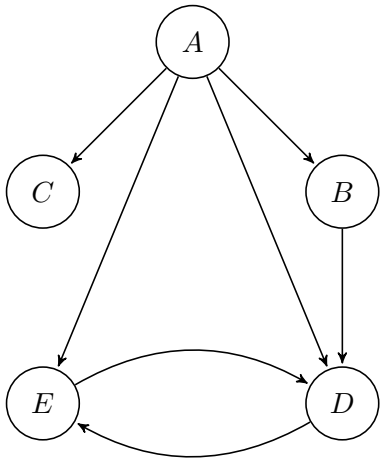
DURÉE : HORAIRE(1,59,33) [1]

- NUMÉRIQUE ET SCIENCES INFORMATIQUES -

Exercice n° 1 Applications directes sur les graphes

1. Déterminer le dictionnaire des prédécesseurs et la matrice d'adjacence du graphe ci-dessous.



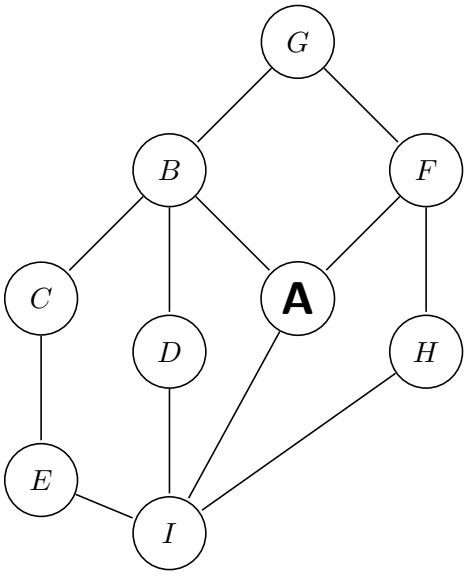
2. a. Tracer le graphe G **NON orienté** dont la matrice d'adjacence est donnée ci-dessous. On notera A, B, C et D les sommets.

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

b. Vérifier que la somme des degrés des sommets du graphe G est égal à deux fois le nombre d'arêtes. (Détailler la vérification)

3. a. Partant du sommet **A**, proposer un parcours en largeur du graphe ci-dessous.

b. Partant du sommet **A**, proposer un parcours en profondeur du graphe ci-dessous.



1. Voir exercice 3

Exercice n° 2 Graphe et algorithmes

1. On considère le dictionnaire des prédécesseurs d'un graphe.

```
parents = {'g':None,'l':'r','i':'u','d':'n','r':'i','a':'l','u':'g','n':'a','e':'d'}
```

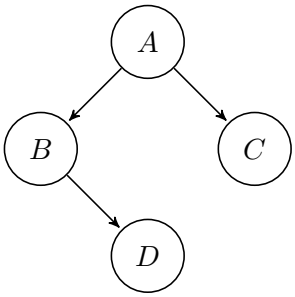
On remarquera que chaque sommet à un unique prédécesseur sauf le sommet `'g'`.

a. Construire le chemin allant de `'g'` à `'e'`.

b. Compléter l'algorithme ci-dessous permettant d'obtenir ce chemin à partir du dictionnaire **parents** :

```
chemin = []
final='e'
while ................. :
    ........................
    ........................
print(chemin)
```

2. Soit G le graphe ci-contre.



a. Construire le dictionnaire des successeurs de G (on le note **DicoSuc**).

b. On considère l'algorithme ci-dessous :

```
def enigme(s1,s2,Dico):
'''
s1 est un sommet de Dico
s2 est un sommet de Dico
Dico est un dictionnaire des successeurs.
'''
    seen={s1:None }
    mystere=[s1]
    while mystere !=[] :
        s=mystere.pop()
        if s==s2 :
            return seen
        else :
            successor=Dico[s]
        for elt in successor :
            if elt not in seen :
                seen[elt]=s
                mystere.append(elt)
    return None
```

i. Pour gérer les données du tableau **mystere**, l'algorithme utilise t-il une structure de pile ou de file ? (Justifier)

ii. Quel est le résultat de : `enigme('A','D',DicoSuc)` ?

iii. Proposer des paramètres **s1**, **s2** et **Dico** qui permettrait d'avoir le résultat :

```
>>> enigme(...,...,Dico):
None
```

On donnera le détail de **Dico**.

Exercice 3 : On définit la classe `Horaire` comme suit :

```
class Horaire:
    def __init__(self, heure, minute, seconde):
        self.__heure = heure
        self.__minute = minute
        self.__seconde = seconde

    def __repr__(self):
        pass

    def __eq__(self, autre_Horaire):
        pass

    def __add__(self, autre_Horaire):
        pass

    def convert_to_h_min_s(nombre):
        """fonction de classe renvoyant un objet de type horaire (h min s) à
        partir d'un nombre entier de secondes passé en passé en paramètre"""
        heure = (nombre//3600)%24
        minute = (nombre%3600)//60
        seconde = nombre%60
        return Horaire(heure,minute,seconde)
        """
        >>> Horaire.convert_to_h_min_s(30564)
        8 h 29 min 24 s
        """
```

1- Ecrire une **méthode de représentation** (`__repr__`)de l'horaire autoréférencé qui renverra une chaine de caractère de la forme `'…h …min …s'` :

```
>>> h1 = Horaire(8,29,24)
>>> h1
'8h 29min 24s'
```

2- Ecrire une **méthode d'égalité** surchargée (`__eq__`) entre 2 instances de la classe `Horaire` : 2 variables de type `Horaire` seront *considérées* comme identiques si elles sont définies par des attributs identiques.

```
>>> h1 = Horaire(8,29,24)
>>> h2 = Horaire(8,29,24)
>>> h1 == h2
True
```

3- Ecrire une **méthode** `convert_to_seconde` qui retournera un nombre entier dont la valeur sera le nombre de secondes correspondant à l'horaire autoréférencé :

```
>>>horloge = Horaire(8,29,24)#soit en secondes : 8*3600 + 29*60 + 24 =
30564
>>>horloge.convert_to_seconde()
30564
```

4- En utilisant les méthodes à votre disposition, écrire une **méthode d'addition** surchargée (`__add__`) entre 2 instances de la classe `Horaires` qui retournera un objet de la classe `Horaire` (= une instance de la classe `Horaire`) résultant de la somme des 2 paramètres :

```
>>> h1 = Horaire(5,50,20)
>>> h2 = Horaire(3,20,10)
>>> h1+h2
'9h 10min 30s'
```

Exercice 4

Afin de gérer numériquement les réservations de places dans chaque wagon de ses trains, la SNCF définit la classe `Wagon` comme ci-dssous.
Attention : il ne faudra pas confondre la classe `Wagon` (qui définit les objets de type `Wagon`) et la *classe* d'un wagon (nombre entier (`int`) qui vaut 1 pour un wagon de *première classe* ou qui vaut 2 pour un wagon de *deuxième classe*) qui caractérise le niveau de confort d'un wagon.

```
class Wagon :
    def __init__(self, classe):
        """constructeur qui renvoie un wagon vide dont la classe est
        passée en paramètre"""
        self.classe = classe
        if classe == 1:
            self.nb_de_sieges = 54
        else:
            self.nb_de_sieges = 88
        self.liste_place=[] #liste des occupants de chaque siège numeroté
        for i in range(self.nb_de_sieges):
            self.liste_place.append("vide")

    def get_nb_de_sieges(self):
        return self.nb_de_sieges

    def get_classe(self):
        return self.classe

    def get_occupant(self, numero_de_place):
        """renvoie le nom de l'occupant du siege dont le numero
        est passé en paramètre (ou "vide" si le siege est libre)"""
        return self.liste_place[numero_de_place]

    def set_occupant(self, numero_de_place, nouvel_occupant):
        self.liste_place[numero_de_place] = nouvel_occupant

    def reserver_place(self, numero_de_place, nom_passager):
        if self.get_occupant(numero_de_place) == "vide":
            self.set_occupant(numero_de_place, nom_passager)
        else:
            print("Réservation impossible")

    def mystere(self):
        res = 0
        for place in self.liste_place:
            if place == "vide":
                res = res + 1
        return res
```

1- Citer les attributs de la classe `Wagon` et préciser leur type.

2- Quel est le nombre de sièges (occupés ou libres) dans un wagon de 2ème classe ?

3- Citer au moins un *accesseur* parmi les méthodes de cette classe `Wagon`.

4- Recopier et compléter la méthode `annuler_reservation` ci-dessous qui permet de libérer le siège occupé par un passager dont le nom est `nom_passager` (à condition que ce passager ait réservé un siège à son nom).

```
    def annuler_reservation(self, nom_passager):
        if ..........................:
            for numero_de_place in range (..........):
                if self.get_occupant(..........) == ..........:
                    ...................
        else:
            print("ce passager n'est pas enregistré dans ce wagon")
```

5- Quelle information, utile pour la SNCF, renvoie la méthode `mystere()` au sujet du wagon autoréférencé ?

La SNCF définit également une classe `Train` afin de gérer, pour chaque train (instance de la classe `Train`) :

- La gare de départ
- La gare d'arrivée
- La listes des wagons (instance de la classe `Wagon`) composant le train

```
class Train:
    def __init__(self, depart, arrivee):
        self.gare_depart = depart
        self.gare_arrivee = arrivee
        self.liste_wagons=[]

    def ajouter_wagon(self, nouveau_wagon):
        self.liste_wagons.append(nouveau_wagon)

    def nb_total_de_sieges(self):
        res = 0
        for wagon in self.liste_wagons:
            res = res + wagon.get_nb_de_sieges()
        return res
```

6- Ecrire une méthode `changer_gare_arrivee` qui permet de modifier l'attribut `gare_arrivee` du train autoréférencé (on parle de *mutateur*).
Exemple d'utilisation :

```
>>> t1 = Train("Paris", "Marseille")
>>> t1.changer_gare_arrivee("Lille")
>>> t1.gare_arrivee
'Lille'
>>> t1.changer_gare_arrivee("Brest")
>>> t1.gare_arrivee
'Brest'
```

7- Ecrire une méthode `nb_places_libres_en_classe` qui renvoie le nombre total de places libres (dont la classe est passée en paramètre) dans la totalité du train.
Exemple d'utilisation :

```
>>> t1.nb_places_libres_en_classe(1)
92
>>> t1.nb_places_libres_en_classe(2)
124
```