2024 Polynésie jour 2

Exercice 1

Partie A

1. Mp - Ar - Ax - Nc avec une distance de 332 km.

2. Mp - Ar - Ax - Nc ou Mp - Ar - Mr - Nc

Partie B

3.

```
G  = {  "Av" : ["Mr", "Ni", "Ax"],
        "Ni" : ["Av", "Ar", "Mp"],
        "Mr" : ["Av", "Ar", "Ax", "To", "Nc"],
        "Ax" : ["Av", "Ar", "Mr", "To", "Nc", "Di"],
        "Ar" : ["Ni", "Mr", "Mp", "Ax"],
        "Mp" : ["Ni", "Ar"],
        "To" : ["Mr", "Ax", "Nc"],
        "Nc" : ["Mr", "To", "Ax", "Di"],
        "Di" : ["Ax", "Nc"]}
```

4. LIFO : Last In First Out et FIFO : First In First Out.

5. FIFO

6. [L'ordre dépend de l'ordre des sommets dans les listes du dictionnaire qui est arbitraire]

```
['Av', 'Mr', 'Ni', 'Ax', 'Ar', 'To', 'Nc', 'Mp', 'Di']
```

7. Proposition A : parcours en largeur.

8.

```
def distance(graphe, sommet):
    dico = {sommet : 0}
    f = creerFile()
    enfiler(f, sommet)
    visite = [sommet]
    while not estVide(f):
        s = defiler(f)
        for v in graphe[s]:
            if not (v in visite):
                dico[v] = dico[s] + 1
                visite.append(v)
                enfiler(f, v)
    return dico
```

9.

```
{'Av': 0, 'Mr': 1, 'Ni': 1, 'Ax': 1, 'Ar': 2, 'To': 2, 'Nc': 2, 'Mp': 2, 'Di': 2}
```

10.
[Erreur dans l'énoncé, `visite` doit être vide à l'initialisation.]

```
def parcours2(G, s):
    p = creerPile()
    empiler(p, s)
    visite = []
    while not estVide(p):
        x = depiler(p)
        if x not in visite:
            visite.append(x)
            for v in G[x]:
                empiler(p, v)
    return visite
```

11. C'est un parcours en profondeur

```
['Av', 'Ax', 'Di', 'Nc', 'To', 'Ar', 'Mp', 'Ni', 'Mr']
```

Exercice 2

Partie A

1.

- Le nœud initial est appelé *racine*.
- Un nœud qui n'a pas de fils est appelé *feuille*.
- Un arbre binaire est un arbre dans lequel chaque nœud a *au plus deux fils*.
- Un arbre binaire de recherche est un arbre binaire dans lequel tout nœud est associé à une clé qui est :
  - supérieure à chaque clé de tous les nœuds de son *sous-arbre gauche*;
  - inférieure à chaque clé de tous les nœuds de son *sous-arbre droit*.

2. 1 - 0 - 2 - 3 - 4 - 5 - 6

3. 0 - 1 - 2 - 6 - 5 - 4 - 3

4. 0 - 1 - 2 - 3 - 4 - 5 - 6

5.

```
arbre_no1 = ABR()
arbre_no2 = ABR()
arbre_no3 = ABR()
for cle_a_inserer in [1, 0, 2, 3, 4, 5, 6]:
    arbre_no1.inserer(cle_a)
for cle_a_inserer in [3, 2, 1, 0, 4, 5, 6]:
    arbre_no2.inserer(cle_a)
for cle_a_inserer in [3, 1, 0, 2, 5, 4, 6]:
    arbre_no3.inserer(cle_a)
```

6.

— arbre_no1 : 5
— arbre_no2 : 3
— arbre_no3 : 2

7.

```
def est_present(self, cle_a_rechercher):
    if self.est_vide() :
        return False
    elif cle_a_rechercher == self.cle() :
        return True
    elif cle_a_rechercher < self.cle() :
        return self.sag().est_present(cle_a_rechercher)
    else :
        return self.sad().est_present(cle_a_rechercher)
```

8.

[Il y a une erreur dans l'énoncé la méthode s'appelle `est_present` et non pas `est_presente`]
Cela sera l'instruction 3 car elle aura 3 appels récursifs (sur 5, 6 et le nœud vide). Sur l'arbre 2 il y aura 4 appels récursifs (sur 4, 5, 6 et le nœud vide). Sur l'arbre 1 il y aura 6 appels récursifs (sur 2, 3, 4, 5, 6 et la nœud vide).

Partie B

9. Un arbre est partiellement équilibré s'il est vide ou si la différence entre les hauteurs de ses deux sous-arbres est inférieure au égale à 1.

10.

L'Arbre_1 n'est pas partiellement équilibré car la hauteur de son sous-arbre gauche est 0 et celle de son sous-arbre droit est 4, la différence est donc de 4.
L'Arbre_2 est partiellement équilibré car la hauteur de son sous-arbre gauche est 2 et celle de son sous-arbre droit est 2, la différence est donc de 0.
L'Arbre_3 est partiellement équilibré car la hauteur de son sous-arbre gauche est 1 et celle de son sous-arbre droit est 1, la différence est donc de 0.

11.

Cela ne peut être que le 2 ou le 3 car le 1 n'est pas partiellement équilibré.
Pour l'Arbre_2. Prenons le sous arbre gauche de l'Arbre_2. Ça racine est 2 et son sous-arbre gauche a une hauteur de 1 alors que son sous-arbre droit a une hauteur de -1, il n'est donc pas partiellement équilibré. Ainsi l'Arbre_2 n'est pas équilibré.
Pour l'Arbre_3. Son sous-arbre gauche a 2 sous-arbres de hauteur 0 et son sous-arbre droit a 2 sous-arbres de hauteur 0. Ils sont donc partiellement équilibré. Ces sous-arbres ont eux-mêmes des sous-arbres vides. Ils sont donc partiellement équilibrés et leurs sous-arbres étant vides, ils sont équilibrés. Ainsi, tous les sous-arbres sont équilibrés et donc l'Arbre_3 est équilibré.

12.

```
def est_equilibre(self):
    if self.est_vide():
        return True
    else:
        return self.est_partiellement_equilibre()
```

```
        and self.sag().est_equilibre()
        and self.sad().est_equilibre()
```

Exercice 3

Partie A

1. R4 - R8 - R1 - R2 ou R4 - R8 - R7 - R2

2.

Nœud R2	
Destination	Coût
R1	1
R3	3
R4	3
R5	2
R6	3
R7	1
R8	2
R9	2

3.

$$c_{\text{FTTH}} = \frac{10^8}{d} = \frac{10^8}{10 \times 10^9} = 0.01$$

4.
Calculons le coût d'une liaison FastEthernet :

$$c_{\text{FE}} = \frac{10^8}{d} = \frac{10^8}{100 \times 10^6} = 1$$

Pour minimiser le coût, il faut éviter les liaisons FastEthernet. Ainsi le chemin sera R4 - R8 - R9 - R1 - R2. Avec un coût de 0.22.

Partie B

5. Elle affiche les noms et prénom des patients dont le numéro de sécurité social commence par un 1 (donc les hommes).

6.
[La façon de traiter une date n'est pas claire. On suppose que c'est comme un text. Sinon il faut faire des inégalités sur des objets DATE.]

```sql
SELECT num_SS
FROM hospitalisation
WHERE service = "orthopédique"
AND date LIKE '%2023';
```

7.

```sql
SELECT type, date
FROM examen
JOIN patient ON patient.num_SS = examen.num_SS
WHERE nom = "Baujean" AND prenom = "Emma";
```

8.

```sql
SELECT patient.nom, patient.prenom
FROM patient
JOIN consultation ON patient.num_SS = consultation.num_SS
JOIN medecin ON consultation.id_medecin = medecin.id_medecin
WHERE medecin.nom = "ARNOS" AND medecin.prenom = "Pierre";
```

Partie C

9. [Leger problème d'indentation pour le return final.]

```python
def mdp_fort(mdp):
    if len(mdp) < 12 :
        return False
    majuscules = 0
    chiffres = 0
    symboles = 0
    for caractere in mdp :
        if caractere.isupper():
            majuscules+=1
        if caractere.isdigit() :
            chiffres+=1
        if caractere in liste_symboles :
            symboles+=1
    if majuscules < 2 or chiffres < 2 or symboles < 2 :
        return False
    return True
```

10.

```python
def creation_mdp(n, nbr_m, nbr_c, nbr_s):
    mdp = ''
    caracteres='abcdefghijklmnopqrstuvwxyz' + \
               'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' + \
               '#@!?%<>=€$+-*/&'
    majuscules = 0
    chiffres = 0
```

```
    symboles = 0
    while len(mdp) < n or majuscules < nbr_m or chiffres < nbr_c or symboles < nbr_s :
        # la variable 'c' contient un caractère
        # choisi aléatoirement dans la variable 'caracteres'
        c = choice(caracteres)
        if c.isupper() :
            majuscules+=1
        if c.isdigit() :
            chiffres+=1
        if c in liste_symboles :
            symboles += 1
        mdp = mpd + c
    return mdp
```

11.

```
def recherche_mot(mdp):
    mot = transforme(mdp)
    trouve = []
    i = 0
    while i < len(mot) :
        if mot[i].isdigit() : # si le caractère est un chiffre
            i = i+1
        elif mot[i] in liste_symboles:
            i = i+1
        else:
            # si le caractère est une lettre, on prend les
            # lettres qui la suivent jusqu'au moment où
            # on trouve un chiffre ou un symbole
            chaine = ''
            while mot[i].isalpha() :
                chaine = chaine + mot[i]
                i = i+1
            trouve.append(chaine)
    return trouve
```

12.

```
def mdp_extra_fort(mdp):
    mots = recherche_mot(mdp)
    for mot in mots:
        if len(mot) > 3 and mot in dicoFR:
            return False
    return True
```