tp_l8dn003_20_04_q2

March 28, 2020

TP n°4: distance de Levenshtein [Corrigé question 2]

La distance de Levenshtein, ou distance d'édition, est une façon de mesurer à quel point deux mots se ressemblent. L'idée de base est d'attribuer un poids à chacune des opérations qui peuvent permettre de passer d'un mot à un autre (suppression ou insertion d'une lettre, interversion de deux lettres, remplacement d'une lettre par une autre…). La distance est calculée à partir de ce nombre d'opérations (éventuellement pondérées). Cette notion est très utilisée pour faire de la correction orthographique, pour gérer les tokens inconnus dans de nombreuses applications de TAL, pour faire de la morphologie computationnelle, *etc*. L'article Wikipedia https://fr.wikipedia.org/wiki/Distance_de_Levenshtein est une bonne source d'information. On y donne même une version de l'algorithme, mais je propose plutôt une approche progressive pour mieux appréhender les aspects algorithmiques du problème: on va implémenter différentes fonctions de mesure de distance qui en faisant différentes hypothèses simplificatrices.

Autre approche : prise en compte de séquences absentes

Ecrire une fonction qui va s'appliquer à deux chaînes de caractères telles que la seconde est obtenue à partie de la première en ôtant une (seule) portion *contiguë*. La fonction `coupe(u,v)` doit renvoyer la taille de la coupe qui a été pratiquée dans u pour obtenir v. Formellement: $\exists a, b, w$ t.q. $u = awb$ & $v = ab$, où a, b et w sont des chaînes de caractères non vides. La fonction doit renvoyer -1 si la condition n'est pas vérifiée (0 signifie que u==v).

Indice algorithmique: il faut chercher d'une part le préfixe commun, d'autre part le suffixe commun: si ces deux sous-chaînes sont non vides, il ne reste plus qu'à compter la taille de w.

Pour faciliter la programmation, on va définir deux fonctions qui vont s'occuper de déterminer la longueur du préfixe et du suffixe commun entre les deux chaînes. L'hypothèse est que si on regarde le mot "dest" (le mot résultant de la coupe) on trouve dans ce mot un indice k tel que `dest[:k]` est un préfixe de "orig" (le mot avant la coupe) et tel que `dest[k:]` est un suffixe de orig.

Autrement dit: `orig = dest[:k] + w + dest[k:]`

Remarque: Dans la définition donnée dans l'énoncé, on a exclu les cas où la coupe se fait au début du mot (alors $k = 0$) et ceux où la coupe se fait à la fin: $k = len(dest)$.

Attention à des cas particuliers rares mais qui doivent être pris en compte: supposons que l'on fasse une coupe de `ni` dans le mot `nininini`. Cela donne `ninini` quel que soit l'endroit où l'on fait la coupe. Autrement dit, dans un tel cas, on ne peut pas savoir s'il faut analyser la coupe comme `nini /ni/ ni` ou comme `ninini /ni/`. Les cas de ce genre sont caractérisés par le fait que la fin du préfixe commun au mot initial et au mot découpé ne coïncide pas avec le début du suffixe commun.

[1]:
```
# g (gauche) = indice du premier caractère différent entre u et v
#            = taille du préfixe commun
# d (droite) = indice du premier caractère différent en partant de la fin
# --> len(v) - (d+1) = taille du suffixe commun
def taille_prefixe(u,v):
    N = len(v)
    g = 0
    while g < N and u[g] == v[g]:
        g += 1
    return g

def taille_suffixe(u,v):
    N = len(v)
    delta = len(u) - len(v)
    d = N - 1
    while d >= 0 and v[d] == u[d+delta]:
        d -= 1
    return N-(d+1)
```

[2]:
```
# Jeu de données pour tester les deux fonctions
# (le 2e mot ne doit pas être plus long que le premier)
paires = [('valapiche', 'valache'), ('toto', 'toto'), ('saxe', 'vax'), ('nininini',␣
 ↪'ninini'),
          ('apolice', 'police'), ('pouliche', 'poule'), ('poule','pou'), ('dans',␣
 ↪'des')]

for (x,y) in paires:
    tp, ts = taille_prefixe(x,y), taille_suffixe(x,y)
    print("%-10s /%6s/ %-10s /%6s/ (%d,%d)"%
          (x,y[:tp],y,y[-ts:] if ts != 0 else '',tp,ts))
```

```
valapiche  /  vala/ valache    /   che/ (4,3)
toto       /  toto/ toto       /  toto/ (4,4)
saxe       /      / vax        /      / (0,0)
nininini   /ninini/ ninini     /ninini/ (6,6)
apolice    /      / police     /police/ (0,6)
pouliche   /  poul/ poule      /     e/ (4,1)
poule      /   pou/ pou        /      / (3,0)
dans       /     d/ des        /     s/ (1,1)
```

Une fois qu'on dispose de la taille des préfixes et suffixes communs (soient p et s), on peut répondre à la question du TP. On ne considère que les cas où le préfixe et le suffixe sont tous deux non nuls : $p > 0$ et $s > 0$. Dans ce cas il faut distinguer deux situations:

1. le préfixe et le suffixe ne se chevauchent pas: $p + s \leq len(v)$. Alors:
   (a) Soit $p + s < len(v)$. Dans ce cas, on ne peut pas dire que v soit le résultat d'une coupe dans u: il y a des caractères dans v qui ne viennent pas de u (on a $u = awb$ et $v = aw'b$ avec $w' \neq \varepsilon$). Il faut renvoyer -1.
   (b) Soit $p + s = len(v)$, et on est exactement dans le cas où $u = awb$ et $v = ab$ avec a, b et w non nuls. La taille de la coupe est la différence de longueur entre u et v.
2. le préfixe et le suffixe se chevauchent : $p + s \geq len(v)$. Alors:
   (a) Soit les deux mots u et v sont identiques (ainsi, par conséquent, que leurs préfixes et suffixes communs). Dans ce cas, la longueur de la coupe est 0 (ce qui correspond à la différence de longueurs).
   (b) Soit les deux mots sont différents. Cela signifie que le préfixe commun a (au moins) un suffixe identique à un préfixe du suffixe commun [**Relisez cette phrase tranquillement en prenant des exemples...**]. Par exemple `u = "nininini"` et `v = "ninini"`, ou `u = "abcdbcde"` et `v = "abcde"`. Dans ce cas on ne peut pas donner la position de la coupe, mais on peut donner la taille: c'est $len(u) - len(v)$.

[3]:
```
def coupe(u,v):
    p = taille_prefixe(u,v)
    s = taille_suffixe(u,v)
    if p > 0 and s > 0 and p+s >= len(v):
        return len(u) - len(v)
    else:
        return -1
```

[4]:
```
for (x,y) in paires:
    c = coupe(x,y)
    if c > 0:
        print("%s a été coupé pour donner %s (taille de la coupe: %d)" % (x,y,c))
```

```
valapiche a été coupé pour donner valache (taille de la coupe: 2)
nininini a été coupé pour donner ninini (taille de la coupe: 2)
pouliche a été coupé pour donner poule (taille de la coupe: 3)
```

[5]:
```
# Test avec des mots piochés dans un texte
def charge_fichier(nf):
    lgns = []
    with open(nf, "r", encoding="utf8") as f:
        for ligne in f:
            lgns.append(ligne.strip())
    return lgns
def charge_lexique(nf):
    liste_lignes = charge_fichier(nf)
    lexique = []
    for l in liste_lignes:
        for w in l.split():
            if w not in lexique:
                lexique.append(w)
    return lexique
```

```
formes = charge_lexique("pensees_pascal_utf8.txt")
formes = formes[200:500]
for i in range(len(formes)):
    for j in range(i,len(formes)):
        x,y = formes[i], formes[j]
        if len(x) < len(y):
            x,y = y,x
            c = coupe(x,y)
            if c > 0:
                print("%-15s : %-10s (%d)" % (x,y,c))
```

```
aperçu          : au         (4)
copies          : ces        (3)
choses          : ces        (3)
couvertes       : ces        (6)
perdu           : peu        (2)
combattraient   : combattent (3)
qu'elle         : quelle     (1)
mais            : mis        (1)
criassent       : crient     (3)
croient         : crient     (1)
faudrait        : fait       (4)
```