer si ce code permet ou non un fonctionnement relativement équilibré des deux threads. Éventuellement, proposer une correction, sans chercher une alternance stricte comme envisagée à la question précédente. ◇

# 2 Problème 2 : Compression de topologie dans un réseau

On se propose d'étudier un réseau de communication dont l'ensemble $S$ des postes est constitué de $n$ éléments. Certains postes sont reliés par des liens de communications. Chaque lien relie deux postes distincts $s_i$ et $s_j$. On appelle $A$ l'ensemble des liens, et on note $m$ son cardinal. On suppose que le graphe $G(S, A)$ est non orienté et connexe. La figure 3 montre un exemple particulier d'un tel réseau avec $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ et $m = 16$.

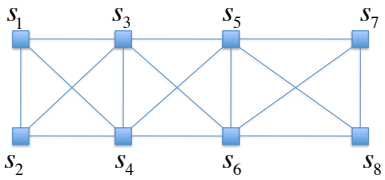


FIGURE 3 – Exemple de réseau

Soit $s \in S$, on appelle $V(s)$ l'ensemble des postes voisins de $s$. Par exemple sur la figure 3, $V(s_1) = \{s_2, s_3, s_4\}$. On suppose que chaque poste $s$ du réseau connaît son ensemble $V(s)$ de voisins.

Pour $E$ un ensemble fini, on note $|E|$ son cardinal. On a donc $|S| = n$ et $|A| = m$.

**Question 6** A quoi est égale la somme $\sum_{s \in S} |V(s)|$ ? En déduire qu'il existe au moins un poste $s \in S$ tel que $|V(s)| \leq \frac{2m}{n}$. ◇

On suppose que le poste $s$ doit transmettre un paquet $P$ vers un poste $d \neq s$. Si le poste $d \in V(s)$, alors le poste $s$ transmet directement vers le poste $d$ en utilisant la procédure TRANSMIT$(P, d)$ qui ne fonctionne que si il existe un lien direct entre les postes $s$ et $d$. Si le poste $d$ n'est pas voisin de $s$ alors le paquet $P$ devra être relayé sur une chaîne de postes qui est donnée par une table de routage. Cette chaîne est appelée la route de $s$ vers $d$, son nombre d'élément, par exemple $k$ est appelé la longueur de la route.

La route de $s$ vers $d$ est une séquence de postes $s_1, \ldots, s_k$ avec $s_1 \in V(s)$, $s_k = d$ et $\forall i < k$ $s_{i+1} \in V(s_i)$.

On se propose d'examiner les informations que les postes doivent échanger pour calculer leur table de routage. La table de routage du poste $s$ est la table qui, à chaque identifiant de poste $d \in S$, associe la route de $s$ vers $d$.

## 2.1 Dissémination de la topologie

La manière la plus efficace pour calculer la table de routage est que chaque poste connaisse entièrement le graphe $G(S, A)$. Dans ce cas le calcul de la table de routage pour le poste $s$ consiste à chercher les chemins les plus courts de $s$ vers tous les postes de $S - \{s\}$.

Pour que l'ensemble des postes connaissent le graphe du réseau on transmet des paquets de dissémination de topologie dont le type est LSA (pour *Link State Advertizement*). Chaque poste $s$ construit un paquet LSA avec la procédure CREATELSA$(s, V(s))$ et l'envoie à ses voisins. Le paquet LSA contient

un champ *originator* qui contient l'identifiant du poste qui le construit et un champ *links* qui est la liste des identifiants des postes qui sont dans $V(originator)$ (et connus par le poste $originator$ lui-même). Ce processus est résumé par le pseudo-code :

**procédure** LINKADVERTIZEMENT($s$)
 $P \leftarrow$ CREATELSA($s, V(s)$)
 **pour chaque** $v \in V(s)$
  **faire** TRANSMIT($P, v$)

Chaque poste $s$ attend et traite les paquets LSA qu'il reçoit par la procédure :

**procédure** LINKDISSEMINATION($s$)
 **tant que vrai**
  **faire**
  $P \leftarrow$ RECEPTION($s$)
  **si** $P \neq$ **null et** TYPE($P$) $= LSA$
  **alors**
  **si** ALREADYRECEIVED($P, s$) $=$ **faux**
  **alors**
  STORE($P, s$)
  **pour chaque** $v \in V(s)$
  **faire** TRANSMIT($P, v$)

La procédure RECEPTION($s$) prend le premier paquet dans la file d'attente des paquets reçu par le poste $s$. La procedure ALREADYRECEIVED($P, s$) donne le résultat **vrai** si un paquet identique à $P$ a déjà été reçu par le poste $s$. La procédure STORE($P, s$) mémorise les données de $P$ de façon à remettre à jour les informations que $s$ possède sur les ensembles $S$ et $A$ du réseau.

**Question 7** Si on suppose que les transmissions se font sans erreur et sans perte, combien de fois un paquet LSA est il transmis dans le réseau ?

Si tous les postes activent une fois leur procédure LINKADVERTIZEMENT(), combien d'identifiants (en ne comptant que les identifiants contenus dans les champs *originator* et *links*) les transmissions des paquets LSA vont-elles totaliser ? ◇

Ce nombre peut être considéré comme le coût de la transmission de la topologie sur le réseau. Après une instruction $P \leftarrow$ RECEPTION($s$), une instruction $\ell \leftarrow$ LASTRELAY($P$) permet de récupérer l'identifiant de l'émetteur de $P$.

**Question 8** Donner une modification très simple de la procédure de LINKDISSEMINATION($s$) qui divise par deux le nombre de transmissions. ◇

## 2.2 Compression de topologie

La quantité d'information pour mettre à jour la connaissance du graphe du réseau dans chaque poste peut être trop importante vis à vis des ressources de communication. Dans ce qui suit, nous introduisons un moyen pour réduire la quantité des informations à échanger.

Soit $s \in S$, nous appelons $L(s)$ l'ensemble des liens qui relie $s$ à ses voisins : $L(s) = \{\{s, v\},\ v \in V(s)\}$. Nous appelons *spanneur* un sous-ensemble $A_r \subset A$ de liens tels que $\forall s \in S : G(S, A_r \cup L(s))$ est un graphe connexe. Comme $L(s)$ est supposé connu de chaque poste $s \in S$, la connaissance d'un spanneur $A_r$ bien plus petit que $A$ suffit pour calculer des tables de routage.

Les figures 4 et 5 montrent deux exemples de spanneur. Les liens qui sont dans le spanneur sont renforcés en gras. Dans la figure 4 on a $|A_r| = 7$ et dans 5 on a $|A_r| = 6$.

**Question 9** Donner la taille minimale d'un spanneur quand le graphe du réseau est complet, c'est-à-dire quand $\forall s \neq t \in S, \{s, t\} \in A$.

Donner la taille minimale d'un spanneur quand le graphe du réseau est une chaîne $S = \{s_1, \ldots s_n\}$ et $A = \{\{s_i, s_{i+1}\},\ i < n\}$, quand $n > 2$. ◇

**Question 10** En utilisant le fait qu'un graphe connexe $G(S, A)$ vérifie nécessairement $|A| \geq |S| - 1$, montrer qu'un spanneur $A_r$ vérifie toujours $|A_r| \geq n - 1 - \frac{2m}{n}$. Vérifier pour le graphe complet et la chaîne. ◇

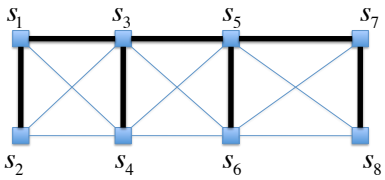


FIGURE 4 – Exemple de spanneur

## 2.3 Spanneur conservatif

Soit $k$ un entier. Un spanneur $A_r$ est dit conservatif à l'ordre $k$ quand, pour tout $(s, d) \in S^2$, si la route la plus courte de $s$ vers $d$ dans $G(S, A)$ est de longueur inférieure ou égale à $k$, alors il existe une route de même longueur entre $s$ et $d$ dans le graphe $G(S, A_r \cup L(s))$.

On définit la distance entre deux sommets $s$ et $d$ d'un graphe comme la longueur de la route la plus courte entre $s$ et $d$. Un spanneur conservatif à l'ordre $k$ conserve donc les distances jusqu'à la longueur $k$. Un spanneur est dit conservatif si il est conservatif à tous les ordres.

Un spanneur conservatif peut s'avérer avantageux car le spanneur étant plus petit que $A$, les nouvelles tables de routages restent néanmoins optimales en terme de distance.

**Question 11** Un des deux spanneurs des figures 4 ou 5 n'est pas conservatif, lequel ? Dire pourquoi. ◇

Un spanneur est trivialement conservatif à l'ordre 1, par contre il n'est pas nécessairement conservatif à l'ordre 2. On va montrer que si le spanneur est conservatif à l'ordre 2, alors il l'est à tout ordre.

**Question 12** Soit $A_r$ un spanneur conservatif à l'ordre 2. Soit $(s, d) \in S^2$, tel que $d$ est à distance 2 de $s$, montrer qu'il existe $v \in V(s)$ tel que $\{v, d\} \in A_r$ ◇

**Question 13** Soit $A_r$ un spanneur conservatif à l'ordre 2. Soit $k$ un entier supérieur ou égal à 1. Soit $(s, d) \in S^2$ tel que la distance de $s$ à $d$ soit $k + 1$. Montrer qu'il existe $t \in S$ tel que $\{t, d\} \in A_r$ et la distance de $s$ à $t$ est égale à $k$. En déduire par récurrence que le spanneur est conservatif. ◇

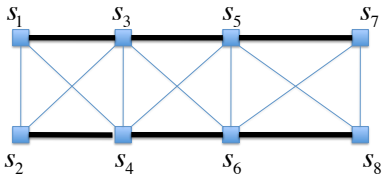


FIGURE 5 – Exemple de spanneur