Rapport de projet en Algorithmes Distribués

Louis DEMOULINS - Serigne Amsatou SEYE

Décembre 2018

Le but principal de ce projet d'algorithme distribué est de transcrire en Java la version synchrone de l'algorithme GHS dont nous avions l'idée pour le faire tourner sur `JBotSim`.

L'algorithme GHS permet de trouver en distribué un arbre couvrant de poids minimum en `O(nlogn)` rounds.

1 Implémentation

1.1 Implémentation de l'algorithme

Tout d'abord, nous avons commencé par implémenter une classe `BasicNode` qui étend de la classe `Node` de la bibliothèque `JBotsim`. Dans cette classe, nous avons redéfini les méthodes `void onStart()`, `void onClock()`, `void onMessage(Message msg)` et `void onSelection()` de la classe parente `Node`.

Dans la méthode `void onStart()`, on initialise le sommet comme étant en même temps la racine et son propre père dans le fragment. Son état est aussi initialisé à `DONE` et la liste de ses voisins qui ne sont pas dans le fragment et celle contenant des messages `"READY"` à vide.

Dans la méthode `void onClock()`, nous avons implémenté le comportement du nœud à chaque round où il n'avait pas besoin d'attendre un message pour effectuer une action en particulier, ou, au contraire, quand il attendait de recevoir un certain nombre de messages pour pouvoir effectuer la suite de ses actions.

La méthode `void onMessage(Message msg)`, permet, selon le type de message reçu, de procéder à sa gestion grâce aux méthodes correspondantes ayant pour préfixe "handle" (e.g. `handle[TypeMessage]()`)

Par ailleurs, nous avons aussi ajouté un ensemble de très petites classes qui ont servi pour envoyer et surtout gérer la réception des messages. Certaines de ces classes n'ont pas d'autres intérêt que de dire "je suis tel message". En outre, certaines sont complètement vides.

Pour les messages dont les classes ne sont pas vides, leurs attributs constituent les paramètres des messages envoyés. Par exemple, pour le message `Frag`, le paramètre est celui du nouveau fragment que nous allons recevoir.

1.2 Codes couleurs

Quand l'algorithme tourne, il y a un code couleur des arêtes et des sommets. Ces couleurs changent en fonction de l'état dans lequel les sommets et arêtes sont.

Le code couleur est le suivant :

Pour les arêtes :

Noire	arête libre (pas dans l'arbre couvrant)
Rouge	arête prise (dans l'arbre couvrant)

Pour les sommets :

Noir	sommet dans l'état `PULSE`
Rouge	sommet dans l'état `CAST`
Jaune	sommet dans l'état `MIN`
Vert	sommet dans l'état `DONE`
Magenta	sommet dans l'état `ROUND_FINISHED`
Bleu	sommet dans l'état `READY`

1.3 Remarques sur l'implémentation

Avec un peu de recul, on se rend compte que le code de la classe `BasicNode` est assez lourd. En effet, avec plus de 400 lignes, on se perd facilement. Une autre façon de faire aurait été de transférer le code qui gère la réception des messages directement dans les classes se trouvant dans le package `MessageObjects` que nous avons introduites (et qui sont, pour certaines, vides). Se faisant, on aurait pu économiser de la place dans `BasicNode`et ainsi avoir une meilleure visibilité de notre code.

Une autre erreur que nous avons faite est le fait d'avoir voulu suivre aveuglément l'algorithme présenté dans le cours. En procédant ainsi, nous avons eu beaucoup de mal à trouver les points qui ont péchés dans notre code et à résoudre les erreurs que nous avons commises.

2 Problèmes rencontrés

L'implémentation d'un tel algorithme n'étant pas triviale, nous avons réussi à faire un certain nombre de round, mais dans certains cas, l'algorithme bloque sans avoir trouvé un arbre couvrant. De plus, la forêt qui se crée ne semble pas être de poids minimum à certains endroits.

Un point positif que l'on peut remarquer, c'est qu'on ne détecte pas de création de cycles : on est toujours en train de créer des arbres et on a un sous-graphe induit par tous les sommets qui donnent une forêt (tous les sommets font partie d'un arbre, on a pas de sommets isolés).

D'après les observations que l'on a pu faire, les sommets ont pour père certains de leurs fils ($pere[u] \in fils[u]$), ou encore certains sommets en milieux de fragment se retrouvent sans père ni fils, et se considérant comme étant leurs propres pères (comme dans le cas du premier round où tous les sommets sont

leurs propres pères, par convention). Il y a donc très certainement une confusion au moment de fusionner les différents fragments, ou avant quand on fait les inversion père-fils.

Néanmoins, trouver la raison d'un tel bug est difficile, et nous avons dû rendre le projet tel quel, non fonctionnel, par manque de temps pour le terminer.