Algorithmes Distribués

12 janvier 2017

Master2 Informatique

Ce devoir est à rendre par courriel à gavoille@labri.fr au plus tard ***jeudi 26 janvier 2017*** *à minuit. Il doit se présenter sous la forme d'un seul fichier d'archive (.tgz ou .zip) contenant un cours compte-rendu (.pdf ou .txt) 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 de $(\Delta+1)$-coloration pour un graphe G de degré maximum Δ arbitraire. 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. On rappelle que l'idée de l'algorithme est de calculer une k-orientation, pour un k assez grand (en fait $k=\Delta$ ainsi qu'une ronde de calcul suffisent), puis d'exécuter en parallèle l'algorithme pour chacune des 1-orientations induites par l'orientation. Le nombre de couleurs visées est alors obtenu par réduction successives.

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 compte-rendu très court, d'une à deux pages décrivant le principe de votre algorithme ainsi que les expériences que vous avez menée.

Vous trouverez ci-dessous quelques liens utiles.

Le sujet et les sources : `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`
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 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. 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 -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 0 -permute` vous pouvez permuter aléatoirement tout en fixant la graine.

```
> ./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`, `expander`, `fdrg`, `gabriel`, `random`, ...

FIN.