Ce document a été mis en ligne par l'organisme FormaV®.

Toute reproduction, représentation ou diffusion, même partielle, sans autorisation préalable, est strictement interdite.

Pour consulter tous nos sujets d'annales et corrigés du Bac Général Maths + NSI, veuillez visiter :
bac-maths-nsi.fr/annales-sujets-et-corriges

Corrigé 25NSIPE2

Exercice 1 (6 points)

Cet exercice porte sur la programmation en Python et la programmation dynamique.

Partie A

1. Les mots 'An', 'algorithm', 'must', 'be' de la citation de Donald Knuth contiennent $2+9+4+2=17$ caractères, le nombre d'espaces nécessaires pour un alignement justifié de 25 caractères est donc $25-17=8$.
2. La seule proposition respectant les règles de l'alignement justifié est `An---algorithm---must--be`.
3. La précondition de la ligne 4 est `assert nb_caracteres+nb-mots-1 <= justification`.
4. On complète les lignes 7, 13 et 16 du code de la fonction `ajout_espace`.

```
7           return liste_mots[0] + " " * nb_espace_total
13              reponse = reponse + " " * (q+1) + liste_mots[i]
16              reponse = reponse + " " * q + liste_mots[i]
```

Partie B

5. On peut ajouter les longueurs des mots et un pour l'espace de séparation tant qu'on ne dépasse pas la justification. On revient à la ligne avant le mot qui dépasse.
6. On complète les lignes 5 à 8 du code de la fonction `affiche_justifie`.

```
1  def affiche_justifie(liste_mots: list[str],
2                       decoupage:list[(int,int)],
3                       justification:int) -> None:
4      # début de la boucle d'affichage justifié
5      for i,j in decoupage:
6          ligne_justifiee = \
7              ajout_espace(liste_mots[i:j], justification)
8          print(ligne_justifiee)
```

Partie C

7. On complète le tableau.

Coût total du découpage : 147					
Indice du mot de début	Indice du mot de fin +1	Nombre de mots	Nombre de caractères	Nombre d'espaces supplémentaires pour atteindre 15 caractères	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. On écrit le code d'une fonction `cout` qui convient.

```
def cout(i,j,liste_mots,justification):
    nb_mots = j-i
    nb_espaces_supplementaires = justification \
                                 -(sum([len(liste_mots[k]) for k in range(i,j)])+nb_mots-1)
    if nb_espaces_supplementaires<0:
        return 1000000
    else:
        return nb_espaces_supplementaires**2
```

9. On a n choix binaires successifs à effectuer pour décrire une possibilité (passer à la ligne ou pas après chaque mot), ce qui donne de l'ordre de $2^n \geqslant 2^{50} \approx 10^{15}$ soit un million de milliards de cas à examiner. C'est une complexité exponentielle. Même avec un milliard de cas traités par seconde, cela nécessiterai une grosse dizaine de jours (et presque 32 ans pour $n = 60$, et plus de 30000 milliards d'années pour $n = 100$), ce n'est pas raisonnable.
10. La boucle des lignes 9 à 22 va être exécutée n fois, avec i variant de $n-1$ à 0 (bornes incluses) et dans cette boucle la fonction `cout` est appelée une fois ligne 12 et $n-i-1$ fois ligne 18. Donc dans un appel à la fonction `justifie_dynamique` la fonction `cout` sera appelée $0+1+2+\cdots+(n-1) = \dfrac{n(n-1)}{2} \approx \dfrac{1}{2}n^2$ soit un ordre de grandeur quadratique en fonction de n le nombre de mots dans la liste.
11. Après la boucle des lignes 9 à 22, `cout_mini[i]` contient le coût minimum pour justifier la partie du texte composée des mots `liste_mots[i:]`, et `indice_mini[i]` contient l'indice `j` tel que couper après le mot `i` amène une ligne correspondant au couple `(i,j)` dans la liste de découpage produite.
    La relation mathématique entre les éléments de la liste `cout_mini` est alors

`cout_mini[i] = min(cout_mini[j] + cout(i, j, liste_mots, justification) for j in range(i+1, n+1))`.

12. On peut changer la dernière ligne en `return decoupage, cout_mini[0]`.

Exercice 2 (6 points)

Cet exercice porte sur les arbres binaires et la représentation binaire.

1. Dans l'arbre de la figure 1, un exemple de feuille est `(j,1)`, la racine de l'arbre est `( -j-f-e-l-i-p-t-a-u,19)`.
2. La profondeur du nœud correspondant au caractère `p` est 4, et son code binaire associé est `1100`.
3. Associer un code plus court aux caractères fréquents permet de minimiser la longueur totale du codage de la phrase.
4. On complète les lignes 3, 4, 5, 6 et 13 de la classe `Noeud`.

```
1  class Noeud:
2      def __init__(self, nom, nb_occu, fils_g, fils_d):
3          self.nom = nom
4          self.nb_occu = nb_occu
5          self.fils_g = fils_g
6          self.fils_d = fils_d
7
8      def __str__(self):
9          """
10         Renvoie une chaîne contenant les données
11         du noeud (nom et nombre d'occurrences)
12         """
13         return '('+self.nom+','+str(self.nb_occu)+')'
```

5. On complète le code de la fonction `liste_occurrences`.

```
1  def liste_occurrences(chaine):
2      dico = dict()  # ou dico = {}
3      for c in chaine:
4          if c in dico:
5              dico[c] = dico [c] + 1
6          else:
7              dico[c] = 1
8      liste_res = []
9      for cle in dico:
10         liste_res.append( (cle,dico[cle]) )
11     return liste_res
```

6. On complète le code de la fonction `tri_liste`.

```
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. On écrit la fonction `conversion_en_noeuds` demandée qui convertit une liste de tuples en liste de nœuds.

```
def conversion_en_noeuds(liste_tuples):
    liste_noeuds = []
    for nom, nb_occu in liste_tuples:
        liste_noeuds.append(Noeud(nom, nb_occu, None, None))
    return liste_noeuds
```

ou bien

```
def conversion_en_noeuds(liste_tuples):
    liste_noeuds = []
    for t in liste_tuples:
        liste_noeuds.append(Noeud(t[0], t[1], None, None))
    return liste_noeuds
```

ou bien

```
def conversion_en_noeuds(liste_tuples):
    return [Noeud(nom, nb_occu, None, None) for nom, nb_occu in liste_tuples]
```

8. On complète le code de la fonction `insere_noeud`.

```
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. On complète le code de la fonction `construit_arbre`.

```
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. La strucuture de données renvoyée par la fonction `codage_arbre` est un tableau associatif, qui en Python s'appelle un dictionnaire.
11. On écrit une fonction `compresse`.

```
def compresse(texte, dico):
    suite = ''
    for c in texte:
        suite = suite + dico[c]
    return suite
```

Exercice 3 (8 points)

Cet exercice porte sur la programmation objet en langage Python, les graphes et les bases de données.

Partie A

1. La modification demandée s'écrit `a4.duree = 12`.
2. L'expression `a2.voisines[2][1]` donne la distance de `a2` à la troisième attraction voisine de `a2`, c'est-à-dire `a4` ; elle vaut `4`.
3. La ligne 7 de ce code, `a3.voisines = [(a1, 5), (a2, 3), (a4, 6)]`, créée les arêtes entre le somet « Train fantôme » et ses trois attractions voisines, en indiquant de plus les temps de trajet.
4. On complète la ligne 8 du code.

```
8 a4.voisines = [(a2, 4), (a3, 6)]
```

5. La modélisation du parc d'attractions est réalisée avec un graphe non orienté car le temps de trajet entre deux attractions est le même dans un sens ou dans l'autre.
6. La durée en minutes de la balade représentée par le tableau `[a1, a2, a3]` est $11+7+6+3+9=36$ minutes, durée obtenue en ajoutant les durées des trois attractions « Grand huit », « Petits chevaux » et « Train fantôme » et les durées pour aller d'une attraction du parcours à la suivante.
7. Le tableau `[a2, a1, a4, a3]` n'est pas une balade du parc d'attractions car le « Grand huit » (`a1`) et la « Grande roue » (`a4`) ne sont pas des attractions voisines.
8. On écrit une fonction booléenne `sont_voisines`.

```
1 def sont_voisines(a1,a2):
2     for b in a1.voisines:
3         if a2==b[0]:
4             return True
5     return False
```

9. On écrit une fonction `est_balade`.

```
1 def est_balade(tableau):
2     for i in range(len(tableau)-1):
3         if not sont_voisines(tableau[i], tableau[i+1]):
4             return False
5     return True
```

10. Le parcours effectué par la fonction `parcours` est un parcours en profondeur (DFS depth first search).
11. Le tableau `balade` contient `[a4, a2, a1, a3]` après l'exécution du code proposé.
12. Le tableau `tableau` contient alors `[a3, a1, a2, None]`.
13. La variable `deja_vues` contient un dictionnaire dont les clés sont les attractions et les valeurs sont des booléens, et permet de ne pas explorer plusieurs fois les mêmes sommets.

Partie B

14. Une clé primaire est une clé qui permet d'identifier de façon unique chaque ligne d'une relation, une clé étrangère est une clé qui permet de référencer une ligne d'une autre relation..
15. La requête `SELECT DISTINCT nom, prenom FROM visiteur WHERE date = '2025-01-11';` convient.
16. La requête `SELECT SUM(prix) FROM photo JOIN visiteur ON photo.id_visiteur=visiteur.id WHERE prenom='Alan' AND nom='Turing' AND '2024-01-01' <= date AND date <= '2024-12-31';` convient.
17. On peut penser que les gérants veulent le nom et le prénom de tous les visiteurs qui ont été pris en photo sur la Grande roue le 26 juillet 2024 à midi trente-quatre.
18. On peut par exemple créer une nouvelle table `tirage` contenant un champ entier `id` comme clé primaire, un champ texte `format` qui contient le format (A5, A6, poster, porte-clés ...), un champ entier `id_photo` qui référence la photo utilisée. On peut alors supprimer le champ `prix` de la table `photo`.

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.