Algorithmique de la mobilité

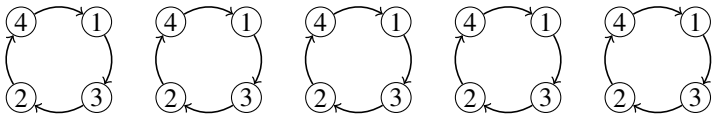
Quelques algorithmes distribués

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...). On considère une exécution *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, etc.).

```
onStart(): // round 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. Vous pouvez représenter les messages envoyés en les écrivant sur les arcs du graphe.
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 chaque `sendAll` génère autant de messages qu'il y a de voisins localement.

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

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

Dans les questions qui suivent, on suppose une exécution *synchrone*.

1. Comment appelle-t-on le paramètre de graphe qui correspondra ici au nombre de rondes qu'il faut exécuter dans le pire des cas (c.à.d. si l'émetteur est placé au pire endroit) ?
2. Combien de messages vont être échangés (en fonction du nombre d'arêtes m) ? Pour quel type de topologie ce nombre est-il maximal ? Si l'on sait à l'avance qu'on est dans ce type de topologie, peut-on faire mieux ? Si oui combien de messages et comment ?
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.
   $\rightarrow$ Élaborez une stratégie (abstraite) pour minimiser le nombre de gros messages envoyés. Combien de gros messages votre algorithme envoie-t-il ?

3 Construction d'arbres couvrants

L'algorithme suivant construit un arbre couvrant dans le réseau. C'est presque le même que dans la présentation. On suppose que le réseau est *synchrone*.

```
Node parent <- null

onStart()
   if (root)
      parent <- moi-même;
      sendAll(); // message quelconque

onMessage(message):
   if (parent == null)
      parent <- message.getSender();
      sendAll(); // idem
```

Proposez des adaptations au pseudo-code de l'algorithme afin de remplir les objectifs ci-dessous. Le format de vos proposition est assez souple (explications ou pseudo-code), mais elles doivent être précises. Nous les coderons réellement en TP.

Astuces : Vous avez le droit de modifier le contenu des messages. Vous pouvez aussi, si besoin, utiliser une méthode `onClock()` que le système appellera automatiquement sur chaque noeud à chaque ronde (juste après l'arrivée des messages).

1. Comment faire en sorte que les noeuds apprennent la liste de leurs enfants dans l'arbre ?
2. Un noeud peut-il détecter qu'il n'aura pas d'enfants dans l'arbre ? Si oui comment ?
3. La racine peut-elle détecter lorsque la construction de l'arbre est terminée ? Si oui, comment ?
4. La racine peut-elle apprendre le nombre de noeuds total dans l'arbre ? Si oui comment, et en utilisant combien de messages ?

4 Plus difficile...

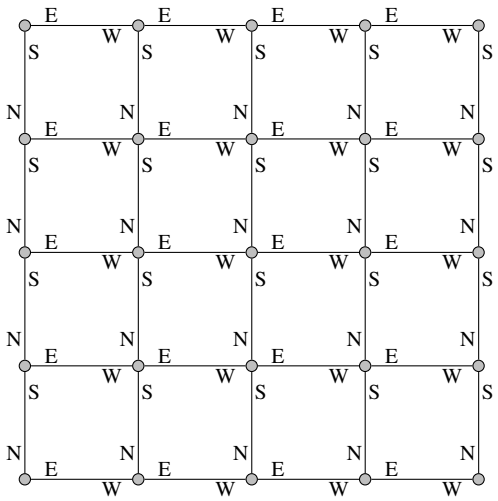
On considère l'algorithme d'élection probabiliste suivant dans un graphe complet, avec exécution synchrone. Initialement tous les sommets sont candidats. A chaque ronde, les sommets candidats tirent à pile ou face. Ceux qui tirent pile envoient un message à tout le monde. Les autres décident

de ne plus être candidats. L'algorithme s'arrête quand il y a strictement moins de deux messages échangés (donc 0 ou 1), ce qui peut être détecté par tout le monde. À ce moment là, si un nœud est encore candidat, il est élu, sinon l'algorithme a échoué.

1. Le temps d'exécution de cet algorithme est-il borné ? est-il fini ? est-il infini ?
2. Quel est la probabilité de succès si $n = 2$, si $n = 3$?
3. En cas d'échec, on recommence (et ainsi de suite). Combien de rondes cela prendra-t-il en moyenne pour réussir l'élection pour $n = 2$.

5 Diffusion dans une grille

On considère une topologie en grille carrée (voir dessin, pour $n = 25$). On considère un processus de diffusion *à partir du noeud le plus en haut à gauche*. Les noeuds savent qu'ils sont dans une grille, mais ils ne savent pas où dans la grille (sauf l'émetteur initial).



1. Combien de messages seront échangés si l'on utilise l'algorithme de diffusion basique vu la semaine dernière ? ("La première fois que je reçois le message, je le retransmets à tous mes voisins avec `sendAll()`.") Le fait que le réseau soit synchrone ou asynchrone a-t-il un impact sur ce nombre ? Si oui, lequel ? (*Pour rappel, on parle de communication* asynchrone *si un message peut prendre un temps arbitraire à être transmis, en supposant toutefois qu'il finira nécessairement par arriver (temps fini).*)
2. On suppose maintenant que les nœudspeuvent utiliser les primitives `sendN()`, `sendS()`, `sendE()`, `sendW()` plutôt que `sendAll()`. En revanche, il ne savent toujours pas où ils sont ni même s'ils ont un voisin dans une direction donnée (ce qui peut induire des messages perdus dans le vide).

   → Écrivez un algorithme qui tient compte de la topologie en grille. Combien de messages sont envoyés (en incluant les messages envoyés dans le vide) ?
3. On suppose maintenant que les nœudspeuvent connaître la provenance d'un message `msg` en utilisant `msg.getOrigin()` dont la réponse est parmi `{N,S,E,W}`. Ils peuvent aussi tester s'ils ont un voisin dans une direction donnée (ex : `hasNeighbor(N)`).

   → Écrire un algorithme qui utilise ces informations pour n'envoyer que $n - 1$ messages.

   Peut-on espérer faire mieux ? Et dans une autre topologie ? Pourquoi ?

6 Élection

L'élection est le problème qui consiste à distinguer un noeud parmi les autres. Un algorithme d'élection doit donc garantir qu'à la fin de l'exécution, un (et un seul) noeud se trouve dans un état différent des autres.

6.1 Élection dans les arbres

On suppose que le réseau est un *arbre*, mais il n'y a pas de noeud distingué au préalable (pas de racine), ni même de relation de parenté. En bref, il s'agit simplement d'un graphe sans cycle. Vous pouvez supposer que les noeuds ont des identifiants uniques (mais piochés dans un intervalle arbitraire). L'exécution est *a*synchrone, donc certains messages peuvent aller plus vite que d'autres. Enfin, les noeuds savent combien de voisins ils ont dans l'arbre ; ils sont aussi capables de choisir sur quel lien ils envoient un message ainsi que d'identifier de quel lien provient un message.

$\rightarrow$ Concevoir un algorithme d'élection qui fonctionne dans ce type de topologie. Vous pouvez utiliser le format de votre choix pour décrire l'algorithme (pseudo code ou explications textuelles), du moment que c'est clair et non-ambigu.

$\rightarrow$ Combien de messages votre algorithme a-t-il utilisé (au plus) lorsqu'un noeud apprend qu'il est élu ? (on ignorera la partie qui consiste à informer les autres noeuds).

6.2 Nombre minimum de messages ?

Soit la déclaration suivante : "Il n'existe aucun algorithme d'élection *universel* (= qui marche dans *toute* topologie) qui utilise moins de m messages (m = nombre d'arêtes), et ce, que l'on ait ou non des identifiants, et que le réseau soit synchrone ou asynchrone."

Pensez-vous que cela est vrai ? Si non, pouvez-vous penser à un algorithme moins coûteux ?

6.3 Élection probabiliste

On considère l'algorithme d'élection probabiliste suivant dans un graphe complet, avec exécution synchrone. Initialement tous les sommets sont candidats. A chaque ronde, les sommets candidats tirent à pile ou face. Ceux qui tirent pile envoient un message à tout le monde. Les autres décident de ne plus être candidats. L'algorithme s'arrête quand il y a strictement moins de deux messages échangés (ce qui peut être détecté par tous les nœuds). A ce moment là, si un nœud est encore candidat, il est élu. S'ils sont tous éliminés, l'algorithme a échoué.

1. Le temps d'exécution est-il borné ? est-il fini ? peut-il être infini ?
2. Quel est la probabilité de succès si $n = 2$, si $n = 3$?
3. En cas d'échec, on recommence (et ainsi de suite). Combien de rondes cela prendra-t-il en moyenne pour réussir l'élection pour $n = 2$.