Épreuve d'informatique de l'X - 2010 - PT/PSI

Rechercher un mot dans un texte et compter ses occurrences

Partie I. Méthode directe

Question 1. On récupère les tailles du mot et du texte (ligne 2 et 3). Si le suffixe du texte est plus court que le mot, on retourne `faux` (ligne 4). Dans la boucle des lignes 6 à 8, on parcourt simultanément le mot et le suffixe. Si on rencontre deux lettres distinctes, on retourne `faux`. Si la boucle s'exécute complement, c'est que le mot est en tête du suffixe étudié, on retourne alors `vrai` (ligne 9).

```
1 def enTeteDeSuffixe(mot,tab,k):
2     m = len(mot)
3     n = len(tab)
4     if k+m > n:
5         return(False)
6     for i in range( 0, m):
7         if tab[k+i] != mot[i]:
8             return(False)
9     return(True)
```

Question 2. : On regarde si le mot est en tête de l'un des suffixes du texte en étudiant tous les suffixes du texte. Si c'est la cas, on arrête le programme en retournant la valeur `vrai` (ligne 6). Si la boucle s'exécute complément, c'est que le mot n'est pas dans le texte, on retourne alors `faux` (ligne 7).

```
1 def rechercherMot(mot,tab):
2     m = len(mot)
3     n = len(tab)
4     for k in range(n):
5         if enTeteDeSuffixe(mot,tab,k):
6             return(True)
7     return(False)
```

Question 3 : On commence par récupérer les tailles du mot et du texte (lignes 2 et 3). Puis, pour compter les occurrences d'un mot dans un texte, on compte le nombre de fois que le mot est en tête d'un suffixe du texte en étudiant tous les suffixes du texte (boucle des lignes 5 à 7). On retourne le résultat à la ligne 8.

```
1 def compterOccurences(mot,tab):
2     m = len(mot)
3     n = len(tab)
4     compt = 0
5     for k in range( n):
6         if enTeteDeSuffixe(mot,tab,k):
7             compt += 1
8     return(compt)
```

Question 4 : Pour compter les nombres d'apparitions de chacune des lettres de l'alphabet, on commence par créer et initialiser un tableau `freq` de 26 cases (ligne 2). Ensuite, on parcourt le texte et, pour chaque lettre rencontrée, on incrémente le compteur associé dans le tableau `freq` (boucles des lignes 3 à 6). À la sortie de la boucle, on retourne le tableau des fréquences (ligne 7).

```
1 def frequenceLettre(tab) :
2     freq = [0 for i in range(26)]
3     n = len(tab)
4     for i in range(n) :
5         lettre = ord(tab[i]) - ord('a')
6         freq[lettre] += 1
7     return(res)
```

Question 5 : Pour afficher les fréquences des bigrammes d'un texte, on étudie chacun des bigrammes du texte (ce sont les séquences de deux lettres consécutives du texte). Il y a au plus $N = 26 \times 26 = 676$. bigrammes possibles. On stocke les fréquences d'apparitions de chaque bigramme dans le tableau «`freq`» dans le seul but d'éviter d'afficher plusieurs fois la fréquence d'apparition d'un même bigramme et d'éviter de recalculer la fréquence d'apparition d'un bigramme déjà étudié. La boucle des lignes 5 à 12 permet d'étudier tous les bigrammes possibles du texte. Chaque bigramme correspond à une seule des cases du tableau `freq`. L'indice de la case est calculé à la ligne 8. Si la fréquence d'apparition n'est pas nulle, c'est que le bigramme a déjà été étudié donc on ne fait rien, sinon on calcule et on affiche la fréquence d'apparition du bigramme (lignes 11 à 13).

```
1  def afficherFrequenceBigramme(tab) :
2      N = 26*26;
3      freq = [0 for i in range(N)]
4      n = len(tab);
5      for i in range(n-2):
6          a = tab[i]
7          b = tab[i+1]
8          c = (ord(a)-ord('a'))+26*(ord(b)-ord('b'))
9          bigramme = tab[i:i+2]
10         if freq[c] == 0 :
11             freq[c] = compterOccurences(bigramme,tab)
12             print("%s est un bigramme de fréquence %d \n"\
13                                     % (bigramme,freq[c]))
```

Partie II. Tableau des suffixes

Question 6 : Pour comparer deux suffixes, on parcourt simultanément les deux suffixes. Dès qu'ils ont des lettres différentes, on retourne le résultat de la différence entre ces deux lettres (ligne 5). Si on a fini de parcourir le suffixe le plus court, sans quitter la boucle, c'est que l'un des suffixes est en tête de l'autre. On retourne alors la différence des longueurs des deux suffixes (ligne 6).

```
1  def comparerSuffixes(tab, k1, k2):
2      n = len(tab)
3      for i in range(n-max(k1,k2)) :
4          if tab[k2+i] <> tab[k1+i]:
5              return(ord(tab[k2+i-1]) - ord(tab[k1+i-1]))
6      return(k1-k2)
```

Question 7 : Pour calculer le tableau des suffixes, on commence par créer le tableau `tabS` $= [0, 1, 2, 3, \ldots, n-1]$. (où n est la taille du texte) (ligne 6). On effectue ensuite un tri par bulles sur le tableau `tabS` en utilisant comme relation d'ordre sur l'ensemble $\{1, \ldots, n\}$, la relation d'ordre induite par la fonction «`comparerSuffixes`» (double boucle des lignes 7 à 10).

```
1  def permute(tab, i, j):
2      tab[i],tab[j] = tab[j],tab[i]
3
4  def calculerSuffixes(tab):
5      n = len(tab)
6      tabS = [i for i in range(n)]
7      for i in range(n-1):
8          for j in range(n-1-i) :
9              if comparerSuffixes(tab,tabS[j],tabS[j+1]) > 0:
10                 permute(tabS,j,j+1)
11     return(tabS)
```

Partie III. Exploitation du tableau des suffixes

Question 8 : Pour comparer un mot et un suffixe, on adapte la fonction «`comparerSuffixes`». On parcourt simultanément le mot et le suffixe. Dès qu'ils ont des lettres différentes, on retourne la différence entre ces deux lettres (ligne 7). Si on sort de la boucle, on arrive à la ligne 8 et alors

- soit le mot est en tête du suffixe et alors on retourne 0,
- sinon le suffixe est strictement en tête du mot, et alors on retourne 1 (ligne 11).

```
1  def comparerMotSuffixe(mot, tab, k):
2      n = len(tab)
3      m = len(mot)
4      for i in range(min(m,n-k)) :
5          if mot[i]<> tab[k+i]:
6              return(ord(mot[i]) - ord(tab[k+i]))
7      if m <= n+1-k :
8          return(0)
9      else :
10         return(1)
```

Question 9 : Pour recherche un mot dans un texte, on procède par recherche dichotomique du mot dans le tableau des suffixes du texte en appliquant la méthode proposée par l'énoncé. Les variables a et b représentent respectivement le plus petit et le plus grand des indices du tas dans lequel on effectue la recherche. Initialement, on effectue la recherche sur tout le tableau, ce qui explique les initialisations de a et b aux lignes 2 et 3. On effectue la boucle des lignes 4 à 12 tant que $b - a \geqslant 0$. Dans cette boucle, on calcule l'indice m de l'élément médian du sous-tas dans lequel on effectue notre recherche. On compare le mot avec cet élément médian. S'il y a égalité, on arrête le programme et on retourne la valeur `vrai` (ligne 8). Sinon on restreint le domaine de recherche suivant la position de l'élément médian par rapport au mot cherché. Si on sort de la boucle sans avoir quitté le programme, c'est que le mot n'est pas présent dans le texte, on retourne alors la valeur `faux` (ligne 13).

```
1  def rechercherMot2(mot, tab, tabS):
2      a = 0
3      b = len(tabS)-1
4      while b-a >= 0 :
5          m = (a+b)/2
6          comp = comparerMotSuffixe(mot,tab,tabS[m])
7          if comp == 0 :
8              return(True)
9          elif comp > 0 :
10             a = m+1
11         else :
12             b = m-1
13     return(False)
```

Question 10 : Pour la fonction «rechercherMot», l'ordre de grandeur des comparaisons est en O(n.m). Pour la fonction «rechercheMot2» est en O(log(n).m).
L'intérêt du tableau des suffixes est de réduire drastiquement le temps de recherche dans le cas d'un moteur de recherche internet à la partie «texte» est très grande en comparaison du ou des mots recherchés.

Question 11 : Pour rechercher le premier suffixe contenant un mot donné, on adapte le principe de la recherche dichotomique mise en œuvre dans la fonction «`rechercherMot2`». Les variables a et b représentent respectivement le plus petit et le plus grand des indices du tas dans lequel on effectue la recherche. À chaque étape, on divise par 2 la taille du tas dans lequel on effectue la recherche. Contrairement à la fonction «`rechercherMot2`», on ne s'arrête pas dès que le mot est un préfixe du suffixe désigné par l'élément médian m. Les instructions conditionnelles des lignes 7 à 12 gèrent les différents cas de figure. Si `mot` est en tête du suffixe d'indice `tabS[`m`]`, alors c'est que l'indice cherché est dans l'intervalle $[\![a, m]\!]$ et b reçoit m (ligne 8). Si `mot` est après le suffixe d'indice `tabS[`m`]`, alors c'est que l'indice cherché est dans l'intervalle $[\![m+1, b]\!]$ et a reçoit $m+1$ (ligne 10). Sinon, c'est que l'indice cherché est dans l'intervalle $[\![a, m-1]\!]$ et b reçoit $m-1$ (ligne 12). On quitte la boucle conditionnelle des lignes 4 à 12 quand le tas dans lequel on effectue la recherche est réduit à un élément. À la sortie de la boucle, on teste pour savoir si le mot est un préfixe du suffixe correspondant. Si ce n'est pas le cas on retourne -1 (ligne 16), sinon

on retourne b qui est, en sortie de boucle le plus petit indice i de `tabS` tel que `mot` apparaît en tête du suffixe numéro `tabS[`i`]` du texte `tab`.

```
1  def rechercherPremierSuffixe(mot, tab, tabS):
2      a = 1
3      b = len(tabS)
4      while b-a > 0 :
5          m = (a+b)/2
6          comp = comparerMotSuffixe(mot,tab,tabS[m])
7          if comp == 0 :
8              b = m
9          elif comp > 0 :
10             a = m+1
11         else :
12             b = m-1
13     if comparerMotSuffixe(mot,tab,tabS[b]) == 0 :
14         return(b)
15     else :
16         return(-1)
```

Question 12 : Pour compter le nombre des occurrences d'un mot dans un texte, on détermine les indices m et M des premier et dernier suffixes du tableau des suffixes du texte `tabS` contenant le mot (ligne 3 et 4). Si le mot est présent dans le texte, on retourne $(M-m+1)$ qui correspond au nombre d'apparition du mot dans le texte (ligne 5). Si ce n'est pas le cas, on retourne 0.

```
1  compterOccurences2:=proc(mot::array,tab::array,tabS::array)
2  def compterOccurences2(mot, tab, tabS) :
3      m = rechercherPremierSuffixe(mot,tab,tabS)
4      M = rechercherDernierSuffixe(mot,tab,tabS)
5      if m != -1 :
6          return(M-m+1)
7      else :
8          return(0)
```

Question 13 : Pour afficher les K-grammes et leurs fréquences d'apparition, on étudie tous les K-grammes possibles (boucles des lignes 5 à 16). Dans la double des lignes 5 à 16, on commence par créer le $i^{ème}$ K-gramme (boucle des lignes 3 à 8). Pour éviter d'étudier plusieurs fois le même K-gramme, on vérifie que l'indice du premier suffixe contenant le K-gramme dans le tableau des suffixes `tabS` coïncide avec i (ligne 5). Si c'est le cas, on affiche le K-gramme et le nombres d'occurrence du K-gramme dans le texte.

```
1  def afficherFrequenceKgramme(tab, tabS, k):
2      n = len(tab)
3      for i in range(n-k+1) :
4          mot = tab[i::i+k]
5          if tabS[rechercherPremierSuffixe(mot,tab,tabS)] == i :
6              afficherMot(tab,i,k);
7              print("%s est de fréquence %d\n" % \
8                                  (mot,compterOccurrences2(mot,tab,tabS)))
```