File Challenge Questions

Santiago Garcia Acosta

February 13, 2018

a.) An algorithm to solve this problem would be one that reads each file into a string with delimiters and then separates the string by spaces and new line delimiters, mapping the text at the beginning of lines (identifiers) to the text at the rest of their lines. The algorithm would then iterate over the mappings for each file and compare the keys (identifiers) of one file's maps to the keys of the other file. If they are found to be identical, then the algorithm will combine their mapped statements into a new statement. The identifier would then be mapped to the new statement in a new mapping for the file that is to be produced, and write the new file once all the identifiers for both files have been compared with one another.

b.) *Pseudocode:*

Algorithm 1: filechallenge

```
Input: $file1$, $file2$ being file locations in the user's computer
string1 ← result from reading $file1$ into a string + "\n";
string2 ← result from reading $file2$ into a string + "\n";
$strings \leftarrow (string1, string2)$;
Initialize $map1$ to an empty mapping;
Initialize $map2$ to an empty mapping;
$maps \leftarrow (map1, map2)$;
for $i \leftarrow 0$ to 1 do
    $string \leftarrow strings[i]$;
    $idx \leftarrow 0$;
    $idxbase \leftarrow 0$;
    $idfound \leftarrow 0$;
    for $char \in string$ do
        if $char =$ " " and $idfound = 0$ then
            $idfound \leftarrow 1$;
            $idval \leftarrow string[idxbase : idx]$;
            $idloc \leftarrow idx$;
        if $char =$ " \n " and $idfound = 1$ then
            $maps[i]_{idval} \leftarrow string[idloc + 1 : idx]$;
            $idfound \leftarrow 0$;
            $idxbase \leftarrow idx + 1$;
        $idx \leftarrow idx + 1$;
$newstring \leftarrow$ "";
for $delim \in maps[0]$ do
    if $delim \in maps[1]$ then
        $newstring \leftarrow newstring + delim +$ " " $+ maps[0]_{delim} +$ " " $+ maps[1]_{delim} +$ "\n";
$newfile \leftarrow$ generated file from $newstring$;
return $newfile$ ;
```

The pros of this approach are that it is simple and easy to understand and, of course, solves the problem given. However, it can be computationally intensive for larger file sizes as the algorithm has a running time of $O(n^2)$ where n is the number of characters in the string for each file, assuming each file has the same number of characters. Another pro of this however is the ease with which it can be implemented with common day programming languages, as a lot of what is shown in the algorithm can be done in a single line using built-in functions.