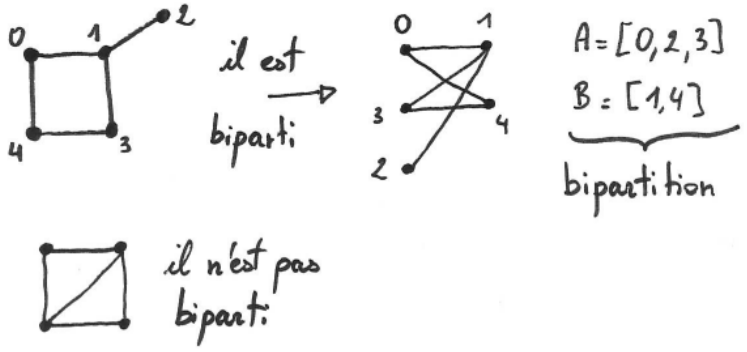
Graphes : interrogation de TP

IMPORTANT : vous travaillerez directement dans le fichier `interroTP.py` ci-joint. Le fichier que vous allez rendre doit pouvoir être directement exécuté. Si une partie ne compile pas, mettez-la en commentaire avant de rendre votre travail. A la fin du code, on donne des tests pour les questions 1,2 et 3 qui doivent passer pour valider votre travail. Pensez à les regarder !

Un graphe non orienté $G = (V, E)$ est dit *biparti* s'il est possible de répartir ses sommets en deux ensembles A et B (il s'agit donc d'une partition des sommets, $A \cup B = V$ et $A \cap B = \emptyset$) de sorte que toutes les arêtes du graphe aient une extrémité dans A et l'autre dans B (ou, autrement dit, qu'aucune arête n'ait ses deux extrémités d'un même côté). Vous connaissez par exemple les graphes bipartis complets, qui sont bipartis, avec le maximum d'arêtes possibles étant donnés les cardinaux des ensembles A et B.



Question 0. **IMPORTANT** Renommez le fichier `interroTP.py` sous la forme `nom_prenom.py` (avec votre nom et prénom !).

Rappels de python

`range(a,b)` renvoie la liste [a,a+1,...,b-1]
`x in uneliste` renvoie un booléen indiquant si x est dans la liste
`x not in uneliste` en est la négation

Question 1. **Graphes bipartis complets** Complétez la fonction `grapheBipartiComplet` (ligne 124 du code) qui, étant donnés deux entiers p et q, renvoie un `GrapheNO` correspondant au graphe biparti complet dont les côtés sont de cardinaux respectifs p et q (pensez aussi à regarder le test à la fin du code, sans le modifier !)

Question 2. **Vérification d'une bipartition** On appelle *bipartition* la répartition des sommets du graphe en deux sous ensembles A et B comme dans la définition plus haut. On demande ici de compléter la méthode `bipartition_valide` de la classe `GrapheNO` (ligne 45 du code) qui prend en paramètres deux listes `A` et `B` et vérifie qu'il s'agit bien d'une bipartition. On supposera que les deux listes de sommets forment bien une partition des sommets (inutile de le vérifier).

Question 3. **Recherche d'une bipartition** Un algorithme simple pour trouver une bipartition est le suivant : on effectue un parcours en largeur depuis le sommet 0, que l'on place dans A, et au fur et a mesure que l'on rencontre de nouveaux sommets, si leur prédecesseur est dans A, on place le nouveau sommet dans B, et inversement. A la fin, on peut vérifier qu'il s'agit bien d'une bipartion avec la méthode de la question 2 pour savoir si le graphe est biparti ou non.
Complétez la méthode `est_biparti` de la classe `grapheNO` (juste après la méthode précédente) qui renvoie :

— `False`, si le graphe n'est pas biparti
— le couple [A, B] où A et B forment la bipartition si le graphe est bien biparti.