Université de Bordeaux
Master 2 Bio-Informatique

Année 2018 - 2019

Visualisation d'Information
TP 1 - Manipulation de graphe dans Tulip Framework -

Objectif : Prendre en main l'outil qui sera utilisé pendant les TPs et le projet de ce cours. Vous allez pour cela générer un graphe, le manipuler, le dessiner, le colorier, etc. Tout cela sera programmé directement dans l'interface de Tulip à l'aide de la vue « Python Script ».

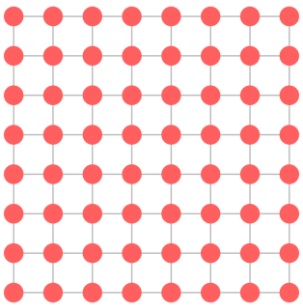
Fichiers fournis : Vous trouverez à l'adresse :
www.labri.fr/perso/bourqui/downloads/cours/Master/2018
les supports de cours et de TP.

Commencez par télécharger le code nécessaire à ce TP.

Documentation : http://tulip.labri.fr/Documentation/current/tulip-python/html/index.html

Partie 1 : Création et manipulation de graphe

La première étape de ce TP consiste à construire une grille régulière. Par exemple,



est une grille régulière 8x8.

Lancer le logiciel Tulip depuis un terminal
/net/ens/tulip/bin/tulip_perspective

Dans l'interface de Tulip, charger un graphe vide (Import>Empty graph) et ouvrez une vue Python Script sur ce graphe (à gauche cliquez sur « Python IDE » puis «Load main script from file» et utilisez le fichier tp_grille.py).

Question 1 : Implémentez la fonction

```
def construireGrille(gr, lignes, colonnes, nodes)
```

qui permet à partir d'un graphe vide `gr` de construire une grille régulière de `lignes` x `colonnes` sommets

et de stocker dans la matrice nodes les sommets de cette grille. Par exemple, nodes[i][j] contiendra le sommet situé à la ligne i et à la colonne j.

Question 2 :

1. Implémentez la fonction

def dessinerModeleForce(gr, layout)

qui permet de dessiner le graphe avec un algorithme par modèle de force (« FM^3 (OGDF) ») où gr est le graphe à dessiner et layout la propriété de position (LayoutProperty) que l'algorithme devra modifier.

2. Définir en utilisant la liste de paramètres de l'algorithme la longueur d'arête désirée (ici 2). Pour cela, vous pouvez récupérer les paramètres d'un algorithme avec :

tlp.getDefaultPluginParameters(<nom_de_l_algo>, [graphe])

Question 3 : Le dessin précédent n'est pas satisfaisant, nous voudrions un dessin régulier (puisque la grille est régulière). Implémentez la fonction :

def dessinerRegulier(nodes, layout, decalageX, decalageY)

qui permet de dessiner le graphe (donné sous la forme de la matrice nodes). decalageX (respectivement decalageY) correspond à l'espace horizontal souhaité entre deux sommets.

Question 4 : Implémentez la fonction :

def calculerDist(root, gr, dist)

qui permet de calculer pour chaque sommet, sa distance au sommet root donné en paramètre. *dist* est une propriété *double* (DoubleProperty) qui associera à chaque sommet cette distance.

Question 5 : Implémentez la fonction :

def colorierGraphe(gr, color, metric)

qui permet de colorier les sommets en fonction d'une mesure.

Partie 2 : Modélisation d'un jeu d'échec

Nous allons maintenant tenter de modéliser un jeu d'échec (de manière évidement naïve) et essayer de voir quelles sont les cases accessibles par quel type de pièce en combien de déplacements. Nous nous limiterons dans ce TP au cavalier et à la reine.

Question 6 : Implémenter la fonction :

def echiquier(nodes, decalage, size, color):

qui permet de modifier la grille afin de la transformer visuellement (couleur, position, taille) en échiquier.

Question 7 : Nous allons maintenant modéliser les déplacements possibles d'un cavalier sur un échiquier. Implémentez la fonction :

def deplacementCavalier(gr, g, nodes):

qui permet de fabriquer un sous-graphe de gr et de l'affecter à g. Les sommets de ce sous-graphe sont les sommets de l'échiquier et 2 sommets sont connectés si et seulement si un cavalier peut se déplacer d'une case a l'autre.

Question 8 : Aux échecs, toutes les cases sont-elles accessibles aux cavaliers ?

Question 9 : Colorier les sommets du graphe en fonction du nombre de déplacements pour atteindre chaque case (pour un cavalier).

Question 10 : Qu’en est-il de la reine ?