Devoir Maison d'ITC N°1
pour la semaine du 06/11/2023
à rendre en séance de TP

Consignes

- La qualité, la lisibilité et l'efficacité du code entreront pour une part importante dans l'appréciation de la copie. L'indentation du code doit figurer clairement sur la copie, il est fortement conseillé d'y ajouter une barre verticale sur la gauche afin de préciser clairement les portions de code ayant la même indentation.
- **Le crayon à papier ne sera pas corrigé.**
- **Il est rappelé également que recopier une correction sur internet est complètement inutile pour tout le monde.**

Exercice 1 Un problème clé lié au volume d'informations numériques (sur le web par exemple) est l'efficacité de la recherche d'un mot dans une page ainsi que son nombre d'apparitions. Ces informations peuvent être obtenues simplement en parcourant le texte. Toutefois, cette approche simple conduit à des algorithmes lents vu la taille des textes considérés. La suite du problème propose des réalisations plus efficaces.

Introduisons d'abord quelques notions élémentaires :

- Un mot et un texte sont des chaînes de caractères sans majuscules, accent, espace et ponctuation.
- Le mot `mot` de taille m apparaît dans le texte `txt` de taille n, si et seulement si il existe un texte extrait de `txt` qui est égal à `mot`, caractère par caractère. Par exemple le mot `bons` apparaît dans le texte `desbonbons`, mais pas dans le texte `unbonbon`.
- Le *suffixe numéro k du texte `txt`* de taille n est le texte extrait `txt[k:n]`. Par exemple le suffixe numéro 2 du texte `unbonbon` est le texte extrait `bonbon`.
- On note que le mot `mot` apparaît dans le texte `txt` si et seulement si il existe un suffixe tel que `mot` apparaît en tête de ce suffixe (`mot` et `txt[k:k+m]` sont égaux, caractère par caractère pour un certain entier `k`). Par exemple le mot `bonb` apparaît en tête du suffixe 2 du texte `unbonbon` (`mot` et `txt[2:2+4]` sont égaux sur cet exemple).

Partie I — Quelques fonctions de bases

1. On considère la fonction `present(a, txt)` suivante :

```
def present(a: str, txt : str) -> bool :
    n = len(txt)
    i = 0
    while i < n and txt[i] != a:
        i = i+1
    return i < n
```

   1.1) On exécute la fonction `present('o', 'ecologie')`; quel est le contenu de la variable `i` en sortie de boucle? Que renvoie cette fonction?

   1.2) On exécute la fonction `present('f', 'ecologie')`; quel est le contenu de la variable `i` en sortie de boucle? Que renvoie cette fonction?

   1.3) À quoi correspond le résultat renvoyé par la fonction `present`?

2. Écrire une fonction `occurence(a: str, txt: str)->int` qui renvoie le nombre de fois où le caractère contenu dans la variable `a` apparaît dans la chaîne de caractères `txt` (et renverra 0 si `a` n'est pas présent dans `txt`).
3. Écrire une fonction `table(txt: str)->dict` qui renvoie un `dictionnaire` dont les clés sont les lettres présentes dans la chaîne de caractères `txt`, et les valeurs sont les occurrences de ce caractère. Par exemple, `table('ecologie')` devra retourner le dictionnaire `{'e' : 2, 'c': 1, 'o' : 2, 'l' : 1, 'g' : 1, 'i' : 1}`.
4. On considère la fonction :

```
def mystere(mot: str, txt: str)->bool:
    n = len(mot)
    if n <= len(txt) and txt[:n] == mot :
        return True
    else:
        return False
```

   À quelle condition cette fonction retourne `True`?

Partie II — Méthode directe de recherche

5. On considère la fonction :

```
def enTeteDeSuffixe(mot: str, txt: str, k: int) -> .... :
    n, m = len(txt), len(mot)
    if .... > .... :
        return False
    else:
        return mystere(...., ....)
```

Recopier et compléter la fonction `enTeteDeSuffixe(mot, txt, k)` pour qu'elle renvoie `True` si le mot `mot` apparaît en tête du suffixe numéro k du texte `txt`, et `False` sinon. On pourra supposer que k est un indice valide du texte `txt`, c'est-à-dire tel que `txt[k]` ne retourne pas d'erreur.

6. Écrire une fonction `rechercherMot(mot: str, txt: str)->bool` qui renvoie `True` si le mot `mot` apparaît dans le texte `txt`, et `False` sinon (on pourra utiliser la fonction précédente).

On souhaite dans cette dernière question obtenir plus précisément le nombre d'occurrences d'un mot dans un texte, avec recouvrement autorisé : on compte le nombre de répétitions du mot dans le texte, sans contrainte aucune. Par exemple, dans le texte `quelbonbonbon` (quel bon bonbon) le nombre d'occurrences de `bonbon` est 2, même si ces occurrences se recouvrent.

				b	o	n	b	o	n			
							b	o	n	b	o	n
q	u	e	l	b	o	n	b	o	n	b	o	n

7. Écrire une fonction `compterOccurrences(mot: str, txt: str) -> int` qui renvoie le nombre d'occurrences de `mot` dans le texte `txt`. La fonction devra faire appel à une ou plusieurs fonctions précédentes.

Correction
du Devoir Maison d'ITC N°1

Solution 1

Partie I — Quelques fonctions de bases

1.
```
def present(a: str, txt : str) -> bool :
    n = len(txt)
    i = 0
    while i < n and txt[i] != a:
        i = i+1
    return i < n
```

1.1) Lorsqu'on exécute la fonction `present('o', 'ecologie')`, la variable n a la valeur 8, et la variable a contient le caractère `"o"`. Tant que $i < 8$ et que le caractère d'indice i de la chaîne `txt` n'est pas la lettre `"o"`, on ajoute 1 à i, donc ici la boucle va s'arrêter lorsque $i = 2$. La fonction renvoie alors `i<n` qui est une expression booléenne et qui vaut **`True`** ici, car $i = 2$ et $n = 8$, donc la fonction renvoie **`True`**.

1.2) Lorsqu'on exécute la fonction `present('f', 'ecologie')`, la variable n a la valeur 8, et la variable a contient le caractère `"f"`. Tant que $i < 8$ et que le caractère d'indice i de la chaîne `txt` n'est pas la lettre `"f"`, on ajoute 1 à i, donc ici la boucle va s'arrêter lorsque $i = 8$ car le caractère `"f"` n'est pas dans la chaîne. La fonction renvoie alors `i<n` qui est une expression booléenne et qui vaut **`False`** ici, car $i = 8$ et $n = 8$, donc la fonction renvoie **`False`**.

1.3) Finalement, la fonction renvoie **`True`** si le caractère contenu dans l'argument a est présent dans la chaîne `txt`, et renvoie **`False`** sinon.

```
def occurence(a: str, txt: str)->int:
    """ renvoie le nombre d'occurrences du caractère contenu \
    ↪ dans l'argument a dans la chaîne txt """
    compteur = 0
    for c in txt :  # parcours par caractère
        if c == a:  # si on tombe sur le caractère contenu \
        ↪ dans a
            compteur += 1  # on ajoute 1 au compteur
    return compteur  # on renvoie la valeur du compteur
```

```
>>> occurence("b", "bonbon")
2
```

2.
```
def table(txt: str)->dict:
    """ renvoie un dictionnaire dont les clés sont les lettres \
    ↪ présentes dans txt,
        et les valeurs sont les occurrences de ce caractère. """
    resultat = {}  # dictionnaire vide
    for c in txt:  # parcours par caractère
        if c in resultat:  # ce caractère a déjà été rencontré
            resultat[c] += 1  # on ajoute 1 à la valeur
        else:  # première fois qu'on voit ce caractère
            resultat[c] = 1  # on crée une entrée dans le \
            ↪ dictionnaire
    return resultat  # on renvoie le dictionnaire créé
```

```
>>> table("ecologie")
{'e': 2, 'c': 1, 'o': 2, 'l': 1, 'g': 1, 'i': 1}
```

4. Pour que la fonction `mystere(mot: str, txt: str)->bool` renvoie la valeur **`True`**, il faut que la condition dans le test soit vérifiée, c'est à dire que la longueur n du `mot` ne dépasse pas celle de `txt`, et que les n premiers caractères de `txt` correspondent exactement aux caractères de `mot`. Autrement dit, la fonction renvoie **`True`** lorsque la chaîne `txt` commence par `mot`.

Partie II — Méthode directe

5. La fonction `enTeteDeSuffixe(mot, txt, k)` complétée, on utilise la fonction précédente :

```
def enTeteDeSuffixe(mot: str, txt: str, k: int) -> bool :
    """ indique si mot apparaît en tête du suffixe numéro k de \
    ↪ txt """
    n, m = len(txt), len(mot)  # longueur de la chaîne et \
    ↪ longueur du mot
    if k+m > n :  # si la longueur du mot plus k dépasse la \
    ↪ longueur de txt
        return False  # il n'est pas possible de trouver mot à \
        ↪ partir de l'indice k dans txt
    else:  # sinon on renvoie si le suffixe k commence par les \
    ↪ caractères de mot ou pas
        return mystere(mot, txt[k:n])
```

```
>>> enTeteDeSuffixe("bonb", "quelbonbonbon", 4)
```

```
True
>>> enTeteDeSuffixe("bonb", "quelbonbonbon", 7)
True
>>> enTeteDeSuffixe("bonb", "quelbonbonbon", 0)
False
>>> enTeteDeSuffixe("bonb", "quelbonbonbon", 10)
False
```

6. La fonction `rechercherMot(mot: str, txt: str)->bool` va chercher si parmi tous les suffixes de `txt`, il y en un qui commence par `mot` :

```
def rechercherMot(mot: str, txt: str)->bool:
    """ renvoie True si mot apparaît dans txt, et False sinon \
    ↪ """
    k = 0  # premier suffixe
    n = len(txt)  # longueur du texte
    while (k < n) and (not enTeteDeSuffixe(mot, txt, k)):
        # tant que k ne dépasse pas la longueur de txt et que \
        ↪ le suffixe k ne commence pas par mot
        k += 1  # on passe au suffixe suivant
    # en sortie de boucle : si le mot n'est pas présent alors \
    ↪ k va atteindre la valeur n,
    # et si le mot est présent, alors ce sera en tête d'un \
    ↪ certain suffixe k, avec k<n
    return (k < n)  # on renvoie donc False lorsque k=n, et \
    ↪ True lorsque k<n
>>> rechercherMot("bobn", "quelbonbonbon")
False
>>> rechercherMot("bonb", "quelbonbonbon")
True
```

7. La fonction `compterOccurrences(mot: str, txt: str) -> int` : cette fois il faut tester tous les suffixes de `txt` et compter ceux qui commencent par `mot`.

```
def compterOccurrences(mot: str, txt: str) -> int:
    """ renvoie le nombre d'occurrences de mot dans txt """
    compteur = 0
    n = len(txt)  # longueur de la chaîne
    for k in range(n):  # on teste tous les suffixes
        if enTeteDeSuffixe(mot,txt,k):  # si ce suffixe \
        ↪ commence par mot
            compteur += 1  # on ajoute 1 au compteur
    return compteur  # on renvoie la valeur de compteur
```

```
>>> compterOccurrences("bonbon", "quelbonbonbon")
2
```

Remarque : on pourrait se limiter dans la boucle à : `for k in range(n-len(mot)+1)` pour éviter de tester les suffixes dont on sait à l'avance qu'ils seront trop courts.