

基于英文缩写判断算法及其在文件相似度分析中的应用

2113958 杜旖芃

一、问题定义

(一) 问题背景

1. 单词缩写

在英语里，缩写（abbreviation）是一个很常见的操作，它能够简化长词或短语，使英文单词更加简洁和易于使用。然而，判断某个缩写是否代表某个单词是一件比较困难的事。

总体上来讲，我们考虑比较基础的情形，英文单词转化为缩写有以下几个原则：

1. 连接每个单词的首字母，如 USA 表示 United States of America。在这种情况下，原单词的部分小写字母和空格被删去。
2. 可能某个单词被大写，如 department 被缩写为 Dept.（我们并不需要过多担心后面可能附加有的“.”，因为这往往能够被轻易去除）。在这种情况下，不仅删去了部分小写字母，也将部分字母（这里的 D）大写。

假设我们将英语短语 by the way 缩写为 BTW，这也就是将 by, the, way 三个单词的第一个字母大写，然后删去其余小写字母得到的。

图 1 为我们展示了常见的一些英文短语或单词的缩写，符合上述的基本原则。

下面总结一个基于字符的，确定字符串 **a** 是否可以通过缩写得到字符串 **b** 的策略

1. 删去 **b** 中所有可能存在的特殊字符或空格，删去 **a** 中所有可能存在的特殊字符或空格。
2. 利用一个算法，判断：能否将 **a** 中部分字母大写，然后将 **a** 中部分小写字母删去，最后得到的结果等于 **b**。事实上，为了方便后续动态规划的求解，我们将 **b** 的字母全部大写，然后只需要判断能否将 **a** 中部分字母大写，然后将 **a** 中剩余所有小写字母删去，最后得到的结果等于 **b**。

由于第一步操作往往是简单的（至少可以在线性时间内完成），所以简单地考虑的话，只需要考虑步骤 2 这一问题。

縮寫	全文	中譯
FYI	for your information	供您參考
ASAP	as soon as possible	盡快
BTW	by the way	附帶一提
NP	no problem	沒問題
TIA	thanks in advance	先謝了
e.g.	exempli gratia (拉丁文)	舉例來說
i.e.	id est (拉丁文)	也就是
attn.	attention	注意、敬請知悉
ths.	thanks	謝謝
rgds.	regards	敬上
info.	information	訊息
incl.	including	包括
vs.	versus	對抗、與...相對
pls.	please	請
P.S.	postscript	附錄、附筆
ver.	version	版本
Q1	first quarter	第一季
int'l	international	國際性的
YTD	year to date	年初到現在

图 1

此外，本问题的求解还能判断受污染的文件与原文件是否具有一定的相似度。即，此问题的求解算法可以运用于：假设有一份文件受到了污染，部分大写字符变为了小写，或是插入了一些小写字母，我们需要判断此文件属于原来的哪个文件，进而进行修复。我们可以利用此算法进行字符比对分析（还需要先对原文件进行特殊处理，如全部转为大写字符等），得到近似的结果。

（二）对问题的形式化定义

假设我们可以对字符串 *a* 进行以下两个操作之一：

1. 将字符串 *a* 某些小写字母大写
2. 删除 *a* 中剩下的所有小写字母

问题一：给定两个无特殊字符、无空格的字符串 *a* 和 *b*，判断 *a* 能不能通过以上两种操作变为 *b*。

问题二：进一步地，给定两个文件 `PollutedFile.txt`（受污染文件）和 `OriginalFile.txt`（其中存在空格等特殊字符），使用此算法比对分析受污染的文件是否可能来源于 `OriginalFile.txt`。

二、问题分析

（一）错误的解法

首先，一个比较直接的想法是直接遍历 *a* 和 *b* 的每一个字符，例如 *a* = "String", *b* = "STR"，我们通过两个索引遍历 *a*，如果有和 *b* 相等的字符（不论大小写）那么 *b* 的索引加 1，*a* 的索引也加 1。否则如果 *a* 为大写，那么说明已经不能构成缩写；如果 *a* 为小写，那么只将 *a* 的索引加 1。这个算法（实际上是错误的）可以使用伪代码描述为：

```
定义 i = 0, j = 0
while j < b.size() and i < a.size() then
    if a[i] 与 b[j] 在不区分大小写的条件下相等 then
        i++
        j++
    else
        if a[i] 为大写
            return false
        end if
        i++
    end if
end while
if j + 1 == b.size() then
    return true
else
    return false
end if
```

然而这样做会导致错误的结果：此算法没有考虑到 *a* 不能具有过多的大写字母，例如 *a* = "strT", *b* = "STR"，按照此算法的逻辑，我们会依次遍历 *a* 的 "str" 部分和 *b* 的 "STR" 部分，然后得出结论 *b* 可以作为 *a* 的缩写；然而，根据问题定义，*a* 包含有无法被删除的 "T"，必然导致其无法缩写为 *b*。

如果我们加上一个对 **a** 中所有大写字符的检测（由于缩写匹配时并不要求大小写，我们必须在算法开始的时候检测），判断 **a** 中所有大写字符必须在 **b** 中出现，仍然会有错误判断的情况：**a** = "staTe", **b** = "STE"。这个例子让此算法的本质缺点一览无遗：如果按照之前的原则，**a** 的 "st" 会与 **b** 的 "ST" 进行匹配，而之后 **a** 遍历到 "T" 时，**b** 已经没有 "T" 可供消去。本应返回 **True**，算法却返回了 **False**。

(二) 思路的修正

反思上面的错误算法，我们发现，上面算法失败的原因是其偏向于“贪心”——每次 **a** 和 **b** 比对时，一旦有字符的匹配那么就会“消去”**b** 中的一个字符，而这很有可能导致局部最优而非整体最优的问题。

为了解决这一问题，我们应当使用动态规划。动态规划能够更好地进行调整，进而得到全局最优解。接下来，我们在“算法设计”版块里给出这样的正确算法。

三、算法设计

(一) 动态规划

吸取前面错误算法的经验，我们利用动态规划解决此问题。

如图 2 所示，我们初始化一个二维数组（备忘录），行表示字符串 **a**，列表示字符串 **b**。这个二维数组的每一个元素用于表示当前的 **a** 和 **b** 是否能够满足题设对应的缩写要求。



图 2

接下来就是按照动态规划的套路，对这个表进行逐步填充。仍然以上图为例，我们先初始化第一列元素（除了第一行的剩下元素），此时 **b** 为空，那么只需要看 **a** 中是否有大写字母：如果当前的 **a** 前面有大写字母，那么说明是不能缩写到空字符的，即设为 **False**，否则设为 **True**。如图 3 所示，此时我们将第一列的值初始化完毕。

`abbreviation("dAbC", "")`
Even if we remove all lowercase
letters, the capital A and C
cannot be removed

	""	"A"	"B"	"C"
""	T			
"d"	T			
"A"	F			
"b"	F			
"C"	F			

图 3

接下来，为了逐步填充剩下的元素，我们需要仍然以列的形式进行填写。例如，第二列第一个元素应当为 **F**（没有字符，不可能有缩写）；第二行第二列元素为 **F**，因为当前 **"d"** 并不能匹配上 **"A"**，如图 4 所示。

以此类推，我们可以把第二列的元素值全部填充完毕——第三行第二列：两个大写的 **"A"**，“消去”，看左上角发现为 **T**，所以该元素也为 **T**；第四行第二列：是小写的 **"b"**，那么不受影响；第五行第二列：大写的 **"C"**，与当前 **b** 的 **"A"** 不匹配，直接置为 **F**。

abbreviation("d", "A")
a lowercase d cannot be
capitalized to match A, and
removing it doesn't change the
situation

	""	"A"	"B"	"C"
""	T	F		
"d"	T	F		
"A"	F			
"b"	F			
"c"	F			

图 4

我们这里总结一下自第二列起的递推关系：假设使用二维数组 `opt` 存储，那么要判断当前 `a` 的元素是否可以缩写为当前的 `b`，也即求 `opt[i][j]`，我们有几种情况：

1. 当 `a[i-1]` 是大写字母时：

- 如果 `a[i-1]` 与 `b[j-1]` 相等，那么说明可以同时“消去”这两个元素。（当前的 `a[i-1]` 可以匹配当前的 `b[j-1]`，因此我们继续判断前一个位置的匹配情况，即 `opt[i][j] = opt[i-1][j-1]`。）
- 如果 `a[i-1]` 与 `b[j-1]` 不相等，由于都是最末尾元素，且无法将大写字母 `a[i-1]` 变为 `b[j-1]`，因此无法进行匹配，即 `opt[i][j] = false`。

2. 当 `a[i-1]` 是小写字母时：

- 如果 `a[i-1]` 与 `b[j-1]` 在不区分大小写的条件下相等，则当前的小写字母 `a[i-1]` 可以匹配当前的 `b[j-1]`。此时我们有两种选择：
 - 选择一：将小写字母 `a[i-1]` 转换为大写字母，匹配当前 `b` 的这一个元素。这时，我们需要判断前一个位置的匹配情况，即 `opt[i][j] = opt[i-1][j-1]`。

- 选择二：删除小写字母 $a[i-1]$ ， a 的前一个元素匹配当前位置。这时，我们需要判断当前位置的前一个位置的匹配情况，即 $opt[i][j] = opt[i-1][j]$ 。

因此，合并后的代码为： $opt[i][j] = opt[i-1][j-1] || opt[i-1][j]$ 。

- 如果 $a[i-1]$ 与 $b[j-1]$ 在不区分大小写的条件下不相等，那么无法将小写字母 $a[i-1]$ 变为 $b[j-1]$ ，因此只能选择删除小写字母 $a[i-1]$ ， a 的前一个元素匹配当前位置，即 $opt[i][j] = opt[i-1][j]$ 。

伪代码如下：

```
定义 opt 数组，大小为 (a.size() + 1) × (b.size() + 1)
初始化第一列和第一行
for j = 1 to b.size() do
    for i = 1 to a.size() do
        if isupper(a[i - 1]) then
            if toupper(a[i - 1]) == b[j - 1] then
                opt[i][j] = opt[i - 1][j - 1]
            else
                opt[i][j] = false
            end if
        else
            if toupper(a[i - 1]) == b[j - 1] then
                opt[i][j] = opt[i - 1][j - 1] or opt[i - 1][j]
            else
                opt[i][j] = opt[i - 1][j]
            end if
        end if
    end for
end for
return opt[a.size()][b.size()]
```

（二）空间复杂度的优化

我们注意到，上述解答的 opt 数组每一次运算只会计算一列的值。因此，实际上我们的 opt 只需要初始化为两列即可，在每一次循环的时候移动数组的第二列到第一列。这样，我们的空间复杂度就由 $O(m+n)$ 降至了 $O(m)$ （ m 表示 a 的长度， n 表示 b 的长度）。

也就是说，备忘录数组 opt 的定义修改为 $bool\ opt[MAX_LENGTH + 1][2]$ ，然后在每一轮外层循环结束将第 1 列拷贝到第 0 列。

实际上，只需要在上面的伪代码中稍作修改即可。在实际应用中，我们的字符可能是非常非常长的，应用如接下来的文件处理时，省下的这一些空间相当的关键。

简要的伪代码如下：

```
定义 opt 数组, 大小为 (a.size() + 1) × 2
初始化第一列和第一行
for j = 1 to b.size() do
    for i = 1 to a.size() do
        将 opt 压缩至两行运算
    end for
    拷贝 opt 的第二列至第一列
end for
return opt[a.size()][0]
```

（三）文件处理

为了解决我们的第二个问题：受污染文件的源判断，我们还需要处理好文件的读入，以及特殊字符的去除等。

对原文件和受污染文件的内容进行特殊字符去除处理（包括数字），可以使用类似代码中的循环读取字符并判断的方式，将非特殊字符添加到新的字符串中。原文件在去除特殊字符后，还需要转为大写，匹配我们的函数。

接着，调用前文的函数即可。

四、复杂度分析

假设 m 表示 a 的长度， n 表示 b 的长度。

1. 时间复杂度：根据上述算法的设计，我们可以看见无论有没有空间的优化，都需要遍历 $b.size() * a.size()$ 这么多步，也就是说时间复杂度为 $O(mn)$ 。
2. 根据上文，通过我们的优化，空间复杂度由 $O(m+n)$ 降到了 $O(m)$ 。

五、测试用例

（一）问题一

测试用例：

- *Input*

第一行包含一个整数 q ，表示查询的数量。

接下来的 $2q$ 行按照以下格式排列：

- 每个查询的第一行包含一个字符串 a 。
- 每个查询的第二行包含一个字符串 b 。

- *Constraints*

字符串 a 仅包含大写和小写的英文字母， $ascii[A-Za-z]$ 。

字符串 b 仅包含大写的英文字母， $ascii[A-Z]$ 。

- *Output*

对于每个查询，如果可以通过操作使得字符串 s 等于字符串 t ，则在新的一行上打印 YES。否则，打印 NO。

- *Sample Input*

10
Pi
P
AfPZN
APZNC
LDJAN
LJJM
UMKFW
UMKFW
KXzQ
K
LIT
LIT
QYCH
QYCH
DFIQG
DFIQG
sYOCa
YOCN
JHMwY
HUVPW

- *Sample Output*

YES
NO
NO
YES
NO
YES
YES
YES
NO
NO

更多的测试数据见“测试数据”文件夹。

（二）问题二

测试用例：

- *Sample Input*

OriginalFile.txt:

There's something down there. It's Gollum.

Gollum?

He's been following us for three days.

He escaped the dungeons of Baraddur?

Escaped, or was set loose. Now the Ring has brought him here. He will never be rid of his need for it. He hates and loves the Ring, as he hates and loves himself. Smeagol's life is a sad story. Yes, Smeagol he was once called. Before the Ring found him. Before it drove him mad.

It's a pity Bilbo didn't kill him when he had the chance.

Pity? It is pity that stayed Bilbo's hand. Many that live deserve death. Some that die deserve life. Can you give it to them, Frodo? Do not be too eager to deal out death and judgment. Even the very wise can not see all ends. My heart tells me that Gollum has some part to play yet, for good or ill, before this is over. The pity of Bilbo may rule the fate of many.

I wish the Ring had never come to me. I wish none of this had happened.

So do all who live to see such times. But that is not for them to decide. All we have to decide is what to do with the time that is given to us. There are other forces at work in this world, Frodo, besides the will of evil. Bilbo was meant to find the Ring. In which case, you also were meant to have it.

PollutedFile.txt:

.there's sovmhething down gthere. It's gollum.
Gollum?
He's been following us for three davys.
He esbccwaped the dungenesons of baraddur?
Escaped,ors wags set loose.Now the Ring has brought him here.He
will never be rivd of his need for it.He hates and loves thse
Rixcng, as hfsde hates and loves himself.Smeagol's life is a sad
story.Yes, Smeagol he was once called. Before the Ring found
him.Befaore it drove him! mad.bvefore
It's a pity Bilbo didn't kidsll him when he had the chance.given
Psity?It is pity that stayed Bilbo's handg.Mwany that live deserve
death. Some that die deserve life.Can your give it to tahem,
Frodo?Do not be too eager to deal vout death and judgment.Even the
very wise can not see all vends.My heart tells me that Gollum has
some part to plasay yet, for jgood or ill,before this is over.The
pity of Bilbo may rulbe the fate of many.f
I wish the Ring had never come to mse.I wish none of this had
happened.
So do all who live to see such times. But that is not for them to
decide.All we have to decide is what to do with the time that is
given to us.Therec are other forces at work in this world, Frodo,
besides the will of evil.Bilbo was meant to find the Ring.In which
case, you also were meant to have it.
davys.

- *Sample Output*

YES

更多的测试数据见“测试数据”文件夹。

六、效果展示

下面，对于两个问题，分别展示一组样例测试。

使用优化空间复杂度的算法，验证上述的样例 1，测试通过，如图 5 所示，与预期结果一致。

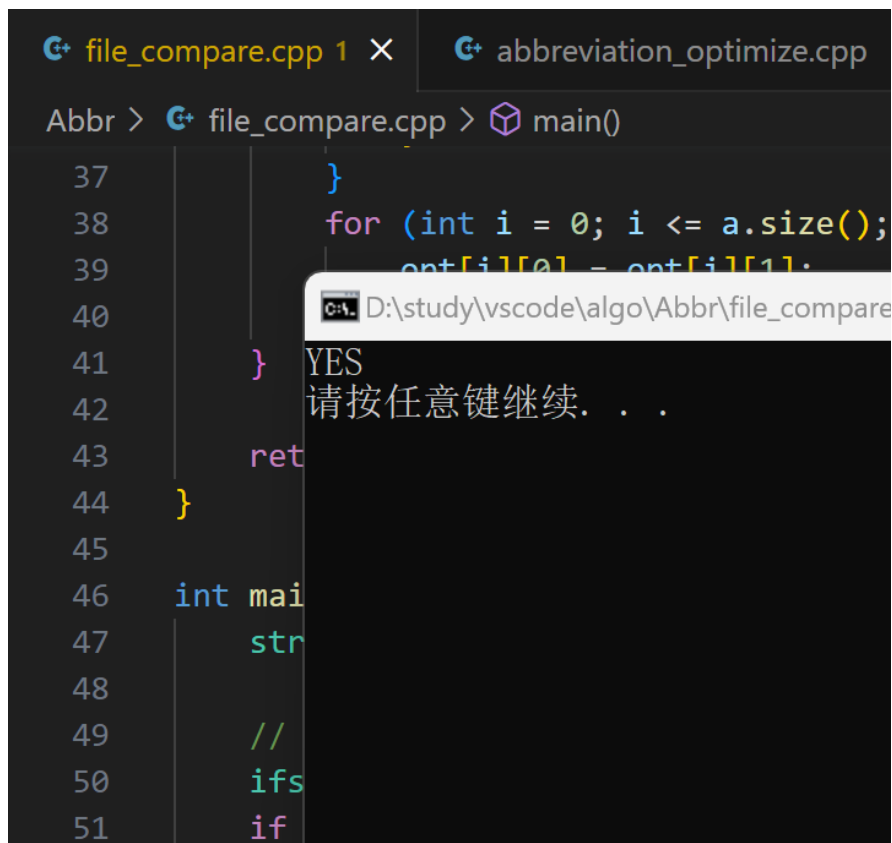
The image shows a VS Code editor window with three tabs: `file_compare.cpp`, `abbreviation_optimize.cpp` (active), and `abbreviation.cpp`. The active tab contains the following C++ code:

```
Abbr > abbreviation_optimize.cpp > abbreviation(string, string)
17     for(int j = 1; j <= b.size(); j++) {
18         // ...
19         // ...
20         // ...
21         // ...
22         YES
23         AfPZN
24         APZNC
25         NO
26         LDJAN
27         LJJM
28         NO
29         UMKFW
30         YES
31         KXzQ
32         K
33         NO
34         LIT
35         YES
36         QYCH
37         QYCH
38         YES
39     }
40     DFIQG
41     YES
42     sYOCa
43     YOCN
44     NO
45     JHMYW
46     HUPW10
47     NO
48     请按任意键继续. . .
```

The output of the program is displayed in the terminal window at the bottom, showing the same sequence of strings as the code. The terminal also shows the message "Build finish" and "Executing task: C/C++: cl.exe build active file".

图 5

使用本算法匹配污染文件，我们可以成功地验证上述样例，如图 6 所示。



```
Abbr > G+ file_compare.cpp 1 X G+ abbreviation_optimize.cpp
Abbr > G+ file_compare.cpp > main()
37     }
38     for (int i = 0; i <= a.size();
39         cout[i][0] = cnt[i][1];
40
41     } YES
42     请按任意键继续. . .
43     ret
44 }
45
46 int mai
47     str
48
49     //
50     ifs
51     if
```

图 6

七、总结

本次大作业研究了判断英文缩写合理性的算法，分析了为什么贪心算法在此问题是失效的，并利用动态规划设计了一个判断英语词语缩写的算法，最后分析了复杂度，并改进了空间复杂度；同时，进一步地将算法运用于复杂的文本处理——判断污染文件相似度。

进一步思考：尽管本报告指出该问题的求解方法可应用于判断受污染的文件与原文件之间的相似度——通过对原文件进行预处理（如转为大写字符），然后利用该算法进行字符比对分析，可以得到近似的结果。然而实际文件的污染程度可能更为严重，本次实验提及的方案只能处理大写字符向小写字符转换或添加小写字符等简单污染，具有局限性。同时，该算法也无法处理复杂的缩写规则或考虑语义上的相似性。在实际应用中，可能需要结合其他技术和方法来解决更复杂的缩写判断和文件相似度问题。

总体而言，基于字符的缩写判断策略为英文单词缩写的判断问题提供了一种简单有效的方法，并可应用于文件的检测。同时，进一步的研究和改进仍然是有益的，以解决更复杂的缩写和相似度判断问题。