

A dark blue vertical bar is on the left. A blue arrow points right from it, containing the text '2018/2019'.

2018/2019

# Table de Routage

Groupe :

- BRAHIMI Lounes
- ADDOUCHE Mouloud

IN403 - Algorithmique de graphe

Mautor Thierry

Several thin, curved lines in dark blue and light grey originate from the left side and curve upwards and to the right.

Licence 2 Informatique

UNIVERSITE DE VERSAILLES SAINT-QUENTIN-EN-YVELINES - UFR DES SCIENCES

## Introduction :

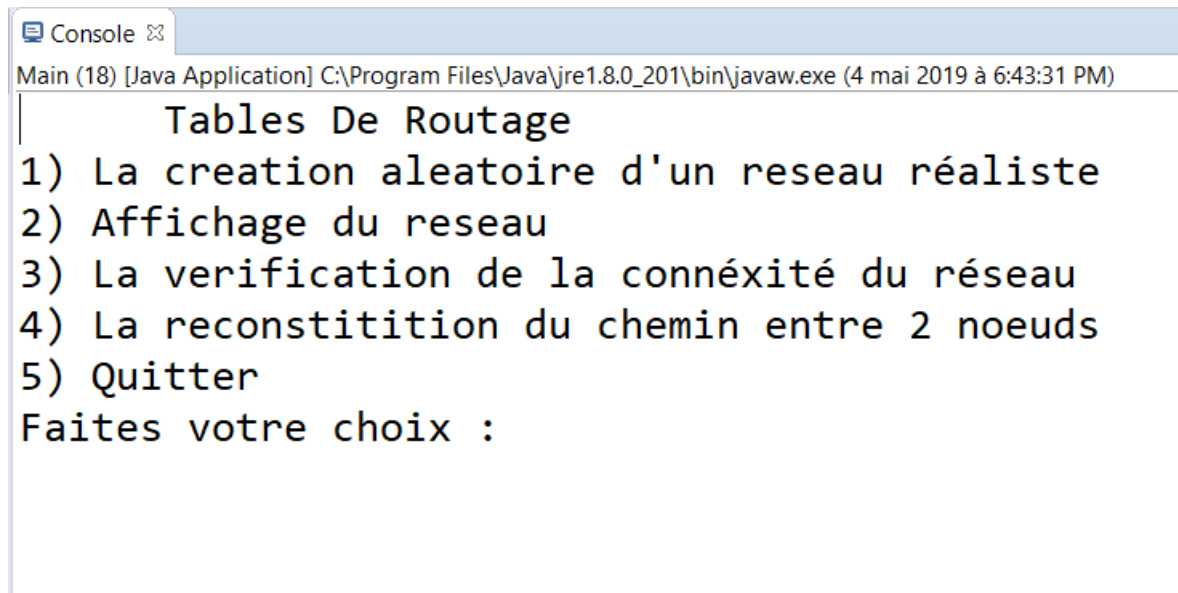
Notre projet est composé de deux versions programmées en java, une qui nous offre un affichage de 100 nœuds sur la console avec une interaction avec l'utilisateur grâce au clavier, une deuxième avec un affichage graphique de 20 nœuds (4 backbone, 6 de type 2, 10 de type 3) avec une interaction grâce à la souris.

Le côté algorithmique en soi est le même, la différence réside dans le fait que dans la première version l'affichage des nœuds et des changements (le plus court chemin) se font sur la console par le biais de `System.out.println()`, dans notre autre version l'affichage est graphique grâce à `JFrame` qui est propre à java et la bibliothèque `JGraphx`.

L'aspect algorithmique des deux versions est le même et est basé sur les mêmes classes : (Graphe Lien Nœud Outils), l'affichage sur la console est généralement basé sur notre class `Main`, dans notre version graphique il se repose sur `Main` et `Clic` qui se servent des classes contenues dans les packages de `JGraphix`.

## L'algorithme du projet et l'affichage console :

Tout commence dans notre méthode principale `main()` ou on affiche sur la console un menu proposant des choix à l'utilisateur :



```
Console
Main (18) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (4 mai 2019 à 6:43:31 PM)

Tables De Routage
1) La creation aleatoire d'un reseau réaliste
2) Affichage du reseau
3) La verification de la connéxité du réseau
4) La reconstitution du chemin entre 2 noeuds
5) Quitter
Faites votre choix :
```

Un Object de type `Graphe` est créé, cette class contient tous les algorithmes principales et de que l'on choisit de créer un réseau la méthode `init_graph()` est appelée elle va se charger de la création des 100 Object de la class `Nœud` qui représente comme son nom l'indique un nœud, elle les crée en leur affectant un type (1 pour backbone, 2 ou 3) et un id (séquentiellement) et elle les affecte dans un tableau de `Nœud`, puis la méthode `creation_Aleatoire()` est appelée qui se charge de créer des Object de type `Lien` qui représente les arrêtes dans notre sémantique entre les différents nœuds et cela de façon aléatoire en respectant le pourcentage de proportion, suit la méthode `Remplir_Table_Routage_Graphe()`

qui se charge de calculer les plus courts chemin de chaque nœud vers tous les sommets et les stocks dans une structure de type HashMap nommée Table\_Routage dans Nœud, ainsi notre reseau de 100 sommets est créé.

```

Console
Main (18) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (4 mai 2019 à 6:58:44 PM)

Tables De Routage
1) La creation aleatoire d'un reseau réaliste
2) Affichage du reseau
3) La verification de la connéxité du réseau
4) La reconstitution du chemin entre 2 noeuds
5) Quitter
Faites votre choix :
1
Reseau Créé
Tables De Routage
1) La creation aleatoire d'un reseau réaliste
2) Affichage du reseau
3) La verification de la connéxité du réseau
4) La reconstitution du chemin entre 2 noeuds
5) Quitter
Faites votre choix :

```

Si l'utilisateur clic sur 2 (Affichage du réseau) grâce a la méthode Affichage\_Noead présente dans Nœud pour chaque sommet on fait une représentation en liste chaîné des sommets adjacents avec les poids des arrêtes entre parenthèses.

```

Console
Main (18) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (4 mai 2019 à 7:15:27 PM)
N13 : 1(14) + 3(14) + 12(15) + 14(20) + 15(11) + 33(28) + 34(33) + 35(46) + 36(24) + 46(48) + 55(42) + 71(21) + 72(19) + 8
N14 : 1(13) + 12(12) + 13(20) + 15(11) + 32(44) + 69(32) + 70(25) + 71(33) + 77(25) + 89(50)
N15 : 2(16) + 3(18) + 13(11) + 14(11) + 32(19) + 66(40) + 68(24) + 82(30) + 83(18) + 93(30)
N16 : 2(15) + 17(10) + 18(17) + 48(33) + 51(18) + 54(31) + 56(22) + 67(19) + 82(33) + 88(23)
N17 : 7(16) + 16(10) + 18(20) + 70(50) + 78(49) + 80(37)
N18 : 5(20) + 16(17) + 17(20) + 19(14) + 43(35) + 52(41) + 62(31) + 84(18) + 93(30)
N19 : 6(16) + 7(11) + 18(14) + 20(13) + 21(19) + 37(27) + 44(36) + 58(19) + 80(48) + 84(18) + 92(31)
N20 : 7(17) + 19(13) + 21(19) + 30(31) + 34(25) + 38(36) + 39(44) + 40(45) + 45(37) + 63(50) + 74(28) + 79(22) + 89(50)
N21 : 6(11) + 8(16) + 19(19) + 20(19) + 44(36) + 49(29) + 51(24) + 72(35) + 74(40) + 76(25) + 87(44) + 95(20) + 97(18) + 1
N22 : 3(17) + 23(12) + 24(11) + 25(18) + 41(19) + 42(41) + 45(17) + 48(36) + 56(34) + 58(29) + 63(43) + 75(29) + 88(45) +
N23 : 8(20) + 22(12) + 24(17) + 29(21) + 35(38) + 40(19) + 55(26) + 57(23) + 61(34) + 65(49) + 87(28) + 98(48) + 100(49)
N24 : 2(20) + 3(15) + 22(11) + 23(17) + 25(13) + 41(46) + 42(25) + 46(34) + 50(38) + 73(26) + 75(46) + 76(43) + 79(30) + 5
N25 : 6(11) + 8(12) + 22(18) + 24(13) + 26(17) + 43(22) + 49(50) + 61(18) + 81(19) + 90(31)
N26 : 4(13) + 25(17) + 27(12) + 47(18) + 60(22) + 67(28) + 68(50) + 92(32) + 94(40)
N27 : 1(13) + 6(13) + 26(12) + 28(13) + 52(31) + 53(31) + 64(29) + 90(24) + 95(18) + 96(36)
N28 : 1(16) + 1(15) + 3(19) + 27(13) + 39(43) + 59(29) + 64(24) + 66(36) + 85(45)
N29 : 12(17) + 23(21) + 56(37)
N30 : 10(40) + 20(31) + 40(18)
N31 : 9(31) + 10(37) + 41(43)
N32 : 14(44) + 15(19) + 59(40)
N33 : 11(40) + 13(28) + 57(38)
N34 : 13(33) + 20(25) + 53(33)
N35 : 13(46) + 23(38) + 80(37)
N36 : 11(38) + 13(24) + 60(35)
N37 : 10(36) + 19(27) + 71(20)
N38 : 11(18) + 20(36) + 85(50)
N39 : 20(44) + 28(43) + 47(26)

```

Si l'utilisateur clic sur 3(vérification de la connexité du réseau) la méthode `Parcours_Profondeur_Graph()` qui retourne un entier, si il est connexe la méthode nous renvoie 10 000 car chaque nœud constitue un chemin de 100 sommets.

Si l'utilisateur clic sur 4(la reconstitution du chemin entre les 2 nœuds) l'algorithme fera objet d'une interaction entre la machine et l'utilisateur tel qu'il demandera de lire le Nœud source et la Nœud destination, une fois chose faite la première méthode appelé `Parcourir()` dans `Graphe` affiche tous les chemins possibles entre les deux sommets choisie avec la durée de chaque chemin, enfin la méthode `Affichage_Plus_Court_Chemin()` écrit depuis notre `TableHachage` le plus court chemin calculé à la création du réseau.

```
Console
Main (18) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (4 mai 2019 à 9:52:43 PM)

Tables De Routage
1) La creation aleatoire d'un reseau réaliste
2) Affichage du reseau
3) La verifcation de la connéxité du réseau
4) La reconstitution du chemin entre 2 noeuds
5) Quitter
Faites votre choix :
4
Veuillez Rentrer l'indice du noeud émetteur
2
Veuillez Rentrer l'indice du noeud destinataire
9
Les chemins possibles :
2 -> 3 -> 11 -> 9 = 39
2 -> 6 -> 10 -> 9 = 39
2 -> 9 = 16

Le plus court chemin : 2 -> 9
```

Enfin si l'utilisateur insert 5 le programme se termine.

```
5) Quitter
Faites votre choix :
5
Programme Terminé
```

# La version affichage graphique :

Les grands changements par rapport à la version précédente (console) se résument dans l'apparition de la classe Clic et une class Main qui change complètement et l'ajout de la bibliothèque JGraphx sous forme de packages qui contiennent les class qui permettent notre affichage.

Pour une meilleur représentation et visibilité ons a fait l'affichage graphique sur 20 nœuds sans changer l'aspect algorithmique qui est le même dans les deux versions.

Tout commence dans la méthode main, une instance de l'Object Graphe est instanciée tout comme dans la version console on appelle les méthodes Init\_Graphe(), Creation\_Aleatoire() et Remplir\_Table\_Routage\_Graphe() qui se chargent comme indiqué dans le premier paragraphe de **L'algorithme du graphe et l'affichage console** de crée le réseau et de calculer les plus courts chemins pour les stocker dans une structure HashMap.

Un Object de type Clic (qui hérite de JFrame de la bibliothèque java) est instancié c'est un constructeur a 1 paramètre à qui on fait passé notre Graphe, un Object de type mxGraph est instancié on la nommé graphe, on a utilisé la méthode insert vertex pour afficher des ellipses qui représentent nos sommets, puis on boucle 20 fois pour crée les arrêtes avec tous les sommets adjacents de chaque nœud, quand appel la méthode Dijkstra\_Graphique() depuis le main un mouseListener est ajouter à notre fenêtr( pour récupérer le clic de la souris), dé que l'utilisateur clic sur deux sommets différents on recupère le plus court chemin dans notre table de hachage et on change la couleur des sommets en rouge pour une meilleur illustration.

