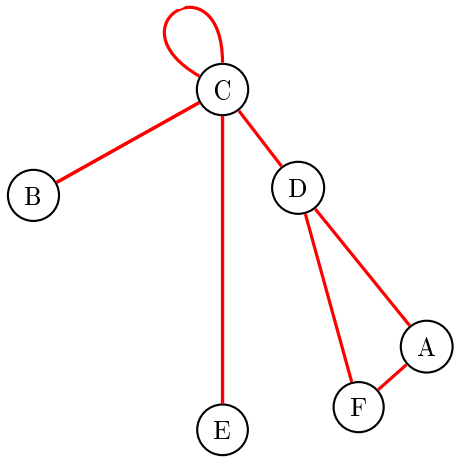
On reprend les notations du tp précédent. Soit $G = (V, E)$ un graphe non orienté. On rappelle que deux sommets $v_i, v_j \in V$ sont dits adjacents s'ils sont reliés par une arête, soit $\{v_i, v_j\} \in E$ (on notera l'arête $\{v_i, v_j\}$ simplement $v_i v_j$.
On appelle chaine dans un tel graphe (resp. chemin pour un graphe orienté) toute suite $(v_1, v_2, \ldots, v_n) \in V^n$ telle que $\forall i \in \{1, \ldots, n-1\}$ les sommets v_i et v_{i+1} sont adjacents. On dira alors que c'est une chaine (resp. chemin) de v_1 à v_n de longueur n.
Si de plus $\forall i, j \in \{1, \ldots, n\}$ $v_i \neq v_j$ alors la chaine (resp. chemin) est dite élémentaire.
Ainsi, si l'on reprend le graphe du tp précédent (A, C, E) est une chaine élémentaire alors que (A, C, E, B, C, D) ne l'est pas.
On se propose d'ajouter deux méthodes à notre classe *Graphe* :
Une méthode **trouve_chaine**, qui prend en paramètres d'entrée deux sommets v_1 et v_2 et qui renvoie une chaine **élémentaire** de v_1 à v_2.
On pourra également passer une chaine (une liste) en paramètre d'entrée (pour éventuellement rappeler, récursivement, notre méthode sur une sous-chaine) :

```
def trouve_chaine(self, sommet_dep, sommet_arr, chain=None):
```

Une méthode **trouve_toutes_chaines**, qui prend en paramètres d'entrée deux sommets v_1 et v_2 et qui renvoie toutes les chaines **élémentaires** de v_1 à v_2.
On pourra également passer une chaine (une liste) en paramètre d'entrée (pour éventuellement rappeler, récursivement, notre méthode sur une sous-chaine) :

```
def trouve_tous_chemins(self, sommet_dep, sommet_arr, chem=[]):
```

On testera ces méthodes sur le graphe du tp précédent ainsi que sur le graphe ci dessous :



On rappelle que le degré ou la valence d'un sommet v, noté $\deg(v)$ (ou simplement $d(v)$), est le nombre d'arêtes incidentes à ce sommet, où une **boucle compte double.**.
Le degré maximum d'un graphe G sera noté $\Delta(G)$ et le degré minimum $\delta(G)$
Lorsque tous les sommets sont de même degré, d, le graphe est dit régulier de degré d.
Ecrire une classe, `Graphe2`, qui hérite de la classe `Graphe` et qui met en oeuvre les méthodes suivantes :

```
class Graphe2(Graphe):

    def sommet_degre(self, sommet):
        """ renvoie le degre du sommet """

        return degre

    def trouve_sommet_isole(self):
        """ renvoie la liste des sommets isoles """

        return isoles

    def Delta(self):
```

```
        """ le degre maximum  """

        return max

    def list_degres(self):
        """ calcule tous les degres et renvoie un
            tuple de degres decroissant
        """

        return degres
```

Vérifier le lemme des poignées de mains sur les graphes précédents.