Rapport Projet Table de routage

Wissam Serhan et Thomas Pittis

Notre projet est codé en Java et contient plusieurs classes que nous allons chacune expliquer.

La classe *Lien* permet de changer et/ou retourner le nombre de liens que possède un nœud avec les autres nœuds du réseau. Pour cela elle va utiliser une variable appelée *valeur* qui va être utilisée par le constructeur *Lien* qui attribue un poids 0 ou 1 à la variable *valeur* si le lien existe entre deux nœuds. Ici on aura un lien "imaginaire" entre chaque nœud du réseau. On va attribuer un poids à chacun des liens : 0 si il n'existe pas réellement et 1 si il existe bien. Si on veut changer cette valeur on utilise la fonction *chgValeur* qui permet de la changer. Si on veut savoir quel est la valeur d'un lien choisi on fait appel à la fonction *getValeur* qui retourner cette dernière.

La classe *Nœud* quant à elle va utiliser une structure de donnée appelée *Nœud*. Elle va permettre de lier toutes les données entre elles et de simplifier leur traitement, ces données permettent de créer un nœud. On va retrouver dans cette structure plusieurs variables : *nbLien* qui donne le nombre de liens que possède un nœud, *identifiant* qui va donner l'identifiant du nœud pour le différencier ainsi que *type* qui donne le type de nœud que c'est (1 pour Tier1, 2 Tier2 et 3 Tier3). Elle possède aussi une chaîne de caractère *message* qui va contenir le message du nœud s'il en a un.

Le constructeur *Nœud* qui prend en paramètre l'*identifiant* et le *type* du nœud va initialiser l'identifiant, le type du nœud ainsi que de dire que le nœud ne possède aucuns liens et aucun message.
Une fois que le message est transmis d'un nœud à l'autre il faut changer la valeur de message du transmetteur est le mettre en *Null* ainsi que de faire passer la chaîne de caractère du message au nœud receveur. La fonction *newMessage* va donc changer le message que contient le nœud.
Pour pouvoir transmettre le message on doit connaître le nombre de liens que possède le nœud, on utilise la fonction *getNbLien*. Si on veut créer un lien on utilise la fonction *addLien*.
Le dernier nœud devra transmettre le message, de ce fait il va utiliser la fonction *getMessage* qui va renvoyer la chaîne de caractère du message.

La classe *Reseau* est très importante pour le programme. C'est elle qui va créer tous les nœuds, créer les liens potentiels de base, initialiser le réseau et initialiser les liens pour les 3 niveaux existants et finalement transférer un message d'un nœud de départ à un nœud d'arrivée.

On initialise le constructeur de la classe *Reseau*, elle va permettre d'attribuer à *tableauDesLiens* le tableau à 2 dimensions *Lien* et à *tableauDesNoeuds* le tableau *Nœud*.
Le tableau à deux dimensions va permettre de créer un lien de valeur *0* entre chaque nœud, il va commencer pour *i=0* et *j=0* et faire une boucle jusqu'à que *j* soit égal à *99*. Une fois atteint cette limite *i* va augmenter de *1* et donc au final créer un lien de valeur *0* entre chaque nœud du système. Si le lien est créé entre *56* et *23* alors le lien inverse *23-56* sera alors aussi créé. La fonction *initTableauDesLiens* permet de créer tous les liens potentiels de base.
On va créer une structure de données *Reseau* qui regroupe le tableau à deux dimensions appelé Lien qui contient donc les valeurs de *tableauDesLiens*, mais aussi le tableau *Nœud* qui lui contient les valeurs de *tableauDesNoeuds*.

Tous ces nœuds ont été au préalable créés par la fonction *initTableauDesNoeuds*. Elle a créé *8* nœuds de type 1, *20* nœuds de type 2 et *72* nœuds de type 3. Ils sont tous différenciables grâces à leur numéro d'identification qu'ils ont perçu lors de leur création.

Par la suite on va initialiser le réseau en englobant les fonctions *initTableauDesNoeuds* et *initTableauDesLiens* dans la fonction *initReseau*. Ceci va juste lier tous les nœuds ensembles mais il faut maintenant vérifier le type de chaque nœud et déterminer le poids de chaque lien.
Nous allons donc commencer par initialiser les liens du *backbone* de type 1. On va parcourir notre double tableau pour trouver nos *8* premiers nœuds. On définit ensuite qu'il y a *75%* de chances que le lien entre deux nœuds de type 1 existe, ainsi que le temps d'attente aléatoire de communication.
On attribue aussi une valeur à chacun des liens de cette catégorie par une valeur comprise entre 5 et 10.
Il faut ensuite faire la même chose pour les liens de type 2. On parcourt chaque noeud ayant comme numéro d'identifiant une valeur comprise entre 8 et 28. Il va donc définir aléatoirement les noeuds de type 2 qui seront liés au backbone parmis les 20. Les noeuds du Tier 2 vont aussi être aléatoirement reliés à deux ou trois noeuds du même Tier. La valeur attribuée à ces liens est comprise entre 10 et 20. Pour finir on fait les mêmes choses que précédemment avec les noeuds du Tier 3. On relie chaque noeud du Tier 3 à deux noeuds de niveau deux ainsi qu'à un noeud de niveau 1. Les liens seront valués entre 15 et 50. Pour vérifier que le graphe est connexe, nous allons parcourir le graphe en profondeur avec la fonction *parcoursProfondeur*. Elle sera utilisée dans la fonction *estConnexe*, qui commence au niveau d'un noeud et qui colorie ce noeud si il possède un lien avec un autre noeud. Si le lien existe alors il va se déplacer dans l'autre noeud et le colorier si lui aussi possède un autre lien avec un noeud non coloré. Sinon on va se déplacer dans un autre noeud coloré et vérifier si il possède un lien avec un noeud non coloré. La fonction va tourner en boucle recursive jusqu'à qu'il n'y ai plus aucun noeud non coloré. Si c'est le cas il va renvoyer que le réseau est connexe. On attribue aussi une valeur à chacun des liens de cette catégorie par une valeur comprise entre 5 et 10.

Pour déterminer la table de routage on créer une nouvelle classe *Routage* qui créer un attribut appelé voisin. On va commencer par dire que tous les nœuds ne possèdent aucuns voisins. Plus tard on va vérifier pour chaque noeud si il possède un voisin. Pour cela on utilisera l'algorithme de Dijkstra pour chaque noeud et pour chacune des 99 destinations (oui ça fait beaucoup) on déterminera le plus court chemin puis on inserera le noeud voisin du noeud de depart au plus court chemin. Cela donnera cela : tableDeRoutage [noeudDeDepart][noeudDarrive] = 1erVoisinDuPlusCourtChemin. Et ainsi pour chaque noeud. L'idée est là cependant petit soucis technique concernant l'algorithme de Dijkstra, on a donc bricolé une fonction qui simule une table de routage cohérente (marche 1 fois sur X) car ce fonctionnement créé des boucles infini, les noeuds se passent et repassent le message. A revoir.

Pour envoyer un message d'un nœud à un autre on utilise les fonctions chemin et *transfereMessage*.
La première va afficher un message demandant à l'utilisateur de saisir l'identifiant du nœud de départ de son choix ainsi que celui d'arrivée ainsi que le message à transférer. Le message va circuler de proche en proche c'est-à-dire que l'on va regarder la table de routage pour chaque noeud, voir à quel voisin il est judicieux de transférer le message en fonction du noeud d'arrivé,, lui transferer et recommencer avec ce noeud pour noeud de depart.