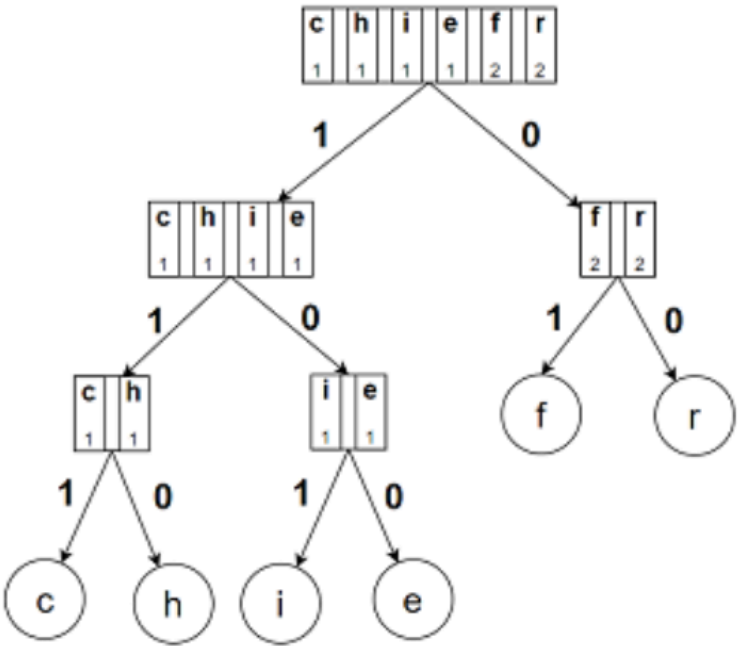
Éléments de correction
sujet 09

Exercice 1

1. 010
2. espion
3. parcours en largeur
4. à gauche : 1+1+1+1+1+1+1+2+2 = 11
   à droite : 3+4+4 = 11
5. 5
6. codage ascii : 22 caractères soit 22 octets soit 176 bits
   Shannon-Fano : 75 bits
   Nous avons donc bien un gain
7. 

8.
```
def creer_dico_occ(texte):
    dico = {}
    for symbole in texte:
        if symbole in dico:
            dico[symbole] = dico[symbole] + 1
        else:
            dico[symbole] = 1
    return dico
```
9.
```
def somme_occ(tab):
    s = 0
    for elt in tab:
        s += elt[1]
    return s
```

10.
```
def shannon(symbole, tab):
    if len(tab) == 1:
        return ""
    else:
        t1, t2 = separe(tab)
        if symbole in [elt[0] for elt in t1]:
            return "1" + shannon(symbole, t1)
        else:
            return "0" + shannon(symbole, t2)
```
11. Les appels récursifs s'arrêtent quand tab contient un seul caractère (`if len(tab) == 1`)
12.
```
def encode_shannon(texte):
    dico = creer_dico_occ(texte)
    tab = creer_tab_trie(dico)
    dico_shannon = {}
    for symbole in dico:
        dico_shannon[symbole] = shannon(symbole, tab)
    code = ''
    for caractere in texte:
        code = code + dico_shannon[caractere]
    return code
```

Exercice 2

1. Parce qu'il est possible d'avoir plusieurs adhérents avec le même nom. Or, la clé primaire doit être unique pour chaque entrée.
2. Donne le nom du jeu et le nom de l'éditeur de tous les jeux. Les jeux étant classés par ordre alphabétique de leur nom.
3.
```
SELECT nomJeu
FROM emprunt
WHERE dateRendu = NULL
```
4.
```
SELECT nom, prenom
FROM adherent
JOIN emprunt ON emprunt.idAdherent = adherent.idAdherent
WHERE nomJeu = 'Catan'
```
5.
```
UPDATE emprunt
SET dateRendu = '2025-06-03'
WHERE idEmprunt = 1538
```

6.
```
SELECT nomJeu, categorie
FROM jeu
WHERE anneeSortie >= 2010-01-01 AND ageMinimum < 10
```
7.
clés étrangères de participation :
- idAdherent (lien vers idAdherent de adherent)
- nomEvenement (lien vers nom de evenement)

8.
```
dict_emprunts = {}
for jeu in tab:
    if jeu in dict_emprunts:
        dict_emprunts[jeu] += 1
    else:
        dict_emprunts[jeu] = 1
```
9.
```
def le_podium(dico):
    podium = [[], [], []]
    scores_podium = [0, 0, 0]
    for i in range(3):
        for jeu, score in dico.items():
            if score > scores_podium[2-i] and score not in scores_podium:
                scores_podium[2-i] = score
    for i in range(3):
        for jeu, score in dico.items():
            if score==scores_podium[i]:
                podium[i].append(jeu)
    return podium
```

Exercice 3

1. PGRDX
2.
```
def indice(L, element):
    for i in range(len(L)):
        if L[i] == element:
            return i
```
3.
```
def lettres_vers_indices(txt):
    t = []
    for c in txt:
        t.append(indice(alphabet,c))
    return t
```

4.
```
def chiffrement(msg, cle):
    assert len(cle) >= len(msg), 'impossible'
    indices_msg = lettres_vers_indices(msg)
    indices_cle = lettres_vers_indices(cle)
    n = len(msg)
    indices_msg_chiffre = []
    for k in range(n):
        ind = indices_msg[k] + indices_cle[k]
        if ind >= 26:
            ind = ind - 26
        indices_msg_chiffre.append(ind)
    msg_chiffre = indices_vers_lettres(indices_msg_chiffre)
    return msg_chiffre
```
5. On obtient une AssertionError `'impossible'` car la longueur de la clé est inférieure à la longueur du message.
6. BRAVO
7. Pour chaque caractère du message chiffré, on soustrait, à sa position dans l'alphabet, la position du caractère associé dans le masque. Si la valeur obtenue est strictement négative, on ajoute 26. On obtient finalement la position du caractère en clair dans l'alphabet.
8.
```
def dechiffrement(msg, cle):
    assert len(cle) >= len(msg), 'impossible'
    indices_msg = lettres_vers_indices(msg)
    indices_cle = lettres_vers_indices(cle)
    n = len(msg)
    indices_msg_dechiffre = []
    for k in range(n):
        ind = indices_msg[k] - indices_cle[k]
        if ind < 0:
            ind = ind + 26
        indices_msg_dechiffre.append(ind)
    msg_dechiffre = indices_vers_lettres(indices_msg_dechiffre)
    return msg_dechiffre
```
9. Dans un algorithme de chiffrement symétrique, la même clé secrète est utilisée pour chiffrer et déchiffrer des messages. Un algorithme de chiffrement asymétrique repose sur l'existence d'une paire de clés : la clé privée, qui doit être gardée secrète par son propriétaire, et la clé publique qui peut être communiquée à tous.

10. Bob doit utiliser sa clé privée pour déchiffrer le message envoyé par Alice (Alice aura utilisé la clé publique de Bob pour chiffrer le message).
11. Une tierce personne peut utiliser la clé publique de Bob pour envoyer un message à Bob. Ce message sera déchiffré par Bob, il n'aura aucun moyen de s'assurer que le message vient bien d'Alice.
12. HTTPS est la version sécurisée de HTTP, le but de HTTPS est de sécuriser les communications. HTTPS s'appuie sur le protocole TSL (Transport Layer Security) anciennement connu sous le nom de SSL (Secure Sockets Layer)
    Comment chiffrer les données circulant entre le client et le serveur ?
    Les communications vont être chiffrées grâce à une clé symétrique. Problème : comment échanger cette clé entre le client et le serveur ? Simplement en utilisant une paire clé publique / clé privée !
    Voici le déroulement des opérations :
    - le client effectue une requête HTTPS vers le serveur, en retour le serveur envoie sa clé publique (KpuS) au client
    - le client "fabrique" une clé K (qui sera utilisé pour chiffrer les futurs échanges), chiffre cette clé K avec KpuS et envoie la version chiffrée de la clé K au serveur
    - le serveur reçoit la version chiffrée de la clé K et la déchiffre en utilisant sa clé privée (KprS). À partir de ce moment-là, le client et le serveur sont en possession de la clé K

    le client et le serveur commencent à échanger des données en les chiffrant et en les déchiffrant à l'aide de la clé K (chiffrement symétrique).
13. Le chiffrement symétrique est beaucoup moins "gourmand" en termes de calculs que le chiffrement asymétrique. Faire l'ensemble de la communication avec le chiffrement asymétrique demanderait beaucoup trop de ressources.
14. Les 2 machines ne sont pas sur le même réseau local (192,.168.110.0/24 et 192,168,100,0/24), le ping ne peut donc pas aboutir.
15. 255.255.255.224
16. 5 bits pour la partie machine, on a donc 2^5 = 32 adresses possibles
17. 10000110
18. Pour l'adresse 192.168.110.134/27 (Zoé) l'adresse réseau est 192.168.110.128
    Pour l'adresse 192.168.110.115/27 (commande n°1) l'adresse réseau est 192.168.110.32
    Pour l'adresse 192.168.110.153/27 (commande n°2) l'adresse réseau est 192.168.110.128
    La commande ping fonctionne donc pour la commande n°2