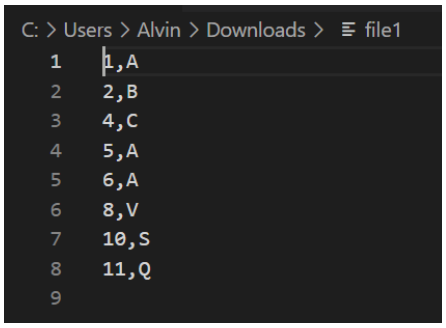
Alvin To
CSE 168
10/18/2022

Lab 2 - Distributed Data Analysis

The option that I worked on for this lab assignment was Option 2: Data clean. For the map function, we first need to identify the way the strings in our files are formatted.



```
1,A
2,B
4,C
5,A
6,A
8,V
10,S
11,Q
```

Referencing the WordCount.java example, we acknowledge that a StringTokenizer object can be used for a loop to iterate over the file line by line. Once we have access to individual lines, we can then use a split() function with comma as our separator so that we can individually access the key and the value for each line. Within our Mapper class, we initialized a counter (starts at 1) and prev_value variable in order to keep track of the current iteration count and the previously iterated value, respectively. On each iteration of our loop, we check if our counter is equal to our key; if they are equal, then no key is missing and we can write our key-value pair to the output like normal, then store our value in prev_value and increase our counter by 1. Otherwise, if our counter is not equal to our key, it means that a key is missing and that our key is greater than our counter by 1. In this case, we write our counter minus 1 and prev_value as a key-value pair to the output, and also write our current key-value pair as another output. Since we are writing 2 key-value pairs in this logical branch, we increase our counter by 2 and continue to store our current value in prev_value. Looking at file1 from above as an example, our expected output is: (1,A), (2,B), (2,B), (4,C), (5,A), (6,A), (6,A), (8,V), (8, V), (10,S), (11,Q).

I am unsure whether this mapper solution works on distributed machines, as the code may not know which prev_value to store. My code is tested and works as intended on a singular machine however.

For the reduce function, we first initialize a counter variable equal to our key. We simply iterate over our list of values parameter, and write our counter and value as a key-value pair to the output, then increase our counter variable after each iteration. For example, if our input for the reduce function is reduce(key: 2, val: [B, B]), our expected output is (2,B), (3,B). Because our counter was initialized to 2 with these inputs, we write the counter and value to our output as (2,B), increment our counter, then write the counter and value again to our output as (3,B). After performing the reduce function on all our key inputs, we should result in our expected output of: (1,A), (2,B), (3,B), (4,C), (5,A), (6,A), (7,A), (8,V), (9, V), (10,S), (11,Q).