Student Name: Kendrick Kee
UOW ID: 7366814

Assume that a txt file named “crime-stories.txt” exists as such:

Crime-stories.txt

“each line in the text file”
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Tempore, voluptate
reprehenderit praesentium veniam ex culpa dolor rem vel Lorem ipsum obcaecati
sit modi doloremqu Magni voluptatum optio fuga quam quas repellendus asperiores
qui, quidem quas sit temporibus Lorem ipsum optio fuga modi doloremque in quam quas
obcaecati, harum error totam enim tenetur laborum fuga optio fuga quam quas
…

And “Patterns.txt” exist as such:

Patterns.txt

“each line in the text file of patterns”
Lorem ipsum
dolor rem fugiat
sit modi Magni asperiores
qui, sit temporibus
enim tenetur laborum
…

Based on the problem statement, the implementation shows similar parallels to “Grepping” a file using regular expressions.

<u>Map Phase:</u>

Using the assumed “Match(text-line, text-pattern)” function, I would begin by mapping each line of Patterns.txt into a <key,value> pair where the each pattern is the key and the value is assigned to 0 for each occurrence as no matching has occurred yet.

Each mapper will then take a line from “crime-stories.txt” as input and use the Match(text-line, text-pattern)” function for each key present in the patterns map. If Match(text-line, text-pattern)” returns true, the value of the key in patterns will be incremented by 1. In which summing the frequency of each pattern occurrence.

```
//String line = value.toString
//Patterns Map <String, Interger> for each line in “patterns.txt”
//for each entry/key in Patterns Map
        //if match(line,key) == True
                //context.write(key,++key.getValue())
```

This will create and output of the matching pattern and its count/frequency like an example below:

```
{
  “Lorem ipsum”:2,
   “dolor rem fugiat” :1,
    Etc…. : x…k
}
```

Reduce Phase:

Each reducer will sum the frequencies of each matching string from patterns.txt which can be optimized by running a combiner that sums the frequency of the strings from the map output. If the sum/frequency is 0, it will not be written to the output file to reduce on time and space complexity.

```
//sum = 0
//for each value in each matching string
        //sum += value.get()
//if sum > 0
        //context.write(key,sum)
```

This will sum sample maps from:

```
{
  Lorem ipsum: 2 ,
  dolor rem fugiat: 1
}

{
  sit modi Magni asperiores: 1,
  Lorem ipsum: 1
}

{
  qui, sit temporibus : 1,
  enim tenetur laborum : 0
}
```

Into 1 single map/output as such:

```
{
          Lorem ipsum: 3,
         dolor rem fugiat: 1,
         sit modi Magni asperiores: 1,
         qui, sit temporibus : 1
}
```