

M2 – NSD (Practical Work 1)

Etudiants : Adel Zeghlache, Charles Rouillard

Ce document contient les réponses aux questions du premier TP, des remarques ainsi que des exemples d'exécution.

Le tableau ci-dessous montre les fichiers utilisés pour chaque exercice.

Exercice	Fichiers
2	SizeGraph.java
3	DegreeGraph.java
4	CleanGraph.java
5.1	EdgesList.java
5.2	Matrix.java Matrix.cpp (facultatif)
5.3	adjancycastle.cpp
7	adjancycastle.cpp
8	DegreeDist.java plot_graph

Pour compiler tous les programmes Java il suffit d'entrer la commande `javac *.java`

Puis pour chaque exercice :

2: `java SizeGraph <filename> or <path>`

3: `java DegreeGraph <filename> or <path>`

4: `java CleanGraph <filename> or <path>`

5.1: `java EdgesList <filename> or <path>`

5.2: `java Matrix <filename> or <path>`

8: `java DegreeDist <filename> or <path>`

Pour compiler tous les programmes C++, il suffit d'entrer la commande suivante en fonction de l'exercice :

5.2 : `g++ matrix.cpp -o matrix, et puis ./matrix <filename> or <path>`

Exercice 2 : (exécution sur le fichier texte fourni)

'There is 20 edges and 12 nodes'

Exercice 3 : (exécution sur le fichier texte fourni)

The node 1 has a degree of 5
The node 12 has a degree of 2
The node 2 has a degree of 7
The node 3 has a degree of 1
The node 4 has a degree of 2
The node 5 has a degree of 7
The node 6 has a degree of 3
The node 7 has a degree of 2
The node 8 has a degree of 2
The node 9 has a degree of 4
The node 10 has a degree of 3

Exercice 4 : (exécution sur le fichier texte fourni)

From :

1 1
1 2
2 1
2 2
3 2
5 4
6 5
9 8
7 7
8 9
9 5
5 1
2 6
6 5
5 4
2 9
5 10
10 11
10 12
12 11

To:

1 2
3 2
5 1
2 6
5 4
10 11
10 12
6 5
2 9
9 5
9 8
5 10
12 11

Exercice 5.1 :

(1,1)
(1,2)

(2,1)
 (2,2)
 (3,2)
 (5,4)
 (6,5)
 (9,8)
 (7,7)
 (8,9)
 (9,5)
 (5,1)
 (2,6)
 (6,5)
 (5,4)
 (2,9)
 (5,10)
 (10,11)
 (10,12)
 (12,11)

Exercice 5.2 (version C++ et Java)

00000000000000	0110010000 000
0111001001000	0010000000000
0000010000000	0100101001100
0010010000000	0000000100000
0000000001000	0010010010000
0000010000011	0000000000101
0000000000110	

2 versions car problème de mémoire, en C++ le type bool fait 1 octet alors que en Java 8, donc possibilités de tester sur 2 plus grands fichier avec la version C++, mais il reste tout de même très gourmand en mémoire (fichier de plus de 40k nodes pour 8Gb de RAM est le maximum)

Exercice 5.3 : AdjacencyList.cpp

```
Problems Tasks Console Properties
<terminated> (exit value: 0) graph.exe [C/C++ Application] C:\Users\Adel\workspace\graph\Debug\graph.exe (25/09/2017 15:21)
The Neighborhood of 104090 contains: 66108 66285
The Neighborhood of 104091 contains: 66111
The Neighborhood of 104092 contains: 66111
The Neighborhood of 104093 contains: 66141
The Neighborhood of 104094 contains: 66211
The Neighborhood of 104095 contains: 66227
The Neighborhood of 104096 contains: 67410
The Neighborhood of 104097 contains: 67651
The Neighborhood of 104098 contains: 68677
The Neighborhood of 104099 contains: 68748
The Neighborhood of 104100 contains: 68841
The Neighborhood of 104101 contains: 69844
The Neighborhood of 104102 contains: 70103
The Neighborhood of 104103 contains: 71824
*****
There is 1 nodes of degree 0
Volume: 2193083 Edges
Size: 104103 Vertices
Density: 0.000404728
Average degree: 42.1329 edges / vertex
Min Degree: 0 Edges
Max Degree: 2980 Edges
```

Résultat avec le fichier **out.livemocha** : <http://konect.uni-koblenz.de/networks/livemocha>

Exercice 6 : Scalability (EdgesList vs adjancyList vs adjancy Matrix)

Après avoir exécuté les programmes avec des bases de données de tailles différentes nous avons remarqué que certaines implémentations étaient plus optimisées que d'autres en termes de mémoire. En effet avec la liste des Edges nous pouvons aller jusqu'à Nœuds, avec la matrices d'adjacence seulement 40 000 nœuds, et avec la liste d'adjacence jusqu'à 250 000 nœuds.

Ces résultats dépendent bien sûr de nos implémentations. Pour la matrice d'adjacence nous avons, au départ, utilisé un tableau à 2 dimensions de **Ints** mais puisque 1 **Int** est stocké sur **4 octets**, ce n'était pas optimale, nous avons alors utilisé des tableaux de Boolean pour diviser la mémoire occupée par 4. Nous aurions pu améliorer ce résultat en utilisant des matrices de bits.

Concernant les liste d'adjacence nous avons utilisé des listes de vecteurs ce qui est moins optimal qu'un stockage compact avec des (mallocs, reallocs...) mais plus simple à manipuler.

Conclusion :

La matrice d'adjacence nécessite beaucoup de mémoire comparé aux deux autres solutions.

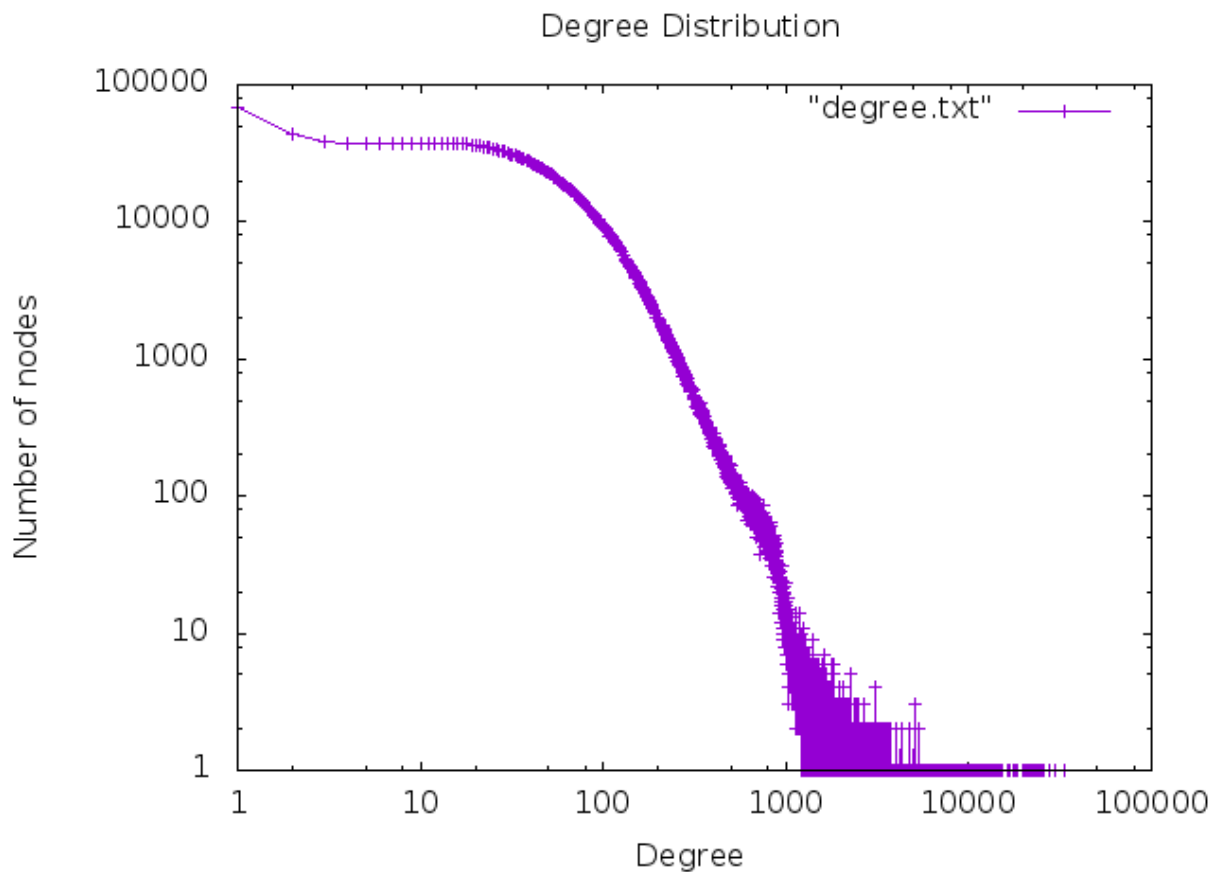
Exercice 7 :

Il n'est pas nécessaire de stocker le graphe pour effectuer ces calculs, mais le fait de stocker le graphe améliore grandement le temps de calcul car nous n'avons plus besoin de parcourir le fichier plusieurs fois.

Nous avons utilisé la liste d'adjacence pour cet exercice, mais nous aurions pu utiliser les autres.

Exercice 8 :

Une fois la commande pour cet exercice effectué, il en sort un fichier texte "degree.txt", il suffit alors d'appeler le script gnuplot 'plot_graph' (fourni dans l'archive) avec la commande "gnuplot plot_graph", il en sors alors un fichier "degree_dist.png". Ci-dessous l'image obtenu pour un fichier a 1 milliards de nœuds (fichier out.orkut-links), résultat mis sur une échelle logarithmique.



Nous pouvons observer qu'en affichage en log, que nous avons une courbe proche d'une loi de puissance. En effet il y a une ligne droite à partir d'un certain seuil (environ 10 000).