Éléments de correction sujet 12 (2024)

Exercice 1

Partie A

1. Mp -> Ar -> Ax -> Nc avec 332 Km
2. Mp -> Ar -> Mr -> Nc ou Mp -> Ar -> Ax -> Nc

Partie B

3.
```
G = {
    'Av' : ['Mr', 'Ni', 'Ax'],
    'Ni' : ['Av', 'Ar', 'Mp'],
    'Mp' : ['Ni', 'Ar'],
    'Ar' : ['Mr', 'Mp', 'Ax', 'Mr'],
    'Mr' : ['Av', 'Ar', 'Ax', 'To', 'Nc'],
    'Ax' : ['Av', 'Ar', 'Mr', 'To', 'Nc', 'Di'],
    'To' : ['Mr', 'Nc', 'Ax'],
    'Nc' : ['Mr', 'To', 'Ax', 'Di'],
    'Di' : ['Nc', 'Ax']
}
```
4. LIFO : Last In First Out ; FIFO : First In First Out
5. Une file est une structure de type FIFO
6. Av - Mr - Ni - Ax - Ar - To - Nc - Mp - Di
7. proposition A : parcours en largeur (on utilise une file, on a donc bien un parcours en largeur)
8.
```
def distance(graphe, sommet):
    f = creerFile()
    enfiler(f, sommet)
    visite = [sommet]
    dist = {}
    dist[sommet] = 0
    while not estVide(f):
        s = defiler(f)
        for v in graphe[s]:
            if not (v in visite):
                dist[v] = dist[s] + 1
                visite.append(v)
                enfiler(f, v)
    return dist
```

9.
```
{'Av': 0, 'Mr': 1, 'Ni': 1, 'Ax': 1, 'Ar': 2, 'To': 2, 'Nc': 2,
'Mp': 2, 'Di': 2}
```
10. Il y a une erreur dans l'énoncé, à la ligne 3 de l'algo : la liste visite doit être vide au départ.

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

11. On a un parcours en profondeur :
['Av', 'Ax', 'Di', 'Nc', 'To', 'Mr', 'Ar', 'Mp', 'Ni']

Exercice 2

Partie A

1. - Le noeud initial est appelé racine
   - Un noeud qui n'a pas de fils est appelé feuille
   - Un arbre binaire est un arbre dans lequel chaque noeud a au maximum 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_inserer)
for cle_a_inserer in [3, 2, 4, 1, 5, 0, 6]:
    arbre_no2.inserer(cle_a_inserer)
for cle_a_inserer in [3, 1, 5, 0, 2, 4, 6]:
    arbre_no3.inserer(cle_a_inserer)
```

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.
erreur énoncé : est_presente à la place de est_present

arbre_no3.est_presente(7) car l'arbre_no3 a une hauteur plus faible, il y aura donc moins d'appels récursifs à faire pour constater que 7 n'est pas présent dans cet arbre.

Partie B

9.
Après l'étude de la méthode `est_partiellement_equilibre`, on peut dire qu'un arbre est partiellement équilibré s'il est vide ou si la différence de hauteur entre son sous arbre gauche et son sous-arbre droit est inférieure ou égale à 1.

10.
Arbre_1 n'est pas partiellement équilibré, car la hauteur du sous-arbre gauche de 1 est égale à 0 alors que la hauteur du sous-arbre droit de 1 est égale à 4.
Arbre_2 et Arbre_3 sont partiellement équilibrés.

11.
Arbre_2 n'est pas équilibré car la différence de hauteur du sous-arbre gauche de 2 et du sous-arbre droit de 2 est supérieure à 1. Seul 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.

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

3. $c = 10^8 / 10^{10} = 0,01$
4. R4 - R8 - R9 - R1 - R2

Partie B

5. Cette requête renvoie les noms et les prénoms des patients qui ont un numéro de sécurité sociale qui commence par à 1 (c'est-à-dire les hommes)
6.
```
SELECT num_SS
FROM patient
WHERE service = ‘orthopédique’ AND date > 31/12/2022 AND date < 01/01/2024
```

ou

```
SELECT num_SS
FROM patient
WHERE service = ‘orthopédique’ AND date LIKE ‘%2023’
```

7.
```
SELECT type, date
FROM patient
JOIN examen ON patient.num_SS = examen.num_SS
WHERE nom = ‘Baujean’ AND prenom = ‘Emma’
```
8.
```
SELECT patient.nom, patient.prenom
FROM consultation
JOIN patient ON patient.num_SS = consultation.num_SS
JOIN medecin ON medecin.id_medecin = consultation.id_medecin
WHERE medecin.nom = ’ARNOS’ AND medecin.prenom= ‘Pierre’
```

Partie C

9.
```
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 majuscule < 2 or chiffres < 2 or symboles < 2 :
            return False
    return True
```

10.

```
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 :
        c = choice(caracteres)
        if c.isdigit() :
            chiffres = chiffres + 1
        if c.isupper() :
            majuscules = majuscules + 1
        if c in liste_symboles :
            symboles = symboles + 1
        mdp = mdp + c
    return mdp
```

11.

```
def recherche_mot(mdp):
    mot = transforme(mdp)
    trouve = []
    i=0
    while i < len(mot) :
        if mot[i].isdigit() :
            i = i+1
        elif mot[i] in liste_symboles:
            i = i+1
        else:
            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 m in mots:
        if m in dicoFR and len(m) > 3:
            return False
    return True
```