

Méthodes Empiriques et Langages de Script

TP 9

G. A. Musillo
musillo4@etu.unige.ch

January 8, 2007

1 Exercice 1: la programmation dynamique de la distance de Levenshtein

Rappelez-vous que la distance de Levenshtein ou distance d'édition qui sépare deux chaînes s et t est le plus petit nombre d'opérations d'édition qu'il faut effectuer pour transformer s en t . Trois opérations d'édition sont permises: l'insertion d'un caractère de t dans s , la suppression d'un caractère de s et la substitution d'un caractère de s par un caractère de t . Rappelez-vous également qu'on représente une séquence d'opérations au moyen d'un alignement. Par exemple, l'alignement suivant représente quatre opérations qui permettent de transformer $gtgcc$ en $ggcga$:

$$\left\langle \begin{array}{cccccc} g & t & g & c & c & - & a \\ g & g & - & c & - & g & a \end{array} \right\rangle$$

L'alignement précédent n'est pas optimal. En effet, il est possible d'aligner $gtgcc$ et $ggcga$ en deux opérations:

$$\left\langle \begin{array}{cccccc} g & t & g & c & c & a \\ g & - & g & c & g & a \end{array} \right\rangle$$

Calculer l'alignement optimal entre deux chaînes de caractères revient à calculer la distance de Levenshtein. L'algorithme qui permet de calculer la distance de Levenshtein entre une chaîne $s = a_1 \dots a_m$ et une chaîne $t = b_1 \dots b_n$ est défini de façon inductive.

Soit $d(i, j)$ la distance minimale qui sépare le préfixe $a_1 \dots a_i$ de s du préfixe $b_1 \dots b_j$ de t . Supposez que vous connaissiez les distances minimales (qui sont les hypothèses d'induction):

- $d(i-1, j-1)$, qui exprime la distance qui sépare $a_1 \dots a_{i-1}$ et $b_1 \dots b_{j-1}$
- $d(i-1, j)$, qui exprime la distance qui sépare $a_1 \dots a_{i-1}$ et $b_1 \dots b_j$

- $d(i, j - 1)$, qui exprime la distance qui sépare $a_1 \dots a_i$ et $b_1 \dots b_{j-1}$

Considérez la première distance $d(i - 1, j - 1)$: si les caractères a_i et b_j sont égaux, alors la distance $d(i, j)$ est égale à $d(i - 1, j - 1)$. Sinon, elle est égale à $d(i - 1, j - 1) + 1$, puisqu'il serait nécessaire de substituer b_j à a_i pour transformer $a_1 \dots a_i$ en $b_1 \dots b_j$.

Si vous connaissiez la seconde distance $d(i - 1, j)$, pour transformer $a_1 \dots a_i$ en $b_1 \dots b_j$, il suffirait de supprimer le caractère a_i . La distance $d(i, j)$ vaudrait donc $d(i - 1, j) + 1$.

Enfin, étant donné la troisième distance $d(i, j - 1)$, il suffirait d'insérer le caractère b_j après le caractère a_i . Et la distance $d(i, j)$ vaudrait donc $d(i, j - 1) + 1$.

Nous avons montré que $d(i, j)$ ne dépend que de $d(i - 1, j - 1)$, $d(i - 1, j)$ et $d(i, j - 1)$. Par conséquent, pour calculer la distance d'édition $d(i, j)$ minimale, il suffit de considérer le minimum de $d(i - 1, j - 1)$, $d(i - 1, j)$ et $d(i, j - 1)$:

$$d(i, j) = \min \begin{cases} d(i - 1, j - 1) & \text{si } a_i = b_j \\ d(i - 1, j - 1) + 1 & \text{si } a_i \neq b_j \\ d(i - 1, j) + 1 \\ d(i, j - 1) + 1 \end{cases}$$

On peut disposer enregistrer ces distances dans une matrice $(m + 1) \times (n + 1)$:

	ϵ	b_1	b_2	\dots	b_n
ϵ	$d(0, 0)$	$d(0, 1)$	$d(0, 2)$	\dots	$d(0, n)$
a_1	$d(1, 0)$	$d(1, 1)$	$d(1, 2)$	\dots	$d(1, n)$
a_2	$d(2, 0)$	$d(2, 1)$	$d(2, 2)$	\dots	$d(2, n)$
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots
a_m	$d(m, 0)$	$d(m, 1)$	$d(m, 2)$	\dots	$d(m, n)$

Nous savons que, pour calculer $d(i, j)$, il suffit de connaître $d(i - 1, j - 1)$, $d(i - 1, j)$ et $d(i, j - 1)$:

$$\begin{array}{ccc} d(i - 1, j - 1) & & d(i - 1, j) \\ & \nearrow & \uparrow \\ d(i, j - 1) & \leftarrow & d(i, j) \end{array}$$

Donc, pour démarrer le calcul récursif de la distance, il suffit à présent de pouvoir calculer les deux cas de base de la récursion: $d(i, 0)$ et $d(0, j)$. La distance $d(i, 0)$ vaut évidemment i (il faut i suppressions pour transformer la chaîne $a_1 \dots a_i$ en la chaîne nulle). Il faut tout aussi évidemment j insertions pour transformer la chaîne nulle en la chaîne $b_1 \dots b_j$. On obtient finalement la matrice de base qui permet tous les calculs nécessaires:

	ϵ	b_1	b_2	...	b_n
ϵ	0	1	2	...	n
a_1	1	$d(1,1)$	$d(1,2)$...	$d(1,n)$
a_2	2	$d(2,1)$	$d(2,2)$...	$d(2,n)$
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots
a_m	m	$d(m,1)$	$d(m,2)$...	$d(m,n)$

La formulation inductive que nous avons donné de la distance minimale entre deux chaînes suggère une implémentation simple: utiliser un tableau 2-dimensionnel, parcouru par les indices i (de la chaîne s) et j (de la chaîne t), qui enregistre les distances calculées. C'est ce que fait le programme récursif suivant (qui n'est rien d'autre que la réécriture en Perl des formules des cas basiques et des hypothèses d'induction):

```
#!/usr/bin/perl
use strict;
use warnings;

my @s = split //, $ARGV[0];
my @t = split //, $ARGV[1];
my ($m, $n) = ($#s + 1, $#t + 1);
my @d;

for (my $i = 0; $i <= $m; ++$i) {
    for (my $j = 0; $j <= $n; ++$j) {
        $d[$i][$j] = -1;
    }
}
for (my $i = 0; $i <= $m; ++$i) {
    $d[$i][0] = $i;
}
for (my $j = 0; $j <= $n; ++$j) {
    $d[0][$j] = $j;
}
sub min {
    my ($a, $b, $c) = @_;
    return ($a < $b ? ($a < $c ? $a : $c) : ($b < $c ? $b : $c));
}
sub edit {
    my ($i, $j) = @_;
    if ($d[$i][$j] != -1) {
        return $d[$i][$j];
    }
    if ($s[$i - 1] eq $t[$j - 1]) {
```

```

    $d[$i][$j] = edit($i - 1, $j - 1);
} else {
    $d[$i][$j] = 1 + min(edit($i - 1, $j - 1),
        edit($i - 1, $j),
        edit($i, $j - 1));
}
return $d[$i][$j];
}
print &edit($m, $n), "\n";

```

Le code qui suit est une version itérative de l'algorithme qui exploite, elle aussi, un tableau 2-dimensionnel afin de ne pas recalculer ce qui l'a déjà été.

```

#!/usr/bin/perl
use strict;
use warnings;

my ($s, $t) = @ARGV;
print &leven($s, $t) . "\n";

sub min {
    my $min = $_[0];
    foreach my $i (@_) {
        $min = $i if ($i < $min);
    }
    return $min;
}

sub leven {
    my ($s, $t) = @_;
    my ($len_s, $len_t) = (length $s, length $t);
    return $len_s if ($len_t == 0);
    return $len_t if ($len_s == 0);
    my @d;
    for (my $i = 0; $i <= $len_s; ++$i) {
        for (my $j = 0; $j <= $len_t; ++$j) {
            $d[$i][$j] = 0;
            $d[0][$j] = $j;
        }
        $d[$i][0] = $i;
    }
    my @a_s = split //, $s;
    my @a_t = split //, $t;
    for (my $i = 1; $i <= $len_s; ++$i) {
        for (my $j = 1; $j <= $len_t; ++$j) {

```

```

    my $subst = ($a_s[$i - 1] eq $a_t[$j - 1]) ? 0 : 1;
    $d[$i][$j] = min($d[$i - 1][$j] + 1,
        $d[$i][$j - 1] + 1,
        $d[$i - 1][$j - 1] + $subst);
}
}
return $d[$len_s][$len_t];
}

```

On vous demande de vous assurer d'avoir bien compris les deux versions, récursive et itérative, de l'algorithme. Pour vous en assurer, vous devez les commenter et remplir à la main la matrice d'édition suivante:

	<i>ε p o l y n o m</i>
<i>ε</i>	
<i>e</i>	
<i>x</i>	
<i>p</i>	
<i>o</i>	
<i>n</i>	
<i>e</i>	
<i>n</i>	
<i>t</i>	

Vos commentaires et votre matrice d'édition doivent m'être remis au plus tard mercredi prochain 17 janvier.

On vous demande également de modifier une des versions données de telle sorte que la séquence des opérations qui mène à la solution soit enregistrée dans le table bi-dimensionnelle et affichée. Remettez-moi votre programme au plus tard le mercredi 24 janvier. Votre programme doit pouvoir afficher la séquence des opérations (insertion, suppression ou substitution) ainsi que les caractères sur lesquelles elles opèrent. Il vous faudra donc garder une trace des opérations effectuées et parcourir la matrice de la cellule (m, n) à la cellule $(0, 0)$ en suivant les traces que vous aurez enregistrées.

Pour montrer que vous avez bien compris le calcul d'un alignement optimal, vérifiez que la matrice ci-dessous des mots *acga* et *atgcta* est bien formée et mettez en évidence dans cette matrice au moins deux alignements optimaux. Indication: un pas vertical correspond à la suppression d'un caractère de la chaîne de départ, un pas horizontal vers la gauche correspond à l'insertion d'un caractère de *t* dans la chaîne de départ *s* et un pas diagonal correspond à une substitution.

	ϵ	a	t	g	c	t	a
ϵ	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
c	2	1	1	2	2	3	4
g	3	2	2	1	2	3	4
a	4	3	3	2	2	3	3