Exercice 1 (6 points)

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

Dans cet exercice, on se réfère à la citation suivante de Donald Knuth :

« An algorithm must be seen to be believed. »

En typographie, il existe plusieurs manières d'aligner un texte : aligner à gauche, centrer, aligner à droite et justifier.

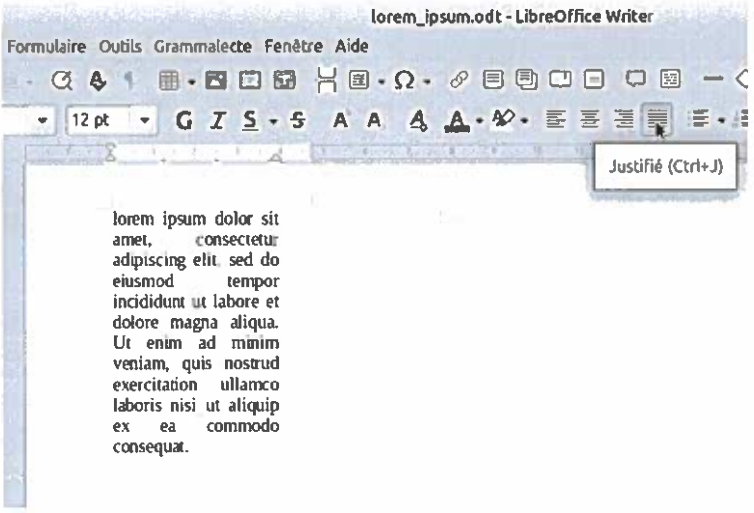

Figure 1. Justification d'un texte à l'aide d'un traitement de texte

L'alignement justifié permet d'obtenir que chaque ligne ait la même longueur appelée *justification*. Pour cela et pour chacune des lignes, on ajoute si nécessaire des espaces supplémentaires. Dans tout cet exercice, on désigne par *une espace* exactement un caractère d'espacement, tel que contenu dans la chaine Python `' '`.

On répartit ces espaces supplémentaires ainsi :

- s'il n'y a qu'un mot on insère les espaces à droite de celui-ci ;
- sinon
  - on effectue la division euclidienne du nombre total d'espaces nécessaires pour compléter la ligne par le nombre d'emplacements inter-mots (entre les mots) ;
  - on répartit le quotient d'espaces entre chaque emplacement inter-mot ;
  - puis, s'il reste des espaces à distribuer, on les répartit une par une de gauche à droite.

Dans toute la suite, pour simplifier, on considère que tous les caractères, espaces comprises, ont la même largeur. Les mots ne sont ni coupés, ni décorés (gras, italique, etc.).

Partie A

1. On considère les quatre premiers mots `'An'`, `'algorithm'`, `'must'`, `'be'` de la citation de Donald Knuth avec un alignement justifié de 25 caractères, c'est-à-dire que la ligne contient 25 caractères au total, en comptant les lettres et les espaces.

   Montrer que, pour cette justification, le nombre d'espaces nécessaires est de 8.

2. On souhaite répartir ces 8 espaces entre ces quatre mots.

   Déterminer la seule proposition, parmi les quatre propositions ci-après, respectant les règles de l'alignement justifié définies précédemment, pour une justification de 25 caractères. Le caractère - représente ici une espace.

   - `An--algorithm---must---be`
   - `An----algorithm--must--be`
   - `An---algorithm---must--be`
   - `An---algorithm--must---be`

On considère le code de la fonction `ajout_espace`, donnée ci-après, qui prend en paramètres une liste `liste_mots` non vide de chaînes de caractères représentant un ensemble de mots et un entier `justification` représentant la justification. La fonction renvoie une chaîne de caractères constituée des mots de `liste_mots` à laquelle on ajoute des espaces pour la justifier selon `justification`.

La fonction `sum` en Python est utilisée pour calculer la somme des éléments contenus dans une liste.

On rappelle qu'en Python, le caractère `\` placé en fin de ligne permet de continuer l'expression Python sur la ligne suivante, comme si on n'avait en fait pas sauté de ligne. Cela sert uniquement à améliorer la présentation du code en évitant d'avoir des lignes trop longues. Par exemple les lignes :

```
reponse = reponse + " " * ... \
            + liste_mots[i]
```

signifient exactement la même chose que la ligne :

```
reponse = reponse + " " * ... + liste_mots[i]
```

```
1 def ajout_espace(liste_mots: list[str],
```

```
2                   justification: int) -> str:
3       nb_caracteres = sum([len(mot) for mot in liste_mots])
4       nb_mots = len(liste_mots)
5       assert nb_caracteres + ...
6       nb_espace_total = justification - nb_caracteres
7       if nb_mots == 1:
8           return ... + " " * nb_espace_total
9       else:
10          q = nb_espace_total // (nb_mots - 1)
11          r = nb_espace_total % (nb_mots - 1)
12          reponse = liste_mots[0]
13          for i in range(1, r + 1):
14              reponse = reponse + " " * ... \
15                  + liste_mots[i]
16          for i in range(r + 1, nb_mots):
17              reponse = reponse + " " * ... \
18                  + liste_mots[i]
19          return reponse
```

3. Dans la fonction `ajout_espace` ci-dessus, il peut arriver que la liste de mots `liste_mots` soit trop longue pour tenir sur une seule ligne de justification donnée. Compléter la précondition de cette fonction à la ligne 5 avec un critère que doit vérifier la liste `liste_mots`, pour être justifiable sur une seule ligne.

Dans le cas d'une liste d'au moins deux mots, si on a `nb_espace_total` espaces à répartir entre `nb_mots - 1` emplacements inter-mots selon les règles de l'alignement justifié, avec `q` et `r` respectivement le quotient et le reste de la division euclidienne de `nb_espace_total` par `nb_mots - 1`, il suffit de :

- placer `q + 1` espaces pour les `r` premiers emplacements inter-mots ;
- placer `q` espaces pour les emplacements inter-mots suivants allant de `r+1` à `nb_mots - 1`.

4. Recopier et compléter les lignes 8, 14 et 17 du code de la fonction `ajout_espace`.

Partie B

Dans cette partie on cherche à déterminer après quel mot revenir à la ligne. On admet que la justification est supérieure à la longueur des mots considérés, ainsi chaque mot peut tenir sur une ligne.

5. Proposer un algorithme en langage naturel permettant de déterminer après quel mot d'un texte revenir à la ligne, étant donné une certaine justification. On attend uniquement une explication précise de l'algorithme, pas sa programmation en Python.

On utilise une liste de tuples pour modéliser le découpage d'un texte en différentes lignes. Par exemple, pour la liste de mots `['An', 'algorithm', 'must', 'be',`

`'seen', 'to', 'be', 'believed']` la liste de découpage `[(0, 2), (2, 5), (5, 7), (7, 8)]` signifie que :

- la première ligne sera constituée des mots d'indice de 0 à 2 (2 exclu), soit `An algorithm` ;
- la seconde ligne des mots de 2 à 5 (5 exclu), soit `must be seen` ;
- la troisième ligne des mots de 5 à 7 (7 exclu), soit `to be` ;
- et la dernière ligne des mots de 7 à 8 (8 exclu), soit `believed`.

6. Recopier et compléter les lignes 5 à 8 du code de la fonction `affiche_justifie`, donné ci-après, qui prend en paramètres une liste de mots, une liste de découpage et une justification puis affiche dans la console Python les lignes justifiées correspondantes.

   Remarque : `liste_mots[a:b]` est une tranche (ou *slice* en anglais). Elle désigne la liste extraite de la liste `liste_mots` constituée des éléments dont l'indice est compris entre `a` et `b-1` inclus.

```
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 ... in decoupage:
6         ligne_justifiee = \
7             ajout_espace(liste_mots[ ... : ... ], ...)
8         ...
```

Partie C

À l'usage, on se rend compte que la méthode précédente ne produit pas toujours un alignement justifié « esthétique ». Pour remédier à ce problème, les typographes utilisent une formule mathématique qui mesure la qualité esthétique d'une ligne en fonction du nombre d'espaces supplémentaires ajoutées. Entre deux mots d'une même ligne il y a forcément une espace. Sont donc considérées comme espaces supplémentaires celles que l'on doit ajouter en plus.

Dans cette partie, on utilise une fonction qui mesure le défaut d'esthétique, appelé coût inesthétique, que l'on cherche à minimiser.

- Le coût inesthétique d'une ligne est le carré du nombre d'espaces supplémentaires nécessaires à la justification de cette ligne.
- Le coût inesthétique d'un texte pour un découpage donné est la somme du coût inesthétique de chaque ligne.

L'objectif de cette partie est, pour un texte donné, de proposer un découpage minimisant ce coût.

7. Étant donné une justification de 15 caractères, la liste de mots `['An', 'algorithm', 'must', 'be', 'seen', 'to', 'be', 'believed']` et la liste de découpage `[(0, 2), (2, 4), (4, 7), (7, 8)]`, reproduire et compléter les trois dernières lignes du tableau ci-après (chaque ligne du tableau correspond à une ligne du texte découpé).

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				
4	7				
7	8				

8. Écrire le code d'une fonction `cout` qui prend en paramètres deux indices `i` et `j`, une liste de chaînes de caractères `liste_mots` et un entier `justification` représentant le nombre total de caractères souhaités par ligne. Elle renvoie le coût inesthétique de la ligne commençant au mot `liste_mots[i]` et finissant au mot `liste_mots[j-1]` si le nombre de caractères (espaces inter-mots comprises) de l'ensemble de ces mots est inférieur ou égal à `justification` et un million sinon.

   Exemples :

```
>>> liste_mots = ['An', 'algorithm', 'must', 'be', 'seen',
'to', 'be', 'believed']
>>> cout(0, 2, liste_mots, 15)
9
>>> cout(0, 4, liste_mots, 15)
1000000
>>> cout(5, 8, liste_mots, 15)
1
```

9. Étant donnée une liste de `n` mots (avec $n \geq 50$), en remarquant qu'on peut revenir à la ligne ou non après chaque mot sauf le dernier, indiquer s'il est raisonnable de déterminer une solution minimisant le coût inesthétique en testant toutes ces possibilités.

On pose la question à une IA générative, qui propose deux méthodes dont voici un extrait dans la capture d'écran donnée ci-après.

Algorithmes pour l'alignement justifié du texte

Voici deux approches courantes :

1. Heuristique gloutonne :

Les heuristiques sont des algorithmes qui ne garantissent pas de trouver la solution optimale, mais [...]

2. Programmation dynamique:

Dans le cas de l'alignement justifié, on peut décomposer le problème en sous-problèmes correspondant à l'alignement des phrases successives.

L'algorithme fonctionne en construisant une table qui stocke les coûts optimaux d'alignement pour chaque sous-problème. Pour chaque sous-problème, on considère toutes les façons possibles de découper la phrase correspondante et on calcule le coût d'alignement pour chaque découpage. Le coût optimal du sous-problème est ensuite défini comme le minimum des coûts calculés pour tous les découpages possibles.

La première méthode consiste en un algorithme glouton, la seconde utilise la programmation dynamique.

On demande alors à l'IA de générer un code en Python d'une fonction `justifie_dynamique` qui prend en paramètres une liste de chaînes de caractères `liste_mots` et un entier `justification` représentant le nombre total de caractères souhaités par ligne, en utilisant la programmation dynamique. L'IA propose alors le code ci-après.

```
1 def justifie_dynamique(liste_mots: list[str],
2                         justification: int)->list[(int,
3                                                    int)]:
```

```
4       """Renvoie une liste contenant un découpage de
5         'liste_mots' justifiée selon 'justification'."""
6       n = len(liste_mots)
7       # génération de deux listes de n éléments
8       cout_mini = [0] * n
9       indice_retour_ligne_mini = [0] * n
10      # parcours à rebours de la liste des mots
11      for i in range(n-1, -1, -1):
12          # initialisation du coût du découpage du ième
13          # au dernier mot
14          cout_mini[i] = cout(i, n, liste_mots,
15                              justification)
16          indice_mini = n
17          # recherche d'un indice j minimisant le coût
18          # total menant à la justification de  i à j
19          for j in range(i+1,n):
20              best = cout_mini[j]\
21                  + cout(i, j, liste_mots, justification)
22              if  best < cout_mini[i]:
23                  cout_mini[i] = best
24                  indice_mini = j
25          indice_retour_ligne_mini[i] = indice_mini
26      # reconstruction de la liste decoupage en partant
27      # du premier mot indice k=0
28      decoupage = []
29      k = 0
30      while k < n:
31          decoupage.append((k,
32                            indice_retour_ligne_mini[k]))
33          k = indice_retour_ligne_mini[k]
34      return decoupage
```

10. Donner l'ordre de grandeur du nombre d'appels à la fonction `cout` lorsqu'on exécute la fonction `justifie_dynamique` en fonction de `n` le nombre de mots dans la liste.

11. Établir à partir du code de la fonction `justifie_dynamique` la relation qui existe entre les éléments de la liste `cout_mini`.

12. Proposer une modification de la fonction `justifie_dynamique` pour qu'elle renvoie, en plus du découpage, le coût inesthétique de celui-ci.