# PDF challenge – two file

Yanjun Chen

February 5, 2018

## 1 Algorithm

An algorithm for solving this problem is described as follow.
1, Firstly, we read every line of $file1$, split the line on the first occurrence of space, and store the identifier-name pair every line as a key-value pair in a mapping $m$.
2, Then, for every line of $file2$, we split it the same way and find its identifier-name pair ($key$ - $value$).
3, If mapping $m$ has this $key$, we output a line in a new file $file3$ with the following format: $key \quad value\_in\_file1 \quad value\_in\_file2$. Every time we see an identical key, we add in a new line in $file3$ with this format.
4, After we finish reading all lines of $file1$, we will have the $file3$ we want.
Denote the size of file1 $m$ and the size of file2 $n$. The algorithm has a worst-case running time of $O(m + mn) = O(mn)$.

## 2 Pseudo-code

**Algorithm 1:** TwoFileChallenge

**Input:** Two files $file1$ and $file2$, which contain lines of identifiers and space-delimited fields.

**Output:** Another file $file3$, which is the intersection of the two inputs.

```
m ← an empty mapping;
file3 ← a new file;
foreach line in file1 do
    key, value ← split line with the first occurrence of space;
    m[key] ← value;
foreach line2 in file2 do
    key2, value2 ← split line2 with the first occurrence of space;
    if key2 ∈ m then
        Append a new line (key m[key] value2) to file3;
return file3;
```

# 3 Discussion

Pros: The algorithm is easy to implement and straight forward. It has $O(mn)$ complexity, which is acceptable by the nature of this problem.
Cons: The worst-case running time for the algorithm is actually $O(m + mn)$. If file1 and file2 get really large, implementing the algorithm may be slow. Furthermore, the algorithm stores a mapping with size $m$, which could also be problematic if the inputs are large.

In real applications, our computations may be too slow for when facing to really big inputs. The algorithm may also take up too much memory. Thus, I need to care about more details in the algorithm. What I would do is that I would take advantage of the fact that every identifier will only appear once in each file. In this case, if we find a key match between file1 and file2, we can pop out that key so that in the next iteration, we don't need to go over that key again to check that if the new key is the same as it. Doing this will significantly decrease the time for our computations when inputs are large. Also, to save memory, we don't need to store that mapping if we do everything in one big nested loop. The improved version is shown below. It has worst-case running time $O(mn)$ and small memory use.

**Algorithm 2:** TwoFileChallengeImproved

**Input:** Two files $file1$ and $file2$, which contain lines of identifiers and space-delimited fields.

**Output:** Another file $f3$, which is the intersection of the two inputs.

```
1  f1 ← a copy of file1;
2  f2 ← a copy of file2;
3  f3 ← a new file;
4  foreach line1 in f1 do
5      key1, value1 ← split line1 with the first occurrence of space;
6      foreach line2 in f2 do
7          key2, value2 ← split line2 with the first occurrence of space;
           if key1 = key2 then
8              Append a new line (key1 value1 value2) to f3;
9              Delete line2 in f2;
10 return f3;
```