Éléments de correction
sujet 07

Exercice 1

1. La clé primaire permet de différencier sans risque d'équivoque une entrée d'une autre, car elle doit être unique pour chaque entrée
2. On n'aurait pas pu avoir 2 matchs avec les mêmes équipes, la même équipe gagnante avec le même score (ce qui est tout à fait possible)
3.

Henri
Laure
Brigitte
Laure

4.
```
SELECT DISTINCT prenom
FROM joueur
WHERE ann_naiss < 1985
```
5.
```
SELECT nom, ann_naiss, num_port
FROM joueur
WHERE commune = 'Bois-Plage'
```
6.
```
SELECT prenom, joueur.nom
FROM joueur
JOIN equipe ON id_joueur = j_1
WHERE equipe.nom = 'Les Kangourous'
```
7.
```
UPDATE equipe
SET points = 5
WHERE nom = 'Volley Warriors'
```
8.
```
DELETE
FROM joueur
WHERE id_joueur = 35
```
9.
```
SELECT id_match
FROM match
WHERE eq_1 = 12 OR eq_2 = 12
```

10.
```
SELECT id_match
FROM equipe
JOIN joueur ON j_1 = id_joueur
JOIN match ON id_equipe = eq_1
WHERE commune = 'Bois-Plage'
```
11.
```
SELECT joueur.nom, prenom
FROM equipe
JOIN joueur ON j_1 = id_joueur
JOIN match ON id_equipe = eq_1
WHERE eq_gagnante = id_equipe
ORDER BY joueur.nom
```

Exercice 2

1. 0x9C (0x45 + 0x57)
2. On trouve 0xD7 dans les 2 cas, car les 2 mots sont composés des mêmes lettres. L'ordre des lettres n'a pas d'importance car l'addition est commutative.
3.
```
def code_hachage(mot):
    somme = 0
    for caractere in mot:
        somme = somme + ord(caractere)
    return somme % 0x100
```
4. % 0x100 permet de conserver uniquement l'octet de poids faible. La clé est donc comprise entre 0x0 et 0xFF (0 et 255 en décimal)
5. On effectue n comparaisons (complexité O(n))
6. `c in dico` est `True` si la clé `c` et présente dans le dictionnaire `dico`
7.
```
def ajouter_mot_dict(dict_mots, mot):
    code = code_hachage(mot)
    if code in dict_mots :
        lst = dict_mots[code]
        ajouter_mot_liste(lst, mot)
    else :
        dict_mots[code] = [mot]
```
8.

debut	fin
0	5
0	1
1	1

9. En divisant le problème en 2 à chaque appel
10. on a une complexité en log(n)
11.
```
def mot_present(dictMots, mot) :
    code = code_hachage(mot)
    if code in dictMots :
        return est_present(dictMots[code], mot, 0,  len(dictMots[code]))
    else :
        return False
```

Exercice 3

1. 3 - 9 - 1 - 2 - 6
2. de 1 à 9
3. 1
4. `s != self` permet de vérifier que le diviseur n'est pas lui même
   et `self.valeur % s.valeur == 0` permet de savoir si le nombre s est un diviseur
5. `[Sommet(1), Sommet(2), Sommet(3)]` (car [5] correspond à la valeur 6)
6.
```
def creer_jeu(n):
    jeu = []
    for valeur in range(1, n+1) :
        sommet = Sommet(valeur)
        sommet.relier_diviseurs(jeu)
        jeu.append(sommet)
    return jeu
```
7.
```
def lister_diviseurs(self):
    return [sommet.valeur for sommet in self.diviseurs]
```
8.
```
l_div_3 = jeu[2].lister_diviseurs()
l_mult_3 = jeu[2].lister_multiples()
assert 1 in l_div_3
assert 6 in l_mult_3
assert 9 in l_mult_3
```
9.
   On vérifie que la file n'est pas vide avant de défiler
10.
```
def taille(self):
    return len(self.donnees) - self.decalage
```

11.
```
f = File()
f.enfiler(1)
f.enfiler(2)
f.enfiler(3)
assert f.taille() == 3
assert f.defiler() == 1
assert f.defiler() == 2
assert f.defiler() == 3
assert f.est_vide()
```
12.
```
def rechercher_chemins(jeu):
    chemins_np = []
    f = File()
    for sommet in jeu :
        f.enfiler([sommet])
    while not f.est_vide():
        chemin = f.defile()
        dernier = chemin[-1]
        voisins = dernier.diviseurs + dernier.multiples
        prolongeable = False
        for voisin in voisins:
            if voisin not in chemin:
                prolongeable = True
                f.enfiler(chemin + [voisin])
        if not prolongeable:
            chemins_np.append(chemin)
    return chemins_np
```
13.
```
def valeurs_chemin(chemin):
    return [s.valeur for s in chemin]
```
14.
```
chemins = []
for chemin in rechercher_chemins(jeu):
    chemins.append(chemin_valeur(chemin))
```
15.
```
def extraire_plus_longs_chemins(L_chemins):
    L_max = 0
    L_long = []
    for c in L_chemins:
        if len(c) > L_max :
            L_long = [c]
            L_max = len(c)
        elif len(c) == L_max:
            L_long.append(c)
    return L_long
```

16. Non, car la complexité de l'algorithme fait que le nombre d'opérations à effectuer risque de devenir beaucoup trop grand pour un nombre de sommets important. Le temps d'exécution deviendra trop grand.