2 File Challenge

Zhe Yu (zy19@rice.edu)

January 24, 2018

1 ReadFile

1.1 Description

Firstly, we scan two files respectively. Instead of mapping the quantifier to its corresponding names, the algorithm will store all lines in files of the format ”quantifier names” into a sequence. Then we will sort the sequence by the alphabetical order of the contents of each line to get a sorted sequence.

Since the quantifier is unique in each file, thus can only appear at most twice in the sequence. In that sorted sequence, we found it easy to check if a quantifier has two names attached to it. We can just look at the quantifier’s neighbor (quantifier of the previous one) and check if two quantifiers are the same.

At the same time of checking repetition for quantifiers, we will store the quantifier into a map that maps to its corresponding name(s). Then by going through the mapping will we generate our targeted outputs formatted in a certain way.

1.2 Solution of runtime $O(m + n)$

Here we denote the number of lines in file1 to be m, and file2 to be n

Algorithm 1: ReadFile

Input: Filename of file 1, $f_n 1$; Filename of file 2, $f_n 2$

Output: A targeted file, tf, with each unique quantifier has its members merged in it

```
1  Initialize an empty sequence $s$;
2  Initialize an empty mapping $m(quant, name)$;  // Map set of quantifier $quant$
     to its corresponding set of names $name$
3  Create a new empty file $tf$ ;
4  $f1 \leftarrow$ read file with name $f_n 1$ ;
5  foreach line $l$ in $f1$ do
6      Add $l$ to $s$ ;
7  $f2 \leftarrow$ read file with name $f_n 2$ ;
8  foreach line $l$ in $f2$ do
9      Add $l$ to $s$ ;
10 Sort $s$ by the alphabetical order of its elements;  // Each element of $s$ is a
     string
11 $p \leftarrow$ size of $s$;
12 foreach $i < p$ do
13     $c \leftarrow$ split $s_i$ by blank space ;
14     if $i > 0$ then
15         $prev \leftarrow$ split $s_{i-1}$ by blank space;
16         if $prev_0 = c_0$ then
17             Add $c_{1:END}$ to $m_{c_0}$;
18         otherwise do
19             $m_{c_0} \leftarrow c_{1:END}$;
20     otherwise do
21         $m_{c_0} \leftarrow c_{1:END}$;
22 foreach $i \in quant$ do
23     if size of $m_i$ = 2 then
24         $line \leftarrow$ concatenate elements in $m_i$ together with a blank space in between ;
25         $line \leftarrow$ concatenate $i$ to the beginning of $line$ with a blank space in between ;
26         Write a new line $line$ in the file $tf$ ;
27     otherwise do
28         $line \leftarrow m_i$ ;
29         $line \leftarrow$ concatenate $i$ to the beginning of $line$ with a blank space in between ;
30         Write a new line $line$ in the file $tf$ ;
31 return $tf$;
```

1.3 Pros and Cons

1.3.1 Pros

The runtime for this algorithm is definitely an advantage if the files are reasonably large since all we need to do is to scan through both files and check the repetition of the quantifiers. So it is very cheap for us if the files are huge, $i.e.$, there are millions of lines in each file.

We can generate the outputs in such way that the quantifiers are ordered in an alphabetical order. Though it does not matter in this problem, order certainly matters under other scenarios.

1.3.2 Cons

This approach requires checking the repetition which can be costly if we do not limit our quantifier to be unique in each of the files. Therefore, in a more general scenario (where the quantifier is not unique in each file), we probably need to modify our algorithm and that may not put us better than $O((m+n)^2)$. On the contrary, if we just use create a map that maps quantifiers to names and go through each quantifier to merge the repeated ones, then we don't need to have additional modifications. So, if the files are small, $i.e.$, hundreds or thousands of lines, then we may just apply a brute-force algorithm for the sake of generalizations in similar problems.

If the files are too large, then linear time will still be a problem. Then I think we should break files into multiple parts to perform the same algorithm and merge them at last.