Recherche d'un mot dans un texte

Corrigé

MPSI - PCSI

1. Quelques fonction de bases

Q1) a) 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`.

b) 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`.

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

Q2) La fonction `occurence` complétée :

```python
def occurence(a: str, txt: str)->int:
    if txt == '':  # cas terminal, txt est la chaine vide
        return 0
    # autres cas
    else:  # on teste si a est le premier caractère de txt
        if a == txt[0]:  # une occurrence plus le nombre d'occurences dans la suite du texte
            return 1+occurence_rec(a, txt[1:])
        else:  # nombre d'occurences dans la suite du texte
            return occurence_rec(a, txt[1:])
```

Q3) Création du dictionnaire :

```python
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
```

Q4) Pour que la fonction `debut(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`.

2. Méthode directe

Q5) La fonction `enTeteDeSuffixe(mot, txt, k)` complétée, on utilise la fonction précédente :

```python
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 debut(mot, text[k:n])
```

Q6) 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
```

Q7) 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
```

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.

3. Tableau des suffixes

Q8) On exécute `inconnue([10,8,12,4,5])`, on note dans un tableau le contenu des différentes variables à la fin de chaque itération k de la boucle `for` :

k (n° itération)	i_k	elt_k	j_k	R_k
0	X	X	X	[]
1	0	10	0	[10]
2	1	8	0	[8,10]
3	2	12	2	[8,10,12]
4	3	4	0	[4,8,10,12]
5	4	5	1	[4,5,8,10,12]

Cette fonction renvoie une liste (`R`) qui correspond à la liste initiale (`L`) triée dans l'ordre croissant. Il faut remarquer qu'en sortie de la boucle `while`, la valeur de `j` est l'indice où il faut insérer l'élément `elt` (c'est à dire L[i]) dans la liste `R` pour conserver l'ordre croissant de `R`.

Q9) La fonction `triSuffixes` :

```
def triSuffixes(txt: str)->list:
    """ renvoie le tableau des suffixes triés dans l'ordre croissant """
    n = len(txt)
```

```
    # Ici la liste des suffixes est L = [0,1,2,...,n-1]
    R = []
    for i in range(n):
        elt = txt[i:n]  # texte du suffixe n°i
        j = i
        while (j > 0) and (elt < txt[ R[j-1] : n] ):
            # on compare elt avec le texte du suffixe R[j-1]
            j = j-1
        R = R[:j] + [i] + R[j:]  # on insère le suffixe i dans R (et non pas son texte)
    return R
```

Q10) a) On suppose le tableau trié des suffixes (`tabS`) déjà établi :

```
def rec_aux(mot: str, txt: str, tabS: list, d: int, f: int)->bool:
    """ renvoie True si mot est présent dans tabS entre les indices d et f (inclus) """
    if d > f:  # si la zone de recherche est vide
        return False
    else:  # la zone n'est pas vide, on précède par dichotomie
        milieu = (d+f)//2  # ce doit être un entier
        k = tabS[milieu]   # suffixe qui est au "milieu"
        if enteteDeSuffixe(mot, txt, k):  # si le suffixe k commence par mot
            return True  # on a trouvé le mot
        elif mot < txt[k: ]:  # il faut chercher le mot à gauche du milieu
            return rec_aux(mot, txt, tabs, d, milieu-1)
        else :  # il faut chercher le mot à droite du milieu
            return rec_aux(mot, txt, tabs, milieu+1, f)
```

b) La fonction `rechercherMot2` :

```
def rechercherMot2(mot: str, txt: str)->bool:
    """ renvoie True si le mot apparaît dans txt, et False sinon """
    n = len(txt)
    tabS = triSuffixes(txt)  # création du tableau trié des suffixes
    return rec_aux(mot, txt, tabs, 0, n-1)  # appel à la fonction de recherche dichotomique
    ↪ récursive
```

c) **Question bonus**. La fonction `rechercherMot3` (itérative) :

```
def rechercherMot3(mot: str, txt: str)->bool:
    """ renvoie True si le mot apparaît dans txt, et False sinon """
    n = len(txt)
    tabS = triSuffixes(txt)  # création du tableau trié des suffixes
    d, f = 0, n-1  # zone de recherche, indices de départ et de fin
    trouve = False
    while (not trouve) and (d <= f) :
        # tant qu'on a pas trouvé et que la zone de recherche n'est pas vide
        milieu = (d+f)//2
        k = tabS[milieu]   # suffixe qui est au "milieu"
        if enteteDeSuffixe(mot, txt, k):  # si le suffixe k commence par mot
            trouve = True  # on a trouvé le mot
        elif mot < txt[k: ]:  # il faut chercher le mot à gauche du milieu
            f = milieu-1  # l'indice de fin de zone change
        else :  # il faut chercher le mot à droite du milieu
            d = milieu+1  # l'indice de début de zone change
    return trouve
```