NGUYEN HUU Long

VINDAS Karim

6/12/2012

Projet logiciel : Dessin de graphes

Rapport de conception

**Introduction**

Lorsque l’on veut représenter un graphe, il existe d’autant plus de possibilités qu’il y a de nœuds et d’arêtes. Notre but est de représenter un graphe donné, orienté ou non, d’une façon « claire », c’est-à-dire en évitant un maximum les croisements entre les arêtes et en maintenant une distance à peu près uniforme entre deux nœuds voisins.

Une représentation plus lisible peut aider à la compréhension des relations entre les nœuds d’un graphe, comme ci-dessus.

**Spécifications**

Nous construirons une application qui :

* convertisse un fichier texte contenant toutes les infos sur un graphe en un modèle de ce graphe
* représente ce graphe de façon « claire » à l’aide d’un algorithme de dessin de graphes

La vérification du critère de clarté est la plus difficile à mettre en place. Comme précisé en introduction, nous chercherons à minimiser le nombre de croisements entre arêtes tout ne conservant une distance la plus constante possible entre deux nœuds voisins. Mais nous prendrons en compte le fait que l’algorithme puisse ne pas fonctionner dans des cas particuliers, selon le choix de l’algorithme.

**Conception générale**

Choix des technologies

L’application sera programmée en Java, et l’interface graphique sera faite à partir de la bibliothèque Swing (javax.Swing).

Les fichiers texte seront analysés grâce au système d’I/O natif de Java.

Solutions techniques

Les fichiers texte contenant les informations de graphe auront le format suivant :

[n° nœud de référence] [n° nœud voisin 1] … [n° nœud voisin n]

Les graphes seront composés de nœuds indexés et positionnés connaissant leurs voisins. Les arêtes n’auront donc pas de modèle propre. On travaillera sur des graphes non-orientés car cela ne modifie pas le problème.

La représentation du graphe se fera en deux temps :

* on convertira d’abord le fichier texte en un modèle de graphe, dont les nœuds seront disposés arbitrairement sur une grille carrée
* on représentera ce graphe une première fois (afin d’en avoir une idée)
* on appliquera l’algorithme de dessin de graphes, afin de déterminer les nouvelles positions des nœuds, sur appui d’une touche avant de le représenter dans sa nouvelle forme (on parlera « d’équilibrage de graphe »)

Selon l’algorithme choisi, l’initialisation des positions des nœuds et la possibilité de réitérer l’algorithme peuvent être, ou non, pertinentes.

Debug mode

En attendant l’implémentation de l’interface graphique, les graphes seront représentés textuellement, avec les relations entre les nœuds selon le même format que dans le fichier texte, avec les positions des nœuds en plus.

Choix algorithme de dessin de graphes

Nous choisirons un algorithme de dessin basé sur les forces.

[à compléter]

Par conséquent, le choix des positions initiales est important et l’on peut réitérer l’opération si le critère d’arrêt est temporel [à adapter].

Définition des classes



***DrawingApp***

*Requiert les classes SwingUtilities et JFrame de Swing en import*

C’est la classe responsable de la création de la fenêtre graphique ; elle délègue la suite à la classe *GraphPanel*.

Cette classe contient la fonction *main* du programme, ainsi qu’une fonction *createAndShowGUI* qui permet la création du panel.

Méthodes

*main* : lance un thread appelant *createAndShowGUI*

*createAndShowGUI* : crée une JFrame (fenêtre), y ajoute un *GraphPanel* et adapte la taille de la fenêtre en conséquence

***GraphPanel***

*Hérite de la classe JPanel de Swing*

*Requiert les classes Color, Dimension, Graphics et Event::KeyEvent et Event::KeyAdapter de AWT, ainsi que BorderFactory de Swing (optionnel)*

Cette classe précise les paramètres esthétiques de la zone de représentation du futur graphe, comment répondre aux évènements qui y ont lieu et comment représenter le graphe.

Dans le constructeur de cette classe, on détaille l’esthétique de la zone de travail, on appelle *Graph* à parser le fichier texte pour se construire une instance *graph* et on précise à quels évènements réagir et comment.

La classe contient également la méthode *getPreferredSize (override)* requise pour transmettre les informations de dimension à *DrawingApp* après la création de la fenêtre.

Enfin, elle *override* la méthode *paintComponent*, responsable du tracé du graphe.

Attribut

*graph* : Graph utilisé dans l’application

Méthodes

*constructor* : crée un bord à la zone de travail (optionnel), appelle le constructeur de *Graph* pour un paramètre *String* le nom du fichier contenant les informations sur le graphe (fixe pour l’instant) et ajoute un listener sur l’appui de barre espace (par exemple) ayant pour réponse le lancement de l’équilibrage du graphe (méthode *balance* de l’instance de *Graph*) et le *repaint* (appel de *paintComponent*, peut être cumulé à d’autres) sur l’ensemble de la zone de travail

*getPreferredSize* : renvoie la dimension voulue pour la zone de travail, et donc pour la fenêtre

*paintComponent(Graphics g)* : appelle la fonction éponyme sur *g* puis trace les nœuds (cercles remplis d’une couleur associée à son numéro ou directement son numéro si on importe une bibliothèque pour afficher du texte) et les arêtes selon leurs positions actuelles

**Graph**

Cette classe représente un graphe. Elle contient des informations sur l’ensemble de ses nœuds, et éventuellement une position dans le cas où les nœuds seraient représentés de façon relative dans le plan.

Attributs

*nodes* : ensemble des nœuds du graphe ; type de set à définir, probablement List<>

*x, y* : position globale du graphe (optionnel)

Méthodes

*constructor (File file)* : parse le fichier *file* et construit l’attribut *nodes* en fonction

*getNodes* : renvoie *nodes* (aide *balance*)

*addNode*: ajoute un nœud (aide à la construction)

*balance* : applique l’algorithme de dessin de graphes et actualise les positions des nœuds du graphe en fonction

**Node**

Classe représentant un nœud de graphe. Ne précise pas de lui-même son appartenance à un graphe. (On ne travaille de toute façon qu’avec un seul graphe.)

Attributs

*index* : entier indexant le nœud dans le graphe

*x, y*: floats positionnant le nœud

*neighbours* : ensemble des nœuds voisins (quitte à avoir de la redondance, contrairement au fichier texte)

Méthodes

*getX, getY*

*setXY(float x, float y)* : déplace le nœud en (x,y)

*addNeighbour(Node node)* : ajoute *node* en voisin (ou bien set selon la façon dont on parse le fichier texte)