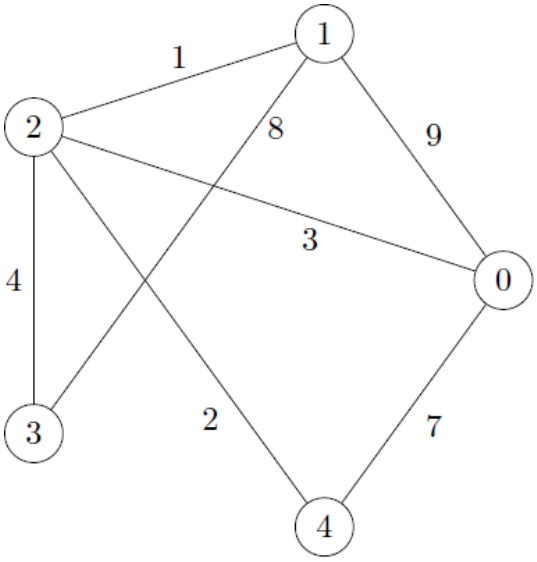
TP 9 – Tris.

Exercice 1 – Implémentation des graphes par une matrice d'adjacence

On considère le graphe `G` suivant, où le nombre situé sur l'arête joignant deux sommets est leur distance, supposée entière.



Question 1 *Construire la matrice $(G_{ij})_{0\leq i,j\leq 4}$, matrice de distances du graphe `G`, définie par : « pour tous les indices i, j, G_{ij} représente la distance entre les sommets i et j, ou encore la longueur de l'arête reliant les sommets i et j ». Cette matrice sera implémentée sous forme d'une liste de listes. (Chaque « sous-liste » représentant une ligne de la matrice d'adjacence. On convient que, lorsque les sommets ne sont pas reliés, cette distance vaut -1. La distance du sommet i à lui-même est égale à 0.*

Question 2 *Écrire une fonction `voisins(G:list, i:int) -> list`, d'argument la matrice d'adjacence `G` et un sommet i, renvoyant la liste des voisins du sommet i.*

Question 3 *Écrire une fonction `arretes(G:list) -> list`, renvoyant la liste des arêtes. Les arêtes seront constitués de couples de sommets (l'arête entre les sommets 0 et 1 sera donnée par `(0,1)`.*

Les instructions suivantes permettent de tracer un graphe.

```
import networkx as nx

def plot_graphe(G):
    Gx = nx.Graph()
    edges = arretes(G)
    Gx.add_edges_from(edges)
    nx.draw(Gx,with_labels = True)
    plt.show()
plot_graphe(M)
```

Question 4 *Écrire et tester la fonction `plot_graphe(G)`.*

Question 5 *Écrire une fonction `degre(G:list, i:int) -> int`, d'argument un sommet i, renvoyant le nombre des voisins du sommet i, c'est-à-dire le nombre d'arêtes issues de i.*

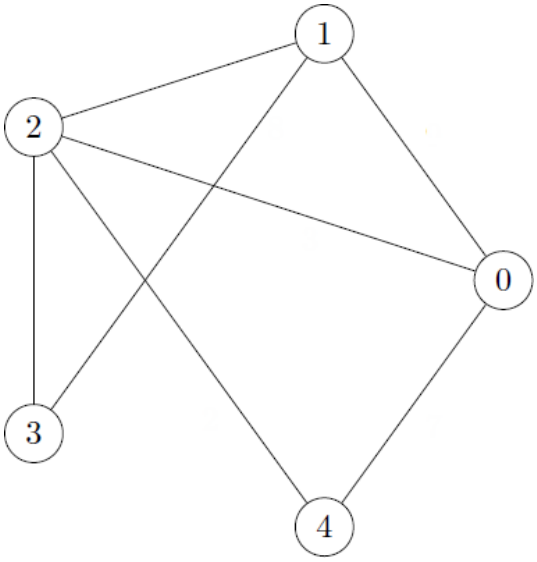
Question 6 *Écrire une fonction* `longueur(G:list,L:list) -> int`*, d'argument une liste* `L` *de sommets de* `G`*, renvoyant la longueur du trajet d'écrit par cette liste* `L`*, c'est-à-dire la somme des longueurs des arêtes empruntées. Si le trajet n'est pas possible, la fonction renverra* -1*.*

Question 7 *Écrire la fonction* `ajout_sommet(G:list, L:list, poids : list) -> None` *permettant d'ajouter un sommet au graphe.* `L` *désigne la liste des sommets auxquels le nouveau sommet est relié,* `poids` *la liste des poids respectifs.* `ajout_sommet` *agit avec effet de bord sur* `G`*.*

Question 8 *Écrire la fonction* `supprime_sommet(G:list, i: int) -> None` *permettant de supprimer le sommet* i *du graphe.*

Exercice 2 – Implémentation des graphes par une liste d'adjacence

On considère le graphe `G` suivant, où le nombre situé sur l'arête joignant deux sommets est leur distance, supposée entière.



Pour implémenter le graphe, on utilise une liste `G` qui a pour taille le nombre de sommets. Chaque élément `G[i]` est la liste des voisins de `i`.
Dans ce cas, `G[0]=[1,2,4]` car Les sommets 1, 2 et 4 sont des voisins de 0.

Question 9 *Construire la liste d'adjacence* `G` *en utilisant la méthode énoncée ci-dessus.*

Question 10 *Écrire une fonction* `voisins_l(G:list, i:int) -> list`*, d'argument la liste d'adjacence* `G` *et un sommet* i*, renvoyant la liste des voisins du sommet* i*.*

Question 11 *Écrire une fonction* `arretes_l(G:list) -> list`*, renvoyant la liste des arêtes. Les arêtes seront constitués de couples de sommets (l'arête entre les sommets 0 et 1 sera donnée par* `(0,1)`*.*

Les instructions suivantes permettent de tracer un graphe.

```
import networkx as nx

def plot_graphe_l(G):
    Gx = nx.Graph()
    edges = arretes_l(G)
    Gx.add_edges_from(edges)
    nx.draw(Gx,with_labels = True)
    plt.show()
plot_graphe(M)
```

Question 12 *Écrire et tester la fonction* `plot_graphe_l(G)`.

Question 13 *Écrire une fonction* `degre_l(G:list, i:int) -> int`, *d'argument un sommet* i, *renvoyant le nombre des voisins du sommet* i, *c'est-à-dire le nombre d'arêtes issues de* i.

Question 14 *Écrire la fonction* `ajout_sommet_l(G:list, L:list) -> None` *permettant d'ajouter un sommet au graphe.* `L` *désigne la liste des sommets auxquels le nouveau sommet est relié.* `ajout_sommet` *agit avec effet de bord sur* `G`.

Question 15 *Écrire la fonction* `supprime_sommet_l(G:list, i: int) -> None` *permettant de supprimer le sommet* i *du graphe.*

Question 16 *Écrire la fonction* `from_list_to_matrix(G:list, i: int) -> list` *permettant de convertir un graphe implémenté sous forme de liste d'adjacence en matrice d'adjacence.*

Question 17 *Écrire la fonction* `from_matrix_to_listmatrix(G:list, i: int) -> list` *permettant de convertir un graphe implémenté sous forme de matrice d'adjacence en liste d'adjacence.*

Exercices d'application sur les piles

Dans les exercices qui suivent, on utilisera des **piles**. Les piles sont des structures de données basées sur le principe LIFO (Last In First Out : le dernier rentré dans la pile sera le premier à en sortir).
Les opérations élémentaires qu'on peut réaliser sur les piles sont les suivantes :

- savoir si une pile est vide ;
- empiler un nouvel élément sur la pile ;
- récupérer l'élément au sommet de la pile tout en le supprimant. On dit que l'on dépile ;
- accéder à l'élément situé au sommet de la pile sans le supprimer de la pile ;
- on peut connaitre le nombre d'éléments présents dans la pile.

Pour implémenter les piles on utilisera le module `deque`. Chacun des éléments de la pile peut être un objet de type différent.

```
from collections import deque

# Créer une pile vide
pile = deque()

# Tester si une pile est vide
len(pile) == 0

# Ajouer l'élément Truc au sommet de la pile
pile.append("Truc")

# Supprimer (et renvoyer) le sommet d'une pile non vide
sommet = pile.pop()
```

Seules ces fonctions éléméntaires seront utilisées dans les exercices suivants.

Exercice 3 – La parenthèse inattendue Dans cet exercice, on souhaite savoir si une chaîne de caractères est bien parenthésée ou non. Une chaîne bien parenthésée est une chaîne vide ou la concaténation de chaînes bien parenthésées.

Exemple

Chaînes bien parenthésées :

— `"()","()()","(())"` et `"(()())"`.

Chaînes mal parenthésées :

— `")(","((","(()"` et `"())"`.

Question 18 *Implémenter la fonction* `parentheses` *répondant aux spécifications suivantes :*

```
def parenthese(s):
    "
␣␣␣␣Retourne␣les␣couples␣d'indice␣parenthèse
␣␣␣␣ouvrante,␣parenthèse␣fermante.
␣␣␣␣Entrée␣:
␣␣␣␣␣␣*␣s(str)␣:␣chaÃ®ne␣de␣caractères␣bien
␣␣␣␣␣␣parenthésée␣constituée␣uniquement
␣␣␣␣␣␣de␣parenthèses.
␣␣␣␣Sortie␣:
␣␣␣␣␣␣*␣Affichage␣des␣couples␣d'indices.
␣␣␣␣"
```

Question 19 *Réaliser un programme permettant de savoir si une chaîne de caractères est bien parenthésée. La structure de pile est-elle nécessaire ?*

Question 20 *Adapter le premier programme pour qu'il puisse traiter des chaînes constituées de parenthèses, de crochets, ou d'accolades. Un mot est alors bien parenthésé si la parenthèse fermante qui correspond à chaque parenthèse ouvrante est du même type.*

Question 21 *Adapter le programme pour qu'il puisse traiter des mots constitués de parenthèses et d'autres caractères, qui n'interfèrent pas avec les parenthèses.*

Question 22 *Écrire une version récursive de la fonction* `parentheses`*.*

Exercice 4 – Inversion

Question 23 *Écrire une fonction qui intervertit les deux éléments situés au sommet d'une pile de taille au moins égale à 2.*

Exercice 5 – Dépile le n^{e}

Question 24 *Écrire une fonction qui dépile et renvoie le troisième élément d'une pile de taille au moins égale à 3. Les premier et deuxième éléments devront rester au sommet de la pile.*

Exercice 6 – Lire le n^{e}

Question 25 *Écrire une fonction qui permet de lire (sans l'extraire) le n-ième élément d'une pile. On prévoira le cas où la pile n'est pas de taille suffisante pour qu'un tel élément existe.*

Exercice 7 – Inversion des extrêmes

Question 26 *Écrire une fonction qui prend une pile non vide en argument et place l'élément situé à son sommet tout au fond de la pile, en conservant l'ordre des autres éléments. Quelle est sa complexité en temps et en espace ?*

Exercice 8 – Inversion de la pile

Question 27 *Écrire une fonction similaire à* `reversed`*, qui prend une pile en argument et renvoie une autre pile constituée des mêmes éléments placés dans l'ordre inverse.*

Question 28 *Si l'on s'autorise à détruire la pile fournie, quelle est la complexité en temps et en espace de cette fonction ? Et si on ne s'y autorise pas ?*

Exercice 9 – Tu coupes ?

Question 29 *Écrire une fonction couper qui prend une pile et la coupe en enlevant de son sommet un certain nombre d'éléments (tirés au hasard) qui sont renvoyés dans une seconde pile.*

Exemple

Si la pile initiale est `[1, 2, 3, 4, 5]`, et que le nombre d'éléments retiré vaut 2, alors la pile ne contient plus que `[1, 2, 3]` et la pile renvoyée contient `[5,4]`.