TP noté Informatique
Cliques maximales
J1 MI 1003, groupe E1, Université Bordeaux

Recommandations

Vous allez écrire tous vos programmes dans un seul et même fichier que vous allez appeler quelque chose du style `NOMPrenom.py`. N'oubliez pas de mettre `graphV3.py` et `bibV3.py` dans le fichier dans lequel vous compilez vos programmes.

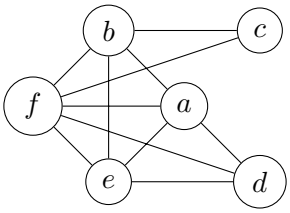
Du reste, n'oubliez pas l'instruction à mettre au début de votre document :

```
from bibV3 import *
```

Vous pouvez également rajouter des commentaires en les précédant par des dièses # afin qu'ils ne soient pas lus lors de l'exécution de votre programme.

Préliminaires théoriques

Une **clique** dans un graphe G est un sous-ensemble de sommets de G tel que chaque sommet est voisin à chaque autre sommet. (Dans un graphe simple, une clique est donc un sous-graphe complet.) Par exemple, dans le graphe



le sous-ensemble de sommets $\{a, b, e, f\}$ forme une clique, tandis que $\{a, b, d, e\}$ n'en est pas une car les sommets b et d ne sont pas voisins.

La **taille** d'une clique est le nombre de sommets qu'elle contient. Par exemple, $\{a, b, e, f\}$ est une clique de taille 4, alors que $\{a, d, e\}$ est une clique de taille 3.

Une clique est dite **maximale** (à ne pas confondre avec *maximum*) s'il n'existe pas une autre clique plus grosse la contenant. Par exemple, $\{b, c, f\}$ est une clique maximale : on ne peut pas rajouter un autre sommet dans ce sous-ensemble sans que celui devienne différent d'une clique. Au contraire, $\{a, d, e\}$ n'est pas une clique maximale car la clique (plus grosse) $\{a, d, e, f\}$ contient $\{a, d, e\}$.

Attention, une clique maximale n'est pas forcément de taille maximale. Par exemple, $\{b, c, f\}$ est bien une clique maximale, mais sa taille (qui est de 3) reste plus petite que celle de la clique $\{a, d, e, f\}$ qui est de 4 !

Programmation

Exercice 1 Écrire un programme `demarquer(G)` qui démarque tous les sommets et toutes les arêtes du graphe `G`.

Exercice 2 Écrire une liste d'instructions (ne pas définir une fonction !) qui marque les sommets `"Paris"`, `"Nantes"`, `"Lyon"` et `"Marseille"` dans le graphe `tgv2005`, et les sommets `a`, `b`, `c`, `d` et `e` dans le graphe `Petersen`.

Exercice 3 Écrire un programme `nombre_marques(G)` qui renvoie le nombre de sommets marqués dans le graphe `G`.

Exercice 4 Écrire un programme `liste_marques(G)` qui renvoie la liste des sommets marqués dans `G`.

Exercice 5 Écrire un programme `est_Clique(G)` qui renvoie `True` si les sommets marqués dans le graphe `G` forment une clique, `False` sinon.

Exercice 6 Écrire un programme `surligne(G)` qui marque toutes les arêtes qui relient deux sommets marqués (et seulement elles).

Exercice 7 Écrire un programme `marques_voisins(G,s)` qui renvoie `True` si tout sommet marqué est voisin de `s`, `False` sinon.

Exercice 8 Écrire un programme `possible_ajout(G)` qui renvoie un sommet non marqué `s` tel que tout sommet marqué est voisin de `s` (si un tel sommet existe) ; et qui renvoie `False` s'il n'en existe pas. (On peut également renvoyer `None`, au choix de l'étudiant).

On cherche maintenant à marquer un sous-ensemble de sommets d'un graphe G formant une clique maximale. L'algorithme serait le suivant :

1. Démarquer tous les sommets et toutes les arêtes de G.
2. Marquer le premier sommet du graphe.
3. Tant qu'il existe un sommet s tel que tout sommet marqué de G admet s pour voisin, marquer ce sommet s.
4. Quand 3. est fini, marquer toutes les arêtes qui relient deux sommets marqués.

Exercice 9 Écrire un programme `premiere_clique_maximale(G)` qui implémente l'algorithme précédent.

On veut maintenant rajouter de l'aléatoire, afin d'obtenir plus de (voire toutes les ?) cliques maximales.

Exercice 10 Grâce à la fonction `melange(l)`, écrire une fonction `sommetrandom(G)` qui renvoie de manière aléatoire un sommet **non marqué**.

Exercice 11 Écrire un programme `possible_ajout_alea(G)` qui renvoie **de manière alétoire** un sommet non marqué `s` tel que tout sommet marqué est voisin de `s` (si un tel sommet existe) ; et qui renvoie `False` s'il n'en existe pas. (On peut également renvoyer `None`, au choix de l'étudiant).

Exercice 12 Utiliser les deux fonctions précédentes pour écrire la version aléatoire de `premiere_clique_maximale(G)`, qu'on appellera plus simplement `clique_maximale(G)`.

Exercice 13 En exécutant plusieurs fois la fonction `clique_maximale(G)`, que semble être la taille maximale d'une clique du graphe `tgv2005` ? du graphe `Europe` ? Et la taille minimale d'une clique maximale ?
On pourra répondre à cette question directement dans votre fichier en précédant chaque ligne de réponse par des dièses #.

Exercice 14 En supposant que les sommets marqués du graphe `G` forment une clique (non encore coloriée) et que `p` est une liste de couleurs distinctes (on dit que `p` est une palette), écrire un programme `colorieclique(G,p)` qui essaye de **bien** colorier les sommets marqués de `G` grâce aux couleurs listés par `p`. Le programme renverra `True` s'il réussit à bien colorier la clique, `False` sinon.

Exercice 15 En supposant que les sommets marqués du graphe `G` forment une clique **partiellement coloriée** et que `p` est une liste de couleurs distinctes, écrire un programme `colclique(G,p)` qui essaye de **bien** colorier les sommets marqués qui étaient de couleur blanche de `G` grâce aux couleurs listés par `p` (sans se préoccuper des potentiels voisins de la clique qui n'appartiendraient pas à la clique). Le programme renverra `True` s'il réussit à bien colorier la clique, `False` sinon.

Exercice 16 On cherche à bien colorier un graphe G grâce à une palette p. Le principe de cet ultime algorithme est le suivant :
Tant qu'il existe un sommet non colorié, on tire aléatoirement une clique maximale contenant un sommet colorié et un sommet non colorié, et on essaye de bien colorier la clique avec p. Si on échoue de colorier la clique maximale, on arrête le programme et on renvoie `False`.
Implémenter cet algorithme sous le nom `colorier(G,p)`.

Fonctions du module *graphV3*

L'argument G est un graphe	
`listeSommets(G)`	retourne la *liste* des *sommets* de G.
`nbSommets(G)`	retourne le *nombre* de *sommets* de G.
`sommetNom(G,etiquette)`	retourne le *sommet* de G désigné par son *nom* (étiquette). Exemple : `sommetNom (Europe, 'Italie').`
`sommetNumero(G,i)`	retourne le *sommet* numéro i dans G ; la numérotation commence à 0.
`dessinerGraphe(G)` ou simplement `dessiner(G)`	demande (très poliment) au logiciel *Graphviz* de dessiner le graphe

L'argument s est un sommet	
`listeVoisins(s)`	retourne la *liste* des *voisins* de s.
`degre(s)`	retourne le *degré* de s.
`nomSommet(s)`	retourne le *nom* (étiquette) de s.
`colorierSommet(s,c)`	colorie s avec la couleur c. Exemples de couleurs : `'red'`, `'green'`, `'blue'`, `'white'`.
`couleurSommet(s)`	retourne la *couleur* de s.
`marquerSommet(s)` `demarquerSommet(s)`	marque ou démarque s.
`estMarqueSommet(s)`	retourne `True` si s est marqué, `False` sinon.
`listeAretesIncidentes(s)`	retourne la *liste* des arêtes *incidentes* à s.

L'argument a est une arête	
`nomArete(a)`	retourne le *nom* (étiquette) de a.
`marquerArete(a)` `demarquerArete(a)`	marque ou démarque a.
`estMarqueeArete(a)`	retourne `True` si a est marquée, `False` sinon.

Arguments : un sommet s et une arête a	
`sommetVoisin(s,a)`	retourne le voisin de s en suivant l'arête a.
L'argument est une liste	
`melange(u)`	retourne une copie mélangée aléatoirement de la liste u. Exemple : `melange(listeSommets(cube))`.

La liste `graphes` contient les graphes `tgv2005`, `fig22`, `Europe`, `Koenigsberg` et `Petersen`. La liste `graphesPlanairesReguliers` contient les graphes `tetraedre`, `cube`, `octaedre` (huit triangles), `dodecaedre` (douze pentagones) et `icosaedre` (vingt triangles). Les fonctions suivantes permettent de construire des graphes de taille variable :

`construireComplet(n)`	retourne le graphe complet K_n . Exemple : `K5 = construireComplet(5)`.
`construireBipartiComplet(m,n)`	retourne K_{mn} . Exemple : `K34 = construireBipartiComplet(3,4)`.
`construireArbre(d,h)`	retourne l'arbre de hauteur h dont chaque sommet possède d fils. Exemple : `arbre = construireArbre(2,3)`.
`construireGrille(m,n)`	retourne la grille rectangulaire avec m lignes et n colonnes. Exemple : `grille = construireGrille(4,6)`.
`construireTriangle(n)`	retourne la grille triangulaire d'ordre n. Exemple : `t5 = construireTriangle(5)`.