Éléments de correction
sujet 03

Exercice 1

1. On doit arriver à 25 caractères, on a pour le moment 17 caractères, on doit donc ajouter 8 espaces.
2. `An(3 espaces)algorithm(3 espaces)must(2 espaces)be`
3. `assert nb_caracteres + (nb_mots-1) <= justification`
4.
```
def ajout_espace(liste_mots, justification):
    nb_caracteres = sum([len(mot) for mot in liste_mots])
    nb_mots = len(liste_mots)
    nb_espace_total = justification - nb_caracteres
    if nb_mots == 1:
        return liste_mots[0] + " " * nb_espace_total
    else :
        q = nb_espace_total // (nb_mots - 1)
        r = nb_espace_total % (nb_mots - 1)
        reponse = liste_mots[0]
        for i in range(1, r + 1):
            reponse = reponse + " " * (q + 1) + liste_mots[i]
        for i in range(r + 1, nb_mots):
            reponse = reponse + " " * q + liste_mots[i]
        #ligne ajoutée sinon par le dernier mot :
        reponse = reponse + liste_mots[-1]
    return reponse
```
5. On place le plus de mots possibles sur la première ligne puis de passer à la 2e ligne (on place là aussi le plus de mots possible) et ainsi de suite
6.
```
def affiche_justifie(liste_mots, decoupage, justification):
    for i,j in decoupage:
        ligne_justifiee = ajout_espace(liste_mots[i:j],
justification)
        print(ligne_justifiee)
```
7.

Coût total du découpage : 147					
i mot deb	i mot fin+1	nbre mots	nombre car	esp sup	coût
0	2	2	11	3	9
2	4	2	6	8	64
4	7	3	8	5	25
7	8	1	8	7	49

8.
```
def cout(i, j, liste_mots, justification):
    n_mots = j - i
    n_caracteres = sum([len(liste_mots[k]) for k in range(i,j)])
    if n_caracteres + (n_mots - 1) > justification:
        return 1000000
    else:
        return (justification - n_caracteres - n_mots + 1) ** 2
```
9. Non, ce n'est pas raisonnable, car la complexité algorithmique est trop grande (2^n)
10. coût quadratique car boucles imbriquées
11. `cout_mini[j] = mininmun(cout_mini[j] + cout(i, j, liste_mots, justification))` pour j compris entre i+1 et n exclu (voir la boucle for)
12. `return [cout_mini[0], decoupage]`

Exercice 2

1. feuille : (j,1) ; racine : (-j-f-e-l-i-p-t-a-u,19)
2. 4 ; 1100
3. plus la profondeur est petite, moins il y a besoin de bits pour coder un caractère et donc plus la compression est importante (pour le p on utilise 4 bits au lieu de 8 sans compression)
4.
```
class Noeud:
    def __init__(self, nom, nb_occu, fils_g, fils_d):
        self.nom = nom
        self.nb_occu = nb_occu
        self.fils_g = fils_g
        self.fils_d = fils_d
    def __str__(self):
        return '(' + self.nom + ',' + str(self.nb_occu) + ')'
```
5.
```
def liste_occurrences(chaine):
    dico = {}
    for c in chaine:
        if c in dico:
            dico[c] = dico[c] + 1
        else:
            dico[c] = 1
    liste_res = []
    for cle in dico:
        liste_res.append((cle, dico[cle]))
    return liste_res
```

6.
```
def tri_liste(liste_a_trier):
    liste_triee = []
    for i in range(0, len(liste_a_trier)):
        element = liste_a_trier[i]
        j = 0
        while (j < len(liste_triee) and element[1] >= liste_triee[j][1]):
            j = j + 1
        liste_triee.insert(j, element)
    return liste_triee
```
7.
```
def conversion_en_noeuds(lst):
    t = []
    for t in lst:
        lst.append(Noeud(t[0],t[1], None, None))
    return t
```
8.
```
def insere_noeud(noeud, liste_noeud):
    j = 0
    while j < len(liste_noeud) and noeud.nb_occu > liste_noeud[j].nb_occu:
        j = j + 1
    liste_noeud.insert(j, noeud)
```
9.
```
def construit_arbre(liste):
    while len(liste) > 1:
        noeud1 = liste.pop(0)
        noeud2 = liste.pop(0)
        nom_noeud_pere = noeud1.nom + "-" + noeud2.nom
        nb_occu_noeud_pere = noeud1.nb_occu + noeud2.nb_occu
        noeud_pere = Noeud(nom_noeud_pere,nb_occu_noeud_pere, noeud1, noeud2)
        insere_noeud(noeud_pere, liste)
    return liste[0]
```

10. Il s'agit d'un dictionnaire (tableau associatif)
11.
```
def compresse(chaine, dico):
    chaine_resultat = ""
    for c in chaine:
        chaine_resultat += dico[c]
    return chaine_resultat
```

Exercice 3

1. `a4.duree = 12`
2. la valeur est 4 ; 3e tuple (indice 2) et 2e valeur (indice 1)
3. Cette ligne permet de créer les arêtes entre l'attraction 3 et les attractions 1, 2 et 4
4. `a4.voisines = [(a2,4), (a3, 6)]`
5. Parce que c'est le même chemin pour aller de l'attraction A à l'attraction B que pour aller de l'attraction B à l'attraction A (avec aussi la même durée).

6. déplacements : de a1 à a2 = 7 ; de a2 à a3 = 3 soit 10 minutes de déplacement. On doit ensuite ajouter la durée des attractions : 11 + 6 + 9 = 26 minutes
   Au total, la balade dure 10 + 26 = 36 minutes
7. Il n'y a pas d'arête entre a1 et a4. Il ne s'agit donc pas d'une balade.
8.
```
def sont_voisines(atr1, atr2):
    for v in atr1.voisines:
        if atr2 == v[0]:
            return True
    return False
```
9.
```
def est_balade(tab):
    for i in range(len(tab)-1):
        if not sont_voisines(tab[i], tab[i+1]):
            return False
    return True
```
10.
   parcours en profondeur (appel récursif)
11.
   On obtient : `[a4, a2, a1, a3]`
12.
   On obtient : `[a3, a1, a2, None]`
13.
   `deja_vu` est un dictionnaire
14.
   Une clé primaire est un attribut qui permet d'indentifier de manière chaque entrée
   Une clé étrangère est un attribut permettant de relier 2 tables (jointure)
15.
```
SELECT DISTINCT nom, prenom
FROM visiteur
WHERE date = '2025-01-11'
```
16.
```
SELECT SUM(prix)
FROM photo
JOIN visiteur ON visiteur.id = id_visiteur
WHERE nom = 'Turing' AND prenom = 'Alan' AND date < '2025-1-1' AND
date > '2023-12-31'
```
17.
   Il voulait connaitre le nom et le prénom des visiteurs ayant été sur la grande roue le 26 juillet 2024 à 12h34
18.
   On peut ajouter une nouvelle table format qui serait lié à la table photo à l'aide d'une clé étrangère.