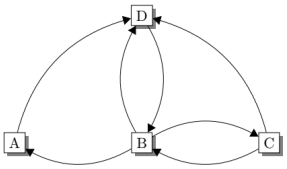
TP6 – Graphes

A – Travaux dirigés

Exercice 1 : Matrices d'adjacence

1. Ecrire la matrice d'adjacence associé au graphe ci-dessous.

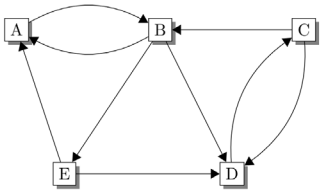


2. Tracer les graphes associés aux matrices d'adjacence donnés : M_1 *(non orienté)*, M_2*(orienté)*.

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

Exercice 2 : Listes d'adjacence

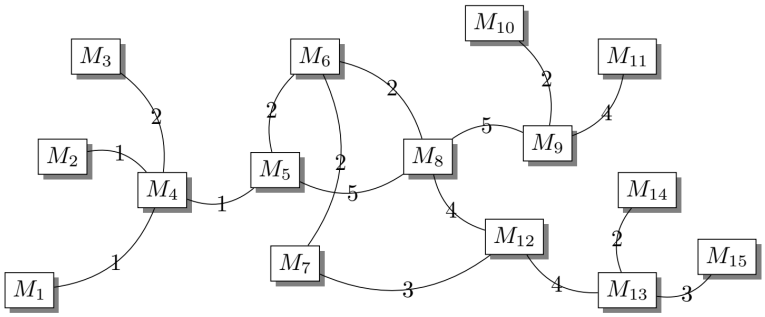
1. Ecrire les listes de successeurs du graphe suivant :



2. Tracer le graphe de la liste de successeurs suivante :

$$\{A : [B], B : [C, E], C : [B], D : [], E : [A, B]\}$$

Exercice 3 : Route la plus rapide



Le schéma ci-dessus représente un réseau d'appareils connectés qui peuvent être des ordinateurs, des smartphones, des boitiers internet, des routeurs. Les arêtes représentent les connexions filaires ou sans fil. Les nombres sont les temps de transmission en unité de temps.

1°) Combien de routes différentes peut prendre un message entre M_1 *et* M_{15} ? Une route ne peut passer qu'une fois par un appareil donné.

2°) Parmi ses routes laquelle est la plus rapide.

B – Travaux pratiques

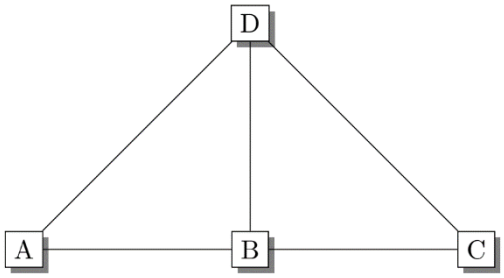
Exercice 4 : Degré d'un graphe

Soit le graphe suivant :

```
graphe={'A':['B','E'],'B':['A','C','E'],'C':['B'],'D':[],'E':['A','B']}
```

Ecrire une fonction deg(g,s) qui retourne le degré du sommet s.

Exercice 5 : Implémentation de graphe



On considère le graphe non orienté suivant (→) :

1. Ecrire la définition de la liste m qui représente la matrice d'adjacence du graphe.
    a. Ecrire une fonction nb_aretes(m) qui prend en paramètre une matrice m représentant un graphe et renvoie le nombre d'arêtes du graphe.
2. Ecrire un dictionnaire en Python, où clés sont les sommets, pour représenter ce graphe « m2 » à l'aide des listes d'adjacence. Ecrire une fonction « sommets(s,g) » qui prend en paramètres un sommet s et un graphe g, sous la forme d'un dictionnaire, et renvoie la liste des sommets liés par une arête au sommet s.

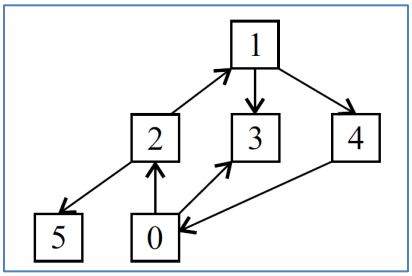
Exercice 6 : Ajout d'un sommet isolé

Les graphes sont supposés non orientés et non pondérés. Une variable g est définie pour représenter un graphe. Un sommet est représenté par un caractère comme 'A'. On pourra tester vos fonctions sur le graphe « m2 » précédent.

1. Ecrire une fonction qui prend en paramètres un graphe g représenté par un dictionnaire des listes d'adjacence et un point s et ajoute le point au graphe en tant que sommet isolé.
2. Ecrire une fonction qui prend en paramètres un graphe g représenté par une liste des listes d'adjacence et un point s et ajoute le point au graphe en tant que sommet isolé.
3. Ecrire une fonction qui prend en paramètres un graphe g représenté par une matrice d'adjacence et complète la matrice pour ajouter au graphe un sommet isolé.

Exercice 7 : Parcours en largeur

On modélise un site web par une page d'accueil qui contient des liens hypertextes permettant d'accéder à d'autres pages du site qui peuvent contenir également des liens hypertextes. Le site web contient 6 pages numérotées de 0 à 5. On considère le graphe G suivant représentant la structure du site web.



On utilise la liste « couleur » pour mémoriser la couleur des sommets. Un sommet est blanc lorsqu'il n'a pas été traité. Lorsqu'on commence à traiter un sommet i, il est gris. Après avoir traité en largeur tous les successeurs (qui deviennent gris) de ce sommet i, le sommet i est noir.

On utilise une deque D pour gérer la file d'attente FIFO.

1. Construire la matrice d'adjacence M du graphe G.
2. Ecrire une fonction « cycle » qui admet comme arguments une matrice d'adjacence M et un sommet de départ « début ». Cette fonction parcourt en largeur le graphe G. La fonction retourne « True » lorsqu'un cycle passe par un sommet (possibilité de retour à ce sommet). La fonction retourne « False » s'il n'existe aucun cycle passant par le sommet S. Pour cela on pourra compléter le code suivant :

```
from collections import deque
def cycle(M,début):
    n=len(M)
    couleur=["blanc" for i in range(n)] #Blanc = sommets non traités
    D=deque() #Création de notre deque vide
    D.append(début) #On ajoute début à l'extrémité droite de D
    couleur[début]="gris" #sommet en cours de traitement
    while len(D)!=0:
        x=D.popleft() #supprime le sommet x à l'extrémité gauche de D
        couleur[x]="noir" #Le sommet a été traité -> "noir"
        for i in range(n): #On va parcourir la matrice
            if...
                ...
                ...
            elif...
                return...
    return False #Le cas échéant
```

Exercice 8 : Conversion

On dispose d'un graphe non orienté sous la forme de listes d'adjacence, par exemple :

$$g = \{"A": ["B", "D"], "B": ["A", "C", " D"], "C": ["B"], "D": ["A", "B"]\}$$

1. La fonction conversion1 prend en paramètre un tel graphe et renvoie la matrice d'adjacence correspondante. Compléter cette fonction. La matrice obtenue avec le graphe g est représentée par la liste :

$$[[0,1,0,1],[1,0,1,1],[0,1,0,0],[1,1,0,0]]$$

```
def conversion1(g):
  sommets={}
  n=0
  for s in g:
    sommets[s]=n
    n=n+1
  mat=[n*[0] for i in range(n)]
  ...
  ...
  ...
  return mat
```

2. La fonction conversion2 prend en paramètre une matrice d'adjacence et renvoie le graphe correspondant. On doit donc retrouver le graphe g à partir de la matrice m :

$$[[0,1,0,1],[1,0,1,1],[0,1,0,0],[1,1,0,0]]$$

```
def conversion2(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
```

Exercice 9 : Polonaise inversée

On considère une expression écrite en notation polonaise inverse. Cette expression est représentée par une liste. Par exemple, l'expression « 83+5 × » est représentée par la liste $[8, 3, '+', 5, '*']$. En mathématiques, la forme d'écriture habituelle est $(8 + 3) \times 5$ et la valeur est 55. On utilise une pile pour réaliser l'algorithme suivant :

```
Pour e dans expression
    si e est un nombre
        alors on l'empile
    sinon
        on dépile deux opérandes
        on effectue l'opération
        on empile le résultat
On renvoie le résultat
```

- Ecrire une fonction calcule qui prend en paramètre une liste représentant une expression comme ci-dessus, notée exp et renvoie la valeur de l'expression. Les opérateurs peuvent être "+", "*", "-", ou"/". Une pile est représentée par le conteneur deque du module collections.
- Tester la fonction en vérifiant les résultats qui suivent :
- $[7, 2, '+', 3, '*'] = 27$; $[2, 5, '*', 4, '+'] = 14$ *et* $[8, 2, '/', 3, '-'] = 1$.