**7.4 编辑距离**

编辑距离算法是由Levenshtein发明的，这是一个动态规划算法。关于该算法的思想，请读者参考相关资料。这里，我们通过一个实例，来讲解这个算法。

假设有字符串s1为jary，和字符串s2为jerry，现在求s1和s2的编辑距离，也就是把s2转换为s1的最少编辑操作步。

首先，我们建立如下的矩阵，并且初始化该矩阵。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | j | a | r | y |
|  | 0 | 1 | 2 | 3 | 4 |
| j | 1 |  |  |  |  |
| e | 2 |  |  |  |  |
| r | 3 |  |  |  |  |
| r | 4 |  |  |  |  |
| y | 5 |  |  |  |  |

从源串的第一个字符(“j”)开始，从上至下与目标串进行对比。

如果两个字符相等，则从此位置的左+1、上+1、左上+0三个位置中取出最小的值，填入该位置；若不等，则从此位置的左+1、上+1、左上+1三个位置中取出最小的值，填入该位置。比如，第一次，源串第一个字符“j” 与目标串的“j”对比，左、上、左上三个位置中取出最小的值0，因为两字符相等，所以填上0。接着，依次对比“j”→“e”、“j”→“r”、“j”→“r”、“j”→“y”等进行处理，直到扫描完目标串，得到的结果如下。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | j | a | r | y |
|  | 0 | 1 | 2 | 3 | 4 |
| j | 1 | 0 |  |  |  |
| e | 2 | 1 |  |  |  |
| r | 3 | 2 |  |  |  |
| r | 4 | 3 |  |  |  |
| y | 5 | 4 |  |  |  |

按照上面的方法，遍历整个源串的各个字符，与目标串的各个字符对比，填写各个单元格，各个单元格的变化如下表所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | j | a | r | y |
|  | 0 | 1 | 2 | 3 | 4 |
| j | 1 | 0 | 1 |  |  |
| e | 2 | 1 | 1 |  |  |
| r | 3 | 2 | 2 |  |  |
| r | 4 | 3 | 3 |  |  |
| y | 5 | 4 | 4 |  |  |
|  |  | j | a | r | y |
|  | 0 | 1 | 2 | 3 | 4 |
| j | 1 | 0 | 1 | 2 |  |
| e | 2 | 1 | 1 | 2 |  |
| r | 3 | 2 | 2 | 1 |  |
| r | 4 | 3 | 3 | 2 |  |
| y | 5 | 4 | 4 | 3 |  |
|  |  | j | a | r | y |
|  | 0 | 1 | 2 | 3 | 4 |
| j | 1 | 0 | 1 | 2 | 3 |
| e | 2 | 1 | 1 | 2 | 3 |
| r | 3 | 2 | 2 | 1 | 2 |
| r | 4 | 3 | 3 | 2 | 2 |
| y | 5 | 4 | 4 | 3 | 2 |

处理完最后一列，则最后一列的最后一个值，为最短编辑距离，即jary和jerry的编辑距离为2。也就是，删除jerry的一个r，然后把e改成a，即可把jerry变成jary。求出编辑距离以后，那么两个字符串的相似度 Similarity = (Max(x, y) – Levenshtein Edit Distance)/Max(x, y)，其中 x、y为源串和目标串的长度。