Exercice 1 (6 points)

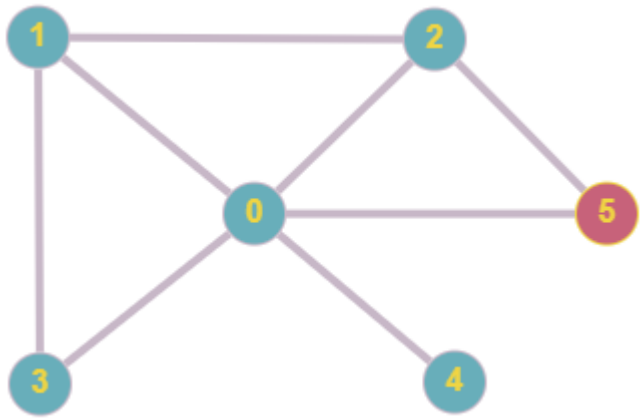
Cet exercice porte sur la programmation Python, la programmation dynamique, les graphes et les réseaux.

Barème évaluation : /+ points Ni : Niveau de difficuté de la question de N1 à N3

1. /+0.25 N1

```
voisins = [[1, 2, 3, 4],
           [0, 2, 3],
           [0, 1],
           [0, 1],
           [0]]
```

2. /+0.25 N1



3. /+0.5 N1

```
voisins = [[1, 2, 3, 4, 5],
           [0, 2, 3],
           [0, 1, 5],
           [0, 1],
           [0],
           [0, 2]]
```

4. /+1 N2

```
import random
def voisin_alea(voisins, s):
    n = random.randrange(len(voisins[s]))
    return voisins[s][n]
```

5. /+0.5 N2

```
1 def marche_alea(voisins, i, n):
2     if n == 0:
3         return i
4     return marche_alea(voisins, voisin_alea(voisins, i), n-1)
```

La fonction marche_alea s'appelle elle-même la ligne 4 donc c'est une fonction récursive (suffisant ?)

6. /+0.5 N2

marche_alea(voisins, i, n) :
• voisins : liste des listes des voisins
• i : entier représentant un sommet
• n : nombre de sommets « à marcher sur »
La fonction modélise une « marche » aléatoire de n « pas » à partir du sommet i et affiche le numéro du dernier sommet marché ;
La fonction modélise un chemin aléatoire du virus qui se propage dans n ordinateurs à partir de l'ordinateur i et affiche le dernier ordinateur du chemin infecté

```
>>> marche_alea(voisins, 0, 8)
'2'
```

```
def marche_alea_MS(voisins, i, n):
    if n == 0:
        return str(i)
    return '*' + marche_alea_MS(voisins, voisin_alea_MS(voisins, i), n-1)
```

```
>>> marche_alea_MS(voisins, 0, 8)
'********2'
```

7. /+1.5 N3

```
def simule(voisins, i, n_test, n_pas):
    results = [0] * len(voisins)
    for _ in range(n_test):
        termine = int(marche_alea(voisins, i, n_pas))
        results[termine] += 1
    for j in range(len(results)):
        results[j] /= n_test
    return results
```

```
>>> simule(voisins, 4, 1000, 1000)
[0.318, 0.19, 0.187, 0.123, 0.064, 0.118]
```

```
def simule_MS(voisins, i, n_test, n_pas):
    results = [0] * len(voisins)
    for _ in range(n_test):
        results[int(marche_alea(voisins, i, n_pas))] += 1
    return [r/n_test for r in results]
```

```
>>> simule_MS(voisins, 4, 1000, 1000)
[0.297, 0.196, 0.176, 0.133, 0.069, 0.129]
```

8. /+0.5 N1

>>> simule(voisins, 4, 1000, 1000)
[0.318, 0.19, 0.187, 0.123, 0.064, 0.118]
l'ordinateur du réseau le plus infecté est le 0 donc c'est celui qu'il est le plus rentable de protéger.

9. /+1 N3

```
1  def etapes_propagation_integrale_MS(voisins, s):
2      infectes = [True] * len(voisins)
3      infection = [False] * len(voisins)
4      etapes = 0
5      infecte = voisin_alea_MS(voisins, s)
6      while infection != infectes:
7          #print(infectes, "↔", infection)
8          infection[infecte] = True
9          etapes += 1
10         infecte = voisin_alea_MS(voisins, infecte)
11     return etapes
```

```
>>> etapes_propagation_integrale_MS(voisins, 2)
20
```

Exercice 2 (6 points)

Cet exercice porte sur les réseaux et les protocoles de routage.

1. masque de sous réseau de 16 bits ; CIDR/16 : **255.255.0.0**
/+0.5 N1
2. adresse du réseau local L2 : **172.16.0.0** /16
/+0.5 N1
3. adresse de diffusion : **172.16.255.255** /16
/+0.5 N1
4. nombre maximum de machines pouvant être connectée à L2 :
$\mathbf{2^{16} - 2} = 65536 - 2 = \mathbf{65534\ machines}$ d'IPv4 différentes autorisées
/+0.5 N1
5. un chemin suivi par un message envoyé de L1 vers L2 :
→ Passerelle + interface des extraits des tables de routage des routeurs
message depuis L1 → **Routeur A** → **Routeur H** → **Routeur D** → destinataire dans L2 (3 sauts)
/+0.5 N2
6. liaison rompue entre Routeur H X Routeur D
Nouveaux chemins possibles protocole RIP :
L1 → **Routeur A** → **Routeur H** → **Routeur C** → **Routeur D** → L2 (4 sauts)
L1 → **Routeur A** → **Routeur B** → **Routeur C** → **Routeur D** → L2 (4 sauts)
/+0.5 N2
7. chemin choisi du 6. :
• L1 → Routeur A → Routeur H → Routeur C → Routeur D → L2 (4 sauts)
règle de routage modifiée pour le Routeur H

Routeur	Réseau destinataire	Passerelle	Interface
H	**L2**	**53.10.10.10**	**53.10.10.9**

• L1 → Routeur A → Routeur B → Routeur C → Routeur D → L2 (4 sauts)
règle de routage modifiée pour le Routeur H

Routeur	Réseau destinataire	Passerelle	Interface
A	**L2**	**193.55.24.2**	**193.55.24.1**

/+0.75 N3
8. Routeur H →Coûts des liaisons pour les 3 valeurs de bande passante ?(débit binaire !) :

$$1\ \text{Gbit\s} : \text{coût} = \frac{10^9}{\text{Gbit\s}} = \frac{10^9}{10^9\ \text{bit\s}} = \mathbf{1}\ \ (\text{s\bit ou s.bit}^{-1})$$

$$10\ \text{Gbit\s} : \text{coût} = \frac{10^9}{10.* 10^9\ \text{bit\s}} = \mathbf{0.1}\ \ (\text{s\bit ou s.bit}^{-1})$$

$$100\ \text{Mbit\s} : \text{coût} = \frac{10^9}{100\ \text{Mbit\s}} = \frac{10^9}{100.10^6\ \text{bit\s}} = \frac{10^9}{10^8\ \text{bit\s}} = 10^1 = \mathbf{10}\ (\text{s\bit ou s.bit}^{-1})$$

/+0.75 N2
9. chemin L1 → L2 protocole OSPF :
• L1 → **Routeur A** → **Routeur G** → **Routeur F** → **Routeur E** → **Routeur D** → L2

$$\begin{array}{lccccccc} & 1\ \text{Gbit\s} & & 10\ \text{Gbit\s} & & 10\ \text{Gbit\s} & & 10\ \text{Gbit\s} \\ \mathbf{coût} = \mathbf{1.3} \mid & 1 & + & 0.1 & + & 0.1 & + & 0.1 \end{array}$$

/+0.75 N2
10. liaison rompue entre Routeur G X Routeur F
Nouveau chemin OSPF
• L1 → **Routeur A** → **Routeur H** → **Routeur F** → **Routeur E** → **Routeur D** → L2

$$\begin{array}{lccccccc} & 1\ \text{Gbit\s} & & 1\ \text{Gbit\s} & & 10\ \text{Gbit\s} & & 10\ \text{Gbit\s} \\ \mathbf{coût} = \mathbf{2.2} \mid & 1 & + & 1 & + & 0.1 & + & 0.1 \end{array}$$

/+0.75 N3

Exercice 3 (8 points)

Cet exercice porte sur la programmation Python, la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.

Partie A

1. avantages (2 attendus) à utiliser une base de données relationnelle plutôt qu'un fichier texte ou un tableur :

- Structuration des données dans des tables
  - éviter les redondances de données
  - vérifier l'intégrité des données
    - éviter les incohérences et les erreurs
- Exécuter des requêtes complexes rapidement et efficacement
- Sécuriser des données
- Partager et communiquer des données
- …

/+0.5 N1

2. Caractéristique d'un attribut pour être utilisé en tant que clé primaire :
existence **et unicité** : n'avoir que des valeurs uniques et non nulles

/+0.5 N1

3.

• clé étrangère # id_client de la table Reservation pour lier cette table à la table Client de clé primaire id_client

• clé étrangère # id_emplacement de la table Reservation pour lier cette table à la table Emplacement de clé primaire id_emplacement

/+0.5 N1

4. Emplacement (id_emplacement, nom, localisation, tarif_journalier)
id_emplacement : clé primaire type INT
nom : type CHAR ou VARCHAR(…) ou TEXT
localisation : type CHAR(2)
tarif_journalier : type FLOAT ou REAL

/+0.5 N1

5.

```
SELECT id_emplacement, nom, localisation
FROM Emplacement
WHERE tarif_journalier = 25 ;
```

id_emplacement	nom	localisation
1	myrtille	A4
4	mandarine	B1
6	melon	A2

/+0.5 N1

6. Requête permettant de donner le nom et le prenom de tous les clients habitant à 'Strasbourg' :

```
SELECT nom, prenom
FROM Client
WHERE ville ='Strasbourg' ;
```

/+0.5 N1

7. Requête permettant d'ajouter le nouveau client :

```
INSERT INTO Client
VALUES (42, 'CODD' , 'Edgar',  '28 rue des Capucines, 'Lyon',  'France', '0555555555') ;
```

/+0.5 N1

8. Requête SQL …:

```
SELECT Client.nom, Client.prenom, Reservation.nombre_personne, Reservation.date_arrivee
Reservation.date_depart, Eplacement.tarif_journalier
FROM Reservation
JOIN Client ON Client.id_client  = Reservation.id_client
JOIN Emplacement ON Emplacement.id_emplacement = Reservation.id_emplacement
WHERE Reservation.id_reservation = 18 ;
```

ou

```
SELECT Client.nom, Client.prenom, Reservation.nombre_personne, Reservation.date_arrivee
Reservation.date_depart, Eplacement.tarif_journalier
FROM Reservation, Client, Emplacement
WHERE Client.id_client = Reservation.id_client
AND Emplacement.id_emplacement = Reservation. id_emplacement
AND Reservation.id_reservation = 18 ;
```

/+1 N2

Partie B

9. En programmation orientée objet, le terme "self" est conventionnellement utilisé comme paramètre pour faire référence à l'instance de la classe à l'intérieur de ses propres méthodes. Cela permet de différencier les attributs et les méthodes propres à l'instance en cours de traitement des attributs et des méthodes.

/+0.5 N1

10.

```
client01 = Client('CODD' , 'Edgar',  '28 rue des Capucines, 'Lyon',  'France', '0555555555')
```

/+0.5 N1

11.

```
def montant_a_regler(triplet):
    """renvoie le montant en euros à régler pour cette réservation """
    client, reservation, emplacement = triplet
    return emplacement.tarif_journlier*reservation.nb_jours() + reservation.nombre_personne*2.20**reservation.nb_jours()
    # return reservation.nb_jours()(emplacement.tarif_journlier + reservation.nombre_personne*2.20)
```

/+0.5 N1

12. Une erreur d'exécution se produit à l'exécution de la fonction facture_est_valide car …

```
25●     if not (len(annee) == 4) or not (2018<=annee<=2024):
    if not (len(annee) == 4) or not (2018<=annee<=2024):
TypeError: '<=' not supported between instances of 'int' and 'str'
```

/+0.5 N2

13. Correction du code pour que l'erreur ne se produise plus

```
25      if not (len(annee) == 4) or not (2018<=int(annee)<=2024):
```

/+0.5 N2

14.

```
27      # Reste à faire vérifier les mois MMM
28      if not mois in calendrier:
29          return False
30      #Reste à faire vérifier le numero xxx
31      if not len(numero) == 3 and not que_des_chiffres(numero):
32          return False
33      return True
```

```
>>> facture_est_valide('2024-MAY-230')
True
>>> facture_est_valide('2012-MAY-230')
False
>>> facture_est_valide('2024-MAI-230')
False
>>> facture_est_valide('2024-JUN-23')
False
```

/+1 N2