Comparaison de similarité entre 2 chaînes de caractères

Vincent FORMAN

Le 21 août 2006

1 Introduction et notations

Soit $\mathcal{S}$ un alphabet (par exemple, $\mathcal{S} = \{0, \cdots, 255\}$ dans le cas des caractères ASCII, $\mathcal{S} = \{a, \cdots, z\}$ dans le cas de l'alphabet courant). $\mathcal{S}^*$ est l'ensemble des mots qu'il est possible de former sur cet alphabet, c'est à dire toutes les suites finies d'éléments de $\mathcal{S}$. Par convention, on notera ϵ le mot vide, et $|m|$ la longueur d'un mot de $\mathcal{S}^*$. Par exemple, $|\epsilon| = 0$. La concaténation des 2 mots p et q s'écrira pq.

2 Définition de la similarité

Soient 2 mots p et q du langage $\mathcal{S}^*$, et α et β 2 lettres de l'alphabet $\mathcal{S}$. On définit les 2 applications λ et μ :

$$\lambda : \mathcal{S}^* \times \mathcal{S}^* \longrightarrow \mathbb{Z}$$

$$\begin{cases} \lambda(\epsilon, q) = 0 \\ \lambda(p, \epsilon) = 0 \\ \lambda(p\alpha, q\beta) = \max \begin{pmatrix} \lambda(p\alpha, q) \\ \lambda(p, q\beta) \\ \begin{cases} 2 + \mu(p, q) & \text{si } \alpha = \beta \\ \lambda(p, q) & \text{si } \alpha \neq \beta \end{cases} \end{pmatrix} \end{cases}$$

$$\mu : \mathcal{S}^* \times \mathcal{S}^* \longrightarrow \mathbb{Z}$$

$$\begin{cases} \mu(\epsilon, \epsilon) = 0 \\ \mu(\epsilon, q) = -1 \qquad \text{si } q \neq \epsilon \\ \mu(p, \epsilon) = -1 \qquad \text{si } p \neq \epsilon \\ \mu(p\alpha, q\beta) = \max \begin{pmatrix} \lambda(p\alpha, q) - 1 \\ \lambda(p, q\beta) - 1 \\ \begin{cases} 2 + \mu(p, q) & \text{si } \alpha = \beta \\ \lambda(p, q) - 1 & \text{si } \alpha \neq \beta \end{cases} \end{pmatrix} \end{cases}$$

On vérifiera facilement que cette définition récursive est cohérente et définit complètement λ et μ. On désignera à partir de maintenant la *mesure de similarité* ou *similarité* des 2 mots p et q par $\mu(p, q)$.

On peut d'abord remarquer que $-(|p|+|q|) \leq \mu(p,q) \leq |p|+|q|$. On peut définir, en utilisant cette remarque, une mesure de similarité normalisée :

$$\mu_{\mathcal{N}}(p,q) = \frac{\mu(p,q)}{|p|+|q|}$$

Cette quantité est comprise entre -1 et 1, et plus facilement exploitable dans des algorithmes de recherche de similitudes.

3 Propriétés de la mesure de similarité

Soient p et q 2 mots de $\mathcal{S}^*$, considérons 2 *découpages parallèles* de p et q (avec s_i et t_i qui sont des mots éventuellement vides tels que $s_i \neq t_i$ et α_i des lettres) :

$$p = s_0\alpha_1 s_1\alpha_2 \cdots \alpha_n s_n$$

$$q = t_0\alpha_1 t_1\alpha_2 \cdots \alpha_n t_n$$

Éventuellement, si p et q n'ont aucune lettre en commun, on pourra obtenir un tel découpage avec $n = 1$. On peut définir la quantité :

$$m = 2n - \sum_{i=0}^{n} \begin{cases} 1 & \text{si } s_i \neq \epsilon \text{ ou } t_i \neq \epsilon \\ 0 & \text{si } s_i = t_i = \epsilon \end{cases}$$

On s'intéresse au maximum $m_{\max}(p,q)$ atteint par m lorsqu'on parcourt tous les découpages parallèles possibles des 2 mots. On admettra la propriété :

$$m_{\max}(p,q) = \mu(p,q)$$

Il est alors facile de vérifier les propriétés suivantes :

$$\mu(p,q) = |p|+|q| \Longleftrightarrow p = q$$

$$1 + \mu_{\mathcal{N}}(a,c) \geq \mu_{\mathcal{N}}(a,b) + \mu_{\mathcal{N}}(b,c)$$

Si on définit

$$d(p,q) = 1 - \mu_{\mathcal{N}}(p,q)$$

cette quantité est comprise entre 0 et 2, et définit une distance sur $\mathcal{S}^*$: la distance de similarité.

4 Programmation efficace

La programmation récursive directe de μ n'est pas recommandée ! Sa complexité serait alors exponentielle, de l'ordre de $\mathcal{O}(3^n)$. Étant donnés 2 mots p et q de longueurs respectives m et n on va utiliser une méthode de programmation dynamique *top-down* pour optimiser le calcul. Pour faire simple on appelera $\mu'(i,j)$ et $\lambda'(i,j)$ les valeurs de μ et λ calculées avec les préfixes composés de i et j lettres de p et q. Dans ces conditions, $\mu'(i,j)$ et $\lambda'(i,j)$ sont obtenus directement à partir de $\mu'(i-1,j)$, $\mu'(i,j+1)$, $\mu'(i-1,j-1)$,

$\lambda'(i-1,j)$, $\lambda'(i,j-1)$ et $\lambda'(i-1,j-1)$. On peut stocker toutes les valeurs intermédiaires du calcul de λ et μ dans 2 matrices d'entiers de taille $m+1 \times n+1$, qu'on remplit de gauche à droite et de bas en haut, pour une complexité algorithmique de $\mathcal{O}(nm)$:

```
function Mu(s1,s2:string):Integer;
var
  i,j,a,b,c,n1,n2:Integer;
  T:array[0..1] of PIntegerArray;
  u:Boolean;
begin
  n1:=Length(s1);
  n2:=Length(s2);
  GetMem(T[0],(n1+1)*(n2+1)*SizeOf(Integer));   Le tableau des valeurs de λ'
  GetMem(T[1],(n1+1)*(n2+1)*SizeOf(Integer));   Le tableau des valeurs de μ'
  for i:=0 to n1 do begin
    T[0,i]:=0;    Initialisation de la récurrence pour λ
    T[1,i]:=-1;   Initialisation de la récurrence pour μ
  end;
  for j:=0 to n2 do begin
    T[0,j*(n1+1)]:=0;    Initialisation de la récurrence pour λ
    T[1,j*(n1+1)]:=-1;   Initialisation de la récurrence pour μ
  end;
  T[1,0]:=0;
  for i:=1 to n1 do
    for j:=1 to n2 do begin
      u:=s1[i]=s2[j];
      if u then
        a:=2+T[1,i-1+(j-1)*(n1+1)]
      else
        a:=T[0,i-1+(j-1)*(n1+1)];
      b:=T[0,i+(j-1)*(n1+1)];
      c:=T[0,i-1+j*(n1+1)];
      T[0,i+j*(n1+1)]:=Max(a,b,c);   Relation de récurrence pour λ
      if u then
        a:=2+T[1,i-1+(j-1)*(n1+1)]
      else
        a:=T[0,i-1+(j-1)*(n1+1)]-1;
      b:=T[0,i+(j-1)*(n1+1)]-1;
      c:=T[0,i-1+j*(n1+1)]-1;
      T[1,i+j*(n1+1)]:=Max(a,b,c);   Relation de récurrence pour μ
    end;
  Result:=T[1,n1+n2*(n1+1)];   La valeur cherchée : μ'(m,n)
  FreeMem(T[0]);
  FreeMem(T[1]);
end;
```