Informatique

2
Dictionnaires et programmation dynamique

TD2-6

Plus longue sous-suite commune

Présentation

Soient deux chaines de caractères m1 et m2. Une sous-suite commune de m1 et m2 est une chaine de caractères telle qu'on la retrouve parmi les lettres de m1 et m2, les 3 parcourues dans le même ordre. Par exemple :

m1= « abdef » ; m2= « agbcjkf »

Voici alors des sous-suites de m1 et m2 :

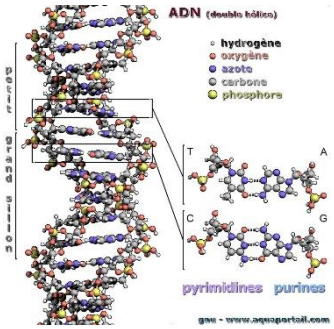
« a » ; « ab » ; « bf » ; « abf »

On souhaite ici déterminer la plus longue sous-suite commune m entre m1 et m2, qui dans l'exemple ci-dessus est : « abf »

Applications

On trouve différentes applications à cette recherche, citons-en quelques-unes :

- Biologie : l'ADN est décrit par une suite de lettres parmi les 4 lettres ATCG. Il arrive qu'il y ait des insertions dans l'ADN. On peut comparer deux informations génétiques et identifier qu'un ADN source a été complété par des lettres via des modifications génétiques, par exemple, AGTCGTCGATCGTAA devenant AGATCGTCGAGTCGTAA. L'algorithme identifiera dans le second la sous-suite du premier.



- Similitudes entre fichier informatiques : pour ne citer qu'une application, on peut mettre en place des algorithmes d'anti-plagia en identifiant une longue phrase commune entre deux fichiers.
- Correction orthographique : Si je tape par erreur « infomatique » au lieu de « informatique », l'algorithme peux identifier automatiquement que le mot attendu était « informatique » car ces deux mots ont la même plus longue sous-suite commune « infomatique »

Algorithme par force brute

Dans un premier temps, on souhaite énumérer l'intégralité des sous suites de deux chaines de caractères pour ensuite trouver la plus longue sous suite commune.

On donne la fonction suivante :

```
def sous_suites_k(m,k):
    n = len(m)
    if n < k or k == 0:
        return #####
    elif k == 1:
        return #####
    else:
        res = []
        L_Mkm1 = sous_suites_k(m,k-1)
        for #####:
            for #####:
                #####
                #####
        return res
```

Question 1: Recopier et compléter la fonction récursive sous_suites_k(m,k) prenant en arguments la chaine de caractères m et l'entier k, et renvoyant une liste contenant les sous listes Mk de k termes de m du type [Mk,ind] où ind est l'indice python de la dernière lettre de m ajoutée dans Mk, 0 si Mk est vide

Vérifier :

```
>>> sous_suites_k('abcde',3)
[['abc', 2], ['abd', 3], ['abe', 4], ['acd', 3], ['ace', 4],
['ade', 4], ['bcd', 3], ['bce', 4], ['bde', 4], ['cde', 4]]
>>> sous_suites_k('abcde',0)
[['', 0]]
```

Question 2: Créer la fonction extraction(L) prenant en argument une liste L composée de listes de listes [Mk,ind] et renvoyant la liste des listes Mk

Question 3: Créer la fonction sous_suites(m) prenant en argument une chaine de caractères m et renvoyant toutes les sous-suites de m

Vérifier :

```
>>> sous_suites('itc')
['', 'i', 't', 'c', 'it', 'ic', 'tc', 'itc']
```

Question 4: Exprimer le nombre de sous-suites d'une chaine composée de n caractères

Question 5: Créer la fonction communes(m1,m2) renvoyant toutes les sous-suites communes des chaines de caractères m1 et m2

Question 6: Créer le code permettant d'afficher la plus longue sous-suite commune entre « lbzhrgmauvto » et « atbcrqkjarvmo »

Question 7: Estimer la complexité de cette procédure

Présentation de la méthode en programmation dynamique

Il est possible d'obtenir la solution précédente en programmation dynamique avec une complexité temporelle en produit des longueurs de chaque chaine de caractères en traitant le sous problème suivant :

Quelle est la longueur de la PLSC entre X_i et Y_i, séquences X et Y contenant leurs i premiers termes

On créer un tableau de dimensions $(n+1)(m+1)$ dans lequel chaque case $D(i,j)$ contient les longueurs des PLSC de X_i et Y_i :

$$\forall \boldsymbol{i} \in [\mathbf{0}, \boldsymbol{n}], \forall \boldsymbol{j} \in [\mathbf{0}, \boldsymbol{m}], \boldsymbol{D(i,j)} = \begin{cases} \mathbf{0}\ \boldsymbol{si}\ \boldsymbol{i} = \mathbf{0}\ \boldsymbol{ou}\ \boldsymbol{j} = \mathbf{0} \\ \boldsymbol{D(i-1, j-1) + 1\ si\ x_i = y_j} \\ \boldsymbol{max}\begin{pmatrix} \boldsymbol{D(i-1,j)} \\ \boldsymbol{D(i,j-1)} \end{pmatrix}\ \boldsymbol{si\ x_i \neq y_j} \end{cases}$$

En effet : Soient $X_n = [x_1, \dots, x_n]$ et $Y_m = [y_1, \dots, y_m]$ deux séquences et $Z_k = [z_1, \dots, z_k]$ une plus longue sous-séquence commune (PLSC) quelconque de X et Y. Appelons X_i, Y_i et Z_i les séquences X, Y et Z contenant leurs i premiers termes. On a :

- Si $x_n = y_m$: $z_k = x_n = y_m$ et Z_{k-1} est une PLSC de X_{n-1} et Y_{m-1}
- Sinon :
  - Si $z_k \neq x_n$: Z_k est une PLSC de X_{n-1} et Y_m
  - Si $z_k \neq y_m$: Z_k est une PLSC de X_m et Y_{m-1}

La dernière case $D(n,m)$ donne la longueur de la PLSC de X et Y

Exemple : $X_4 = [1,2,3,5]$, $Y_4 = [1,2,4,5]$ et $Z_3 = [1,2,5]$

- $(x_3 = 5) = (y_4 = 5)$: $Z_2 = [1,2]$ est une PLSC de $X_3 = [1,2,3]$ et $Y_3 = [1,2,4]$
- $(x_3 = 3) \neq (y_3 = 4)$:
  - $(z_2 = 2) \neq (x_3 = 3)$: Z_2 est une PLSC de $X_2 = [1,2]$ et $Y_3 = [1,2,4]$
- $(x_2 = 2) \neq (y_3 = 4)$:
  - $(z_2 = 2) = (x_2 = 2)$: RAS
  - $(z_2 = 2) \neq (y_3 = 4)$: $Z_2 = [1,2]$ est une PLSC de $X_2 = [1,2]$ et $Y_2 = [1,2]$
- $(x_2 = 2) = (y_2 = 2)$: $Z_1 = [1]$ est une PLSC de $X_1 = [1]$ et $Y_1 = [1]$
- $(x_1 = 1) = (y_1 = 1)$: $Z_0 = [\,]$ est une PLSC de $X_0 = [\,]$ et $Y_0 = [\,]$
  On remarque que tout ce qui est écrit est vrai.

Exemple pour comprendre le « max » de l'algorithme :

- Comparer 'abc**d**' et 'ab**c**' : il faut supprimer **d**, pas **c**. En supprimant **c**, la PLSC aura une longueur de 2 alors qu'en supprimant **d**, elle sera de 3, c'est ce qu'il faut prendre.
- Comparer 'abcd' et 'abce' : il faut supprimer d et e, ce qui sera fait en 2 étapes, le résultat reste de 3 que d ou e soient supprimés.

Soit le tableau suivant :

	j	0	1	2	3	4	5
i			C	H	I	E	N
0		0	0	0	0	0	0
1	N	0					
2	I	0					
3	C	0					
4	H	0					
5	E	0					

Question 8: Compléter le tableau en utilisant l'algorithme décrit

Remarquez que chaque case correspond à la longueur de la PLSC entre les mots jusqu'à cette case, par exemple : Entre NICH et CH, c'est 2 (CH).

Programmation de l'algorithme itératif

Question 9: Mettre en place une fonction PLSC_it(m1,m2) prenant en arguments deux chaines de caractères m1 et m2 et renvoyant le dictionnaire D attenu par construction ascendante
Question 10: Préciser la complexité de la fonction PLSC_it
Question 11: Créer une fonction matrice(f,m1,m2) renvoyant un array représentant le tableau associé au dictionnaire créé par f(m1,m2) et vérifier votre tableau. On définira les valeurs non évaluées par l'algorithme à l'infini (np.inf)

Programmation de l'algorithme récursif

Question 12: Proposer une première fonction récursive PLSC_rec_ij(i,j,m1,m2) renvoyant la longueur de la PLSC de m1 et m2
Question 13: Préciser la complexité de la fonction PLSC_rec_ij lorsque m1 et m2 sont de même longueur n et n'ont aucun caractère en commun
Question 14: Proposer une fonction PLSC_rec_mem(m1,m2) récursive mémoïsée renvoyant le même dictionnaire que PLSC_it

Remarque : vous pourrez vérifier que vous obtenez la même matrice qu'avec PLSC_it, à un détail près à comprendre

Etapes intermédiaires

Que ce soit par l'algorithme récursif ou itératif, on obtient le même dictionnaire. On souhaite remonter les étapes permettant d'établir la plus longue séquence commune entre deux chaines de caractères.

On reprend le tableau précédent :

	j	0	1	2	3	4	5
i			C	H	I	E	N
0		0	0	0	0	0	0
1	N	0	0	0	0	0	1
2	I	0	0	0	1	1	1
3	C	0	1	1	1	1	1
4	H	0	1	2	2	2	2
5	E	0	1	2	2	3	3

On peut retracer un chemin faisant apparaître la PLSC entre « niche » et « chien » en se basant sur l'algorithme précédent car à chaque case, on sait déterminer la case de provenance. En effet, depuis la case $D(i,j)$:

- Si $x_i = y_j$:
  - Déplacement en diagonale $(i-1, j-1)$
  - La lettre $x_i = y_j$ est z_k à ajouter dans PLSC
- Sinon, $x_i \neq y_j$:
  - Si $D(i,j-1) = D(i-1,j)$: On peut indépendamment aller à $(i-1,j)$ ou $(i,j-1)$
  - Si $D(i,j-1) < D(i-1,j)$: Déplacement horizontal $(i-1,j)$
  - Si $D(i,j-1) > D(i-1,j)$: Déplacement vertical $(i,j-1)$

  Fin quand on arrive dans une case violette $i * j = 0$

On a tracé un parcours dans le tableau ci-dessus.

Question 15: Créer la fonction PLSC(m1,m2) créant le dictionnaire et proposant la PLSC de m1 et m2 en remontant la solution comme proposé ci-dessus

```
>>> PLSC('niche','chien')
'che'
```

Question 16: Créer une fonction Tirets(m1,m2,PLSC) prenant en argument les mots m1, m2, et leur PLSC, et renvoyant deux chaines de caractères contenant à leurs places dans m1 et m2 les lettres communes, des tirets sinon

```
>>> Tirets('niche','chien','che')
('--che', 'ch-e')
```