2025 Amérique du nord jour 2 bis

Exercice 1

Partie A

1. Le nombre total de caractères est 17. Il faut donc $25 - 17 = 8$ espaces pour la ligne.

2. `8 // 3 = 2` et il reste deux espaces à répartir. On a donc 3-3-2 comme répartition. Ainsi c'est la troisième proposition : `An---algorithm---must--be`

3. Le nombre de caractères plus le nombre d'espace inter-mots doit être inférieur ou égal à la justification. Or le nombre d'espaces inter-mots est égal au nombre de mots moins un :

```
assert nb_caracteres + nb_mots - 1 <= justification
```

4.

```
def ajout_espace(liste_mots: list[str], justification: int) -> str:
    nb_caracteres = sum([len(mot) for mot in liste_mots])
    nb_mots = len(liste_mots)
    assert nb_caracteres + nb_mots - 1 <= justification
    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]
        return reponse
```

Partie B

5. Le premier mot appartient à la ligne car il est plus petit que la justification. Ensuite on regarde si en ajoutant la taille du mot suivant plus un (pour une espace) la taille reste inférieure à la justification. Si oui, on ajoute le mot à la ligne sinon, il faut passer à la ligne suivante.

6.

```
def affiche_justifie(liste_mots, decoupage, justification) -> None:
    # début de la boucle d'affichage justifié
    for a,b in decoupage:
        ligne_justifiee = ajout_espace(liste_mots[a : b], justification)
        print(ligne_justifiee)
```

Partie C

7.

Coût du découpage total : 147					
ind. mot déb	ind. mot fin+1	nb mots	nb caract.	esp. supp.	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):
    nb_mots = j - i
    nb_espaces = nb_mots - 1
    nb_caract = 0
    for k in range(i, j):
        nb_caract += len(liste_mots[k])
    if nb_caract + nb_espaces <= justification:
        return (justification - nb_caract - nb_espaces)**2
    else:
        return 1000000
```

9. Non, cela n'est pas raisonnable car il y a $2^{(n-1)}$ possibilités. Le test de toutes les possibilités risque de prendre énormément de temps.

10. L'ordre de grandeur est n^2 car il y a deux boucles imbriquées.

11. [Je ne comprends pas vraiment la question, je ne trouve pas de relation simple entre les éléments de `cout_mini`]

12. Il faut faire la somme des coûts de chaque découpage. On peut ajouter ce calcul dans le `while` final :

```
decoupage = []
k = 0
cout_total = 0
while k < n:
    decoupage.append((k, indice_retour_ligne_mini[k]))
    cout_total += cout_mini[k]
    k = indice_retour_ligne_mini[k]
return cout_total, decoupage
```

Exercice 2

1. `(j,1)` est une feuille et `( -j-f-e-l-i-p-t-a-u,19)` est la racine.

2. La profondeur est 4 et le code est `1100`.

3. Le codage binaire sera plus court si les caractères les plus fréquents ont un codage plus court. Et donc la transmission du codage sera plus rapide.

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):
        """
        Renvoie une chaine contenant les donnees
        du noeud (nom et nombre d'occurrences)
        """
        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.

[C'est un tri par insertion particulier car on parcourt la liste triée de gauche à droite et on y insère l'élément à trier. Il faut bien voir que le `len(liste_triee)` varie à chaque itération.]

```
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(liste_tuples):
    liste_noeuds = []
    for c, nb_occu in liste_tuples:
        noeud = Noeud(c, nb_occu, None, None)
        liste_noeuds.append(noeud)
    return liste_noeuds
```

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.occu + noeud2.occu
        noeud_pere = Noeud(nom_noeud_pere, nb_occu_noeud_pere, noeud1, noeud2)
        insere_noeud(noeud_pere, liste)
    return liste[0]
```

10. Dictionnaire

11.

```
def compresse(texte, codage):
    s = ""
    for c in texte:
        s += codage[c]
    return s
```

Exercice 3

Partie A

1. clpa4.duree = 12

2. 4, on accède à l'élément d'indice 1 de la voisine d'indice 2 des Petits chevaux.

3. On ajoute les voisines "Grand huit", "Petits chevaux" et "Grande roue" au "Train fantôme" en donnant les temps de déplacement respectifs 5, 3 et 6.

4. clpa4.voisines = [(a2, 4), (a3, 6)]

5. Les temps de déplacement entre deux attractions ne dépendent pas du sens de parcours.

6. On ajoute les temps de déplacement et la durée des attractions : $d = 11+7+6+3+9 = 36$ min.

7. Il n'y a pas d'arête entre le Grand huit (a1) et la Grande roue (a4).

8.

```
def sont_voisines(a, b):
    for v in a.voisines:
        if b = v[0]:
            return True
    return False
```

9.

```
def est_balade(ls):
    for i in range(len(ls) - 1):
        if not sont_voisines(ls[i], ls[i+1]):
            return False
    return True
```

10. Parcours en profondeur.

11. `[a4, a2, a1, a3]`

12. `[a3, a1, a2, None]`

13. C'est un dictionnaire. Il sert à mémoriser les sommets déjà visités pour qu'ils n'apparaissent qu'une seul fois.

Partie B

14. Une clé primaire est un champ unique d'une relation permettant d'identifier de manière unique une ligne.
Une clé étrangère est un champ permettant de faire le lien avec la clé primaire d'une autre table.

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 prenom = 'Alan' AND nom  'Turing'
AND date >= '2024-01-01' AND date <= '2024-12-31';
```

17. Ils voulaient savoir quels visiteurs avaient été pris en photo le 26 juillet 2024 à 12h34 dans la Grande roue.

18. Il faudrait enlever le champ `prix` de la relation photo. Ajouter une relation `format (id, type, prix)` contenant tous les formats disponibles et leur prix. Ensuite il faut une relation pour faire le lien entre les photos et les formats choisis par les visiteurs : `commande (#id_photo, #id_format)`.