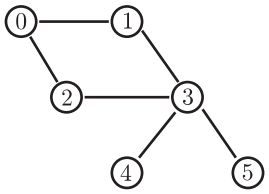
TP n° 2 - Graphes

On va voir dans ce TP deux manières de représenter des graphes : les matrices d'adjacence et les listes d'adjacence.
Un graphe $G = (S, A)$ est constitué d'un ensemble fini S de sommets et d'arêtes joignant certains couples de sommets $A \subseteq S \times S$. On ne s'intéresse ici qu'à des graphes non-orientés c'est-à-dire qu'une arête joignant le sommet i au sommet j joint aussi le sommet j au sommet i.

Exercice 1. *Matrices d'adjacence*

La matrice d'adjacence d'un graphe $G = (S, A)$ à n sommets est une matrice (ou un tableau de tableaux...) à n lignes et n colonnes.
La case (i, j) contient un 1 si les sommets i et j sont reliés par une arête, et un 0 sinon. On considère qu'un sommet n'est pas relié à lui-même, la diagonale (i, i) ne contient donc que des 0.

1. Représentez le graphe suivant sous la forme d'une matrice d'adjacence :



2. Écrivez les fonctions `stable(n)` et `clique(n)` qui renvoient respectivement le stable à n sommets (aucune arête), et la clique à n sommets (toutes les arêtes possibles).

3. Écrivez une fonction `voisins(g, i)` qui prend en argument un graphe `g` et un sommet `i` et renvoie la liste des voisins de `i`.

4. Écrivez les fonctions `nb_aretes(g)` et `degre_total(g)` qui comptent respectivement le nombre d'arêtes du graphe et la somme des degrés des sommets (le degré d'un sommet est le nombre de voisins qu'il a). Quelle relation y a-t-il entre les deux ?

Exercice 2. *Listes d'adjacence*

Une autre manière de représenter un graphe est de lister les voisins de chacun de ses sommets. Ainsi, la liste d'adjacence d'un graphe à n sommets est une liste de n listes. Chaque sous-liste correspond à un sommet i et contient les voisins de i.

1. Donnez la liste d'adjacence du graphe de la question 1 de l'exercice précédent.

2. Écrivez deux fonctions `liste_en_matrice(g)` et `matrice_en_liste(g)` qui permettent de convertir un graphe sous forme de liste en matrice et réciproquement.

3. Ré-écrivez les fonctions `stable`, `clique`, `voisins`, `nb_aretes` et `degre_total` pour qu'elles traitent un graphe sous forme de liste d'adjacence.

Exercice 3. *Connexité*

1. En utilisant la représentation que vous préférez, écrivez une fonction `connecte(g,i,j)` qui détermine s'il existe ou non un chemin dans le graphe `g` allant de `i` à `j`.

2. En utilisant la fonction précédente, écrivez une fonction `connexe(g)` qui détermine si un graphe `g` est *connexe* (ie. tous ses sommets sont connectés par un chemin).