Algorithmes Distribués

6 décembre 2017 - v2

Master2 Informatique

Ce devoir est à rendre par courriel à gavoille@labri.fr au plus tard ***mardi 2 janvier 2018****. Il doit se présenter sous la forme d'un seul fichier d'archive (.tgz ou .zip) contenant un compte-rendu (.pdf) et vos différents fichiers de programmation, voir d'exemples. Important : précisez bien vos noms et prénoms en première page.*

Objectifs. L'objectif de ce devoir qui pourra être réalisé en binôme, est de programmer l'algorithme discuté en cours sur la construction d'un arbre couvrant de poids minimum (MST). L'algorithme à programmer doit être basé sur celui vu en cours. Il s'agit de la version synchrone de GHS qui consiste à fusionner tant que possible des fragments (= des sous-arbres du MST) afin d'obtenir *in fine* un seul fragment couvrant tous les sommets du graphe. La fusion s'effectue à partir de l'arête sortant de coût minimum de chaque fragment. Une des difficultés de l'implémentation distribuée de l'algorithme est qu'au bout d'un moment, les fragments peuvent être de taille très diverse, ce qui fait que les différentes étapes, notamment la demande fusion, peuvent d'exécuter dans chaque fragments dans un nombre de rondes variables.

Vous devez programmer votre algorithme en Java avec la bibliothèque `jbotsim`. Le poids de chaque arête devra correspondre à la distance entre ces extrémités dessinées sur le plan, c'est-à-dire à sa longueur. Cette longueur est donnée par propriété `.length()`. Il sera impératif de valider vos algorithmes en les testant sur différentes topologies sur lesquelles vous connaissez le comportement de votre algorithme (par exemple un arbre). Pour cela vous pourrez utiliser l'éditeur mais aussi le générateur de graphes `gengraph`. Enfin, vous devrez fournir un compte-rendu assez court, 2 à 4 pages environ, décrivant le principe de votre algorithme ainsi que les expériences que vous avez menée.

Vous trouverez ci-dessous quelques liens utiles.

Le sujet : `http://dept-info.labri.fr/~gavoille/AD/UE-AD.html`
La bibliothèque Java : `http://jbotsim.sf.net`
Le générateur de graphes : `http://dept-info.labri.fr/~gavoille/gengraph.c`
L'aide de `gengraph` : `http://dept-info.labri.fr/~gavoille/gengraph.html`
Un IDE conseillé : `https://www.jetbrains.com/idea/download/`

Dans `jbotsim`, bibliothèque développée au LaBRI, faite attention que le cycle d'exécution (une ronde synchrone, le mode par défaut) ne suit pas exactement la même convention que celui vu en cours. Dans le cours, le cycle est une répétition de SEND / RECEIVE / COMPUTE. Dans `jbotsim` c'est plutôt une initialisation avec un tout premier SEND puis une répétition de cycle RECEIVE / COMPUTE / SEND (ici on réagit à la réception synchrone de messages). La différence est donc que les messages réçus sont ceux envoyés de la ronde précédante. Dans le cours, on reçoit en fin de ronde les messages envoyés au début de la

ronde courante. Ainsi il faut une ronde pour connaître les identifants de ses voisins, dans `jbotsim`, il en faut deux.

Pour créer la topologie (c'est-à-dire le graphe) vous pouvez utiliser les fonctions natives de `jbotsim` comme `TopologyGenerator.generateRing()` pour créer un cycle. Par défaut, les nœuds démarrent l'algorithme dès qu'ils sont ajoutés. Pour éviter ceci, il faut créer la topologie avec `false` comme paramètre par défaut, suivit d'un `.start()` explicite sur la toppologie pour que tous démarrent de manière synchrone.

Il est aussi possible d'importer n'importe quelle topologie avec `importGraph()` précédemment générée. Pour cela vous utiliserez `gengraph` un petit programme `.c` développé au LaBRI. Orienté ligne de commande, il vous permettra de générer de nombreuses topologies standards comme des cycles, des arbres, des graphes aléatoires, *etc.*, le tout au format GraphViz (`.dot`) compréhensible par `jbotsim`.

Par exemple :

```
> ./gengraph cycle 10 -width 1 -format dot > test.dot
```

génère au format `.dot` un cycle à 10 sommets dont les identifiants vont de 0 à 9. En ajoutant l'option `-permute` il est possible de permuter les identifiants aléatoirement, ce qui permet de se passer de l'option `shuffleNodeIds()`. En mettant `-seed 17 -permute` comme option à `gengraph` vous pouvez permuter aléatoirement tout en fixant la graine (ici à `17`).

```
> ./gengraph
> ./gengraph -list
> ./gengraph fdrg
```

affiche respectivement l'aide, la liste des graphes disponibles, et l'aide sur un graphe donné, ici un graphe aléatoire ayant une séquence de degrés fixés. Avec `./gengraph fdrg 20 3 . -visu` vous obtiendrez un fichier `g.pdf` contenant un dessin du graphe. Parmi les graphes qui pourront vous servir : `cycle`, `tree`, `clique`, `expander`, `fdrg`, `gabriel`, `random`, ...

FIN.