Algorithmes Distribués

26 novembre 2019 - v2

Master2 Informatique

Ce devoir est à rendre par courriel à gavoille@labri.fr au plus tard ***mercredi 8 janvier 2020****. 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. Il pourra être réalisé seul ou en binôme. Important : précisez bien vos noms et prénoms en première page.*

Objectif. Il est programmer des algorithmes distribués de coloration en partie vue en cours et disponible dans les notes de 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 votre algorithme en Java avec la bibliothèque `jbotsim`. Il est impératif de valider vos algorithmes en les testant sur différentes topologies sur lesquelles vous connaissez le comportement de votre algorithme. Cela pourra être ici un graphe régulier, où tous les degrés sont les mêmes. Pour cela vous pourrez utiliser l'éditeur de `jbotsim` mais aussi le générateur de graphes `gengraph`. Enfin, vous devrez fournir un compte-rendu de quelques pages, décrivant le principe de votre algorithme ainsi que les expériences que vous avez menée.

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 suffisamment 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.

Quelques liens utiles.

- http://dept-info.labri.fr/~gavoille/UE-AD/
- https://jbotsim.io/
- http://dept-info.labri.fr/~gavoille/gengraph.c
- http://dept-info.labri.fr/~gavoille/gengraph.html

À propos de JBotSim. Dans cette bibliothèque développée au LaBRI, 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 précédente. 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.

Dans le context du modèle LOCAL, la façon la plus simple de programme JBotSim est de surcharger la méthode `onClock()` qui est appelée au début de chaque nouvelle ronde. C'est l'équivalent du NEWPULSE du cours. Il faut typiquement : (1) commencer cette méthode par un `getMailbox()` qui renvoie la liste des messages reçus à la ronde précédente ; (2) effectuer un calcul et préparer les nouveaux messages à envoyer ; et (3) envoyer les messages à certains de ces voisins avec `send(Node,Message)`, ou le même message à tous ces voisins avec `sendAll(Message)`.

Pour démarrer et appliquer l'algorithme, les nœuds peuvent avoir besoin de certaines connaissances globales, comme par exemple le nombre sommets du graphe, son degré maximum, une orientation spécifique des arêtes, etc. Il faut dans ce cas ajouter à la classe `Node` des variables globales, disons `dmax`, avec le type `static`. Puis dans le `main()`, après avoir charger la topologie (le graphe G), vous pouvez effectuer le traitement que vous voulez sur G (donc avant que commence l'algorithme distribué), calculer par exemple son degré maximum Δ, et enfin affecter `Node.dmax =` Δ. Ainsi chaque nœuds du graphe (instance de `Node`) disposera de cette donnée. Bien sûr, il ne faut pas profiter de ces variables `static` (donc globale) pour tricher et rendre l'algorithme complètement séquentiel. Un sommet peut accéder à son identifiant grâce à `getID()`.

À propos de Gengraph. Il est possible d'importer n'importe quelle topologie précédemment générée au format `dot`. Pour cela vous utiliserez gengraph un petit programme `C` développé au LaBRI. Orienté ligne de commande, il vous permettra de générer de nombreuses topologies comme des cycles, des arbres, des graphes réguliers aléatoires, etc., le tout au format GraphViz (`.dot`) compréhensible par `jbotsim`. Pour charger un graphe précédemment généré au format `dot`, il suffit de cliquer (clic droit) sur la fenêtre de l'interface et d'aller chercher le fichier `.dot`.

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 puisque sinon les identifiants seront toujours les mêmes. En ajoutant `-seed 7 -permute` aux options de `gengraph` vous pouvez permuter aléatoirement les identifiants tout en fixant la graine (ici

à 7).

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

affiche respectivement l'aide en ligne, la liste des graphes disponibles (plus de 230 ...), et l'aide sur un graphe donné, ici un graphe aléatoire ayant une séquence de degrés fixés. Avec

```
> ./gengraph fdrg 20 3 4 1 . -visu
```

vous obtiendrez un fichier `g.pdf` contenant un dessin du graphe.

Parmi les graphes qui pourront vous servir : `path`, `cycle`, `tree`, `clique`, `cubic`, `fdrg`, `gabriel`, `mesh`, `expander`, `random`, ... Si le dessin de JBotSim n'est pas extraordinaire, ce qui est probable, faites dessiner le graphe par `dot` qui codera les coordonnées des nœuds directement dans le fichier, avec par exemple :

```
> ./gengraph mesh 2 5 -format dot | dot -Kneato > g.dot
```

Ce ne sont que des suggestions. C'est à vous de trouver des exemples pertinents. Enfin,

```
> ./gengraph ? planaire
```

vous donnera une liste des graphes où le mot `planaire` figure dans l'aide, soit environ 80 réponses.

FIN.