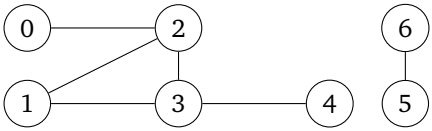
TD — Graphes

Exercice 1. *Graphes (adapté d'un roman de Georges Perec)*

Ces échanges, qui sont suscités aussi bien par des occasions propices de vente ou d'achat (il s'agit alors de faire de la place) que par des inspirations subites, des lubies, des caprices ou des dégoûts, ne se font pas au hasard, et n'épuisent pas les douze possibilités de permutations qui pourraient se faire entre ces quatre lieux et que la figure 1 *met bien en évidence ; ils obéissent strictement au schéma de la* figure 2 *: quand Madame Marcia achète quelque chose, elle le met chez elle, dans son appartement, ou dans sa cave ; de là, ledit objet peut passer dans l'arrière-boutique, et de l'arrière-boutique dans le magasin ; du magasin enfin il peut revenir — ou parvenir s'il venait de la cave — dans l'appartement. Ce qui est exclu, c'est qu'un objet revienne dans la cave, ou arrive au magasin sans être passé par l'arrière-boutique, ou repasse du magasin dans l'arrière-boutique, ou de l'arrière-boutique dans l'appartement, ou enfin passe directement de la cave à l'appartement.*

1. Dessinez les graphes des figures 1 & 2.
2. Comment appelle-t'on les graphes semblables à celui de la figure 1.

Exercice 2. *Graphes (adapté d'un exercice de Marie Durand)*
On donne le graphe non orienté suivant :



1. Donnez le degré du sommet 2.
2. Ce graphe est-il acyclique ? Si non, donnez un exemple de cycle.
3. Ce graphe est-il connexe ? Si non, donnez les ensembles de sommets formant ses composantes connexes.
4. Donnez deux chemins allant du sommet 0 au sommet 4. Combien y a-t'il de chemins possibles entre ces sommets ? Combien de ces chemins sont élémentaires ?
5. Donnez une représentation en Python de ce graphe par liste de listes d'adjacence, utilisant des `list`.
6. De quelle autre façon pourrait-on représenter ce graphe informatiquement ?
7. Écrivez une fonction Python `reachable(G, src, dst)` qui prend en entrée un graphe et les identifiants de deux sommets (représentés comme à la

question précédente) et renvoie `True` (resp. `False`) s'il existe (resp. n'existe pas) un chemin dans le graphe entre les deux sommets.

Exercice 3. *Parcours complet & détection de cycle*

On considère des graphes représentés par tableau (liste) de liste d'adjacence en Python.

1. Écrivez une fonction `wfs` Python telle que `wfs(g,v)` effectue un parcours de graphe d'origine `v` et s'évalue en une `list` dont la case d'indice `i` contient `None` si `i` n'est pas accessible depuis `v`, et sinon `t` avec `t` son ordre de visite par le parcours (l'ordre dans lequel il a été ajouté dans la `list`).
   Votre fonction devra avoir un coût linéaire en la taille de ses arguments.
2. Donnez une caractérisation des graphes non-orientés pour lesquels un résultat d'appel à `wfs` peut contenir des cases à `None`.
3. Modifiez votre fonction `wfs` en une fonction `wfsf` qui visite *tous* les sommets du graphe (désormais non orienté, mais **peu importe sa connexité**) et s'évalue en une paire donnant le nombre de composantes connexes et un tableau contenant en case `i` un entier indiquant sa composante connexe (on numérotera par exemple les $\#C$ composantes connexes par des entiers 0, 1, ..., $\#C-1$).
   Cette fonction devra être de **coût linéaire** en la taille de son argument.
4. Utilisez votre fonction de la question précédente pour écrire une fonction `acyc` telle que `acyc(g)` renvoie `True` si `g` représente (toujours de la même façon) un graphe non-orienté acyclique, et `False` sinon.

Exercice 4. *Bicoloriage*

Pour le besoin de cet exercice, on dit d'un graphe non orienté $G=(S,A)$ qu'il est *biparti* si $A \neq \varnothing$ et S est l'union disjointe de L, R non vides tels que $(u,v) \in A \Rightarrow ((u \in L \wedge v \in R) \vee (u \in R \wedge v \in L))$. (Autrement dit, toute arête de G connecte nécessairement un sommet de L à un sommet de R.) On définit également le *nombre chromatique* d'un graphe G (noté $\chi(G)$) comme le nombre minimal n de couleurs pour lequel G est n-coloriable (c'est à dire que n couleurs sont suffisantes pour que chaque sommet soit colorié d'une couleur différente de celle de tous ses voisins).

1. Montrez l'équivalence (G est biparti) $\Leftrightarrow \chi(G)=2$.
2. Déduisez de cela qu'un arbre de taille >1 est biparti.
3. Montrez que $\chi(C_n)=2$ ssi. $n \geq 3$ est pair.
4. Décrivez un algorithme qui décide si un graphe est 2-coloriable, et le colorie le cas échéant.
5. Proposez-en une implémentation Python pour un graphe représenté comme à l'exercice précédent.