File Challenge

Luis Clague (lc61)

Feb 5, 2018

The Problem

You have two files. Each line of each file consists of a unique identifier followed by space-delimited fields. For each file, a unique identifier is present only once. The list of unique identifiers in the first file may not be the same as in the second file.

From this, you want to produce one file that contains, on each line, a merged line from the input where the unique identifiers match. For example:

File 1
rp1 Mary Smith
st7 John Doe
uk9 Alex Johnson

File 2
po9 Claudine Evert
uk9 Patricia Nelson
st7 Caroline Winchester
pl3 Sarah Harvey

Output
st7 John Doe Caroline Winchester
uk9 Alex Johnson Patricia Nelson

The order of the lines and the fields within the lines does not matter; the below would also be acceptable as output

Output
uk9 Patricia Nelson Alex Johnson
st7 Caroline Winchester John Doe

Describe an algorithm to solve this problem.

The simplest, most brute force algorithm to solve this problem would be to, for every entry in File 1, search File 2 for a match, which would take in the worst case $O(n^2)$ comparisons, the worst case being when there are no matches across both files. Because both lists of identifiers do not have absolute order, I cannot do anything mathematically to optimize the runtime from n^2. Further, because I don't know any more information (how many total matches exist, if a match exists at all) the algorithm to literally compare each entry is the solution I will describe in my pseudo-code.

Write pseudo-code for your algorithm

Algorithm 1: FileMatch

Input: File 1, f_1, File 2, f_2 both with two fields per line: ID denoting the identifier, and $name$ denoting the associated name.

Output: An output file, $final\_file$ that contains, on each line, a merged line from the input where the unique identifiers match.

```
1 final_file ← New File;
2 foreach line ∈ f_1 do
3     foreach inner_line ∈ f_2 do
4         if line_ID = inner_line_ID then
5             final_file_ID ← line_name, inner_line_name
6 return final_file
```

What are the pros and cons to your approach? What computational limitations might force you to do something differently, and what would that be?

In this algorithm there are runtime and spatial concerns. Assuming the files are small enough to fit in memory, the issue of runtime looms large, as this algorithm can be highly unscalable for inputs of increasing size. This can be alleviated somewhat by using a parallelization approach, dividing one file to one computer and finding the unique term, and performing the same task with another computer, then merging the two results. This approach would only be necessary in the edge case that both input files are obscenely large and searching them proves prohibitively long for just one computer.