correction TD2 : Rechercher un mot dans un texte et compter ses occurrences

(d'après Polytechnique PSI-PT 2010 info)

Partie I :

1.
```
def enTeteDeSuffixe(mot,txt,k) :
    l = len(mot)
    return txt[k:k+l] == mot
```

2.
```
def rechercherMot(mot,txt) :
    trouve = False
    k = 0
    while not trouve and k <= len(txt)-len(mot) :
        if enTeteDeSuffixe(mot,txt,k) :
            trouve = True
        k += 1
    return trouve
```

3.
```
def compterOccurrences(mot,txt) :
    nb = 0
    for k in range(len(txt)-len(mot)+1) :
        if enTeteDeSuffixe(mot,txt,k) :
            nb += 1
    return nb
```

4.
```
def afficherFrequenceBigramme(txt) :
    dico = {}
    for k in range(len(txt)-1) :
        clef = txt[k:k+2]
        if clef in dico :
            dico[clef] += 1
        else :
            dico[clef] = 1
    return dico
```

Partie II

1.
```
def rechercherMot2(mot,txt,tabS) :
    if mot < txt[tabS[0]:] or mot > txt[tabS[-1]:] :
        return False
    g,d,m = 0,len(tabS),(g+d)//2
    k = tabS[m]
    trouve = False
    while d-g > 0 and not trouve :
        if enTeteDeSuffixe(mot,txt,k) :
            trouve = True
        elif mot < txt[k:] :
            d = m - 1
        else :
            g = m + 1
        m = (g+d)//2
        k = tabS[m]
    return  trouve
```

2. La fonction de la partie I fait a une complexité en $O(n)$ (on balaye le texte en entier une fois) alors que celle de la partie II a une complexité en $O(\log(n))$, elle est donc plus efficace que la première.

   Il faut par contre comprendre que cela est dû au fait que le tableau des suffixes est passé en argument dans cette fonction : la création du tableau des suffixes (qui est un tri) a une complexité en $O(n\log n)$ (au mieux) mais son coût est nettement supérieur à celui de la fonction de la partie I. Si on ne mettait pas le tableau des suffixes en argument de la fonction et si on commençait par faire calculer ce tableau à partir du texte avant la recherche, la complexité de la recherche dichotomique passerait en $O(n\log(n))$. L'idée est donc de créer le tableau des suffixes une bonne fois pour toute, le stocker en mémoire, puis on l'utilise pour toutes les recherches que l'on veut faire sur ce texte. Il faudrait donc faire évoluer le tableau des suffixes en même temps que le texte : si on modifie le texte,

il faut penser en même temps à actualiser le tableau des suffixes. Ceci est particulièrement intéressant si on fait plusieurs recherche sur le texte : on crée le tableau des suffixes à la première recherche (qui est donc assez longue) puis on l'utilise pour les recherches suivantes, qui sont donc plus rapides.

3.
```
def compterOccurrences2(mot,txt,tabS) :
    if mot < txt[tabS[0]:] or mot > txt[tabS[-1]:] :
        return 0
    # on commence par rechercher une appartition du mot
    g,d = 0,len(tabS)
    trouve = False
    while d-g > 0 and not trouve :
        m = (g+d)//2
        k = tabS[m]
        if enTeteDeSuffixe(mot,txt,k) :
            trouve = True
        elif mot < txt[k:] :
            d = m - 1
        else :
            g = m + 1
    # on cherche le nombre d'apparitions en élargissant les indices
    i = 0
    while m-i >= 0 and enTeteDeSuffixe(mot,txt,tabS[m-i]) :
        i += 1
    j = 0
    while m+j < len(tabS) and enTeteDeSuffixe(mot,txt,tabS[m+j]) :
        j += 1
    if trouve :
        nb = i+j-1
    else :
        nb = 0
    return nb
```

4.
```
def triSuffixes(txt) :
    L = list(range(len(txt))) # la liste des suffixes que l'on va réordonner
    for i in range(1,len(L)) :
        rang_ins = i
        elt = L[i]
        while rang_ins > 0 and  txt[L[rang_ins-1]:] > txt[elt:] :
            L[rang_ins] = L[rang_ins-1]
            rang_ins -=1
        L[rang_ins] = elt
    return L
```