Algorithmes Distribués

16 novembre 2015

Master2 Informatique

Ce devoir est à rendre par courriel à gavoille@labri.fr au plus tard ***lundi 21 décembre 2015*** *minuit. Il doit se présenter sous la forme d'un seul fichier d'archive (.tgz ou .zip) contenant un rapport (.pdf) et vos différents fichiers de programmation, voir d'exemples. Important : précisez bien votre nom et prénom sur la première page du rapport, éventuellement dans le nom de l'archive. La note finale tiendra compte du rapport (de la présentation des algorithmes, des expérimentations, des justifications) et du code.*

Objectifs. L'objectif de ce devoir individuel est de concevoir et de programmer des algorithmes distribués de coloration en partie vue en cours. On considère le modèle LOCAL dont les caractéristique essentielles sont : absence de panne, mode synchrone, sommets avec identifiant, aucune limite sur la taille des messages.

Vous devez programmer vos algorithmes en Java avec la bibliothèque `jbotsim`. Il sera impératif de valider vos algorithmes en les testant sur différentes topologies en utilisant le générateur de graphes `gengraph`. Enfin, vous devrez fournir un rapport de quelques pages décrivant vos algorithmes et leurs principes ainsi que les expériences que vous avez menée. En particulier, pour pourrez discuter de la complexité théorique de votre algorithme par rapport aux résultats expérimentaux.

Coloration des cycles. Dans un premier temps il vous est demandé de concevoir, de programmer et d'expérimenter un algorithme de 3-coloration pour une cycle non orienté. À la différence de l'algorithme vu en cours, chaque processeur n'a accès ni à son prédécesseur ni à son successeur. L'idée de se ramener au cas des 1-orientations et de gérer les conflits éventuels en fin d'algorithme.

Coloration d'un graphe arbitraire. Dans un deuxième temps il vous est demandé de concevoir, de programmer et d'expérimenter un algorithme de $(\Delta + 1)$-coloration pour un graphe général G, où Δ est le degré maximum du graphe. Cette fois, l'idée est de calculer une k-orientation, pour un k assez grand (en fait $k = \Delta$ suffit), puis d'exécuter en parallèle l'algorithme pour les 1-orientations. Le nombre de couleurs visées est alors obtenu par réduction successives.

Complément d'informations. Vous trouverez ci-dessous quelques liens utiles.

Le sujet et les sources : `http://dept-info.labri.fr/~gavoille/UE-AD.html`
La bibliothèque Java : `http://jbotsim.sf.net`

Le générateur de graphes : `http://dept-info.labri.fr/~gavoille/gengraph.c`
Un IDE conseillé : `https://www.jetbrains.com/idea/download/`

Dans les sources vous trouverez des fichiers vous permettant de démarrer plus facile votre projet. Dans un premier temps, il vous faudra écrire `Helper.java` regroupant les fonctions locales de calcul des sommets, notamment la fonction PosDiff(x, y).

Dans `jbotsim`, bibliothèque développée au LaBRI, faite attention que le cycle d'exécution (une ronde) 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 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. Mais il est aussi possible d'importer n'importe quelle topologie avec `importGraph()` précédemment générée. Pour cela vous utiliserez un petit programme `.c` développé au LaBRI, `gengraph`. Orienté ligne de commande, il vous permettra de générer des cycles, des arbres, des graphes aléatoire, *etc.*, le tout dans le format GraphViz `.dot` compréhensible par `jbotsim`.

Quelques exemples :

```
> ./gengraph cycle 10 -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 0 -permute` vous pouvez permuter aléatoirement tout en fixant la graine.

```
> ./gengraph
> ./gengraph -list
> ./gengraph gabriel
```

affiche respectivement l'aide, la liste des graphes disponibles, et l'aide sur un graphe donné, ici le graphe de Gabriel. Avec `./gengraph grabiel 50 -visu` vous obtiendrez un fichier `g.pdf` contenant un dessin du graphe. Parmi les graphes qui pourront vous servir : `cycle`, `tree`, `expander`, `bdrg`, `gabriel`, `random`, ...

FIN.