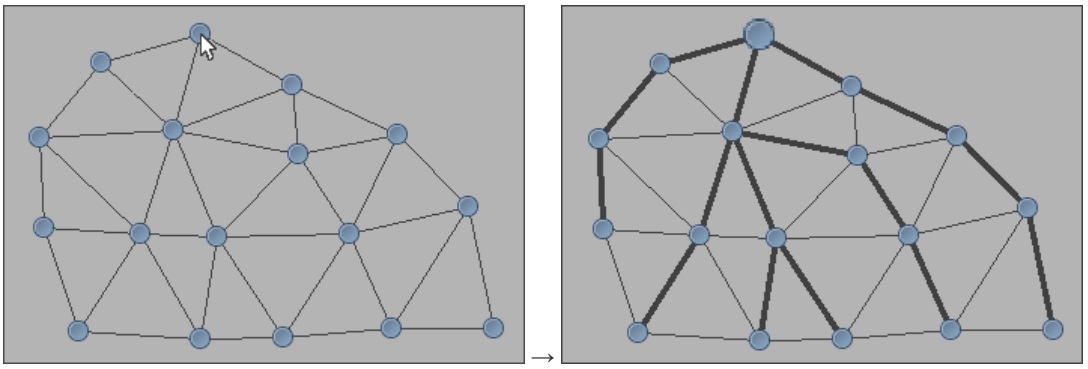
Construction distribuée d'un arbre couvrant

Principe général

Étant donné un réseau et un noeud de départ, il est très facile de construire un arbre couvrant de manière distribuée. Le noeud de départ envoie un message à tous ses voisins. Ensuite, la première fois que chaque autre noeud reçoit un message, il choisit l'émetteur de ce message comme parent et envoie à son tour un message à ses voisins. Cette propagation au sein du réseau va ainsi définir un arbre dont la racine est le noeud de départ (illustration ci-dessous).



Implémentation avec JBotSim

On définit une classe TreeNode qui hérite de Node. Deux événements sont gérés (surchargés) :

- La méthode `onSelection()` : appelée lors de la sélection d'un noeud par l'utilisateur (bouton du milieu de la souris).
- La méthode `onMessage()` : appelée lors l'arrivée d'un message sur un noeud.

1. Complétez le code suivant pour qu'un arbre couvrant se construise de manière distribuée lorsqu'un noeud est sélectionné. La largeur des arrêtes appartenant à l'arbre sera définie à 4 (`node1.getCommonLinkWith(node2).setWidth(4);`)

```
import jbotsim.Message;
import jbotsim.Node;

import java.util.ArrayList;
import java.util.List;

public class TreeNode extends Node {
    Node parent = null;
    List<Node> children = new ArrayList<Node>();

    @Override
    public void onSelection() {
        // le noeud courant est sélectionné
    }

    @Override
    public void onMessage(Message message) {
        // Le noeud courant reçoit un message
    }
}
```

La construction d'un arbre consiste a choisir un parent pour chaque noeud. Le parent de la racine est lui-même. Explications détaillées : Dans le cas de la sélection d'un noeud, le traitement consiste à devenir racine de l'arbre en faisant pointer la variable parent sur soi-même (`parent = this`), puis d'initier la construction en envoyant un message à tous ses voisins (c.à.d. les noeuds avec qui on partage un lien) via `sendAll()` .

Ensuite, lorsqu'un noeud reçoit un message, s'il n'avait pas encore de parent, il choisir l'émetteur du message comme parent (et met le lien correspondant en gras), puis il envoie un message à son tour.

2. Faites en sorte que chaque noeud connaisse ses enfants (dans la variable `children`).

3. Vérifiez le bon fonctionnement de votre classe via la méthode `main()` suivante.

```
public static void main(String[] args) {
    Topology tp = new Topology();
    tp.setDefaultNodeModel(TreeNode.class);
    tp.setClockSpeed(500);
    new JViewer(tp);
}
```

Agrégation de données dans les réseaux de capteurs

On considère un ensemble de capteurs (sensors en anglais) déployés dans une région donnée. Ces capteurs sont capables d'effectuer des mesures, par exemple, mesurer la température locale, le taux d'humidité ou la présence de mouvements. En plus de ces capacités, les capteurs son capables de communiquer sans fil les uns avec les autres. On parle alors de réseaux de capteurs (wireless sensor networks, ou WSN en anglais).

L'idée derrière les réseaux de capteurs est de pouvoir surveiller une région de manière quasi-autonome, chaque capteur étant concerné par un petit sous-ensemble de la région. Un noeud spécial, appelé station de base, est chargée de collecter les informations en provenance des capteurs et éventuellement d'informer les opérateurs humains en cas d'alerte.

Infrastructure

Afin de simplifier l'acheminement des informations (routage), on organise les communications sous forme d'un arbre qui couvre tous les noeuds (arbre couvrant) et dont la racine est la station de base.

Cet arbre est construit de manière distribuée à l'initiative de la station de base.

1. Créez une nouvelle classe de noeud `SensorNode` qui hérite de `TreeNode` (issue du problème précédent). Dans cette nouvelle classe, redéfinir la méthode `onSelection()` pour mettre à jour l'icône serveur.png du noeud qui est choisi comme station de base (c'est à dire comme racine). Vous devriez obtenir un résultat similaire à la figure ci-dessus.

   Les méthodes `onMessage()` et `onClock()` seront aussi redéfinies

   Pour que la classe `SensorNode` fonctionne correctement, elle doit appeler les méthodes `onMessage()` et `onSelection()` de sa classe parente (`TreeNode`) au début de chaque méthode. Cependant, les noeuds capteurs envoient aussi des messages, qui sont différents des messages utilisés par la classe `TreeNode`. On peut donc modifier la classe `TreeNode` pour ajouter une propriété `TreeNodeMessage` aux messages qu'elle envoie (méthode `setProperty('TreeNode', true)` sur l'objet `Message`). Ainsi, dans la class `SensorNode`, lorsqu'un message arrive, s'il a la propriété, il est passé à la méthode de la classe parente (`super.onMessage(message)`), sinon il est traité par la classe `SensorNode`.

2. Testez votre programme. Plutôt que de déployer les noeuds manuellement à chaque exécution, il est recommandé de le faire automatiquement depuis votre main(). Nous aurons souvent besoin de faire des tests...

3. Ajoutez un attribut `value` de type `Double` ainsi qu'une méthode `sense()`, sans argument, qui met à jour cette variable à chaque fois qu'elle est appelée. Pour commencer, on se contentera de tirer au sort une valeur entre 0 et 255, symbolisant par exemple le relevé d'une température. En même temps, changez la couleur du noeud (`setColor()`) afin de représenter la valeur qu'il a captée. Le constructeur `Color(r, g, b)` peut s'avérer pratique pour ça.

4. Ajoutez une nouvelle entrée au menu contextuel permettant de mettre à jour toutes les valeurs captées (appel à sense() sur chaque noeud). Vous pouvez le faire directement dans votre méthode main() (ou tout autre endroit, mais pas dans vos noeuds !) en créant un `CommandListener` à la volée comme indiqué ci-dessous (où tp est votre topologie):

```
JTopology jtp = new JTopology(tp);
jtp.addCommand("Update values");
jtp.addCommandListener((String command) -> {
    if (command.equals("Update values"))
        for (Node node : tp.getNodes())
            ((SensorNode)node).sense();
});
new JViewer(jtp);
```

L'objectif de la station de base est de s'informer des valeurs relevées dans le réseau à intervalle régulier, afin de détecter une éventuelle anomalie. Bien entendu, il existe des bonnes et des mauvaises manières de le faire. En particulier, les capteurs ont de fortes contraintes d'économie d'énergie et on essaie donc de minimiser les communications. On se concentre ici sur les procotoles permettant de remonter l'information une seule fois. Si on veut le faire régulièrement, il suffit d'utiliser ces protocoles de manière répétée.

Protocole basique : Le moyen le plus facile, algorithmiquement, serait que chaque capteur envoie la valeur qu'il a relevée à son parent. Chaque valeur reçue par un noeud serait ensuite retransmise à son propre parent, et ainsi de suite jusqu'à la station de base (racine de l'arbre). Autrement dit, chaque valeur est remontée jusqu'à la racine.

5. Implémentez cette méthode et comptez combien de messages ce protocole engendre ?

6. Souvent, il n'est pas nécessaire de connaître chaque valeur captée. Par exemple, dans le cas d'une surveillance de feu de forêt, on peut se contenter de connaître la valeur maximum captée, et si elle dépasse un seuil, chercher alors à collecter plus d'information.

   Protocole d'agrégation : Dans un protocole d'agrégation, chaque noeud attend de recevoir la valeur de ses enfants avant d'envoyer quoi que ce soit. Il "agrège", soit au fur et à mesure, soit tout d'un coup, ces valeurs entre elles et agrège aussi sa propre valeur (le tout selon une fonction bien définie, ici on prendra la fonction maximum). Une fois terminé, il transmet le résultat à son parent, et ainsi de suite jusqu'à la racine.

   Implémentez cette méthode dans une autre classe, par exemple `CleverSensorNode`, et comptez combien de messages ce protocole engendre.

7. Adaptez votre code pour que chaque noeud ayant des enfants colore en rouge le lien par lequel il a reçu la plus grande valeur (seulement si cette valeur est supérieure à celle qu'il a lui-même capté). Que représente l'ensemble des liens en rouge ?

8. Faites en sorte que lors du redémarrage de la simulation (événements `onStart()`) les noeuds réinitialise tous leurs attributs.

9. Ajouter un `StartListener` qui déplace les noeuds aléatoirement (en vérifiant que le graphe reste connecté grâce à la méthode `Connectivity.isConnected(Topology tp)`), et selection un noeud au hasard. De plus, faire en sorte que lorsque l'arbre est formé, que chaque noeud effectue sa capture et démarre la transmission (ou l'aggrégation) des températures. Pour finir, faire en sorte que, lorsque toutes les températures sont récupérées (ou aggrégé), le nombre de transmission soit sauvegardé dans un fichier log, et la simulation redémarre.

10. Lancez une simulation pour chaque méthode jusqu'a avoir assez de données pour tracer une coubre avec `matplotlib` en python permettant de comparer les performances des deux méthodes.