Project 1 Phase 1

CIS-687 OBJECT ORIENTED DESIGN

PROF. SCOTT ROUECHE

5/1/2022

Anthony Redamonti
SYRACUSE UNIVERSITY

Project 1 Phase 1

1. Link to repository: https://github.com/ColtonWilson/MapReduce_Project_Phase-1

2. "How-To" document and code reviews are in the repository.

3. I was responsible for designing and implementing the Sorting and Reduce packages. I was also responsible for implementing those packages in the Workflow class. I also designed the consistency unit test, which creates two Workflow objects that process the same input and makes sure the output text files match. My partner, Colton Wilson, was responsible for designing and implementing the Executive.cpp file (Main) as well as the FileManagement and Map packages. He was also responsible for implementing the Map package in the Workflow class. He also designed many unit tests for our project.

   The Sorting class creates a struct "word" for each word in the intermediate file.

```
// a private struct "word"

struct word {
	string key;
	int numberOfOccurrences;
};
```

   Originally, the words were placed into a list by the format method. However, the program was experiencing unusually long runtimes, so I changed the container type to a vector and implemented the following algorithm. Each word is placed into the vector with the number of occurrences being equal to 1.

   Vector = {{key: "apple", numberOfOccurrences: 1}, {key: "pear", numberOfOccurrences: 1}, {key: "apple", numberOfOccurrences: 1}, {key: "pear", numberOfOccurrences: 1}}

   Next, the words are sorted (swapped) so that matching words are adjacent.

   Vector = {{key: "apple", numberOfOccurrences: 2}, {key: "apple", numberOfOccurrences: 1}, {key: "pear", numberOfOccurrences: 2}, {key: "pear", numberOfOccurrences: 1}}

   Notice that the first word in the grouping of adjacent words has the correct number of occurrences. The first word is passed to the export method (along with its number of occurrences), and the algorithm knows the location of the next grouping of words based of the number of occurrences of the current grouping.

   The runtime was significantly improved using this algorithm and container type (reduced 60%). The entire works of Shakespeare were processed by the project in 4 minutes and 30 seconds.

4. The project went great. I couldn't have had a better partner. Originally, I had some issues on my PC getting the boost libraries integrated with Visual Studio. I was able to get them working using

the official integration guide provided by Microsoft: Integration Guide. One problem that my partner and I could not overcome is in the Shakespeare file “Cymbeline.txt.” There is a strange exception thrown from within the BOOST library when attempting to tokenize the input. Some of the blank spaces are being interpreted as special symbols like “Í” or “á.” The tokenizer in the BOOST library is throwing the exception when attempting to convert them to ASCII format. We were able to fix this issue by removing all the spaces from the file and adding them back. I recommend using the Shakespeare test files provided in our repository when testing.