BRUNEAU Richard p1402059

Sujet F

Exercice 1:

Question 1:

Déroulons l'algortihme sur une grille comme celle ci-dessous.

```
0 -- 1 -- 2 -- 3
|    |    |    |
4 -- 5 -- 6 -- 7
|    |    |    |
8 -- 9 -- 10 - 11
|    |    |    |
12 - 13 - 14 - 15
```

Etape 1 : Chaque noeud envoie son id et l'id de ses voisins

id envoie à voisin : (id, [voisin du haut, voisin du bas, voisin gauche, voisin droit])

0 envoie à 4 : (0,[-1, 4, -1, 1]) //-1 signifie qu'il n'y pas de voisin
0 envoie à 1 : (0,[-1, 4, -1, 1])
1 envoie à 5 : (1,[-1, 5, 0, 2])
1 envoie à 0 : (1,[-1, 5, 0, 2])
1 envoie à 2 : (1,[-1, 5, 0, 2])
et ainsi de suite, au milieu de la grille on aura
5 envoie à 1 : (5,[1, 9, 4, 6])
5 envoie à 9 : (5,[1, 9, 4, 6])
5 envoie à 4 : (5,[1, 9, 4, 6])
5 envoie à 6 : (5,[1, 9, 4, 6])
toujours la même logique

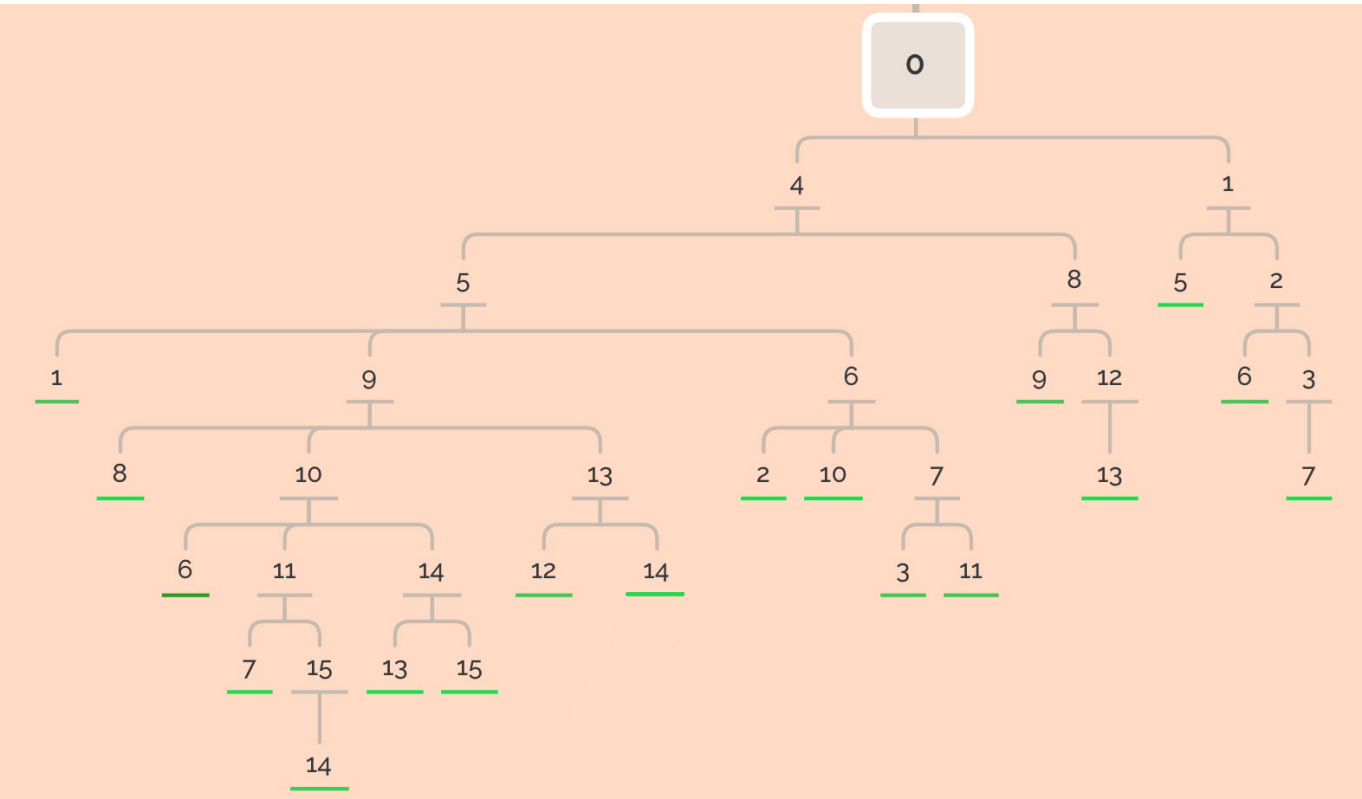
Etape 2 : Chaque noeud reçoit pour la première fois une paire (id, liste de voisin)

Le noeud qui reçoit met à jour sa représentation du graphe.

- En mettant à jour sa liste de noeuds
- En mettant à jour la liste de lien du noeud reçu.

Puis retransmet à chacun de ses voisins l'information (sauf à celui qui vient de la lui envoyer).
Dans notre cas, ça donnerait la situation suivante pour le noeud 0.
0 envoie à 1 et à 4 sa paire (id + liste de voisin).
1 envoie la paire de 0 à 2 et 5.
4 envoie la paire de 0 à 5 et 8

2 envoie la paire de 0 à 3 et 6
5 envoie la paire de 0 à 4 8
8 envoie la paire de 0 à 5 et 8



Nous pouvons tracer le même genre d'arbre pour chacun des noeuds du graphe.

Question 2:

D'après l'algorithme : Un noeud sait qu'il a terminé quand la taille de sa liste de noeuds est égale au nombre de listes dans sa liste qui recence les paires "ID(k), ID-voisins(k)".

C'est également ce que j'aurais implémenté si la classe liste fournissait un moyen de récupérer sa taille.

Dans mon code, j'ai utilisé un tableau pour stocké la grille que j'ai initialisé en mettant toutes les valeurs à -2. Je continue d'iterer tant que j'ai des valeurs à -2 dans mon tableau.

Question 3:

Si nous avions utilisé un langage avec des structures de données complètes, c'est à dire des listes qui donne la taille de la liste, j'aurais utilisé des listes. Or, ici, que ce soit les DList ou les SList, ce n'est pas le cas.

J'ai donc utilisé un tableau pour stocker tout les noeuds découverts par le noeud qui remplit le tableau et un autre tableau, 2D celui-ci, pour connaitre les voisins de chaque noeuds.

Question 4 :

(voir code, fichier exercice2.d)

Question 5 :

Sur une moyenne de 1000 executions :

Pour une grille de 4 x 4 il y a 2 005 messages échangés. Le minimum est de 1 026 messages alors que le maximum est de 7 235 messages.

La consigne demandait de réaliser cette moyenne sur des grilles 10x10 et 50x50, malheureusement, mon PC ne peut pas réalisé ces tentatives. Au maximum, j'ai pu faire sur une grille 7 x 7. Voici les résultats.

Pour une grille de 7 x 7 il y a 280 102 messages échangés. Le minimum est de 132 036 messages alors que le maximum est de 1 308 530 messages.

Cette limite matériel repose sur le processeur, le nombre de coeur dont il est équipé ainsi que les processus déjà en cours.

Exercice 2:

Question 1:

(voir code, fichier exercice2.d)

Question 2:

Sur une moyenne de 1000 executions :
Pour une grille de 4 x 4 il y a 49 messages échangés. Le minimum est de 49 messages alors que le maximum est de 49 messages.

Pour une grille de 7 x 7 il y a 169 messages échangés. Le minimum est de 169 messages alors que le maximum est de 169 messages.

Pour une grille de 10 x 10 il y a 361 messages échangés. Le minimum est de 361 messages alors que le maximum est de 361 messages.

Pour une grille de 45 x 45 il y a 7921 messages échangés. Le minimum est de 7921 messages alors que le maximum est de 7921 messages.

Je me suis arrêté à 45, car j'ai une erreur à la création de thread au-delà. Cette erreur est une limite imposé par l'OS de l'ordinateur.

Question 3:

En rajoutant un temps aléatoire d'environs ~50 millisecondes sur certains noeuds tiré aléatoirement avant l'expedition à certains voisins tirés aussi aléatoirement, il n'y a pas de changement. En effet, le noeud recevra le message, attendra et le transmettra à ces voisins, ne changeant absolument rien au déroulé de l'algorithme.

Sur une moyenne de 1000 executions :
Pour une grille de 4 x 4 il y a 49 messages échangés. Le minimum est de 49 messages alors que le maximum est de 49 messages.

Pour une grille de 10 x 10 il y a 361 messages échangés. Le minimum est de messages 361 alors que le maximum est de 361 messages.

Pour une grille de 45 x 45 il y a 7921 messages échangés. Le minimum est de messages 7921 alors que le maximum est de 7921 messages.

J'ai effectué une variante de ce qui était demandé initialement dans l'énoncé. J'ai utilisé la fonction receiveTimeout qui permet d'attendre la réception d'un message pendant un certain temps. Vous trouverez ce code dans le fichier exercice2sleep2.d. (exercice2sleep2.d). En fonction des messages reçus pendant ce temps d'attente et c'est là qu'entre en compte la variation avec ce qui était demandé, j'envoie au noeud prévu le message uniquement si il ne l'a pas envoyé avant. Je stocke cette information dans un tableau de booléen.

Dans ce cas là, oui, j'ai une variation du nombre de messages. Je vous invite à lire la réponse suivante pour constater les mesures expérimentales que j'ai réalisée.

Question 4 :

Dans le cadre d'une grille 20x20 sur 1 000 executions. J'obtiens en moyenne 1202 messages. Le minimum est de 1169 et le maximum 1249. On obtient un beau gain sur le nombre de message envoyé. En allongeant le temps "d'attente active, on pourrait économiser plus de messages encore, mais cela influera sur le temps total d'execution du programme."

Lors que mon noeud reçoit un message, avant chaque envoie à ses voisins, sauf l'émeteur, le noeud à la possibilité de recevoir un autre messages pendant 50 ms. La source est toujours la même. Le noeud avec l'id égale à 5. C'est une constante dans cette expérimentation afin d'éviter d'avoir trop de facteurs différents.

Exercice 3

Question 1 :

Dans un premier temps, le noeud émetteur (v) émet le message à ses voisins. Quand chaque voisin reçoit le message pour la première fois, il enregistre qu'il a reçu le message de v et il envoie le message à tout ses AUTRES voisins.

Quand chaque noeud reçoit le message une autre fois, il retourne à l'expediteur qu'il l'a déjà reçu.

A chaque fois qu'un noeud reçoit un message d'un de ses voisins il le note. Une fois que tout ses voisins lui ont répondu, il répond au premier noeud qui lui a envoyé le message.

Ainsi, de fil en aiguille, le noeud emetteur saura que tout le monde à reçu le message.

Question 2 :

(voir code, fichier exercice3.d)

Question 3 :

Le nombre de message à augmenté.

Nous avons maintenant, sur une moyenne de 1000 executions :
Pour une grille de 4 x 4 il y a 82 messages échangés. Le minimum est de 82 messages alors que le maximum

est de 88 messages. Je tiens à souligner que j'ai parcouru les résultats pendant l'execution de mon script. J'ai eu quelques fois des nombres au-dessus de 82 tels que 85, 87 et 88. Moins de 20 fois en tout.
Il y a donc eu 82-48 = 34 messages supplémentaires

Pour une grille de 10 x 10 il y a 622 messages échangés. Le minimum est de messages 622 alors que le maximum est de 626 messages. Une fois de plus il y a eu de légères variations.

Pour une grille de 45 x 45 il y a 13 817 messages échangés. Le minimum est de messages 13 817 alors que le maximum est de 13817 messages. Il n'y a pas eu de variations ici.

Question 4 :

Si chaque noeud connait le graphe, l'un des algorithmes possible serait le suivant.

Le noeud emetteur émet le messages à ses voisins. Quand un noeud reçoit le message, il l'acquitte au près du noeud qui lui a envoyé. Ce noeud fait remonter l'acquittement par le noeud qui lui a envoyé le premier message et ainsi de suite.

Comme le noeud émetteur connait le graphe, il a juste à compter les acquitements reçu et en comparant le comptage avec le nombre de noeud.

On pourrait améliorer l'algorithme en incorporant au message d'acquittement l'id du noeud qui envoie l'acquitement. Ainsi le noeud emetteur saurait exactement qui a reçu le message et qui ne l'a pas reçu. Cette amélioration permettrait de ré-emmetre le message uniquement vers les noeuds qui ne l'ont pas reçu.