Initiation à l'informatique - Premier semestre : TP 3

Type abstrait centré-sommets

– Placez-vous dans le répertoire `TPgraphes` (s'il n'existe pas le créer dans `initinfo`) et téléchargez le fichier `graphes.tar.gz` en tapant la commande suivante dans la fenêtre xterm :
```
wget http://dept-info.labri.fr/ENSEIGNEMENT/INITINFO/ressources-initinfo/graphes.tar.gz
```
– Décompressez le fichier `graphes.tar.gz` en tapant la commande (dans la fenêtre xterm) :
```
tar -xzvf graphes.tar.gz
```

Vous devrez mettre en première ligne de vos fichiers `from vgraph import *`
Dans l'interpréteur, si vous voulez tester vos fonctions sur un graphe de la bibliothèque, vous devrez taper `execfile("bibliotheque.py")`

Exercice 3.1

– Ecrire une fonction qui affiche tous les sommets d'un graphe, qui sont de degré `d` donné.
Utiliser votre fonction pour afficher toutes les stations du graphe tram qui sont à l'extrémité d'une ligne, puis pour afficher toutes les stations qui sont au croisement de deux lignes (sans en être l'extrémité).
– Ecrire une fonction `est_voisin (t,s)` qui renvoie True si `t` et `s` sont voisins, False sinon. Tester votre fonction sur le graphe tram.

Exercice 3.2

– Ecrire une fonction `boucle(s)` qui détermine s'il existe une boucle ayant pour extrémités le sommet `s`.
– Ecrire une fonction `sans_boucle(G)` qui détermine si le graphe `G` est sans boucle.

Exercice 3.3

– Ecrire une fonction `degre_min(G)` qui calcule le degré minimum d'un graphe. Vous utiliserez une variable `min`, initialisée avec le degré du sommet numéroté `0`; pour tout sommet ayant un un degré plus petit que `min`, vous devrez changer la valeur de la variable `min`.
Optimiser votre fonction pour le cas où le graphe possède un sommet de degré nul.
– En utilisant un algorithme similaire à celui de la question précédente, écrire une fonction `sommet_degre_max(G)` qui renvoie le (ou un des) sommet(s) de degré maximum d'un graphe.

Exercice 3.4

– Soit `s` un sommet d'un graphe `G`. Ecrire une fonction `voisin_colore(s,c)` qui renvoie `True` si le sommet `s` a un voisin de couleur `c`.
– Ecrire une fonction `colorie_voisins(s,c)` qui colorie tous les voisins blancs de `s` avec la couleur `c`.
– Ecrire une fonction `efface_couleur(G)` qui remet tous les sommets de `G` en blanc.
– tester vos fonctions sur le graphe `g1` de la bibliothèque.

Exercice 3.5

Ecrire une fonction `bien_colorie(G)` qui renvoie `True` si `G` a tous ses sommets coloriés et si tout sommet `s` a une couleur différente de celles de ses voisins, qui renvoie `False` sinon (les sommets de couleur blanche seront sonsidérés comme non coloriés).

Exercice 3.6
On souhaite implémenter l'algorithme de deux-coloration du TD2. Les deux couleurs seront "red" et "green". Les sommets non coloriés seront blancs.

- Ecrire une fonction `sommet_a_colorie(G)` qui cherche un sommet blanc ayant (au moins) un voisin de couleur et renvoie un tel sommet s'il en existe ou `None` s'il n'en existe pas (on pourra utiliser la fonction `voisin_colore(s,c)` écrite précédemment).
- Implémenter l'algorithme suivant :

```
s = sommet_a_colorie(G)
tant que s est différent de None :
   si s a un voisin vert
   alors colorier s en rouge
   sinon colorier s en vert
   s = sommet_a_colorie(G)
```

Exercice 3.7 *Exercice complémentaire*
Un *graphe k-régulier* est un graphe où chaque sommet est de degré `k`.

- Ecrivez un algorithme qui détermine si un graphe est 3-régulier ou non.
- En modifiant l'algorithme précédent, écrivez un algorithme qui détermine si un graphe est `k`-régulier ou non.
- Quelle est la complexité de vos algorithmes ?

Exercice 3.8 *Exercice complémentaire*
On souhaite implémenter un algorithme de coloration avec un ensemble donné de couleurs, que nous appellerons `couleurs`. Dans cet ensemble, il y a une couleur sélectionnée que nous appellerons `couleur_courante`. On dispose des fonctions suivantes :

- `init_color()` qui renvoie l'ensemble `couleurs` (contenant une première couleur selectionnée)
- `get_color(couleurs)` qui donne la `couleur_courante`
- `next_color(couleurs)` qui change la couleur selectionnée (**mais ne renvoie rien**)
- `more_color(couleurs)` qui renvoie `True` s'il existe une couleur qui n'a encore jamais été selectionnée, et `False` sinon.

Pour disposer de ces fonctions, vous devrez taper dans l'interpréteur `execfile("colors.py")` Implémenter l'algorithme suivant :

```
pour tout sommet s de G:
    couleurs = init_color()
    c  est la couleur_courante
    tant qu'il existe un voisin de s colorié avec c
         s'il existe une couleur encore non selectionnée:
             changer la couleur selectionnée
             c  est la nouvelle couleur courante
         sinon:
             retouner False
     colorier s avec c
retourner True
```