Éléments de correction sujet 8 (2024)

Exercice 1

1. On peut avoir 6, 7, 8, 9 ou 10
2.
```
def effectif(val, lst):
    compt = 0
    for v in lst:
        if v == val:
            compt = compt + 1
    return compt
```
3. La liste comporte 9 valeurs, on a donc 9 comparaisons
4.
```
def majo_abs1(lst):
    for v in lst:
        eff = effectif(v, lst)
        if eff > len(lst)//2:
            return v
    return None
```
5. Pour chaque élément de la liste, on parcourt la liste en entier, on a donc 81 comparaisons
6.
```
def eff_dico(lst):
    dico_sortie = {}
    for v in lst :
        if v in dico_sortie:
            dico_sortie[v] = dico_sortie[v] + 1
        else:
            dico_sortie[v] = 1
    return dico_sortie
```
7.
```
def majo_abs2(lst):
    dico = eff_dico(lst)
    for k, v in dico.items():
        if v > len(lst) // 2:
            return k
    return None
```
8. L'élément absolument majoritaire est l'unique élément de la liste
9. Si lst1 (liste contenant n1 élément) ne contient pas d'élément absolument majoritaire, cela signifie, qu'au plus, on retrouve l'élément le plus fréquent de lst1 n1/2 (même raisonnement pour lst2 avec n2/2). Quand on fusionne les 2 listes, on trouve dans le meilleur cas n/2 = n1/2 + n2/2, on ne peut donc pas avoir d'élément absolument majoritaire dans lst.

10. Si lst1 contient un élément absolument majoritaire abs_maj1, il suffit de vérifier si abs_maj1 est aussi absolument majoritaire dans lst (fusion de lst1 et lst2)
11.
```
def majo_abs3(lst):
    n = len(lst)
    if n == 1:
        return lst[0]
    else:
        lst_g = lst[:n//2]
        lst_d = lst[n//2:]
        maj_g = majo_abs3(lst_g)
        maj_d = majo_abs3(lst_d)
        if maj_g is not None:
            eff = effectif(maj_g, lst)
            if eff > n/2:
                return maj_g
        if maj_d is not None:
            eff = effectif(maj_d, lst)
            if eff > n/2:
                return maj_d
```

Exercice 2

1. Non, car on a un [avant le 2 et un) après le 3 (on devrait avoir un])
2.
```
def compte_ouvrante(txt):
    compteur = 0
    for c in txt:
        if c == '(' or c == '[' or c == '{' :
            compteur = compteur + 1
    return compteur
```
3.
```
def compte_fermante(txt):
    compteur = 0
    for c in txt:
        if c == ')' or c == ']' or c == '}' :
            compteur = compteur + 1
    return compteur
```
4.
```
def bon_compte(txt):
    return compte_ouvrante(txt) == compte_fermante(txt)
```
5. exemple : `[()]`

6.
```
class Pile:
    def __init__(self):
        self.contenu = []
    def est_vide(self):
        return len(self.contenu) == 0
    def empiler(self, elt):
        self.contenu.append(elt)
    def depiler(self):
        if self.est_vide():
            return "La pile est vide."
        return self.contenu.pop()
```
7.
pour chaque caractère de la chaine, on doit : vérifier que ce n'est pas une parenthèse ouvrante, vérifier que ce n'est pas une parenthèse fermante et, si c'est une parenthèse fermante, comparer avec l'élément qui se trouve au sommet de la pile. Pour chaque caractère, on a donc au maximum 3 comparaisons. Pour n caractères, on a donc 3n comparaisons. Cet algorithme a donc une complexité linéaire.
8.
```
def est_bien_parenthesee(txt):
    if not bon_compte(txt):
        return False
    p = Pile()
    for c in txt:
        if c in ['(', '[', '{']:
            p.empiler(c)
        elif c in [')', ']', '}']:
            v = p.depiler()
            if c == ')' and v != '(':
                return False
            if c == ']' and v != '[':
                return False
            if c == '}' and v != '{':
                return False
    return True
```

Exercice 3

1. Non, car il est possible d'avoir 2 fois le même titre (exemple Showbiz)
2.

Welcome too the Jungle	Appetite for Destruction

3.
```
SELECT titre
FROM Chanson
WHERE album = ‘Showbiz’ ORDER BY id
```
4.
```
INSERT INTO Chanson
VALUES
(10, ‘Megalomania’, ‘Hullabaloo’, ‘Muse’)
```
5.
```
UPDATE Chanson
SET titre = ‘Welcome to the Jungle’
WHERE titre = ‘Welcome too the Jungle’
```
6.
Le regroupement des données en une seule table entrainerait une duplication des données.
7.
id_album est une clé étrangère qui permet de lier la table Chanson à la table Album
8.
```
Chanson (id : int, titre : str, #id_album : int)
Album (id : int, titre : str, année : int, #id_groupe : int)
Groupe (id : int, nom : str)
```
9.
```
SELECT Album.titre
FROM Album
JOIN Chanson ON id_album = Album.id
WHERE Chanson.titre = ‘Showbiz’
```
10.
```
SELECT Chanson.titre, Album.titre
FROM Album
JOIN Chanson ON id_album = Album.id
JOIN Groupe ON id_groupe = Groupe.id
WHERE nom = ‘Muse’
```
11.
Cette requête permet de déterminer le nombre d’albums du groupe Muse présent dans la base de données
12.
```
assert ordre_lex("", "a") == True
assert ordre_lex("b", "a") == False
assert ordre_lex("aaa", "aaba") == True
```

13.
```
def ordre_lex(mot1, mot2):
    if mot1 == "":
        return True
    elif mot2 == "":
        return False
    else:
        c1 = mot1[0]
        c2 = mot2[0]
    if c1 < c2:
        return True
    elif c1 > c2:
        return False
    else:
        return ordre_lex(mot1[1:], mot2[1:])
```
14.
```
def ordre_lex(mot1, mot2):
    if mot1 == "":
        return True
    elif mot2 == "":
        return False
    else:
        i = 0
        while i < len(mot1) and i < len(mot2):
            c1 = mot1[0]
            c2 = mot2[0]
            if c1 < c2:
                return True
            elif c1 > c2:
                return False
            else:
                i = i + 1
        return len(mot1) <= len(mot2)
```