Projet de Graphes et Algorithmes

Licence 3 MIAGE

Réalisé par :

Ahmed Amine DAKHLI

Asmae YASSINE

Salma Insaf BENAMARA

Fatima Zohra BENACEUR

Encadré par :

M. ELBAZ

Table des matières

Introduction

La notion de graphes apparaît avec les travaux d'Euler sur un problème des ponts qui est devenu très célèbre et depuis la théorie des graphes s'est développée dans divers disciplines telle que la chimie, la biologie, les sciences sociales, les réseaux de communications.

Depuis le 20 e siècle la théorie des graphes constitue une branche à part entière des mathématiques, grâce aux travaux de König, Menger, Cayley puis de berge et d'Erdös.

De manière générale, un graphe permet de représenter la structure, la connexion d'un ensemble complexe en exprimant les relations entre ses éléments : réseaux de communication, réseaux routiers, interaction de diverses espèces animales, circuit électrique….

Les graphes constituent donc une méthode pensée qui permet de modéliser une grande variété de problèmes en se ramenant à l'étude de sommet et d'arcs. Les derniers travaux en théorie des graphes sont souvent effectués par des informaticiens, du fait de l'importance qu'y revêt l'aspect algorithmique.

1. Présentation du sujet :

Dans le cadre de notre 3ème année de licence Méthode Informatiques Appliquées à la Gestion d'Entreprise et licence Informatique notre professeur de module Graphes Algorithmes a demandé de réaliser par groupe un projet de graphes.

Le but de ce projet est de programmer un certain nombre d'algorithmes vus en cours, il est question de créer un logiciel, avec une interface graphique simple permettent de manipuler la notion de graphes.

Le projet doit être codé dans le langage de programmation C++ ou java.

Le chef de projet doit superviser la préparation est la mise en place des différentes classes nécessaires à une programmation efficace de tous les algorithmes. Une analyse détaillée doit être présentée.

Nous avons tenu compte du fait que le graphe peut représenter une situation où les sommets ne sont pas réduits à un entier mais peuvent être des enregistrements complexes qui, néanmoins, sont identifiés par des clés uniques allant de 1 à n (ordre du graphe).

Les différentes opérations comme le calcul des distances, détermination du rang, la détermination des composantes fortement connexes selon Tarjan, le problème selon d'ordonnancement, les chemins les plus courts selon Dijkstra, la détermination d'un arbre recouvrant minimal d'un graphe non orienté Kruskal, et le codage de Prüfer, doivent être implémentés selon les algorithmes étudiés en cours.

Cela nous amène à dire que l'application doit pouvoir traiter de différents graphes (orientés, non orienté, valué, connexe).

Le graphe doit pouvoir être défini sous différentes formes :

- Matrice d'adjacence.
- File de successeurs (fs) et adresses du premier successeur (aps).
- Nous devons aussi pouvoir passer d'une forme à l'autre.

Enfin, il nous est demandé de préparer des exemples concrets d'application de certains algorithmes tels que la gestion de tâches (problème d'ordonnancement) ou la détermination des chemins les plus courts et la détermination d'un arbre recouvrant minimal d'un graphe non orienté selon Kruskal.

2. Cahier des charges :

2.1. Descriptif fonctionnel :

a. ***La saisie :***

La saisie d'un graphe doit pouvoir se faire à partir :

- Du clavier par l'intermédiaire d'une zone de saisie
- D'un fichier contenant toutes les informations permettant de créer un graphe.
- De la souris (en utilisant le logiciel)

b. ***L'affichage :***

En effet, il est aussi possible d'afficher un graphe. Pour se faire il est possible de l'afficher selon trois manières :

- Sur l'écran alphanumérique
- Dans un fichier, ayant le même format qu'un fichier de saisie de graphe
- Graphiquement

c. ***Calcul des distances :***

Il s'agit de calculer la distance entre un sommet donné et tous les autres sommets du graphe. Afin de calculer cette distance, on utilise un parcours en largeur du graphe (algorithme vu en cours). Le parcours en largeur consiste à parcourir le graphe en passant par les sommets directement voisins puis ceux dont il faut passer par un sommet pour les atteindre, et ainsi de suite. Les informations à prendre en compte pour ce type de parcours sont :

- Un sommet déjà visité ne doit pas être revisité.
- On explore les sommets successeurs directs.

d. ***Détermination du rang des sommets :***

Comme le calcul des distances on utilise le parcours du graphe en largeur afin de déterminer la longueur maximale d'un chemin arrivant à ce sommet.

Ce rang peut être infini s'il existe un circuit qui passe par ce sommet ou bien s'il existe un chemin arrivant à ce sommet et contenant un circuit.

e. ***Détermination des composantes fortement connexes selon Tarjan :***

Permet de déterminer les composantes fortement connexes d'un graphe orienté et renvoie une partition des sommets du graphe correspondant à ses composantes fortement connexes. Son principe consiste à lancer un parcours en profondeur depuis un sommet arbitraire les sommets explorés sont placés sur une pile, un marquage spécifique permet de distinguer certains sommets : les racine des composantes fortement connexes c'est-à-dire les premiers sommets explorés de chaque composante. Lorsqu'on termine l'exploration d'un sommet racine v, on retire de la pile tous les sommets jusqu'à v inclus. L'ensembles des sommets retirés forme une composante fortement connexe du graphe. S'il reste des sommets non atteints à la fin du parcours, on recommence à partir de l'un d'entre eux.

f. ***Le problème d'ordonnancement :***

Les problèmes d'ordonnancement consistent à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles des délais et d'enchaînement, ils apparaissent dans tous les domaines de l'économie, l'informatique (tâches :jobs ; ressources : processeurs ou mémoires), la construction (suivi de projet), l'industrie (problèmes d'ateliers, gestion de production), l'administration (emplois du temps).

Étant donné un projet qui nécessite la réalisation de plusieurs tâches soumises à différentes contraintes, on cherche à déterminer la durée totale du projet mais aussi l'ordre et le calendrier d'exécution de ces différentes tâches.

Les tâches sont caractérisées par leur durée (le tâche i à une durée di) et les seules contraintes sont des contraintes de successions dans le temps : la tâche i doit être effectué avant la tâche j.

À partir d'un projet donné, on va donc construire le graphe suivant :

- À chaque tâche i (i de 1 à n) on associe un sommet i du graphe.
- On définit un arc entre les sommets i et j de longueur di si la tâche i doit précéder la tâche j.

g. ***L'algorithme de chemin le plus court de Dijkstra :***

Il s'applique à un graphe connexe dont le poids lié aux arêtes est un réel positif.

Cet algorithme permet de calculer un plus court chemin entre un sommet s et tous les autres sommets accessibles du graphe. G= (S, U, C) et s un sommet de départ.

L'idée de l'algorithme est de construire progressivement l'ensemble X des sommets pour lesquels il existe un chemin à partir de sommet S donnée de coût minimal.

L'algorithme mémorise à chaque étape où on trouve un chemin de coût minimal entre sommet S et sommet T, le prédécesseur de sommet T dans un tel chemin et un mettant à jour les distances entre le sommet S et les successeurs de sommet T.

h. *Détermination d'un arbre recouvrant minimal d'un graphe non orienté selon Algorithme de Kruskal :*

Il s'applique sur un graphe orienté connexe et valué.

L'idée de l'algorithme est la suivante ; Initialement, l'ensemble T des arêtes est vide, on ajoute à T des arêtes une par une en choisissant à chaque étape, une arête qui n'est pas dans T celle de coût minimal et qui ne crée pas de cycle dans T.

Lorsque l'on aura ajouté n-1 arêtes, n le nombre de sommet, on a obtenu un ARM.

Pour savoir si une arête (s, t) crée un cycle avec des arêtes de T, il faut savoir s'il existe déjà une chaîne reliant le sommet s à sommet t dans l'arbre en construction. Autrement dit, si s et t appartiennent à la même composante connexe. Si ce n'est pas le cas, l'ajout de (s, t) à T réunit la composante de s à celle de t en une seule.

i. *Codage de Prüfer :*

Le codage de Prüfer est un moyen de décrire un arbre dont les sommets sont numérotés.

Le codage permet de représenter un arbre numéroté de n sommet avec une suite de n-2 terme.

Une suite P = (x1, x2, x3, x4, ...xn-2) donnée correspond à un et un seul arbre numéroté de 1 à n.dans l'informatique le codage de Prüfer est donc utilisé pour enregistrer la structure d'un arbre de façon plus compacte qu'avec les traditionnelles pointures.

2.2. Contraintes techniques :

La première difficulté que nous avions c'est le choix de la structure à mettre en place dans le projet pour pouvoir implémenter les algorithmes que nous avons vu en cours dans certains été difficile à implémenter et il a fallu prendre en considération les capacités chaque membre. On devait aussi faire beaucoup de recherche pour la mise en place et le fonctionnement de l'interface graphique.

2.3. Contraintes temporelles :

Le projet à durée deux mois et le planning nous a permis de développer le projet dans le temps imparti. Néanmoins à défaut de la contrainte temporelle, nous avons été confrontés à la distance séparant les membres de groupes pour les réunions, impliquant des retards dans avancement.

3. Mise en œuvre :

3.1. Répartitions des tâches :

Le diagramme suivant illustre la répartition des tâches :

L'interface graphique	Amine, Asmae, Batoul, Insaf,
Création des classes	Amine, Insaf
Calcul distance et Prüfer	Batoul
L'ordonnancement et le Rang	Asmae
Tarjan et Kruskal	Insaf
Dijkstra et sauvegarde et récupération d'un graphe dans un fichier	Amine
Rédaction du rapport	Asmae, Batoul

3.2. Les outils utilisés :

Eclipse : est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la fondation Eclipse visant à développer un environnement de production de logiciels libre qui soit extensible, universel et polyvalent, en s'appuyant principalement sur Java.

GitHub : est un service web d'hébergement et de gestion de développement de logiciels utilisant le logiciel de gestion de version Git GitHub propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres le site assure également un contrôle d'accès et de fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

3.3. Description principale des classes :

a. ***Les sommets :***

Un sommet est fait d'un couple de coordonnées (x, y) et d'une étiquette. On fait en sorte que les sommets aient toujours une étiquette ; à cet effet ils en reçoivent une lors de leur initialisation, formée à partir de leur rang de création. L'utilisateur peut ensuite remplacer cette étiquette provisoire par une étiquette utile.

Pour bien faire les choses, on a rendu privées toutes les variables de la classe **Sommet** et on a doté cette classe d'accesseurs (méthodes *get...* et *set...*) publiques. Un rapide coup d'œil à la méthode **setEtiquette** montre l'erreur qu'aurait été de donner libre accès à la variable **etiquette**.

La justification de la manière dont la méthode **toString** transforme un sommet en une chaîne de caractères apparaîtra aux étapes 9 et 10 (sauvegarde et restauration d'un graphe).

La variable de classe **tousLesSommets** et les méthodes de classe associées **nombreSommets**, **trouverSommet** et **iterator** permettent de mémoriser et d'utiliser la collection des sommets du graphe.

Fichier **Sommet.java** ;

b. Les arêtes :

Une arête n'est rien de plus que deux références à des sommets qui sont ses extrémités. La variable de classe **toutesLesAretes**, le type collection d'arêtes (la valeur effective est un objet **Vector**) mémorise la liste de toutes les arêtes créées et en permet le parcours.

Fichier **Arete.java** ;

c. Graphe :

Un graphe est composé d'une matrice d'adjacence, un tableau d'arêtes, le tableau fs et le tableau aps. La classe contient tous les algorithmes demandés dans le projet en forme de fonctions ainsi que les méthodes utilisées pour leurs fonctionnements en plus des tableaux et attributs nécessaires pour pouvoir exécuter un des algorithmes ou plusieurs sur le même graphe. On y trouve aussi les méthodes permettant de faciliter l'affichage des résultats finaux de chaque algorithme exécuté.

Fichier **Graphe.java** ;

d. Graphe orienté et non orienté :

Deux classes héritant de la classe graphe qui se différencient à partir de la façon dont est rempli la matrice d'adjacence dans chaque classe. Graphe orienté contient des arêtes qui en connaît forcément leurs sommets de début exact ainsi que leur sommet d'arrivée, contrairement à graphe non orienté qui contient des arêtes qui ont deux sommets sans connaître forcément lequel et le départ et lequel et l'arrivée.

Fichier **Graphe_oriente.java** ; *Fichier* **Graphe_non_oriente.java** ;

e. Etape 00 – Juste un cadre avec un panneau blanc dedans :

Pour commencer, faisons simple : juste un cadre (qui ne bougera plus dans la suite du travail) avec, pour tout contenu, un panneau blanc et vide. Notre travail consistera surtout à écrire la classe de ce panneau, appelée à se développer par la suite.

Nous avons choisi d'écrire notre (petite) application comme la définition d'*une seule classe*. Cela détermine notre choix : nous devons définir la classe du panneau de dessin (une sorte de **JPanel**) au lieu de la classe du cadre principal (un **JFrame**) ou toute autre classe, afin de pouvoir, le moment venu, redéfinir la méthode **paint** chargée du dessin effectif du graphe.

Fichier **Etape00.java** ;

f. Etape 01 – Une barre d'état pour guider l'utilisateur :

Au lieu d'attribuer tout l'espace à l'intérieur du cadre au panneau où on dessinera le graphe, on partage cet espace en deux : en bas (contrainte **BorderLayout.SOUTH**) on place une *barre d'état* destinée à afficher des indications pour guider l'utilisateur ; puisqu'il ne s'agit que d'afficher des textes convenus, un objet **JLabel** (muni d'une bordure « creuse ») convient parfaitement.

Au-dessus on place le panneau de dessin, c'est-à-dire l'objet en cours de construction. Il faut le placer au centre (contrainte **BorderLayout.CENTER**) et non tout en haut (contrainte **BorderLayout.NORTH**) ; de cette manière le panneau de dessin prend tout l'espace disponible et c'est lui qui s'agrandit lorsque la taille du cadre augmente.

Il y a également la possibilité d'écrire « *unJFrame*.**add(***unComposant***)** » date de Java 5. Jusqu'à Java 1.4 il fallait écrire *unJFrame*.**getContentPane().add(***unComposant***)** ou, parfois, *unJFrame*.**setContentPane(***unConteneur***)**.

Fichier **Etape01.java** ;

g. Etape 02 – Une barre de menus :

On crée et on ajoute au cadre une barre de menus portant deux menus, *Fichier* et *Actions*, qui, pour le moment, n'ont aucune action associée.

Fichier **Etape02.java** ;

h. Etape 03 – Introduction de l'état et des textes d'aide associés (les menus agissent dessus) :

Ici on fait en sorte que les items des menus aient un effet, en leur associant des *actions* qui, pour l'instant, ne font que changer la valeur de la variable **etat** (plus tard, la valeur de cette variable exprimera la signification à donner aux actions de la souris : créer un sommet, le déplacer, etc.).

On met également en place (voyez la méthode **rafraichirBarreEtat**) la gestion de la barre d'état, qui affiche désormais le texte d'aide correspondant à l'état du système, ce texte annonce donc l'effet du prochain clic de la souris.

Fichier **Etape03.java ;**

i. *Etape 04 – Détection des clics de la souris pour créer des sommets :*

A partir de maintenant, si l'état courant est **CREATION_SOMMET**, chaque pression d'un bouton de la souris doit produire la création d'un sommet à l'endroit où se trouve alors le curseur. Pour commencer, les sommets créés ne font qu'afficher leurs coordonnées à la console.

Fichier **Etape04.java** ;

j. *Etape 05 – Affichage des sommets :*

On commence ici à s'occuper d'affichage graphique. Par conséquent on voit apparaître la redéfinition de la méthode **public void paint (Graphics g) ;** qui commence par un appel de la méthode héritée (de **JPanel**), cela est nécessaire pour obtenir la coloration du fond du panneau. Ensuite, elle parcourt la liste de tous les sommets (disponible dans la classe **Sommet**), en dessinant un gros point rouge à l'emplacement de chacun.

A propos de la méthode **paint**, seul Java l'appelle. Quand une application souhaite provoquer un tel appel, elle appelle à la place **repaint()**, qui se charge de notifier à la machine java que l'actuelle apparence graphique du composant est obsolète et qu'il y a lieu de la refaire ; la machine java appellera alors **paint**, lorsqu'elle n'aura rien d'autre à faire (les opérations de dessin ont une priorité minimale).

Fichier **Etape05.java** ;

k. *Etape 06 – Affichage des étiquettes et possibilité de les changer :*

Ici on met en place l'affichage des étiquettes (un ajout dans **paint**, facile à comprendre) et, surtout, la possibilité d'en [re-]définir la valeur : quand l'utilisateur manifeste le souhait de définir une étiquette, on lui demande d'indiquer le sommet qu'il vise en cliquant avec la souris. C'est la méthode **sommetVoisin** qui se charge de déterminer le sommet « sur » (ou plutôt, « pas très loin de ») lequel il a cliqué, puis une boîte de saisie acquiert la nouvelle valeur de l'étiquette.

Fichier **Etape06.java** :

l. *Etape 07 – Possibilité de déplacer les sommets :*

Pour pouvoir faire bouger les sommets nous devons nous intéresser aux événements de déplacement de la souris, ou plutôt aux déplacements avec le bouton pressé (cela s'appelle *mouse dragging*). Notre classe doit donc implémenter l'interface **MouseMotionListener**, dont la méthode **mouseDragged** nous intéresse.

Après qu'il a été mis dans l'état « un déplacement de sommet *va avoir lieu* » (**DEBUT_DEPLACEMENT_SOMMET**), le système passe dans l'état « un déplacement de sommet *est en cours* » (**SUITE_DEPLACEMENT_SOMMET**) lorsque l'utilisateur appuie sur le bouton de la souris sur [ou pas très loin d'] un sommet ; le système sortira de cet état quand le bouton de la souris sera relâché.

Pendant que le bouton sera pressé, tout déplacement de la souris produira le déplacement correspondant du sommet sélectionné.

Fichier **Etape07.java** ;

m.Etape 08 – Création et affichage des arêtes :

On crée une arête en sélectionnant successivement deux sommets. C'est encore une opération à deux temps (deux états) : dans l'état **DEBUT_CREATION_ARETE** l'utilisateur doit sélectionner un sommet ; s'il réussit, le système passe dans l'état **SUITE_CREATION_ARETE** dans lequel l'utilisateur doit sélectionner le deuxième sommet ; l'arête est alors créée.

Dans la méthode **paint** on dessine les arêtes *avant* de dessiner les sommets afin que les sommets soient dessinés *par-dessus* les extrémités des arêtes.

Fichier **Etape08.java** ;

n. Etape 09 – Sauvegarde d'un graphe dans un fichier de texte :

Tout se passe dans la méthode **enregistrement**. Après avoir créé le fichier, à l'aide notamment d'une boîte de dialogue pour fixer le nom et le chemin, on écrit dans un **PrintStream** (ce qui permet d'employer la bonne vieille méthode **println**) :

- Le nombre de sommets,
- Pour chaque sommet : l'étiquette (entre guillemets, pour le cas où elle contiendrait des blancs) et les coordonnées
- Le nombre d'arêtes,
- Pour chaque arête, les étiquettes de ses deux extrémités (entre guillemets pour les mêmes raisons).

Fichier **Etape09.java** ;

La manière d'écrire chaque sommet et chaque arête est déterminée par les méthodes **toString** des classes **Sommet** et **Arete**.

o. Etape 10 – Restauration d'un graphe depuis un fichier :

Le travail est fait par la méthode **restauration** et deux méthodes auxiliaires, **prochainMot** et **prochainEntier**, qui allègent considérablement le programme en regroupant l'obtention de la prochaine unité lexicale, la vérification qu'elle a le type attendu et l'éventuelle conversion dans le cas d'un nombre.

Excellente nouvelle, on apprend qu'un objet **StreamTokenizer** traite correctement une chaîne entre guillemets. Plus précisément, sur la rencontre d'une chaîne ainsi encadrée, l'unité lexicale est le caractère « " » (c'est-à-dire la valeur 34) et le **sval** est la chaîne entre les guillemets toute entière.

Conclusion

Ce projet a été pour nous une expérience très enrichissante sur tous les plans : il nous a permis d'étendre nos domaines de compétence, notamment avec l'utilisation de jPanel pour la création de l'interface graphique ainsi que dans la programmation orienté objet en java (eclipse). De plus la notion de graphe en informatique et ses différents domaines d'utilisation.

La bonne entente entre les membres du groupe a facilité le travail sur ce projet, il y avait une coordination et une entre aide entre nous ce qui nous a permis de réaliser le projet de la manière la plus saine possible.