TD Informatique TSI2

GRAPHS : definition and representation

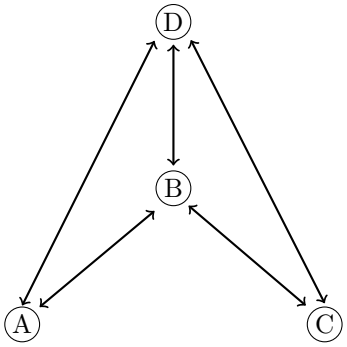
EXERCICE 01

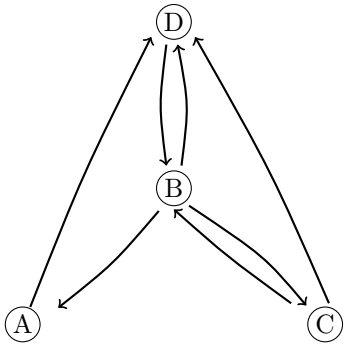
Parmi les assertions suivantes, lesquelles sont vraies ?

A. Un graphe ne peut pas avoit plus de sommets que d'arêtes.
B. Une matrice d'adjacence d'un graphe à n sommets contient $2n$ coefficients.
C. Une matrice d'adjacence d'un graphe orienté est symétrique.
D. L'ordre d'un graphe est le nombre d'arcs ou d'arêtes.
E. Un graphe est connexe si chaque sommet est adjacent à tous les autres.
F. Un chemin qui passe deux fois par le même sommet contient un cycle.
G. Un graphe non orienté d'ordre 5 a au maximum 10 arêtes reliant des sommets distincts.

EXERCICE 02

1. Écrire la matrice d'adjacence (provenance en lignes et destination en colonnes) associée aux graphes ci-dessous :





2. Écrire la liste d'adjacence sous la forme de dictionnaire des graphes ci-dessus.

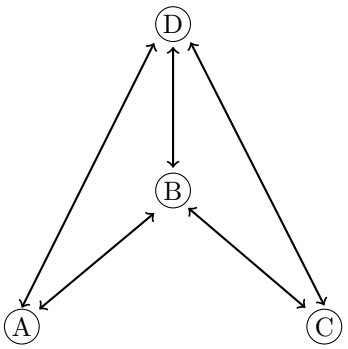
EXERCICE 03

Tracer les graphes associés aux matrices d'adjacence suivantes (provenance en lignes et destination en colonnes) :

$$M_1 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \text{ et } M_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

EXERCICE 04

On considère le graphe 1 suivant :



T.S.V.P →

1. On considère la fonction

```
In [1]: def UNKNOWN(m):
   ...:     nb = 0; n = len(m)
   ...:     for i in range(n):
   ...:         for j in range(i+1,n):
   ...:             if m[i][j] == 1 :
   ...:                 nb = nb + 1
   ...:     return nb
```

   L'appliquer à la matrice d'adjacence du graphe 1 trouvée à l'exercice 02. Que fait `UNKNOWN` ?
2. Écrire un dictionnaire, où les clés sont les sommets, pour représenter le graphe 1 à l'aide des listes d'adjacence.
3. Écrire une fonction `sommets` qui prend en paramètres un sommet `s` et un graphe `g`, sous forme d'un dictionnaire, et renvoie la liste des sommets liés par une arête au sommet `s`.
   L'appliquer au graphe 1.
   Indication : *On teste simplement si `s` est une clé de `g` en utilisant la boucle conditionnelle `if s in g:`*

EXERCICE 05

Les graphes sont supposés non orientés et non pondérés. On appliquera les trois fonctions Python ci-dessous au cas du premier graphe de l'exercice 02 en notant le sommet isolé `'E'`

1. Écrire `ajoute_sommet1(g,s)` qui prend en paramètres un graphe `g` représenté par un dictionnaire des listes d'adjacence, un point `s` et qui ajoute le point au graphe en tant que sommet isolé.
   Indication : *on rappelle que `g[s]` est la valeur de la clé `s` du dictionnaire `g`*
2. Écrire `ajoute_sommet2(g,s)` qui prend en paramètres un graphe `g` représenté par une liste des listes d'adjacence et un point `s` et qui ajoute le point au graphe en tant que sommet isolé.
   Indication : *on rappelle que `[s,[]]` signifie que le sommet `s` n'est relié à aucun autre point*
3. Écrire `ajoute_sommet3(g)` qui prend en paramètres un graphe `g` représenté par une matrice d'adjacence et complète la matrice pour ajouter au graphe un sommet isolé.
   Indication : *D'abord dans une boucle `for L in g:`, on complète chaque ligne avec un `0` en utilisant `append` puis on ajoute une ligne ne contenant que des `0` en fin de `g`*

EXERCICE 06

1. Faire avec Python une boucle qui donne les valeurs de `chr(i)` pour `i` variant de 65 à 65+26.
   Que remarque-t-on ? Taper `chr(20)`, `chr(256)` et `chr(400)`
2. La fonction `conversion` suivante prend en paramètre une matrice d'adjacence `m` donc une liste de listes de bits et renvoie le graphe `g` non orienté correspondant sous forme d'un dictionnaire de liste d'adjacence, c'est-à-dire que les clés sont des points et les valeurs la liste des points reliés directement à la clé écrits dans l'ordre de l'alphabet. La fonction `conversion(m)` est :

```
In [38]: def conversion(m):
    ...:     sommets = {}; n = len(m)
    ...:     for i in range(n):
    ...:         sommets[i] = chr(65+i)
    ...:     g = {}
    ...:     for i in range(n):
    ...:         g[sommets[i]] = []
    ...:         for j in range(len(m[i])):
    ...:             ..............
    ...:                 ..................
    ...:     return g
```

   où vous devez remplacer les pointillés par le bon code Python.
   Appliquer à `m = [[0,1,0,1],[1,0,1,1],[0,1,0,0], [1,1,0,0,]]`