IUT Informatique Bordeaux

M4102C - Algorithmique distribuée

Exemples d'algorithmes

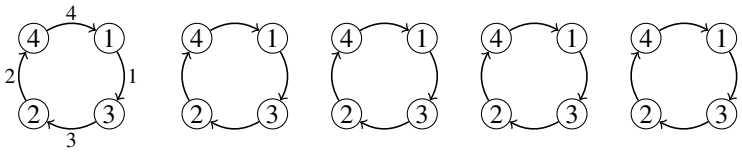
TD1

1 Élection dans un anneau orienté (avec identifiant)

On suppose que la topologie est un anneau *orienté* : chaque sommet (noeud) peut envoyer des messages à son voisin dans le sens des aiguilles d'une montre. Chaque noeud a un *identifiant unique* de type entier. On considère l'algorithme suivant (appelé LCR) permettant d'élire le sommet qui a le plus grand identifiant (mais qui ne le sait pas au départ...). L'exécution est *synchrone* : à chaque ronde, les nœuds recoivent les messages envoyés à la ronde précédente et en envoient de nouveaux (qui seront reçus à la ronde suivante). Tous les noeuds démarrent en même temps (en parallèle).

```
onStart(): // ronde 1
   send(myID)

onMessage(otherID):
   if (otherID > myID)
       send(otherID)
   else
       if (otherID = myID)
           ELECTED <- true
```



1. Exécuter l'algorithme sur le graphe ci-dessus, en écrivant le contenu des messages à côté des arcs correspondants.
2. Quelle est la complexité en *temps* (nombre de rondes) de cet algorithme ?
3. Quelle est sa complexité en *nombre de message*
   (a) Dans l'exemple ci-dessus ?
   (b) Dans le meilleur des cas pour n sommets, si les identifiants sont répartis de la meilleure manière possible ? (dessinez le réseau correspondant pour $n = 4$)
   (c) Dans le pire des cas pour n sommets, si les identifiants sont répartis de la pire façon possible ? (dessinez le réseau correspondant pour $n = 4$)

2 Diffusion d'information

On considère l'algorithme suivant de diffusion (broadcast) dans une topologie arbitraire *non-orientée*, en supposant que l'émetteur initial est déjà informé. On suppose ici que `sendAll()` génère autant de messages qu'il y a de voisins localement. Dans les questions qui suivent, on suppose une exécution *synchrone*.

```
onStart():
   if (informed)
      sendAll(message)

onMessage(message):
   if (! informed)
      informed <- true
      sendAll(message)
```

1. Etant donné un graphe G, combien de rondes prendra la diffusion dans le pire des cas ?
2. Combien de messages seront échangés en fonction du nombre d'arêtes (noté m) ? Pour quel topologie ce nombre est-il maximum ? Si l'on sait qu'on est dans une telle topologie, peut-on faire mieux ? Si oui comment et combien ?

3. Modification de l'algorithme : Revenons à une topologie arbitraire et supposons maintenant que l'information à transmettre est un fichier volumineux. On distingue deux types de messages : le fichier lui-même (gros message) et les messages de coordination de l'algorithme (petits messages). Les petits messages sont gratuits mais les gros messages coûtent cher.
   → Élaborez une stratégie (abstraite) pour minimiser le nombre de gros messages envoyés. Combien de gros messages votre algorithme envoie-t-il ? Êtes-vous sûrs qu'on ne peut pas faire mieux ?

3 Construction d'arbres couvrants (sera codé en TP)

L'algorithme suivant construit un arbre couvrant dans un réseau *synchrone*. C'est presque le même que dans la présentation, en adaptant le code pour que cela ressemble à notre implémentation en TP.

```
Node parent // variable qui désigne le parent du noeud local

onStart(){ // Exécuté par tout le monde au début
   parent <- null
}

onSelection(){ // Exécuté seulement par un noeud qu'on choisira à la souris
   parent <- moi-même; // je suis la racine de l'arbre
   Message message <- new Message("", "TREE"); // voir *
   sendAll(message);
}

onMessage(Message message){
   if (message.getFlag().equals("TREE")){
      if (parent == null)
         parent <- message.getSender();
         sendAll(message); // propage la construction de l'arbre
      }
   }
   // Note: les messages ont à la fois un "contenu" et un "flag". Le flag est
   // optionel, il permet de différiencier entre plusieurs types de message quand
   // l'algorithme devient complexe (p.ex. ici la construction de l'arbre).
}
```

Proposez des adaptations de cet algorithme afin de remplir les objectifs demandés. Écrivez des explications claires sur votre stratégie.

1. Faites en sorte que les noeuds apprennent la liste de leurs enfants dans l'arbre.
   → *Astuce :* Il existe une méthode `send(destination, message)` pour envoyer un message à un seul voisin. Par ailleurs, vous pouvez utiliser les flags pour distinguer entre deux types de messages différents.
2. Comment peut-on détecter lorsque l'on connait tous ses enfants ?
   → *Astuce :* Il existe une méthode `getTime()` que vous pouvez appeler pour connaître le numéro de la ronde actuelle. De plus, vous pouvez surcharger une méthode `onClock()` que le système appellera automatiquement à chaque ronde (après la réception des messages envoyés à la ronde précédente).
3. La racine peut-elle détecter lorsque la construction de l'arbre est terminée ? Si oui, comment ?
   → *Astuce :* Vous aurez besoin d'un troisième type de message (et d'une bonne idée).
4. La racine peut-elle apprendre le nombre de noeuds total dans l'arbre ? Si oui comment ?