Trie 上的动态规划：编辑距离与最长公共子序列的批量计算

DONG Yuxuan <https://dyx.name>

创建于北京时间 2015 年 11 月 16 日
更新于北京时间 2025 年 8 月 19 日
修订版 2

摘要

给定一个称为文本的字符串，和一个词典，词典中的字符串称为模式。通过在 Trie 树上以 DFS 序执行动态规划算法，我们可以批量计算文本与每个模式的最长公共子序列长度与编辑距离。

引言

在一个包含若干模式的词典中找出与给定文本最相似的模式。这是一个常见的计算问题。此问题的朴素解法是计算文本与每个模式的相似度，然后选择相似度最大的模式。最常用的字符串相似度指标是最长公共子序列 (LCS) 与编辑距离。假设文本和模式的最大长度为 M，那么计算的时间复杂度为 $O(M^2)$。如果词典含有 N 个模式，那么总的时间复杂度为 $O(NM^2)$。

这个结果是可以优化的。不管是 LCS 还是编辑距离，他们的计算都是通过以两个字符串的前缀为状态的动态规划算法完成的。字典中各模式之间可能存在大量的公共前缀。这便导致了大量的重复计算。我们可以通过将词典建成一颗 Trie 来合并前缀，并在 Trie 上执行 DFS 来计算所有的相似度。将时间复杂度降低到 $O(RM)$，其中 R 是 Trie 的结点个数。

最长公共子序列的批量计算

回顾字符串 `s`, `t` 的最长公共子序列长度是怎么计算的，用 `f(i,j)` 表示 `s` 的长度为 `i` 的前缀，和 `t` 的长度为 `j` 的前缀的 LCS 长度，我们有如下方程。

```
f(i,j)  = 0,                              if i=0 or j=0
        = f(i-1,j-1)+1,                   if s[i-1] = t[j-1]
        = max(f(i-1,j), f(i,j-1))
```

为了避免词典中重复前缀的重复计算，我们将词典组织为一颗 Trie。对每个结点指针 `x`，用 `x->ch` 表示此节点对应的字符，`x->par` 指向父节点，`x->lcs[i]` 表示文本的长度为 i 的前缀与此结点对应模式前缀的 LCS 长度。`x->lcs[]` 可以通过如下代码计算：

```
x->lcs[0] = 0;
for (i = 1; i <= patlen; ++i)
        if (x->ch == pat[i-1])
                x->lcs[i] = x->par->lcs[i-1] + 1;
```

```
        else
                x->lcs[i] = MAX(x->par->lcs[i], x->lcs[i-1]);
```

这样一来，我们可以在 Trie 上做 DFS。对于每一个遍历到的结点，都通过上面的代码计算 `lcs[]`。当 DFS 完成时，就获得了文本与每个模式的 LCS 长度。

编辑距离的批量计算

令 `f(i,j)` 表示 `s` 的长度为 `i` 的前缀与 `t` 的长度为 `j` 的前缀的编辑距离。

```
f(i,j)  = j,                                    if i=0
        = i,                                    if j=0
        = min(
                f(i,j-1)+1,
                f(i-1,j)+1,
                f(i-1,j-1)+delta(s[i-1],t[j-1])
        )
```

其中 `delta(x, y)` 定义为：

```
delta(x, y)     = 0,    if x=y
                = 1,    if x!=y
```

将词典组织为 Trie，对每个结点指针 `x`，用 `x->ch` 表示此节点对应的字符，`x->par` 指向父节点，`x->len` 表示当前节点对应的模式前缀的长度，`x->ed[i]` 表示文本的长度为 `i` 的前缀与此结点对应模式前缀的编辑距离。`x->ed[]` 数组可以通过如下代码计算：

```
x->ed[0] = x->len;
for (i = 1; i <= patlen; ++i)
        x->ed[i] = MIN(
                x->par->ed[i] + 1,
                x->ed[i-1] + 1,
                x->par->ed[i-1] + delta(pat[i-1], x->ch)
        );
```

与计算 LCS 一样，我们在 Trie 上做 DFS。对于每一个遍历到的结点，都通过上述的方法计算其 `ed[]`。这样便计算了所有的编辑距离。

相关工作

相同的思路在 [1] 中被用来计算编辑距离。

参考文档

[1] Steve Hanov. 2011. Fast and Easy Levenshtein Distance Using a Trie. 取读于 从 http://stevehanov.ca/blog/?id=114