Vivian Chen, Ryan Liang, Hiren Patel
03/20/2020
DS4300: Large Scale Storage & Retrieval

Homework 4: Learning Scala
https://github.com/ryanl35/DS4300_homework/tree/master/Assignment4

Contributions
Note: We all worked together on this assignment so contribution was equally spread.
Vivian Chen: Worked on analysis for the first problem and contributed to understanding of Scala. Fixed up formatting and edited final doc.
Ryan Liang: Worked through the code blocks on the first problem and contributed to working through each problem as a group. Contributed syntax knowledge of Scala and connecting to IntelliJ and Java.
Hiren Patel: Worked with the team to understand the code and create a plan to solve each question. Worked with the team to develop and test our code.

Part A. Scala Warm-up

```
val nulls = Array[Int](0, 0, 0, 0)
for (line <- Source.fromFile(filename).getLines) {
   val toks = line.split(",", -1)
   for (i <- 0 until toks.length)
      if (toks(i) == "") nulls(i) = nulls(i) + 1}
println(nulls.mkString(","))
```

In this code, we have a value named ***nulls*** which is initialized to be an Array of Ints, all valued at 0 currently. Then, we enter a for-each loop that iterates through each line in the source file by using the getLines() function. In each line, we initialize a value named ***toks*** to be the current line split by comma-delimiters, the -1 meaning that the Source will be split as many times as possible and the array can have any length. Now, we have an array of values from the current line that we obtained as a result of splitting the line by a comma. Next, we have a counted-for loop that runs until the end of (the length of the array of) all of the new values. For each of these new values, the code checks if the value at the given index of the array is a "null", or empty String, and if it is, then make the value at the current index of ***nulls*** (the Array of Ints) to be the current value plus 1, meaning that there is a null value at this current index.

```
val nulls = Source.fromFile(filename).getLines
   .map(_.split(",", -1)).map(a => a.map(z => if (z == "") 1 else 0))
   .reduce((x, y) => (x zip y).map { case (u, v) => u + v })
println(nulls.mkString(","))
```

This code is more connected and concise. All necessary functions performed are performed in the same line, and there is no dummy Array created that is an Array of empty Ints (valued at 0). Rather, we obtain the Array we need in the same place where we initialize the variable ***nulls***. This is done by essentially eliminating all of the for loops used in the first code and replacing them with map(), which is a more comprehensive way of applying a function to multiple items through an iteration. First, we retrieve each line by using the function getLines(). Then, we use map() to perform the split() function on each line to split each line by comma-delimiters. We then use map() again on each resulting line that has been split by comma and pass in a function that determines if the value at each index is an empty String or not, and if it is add a 1 at the given index, and if not then add a 0. After that, we use the reduce() function, pass in two arrays, with one array

being the array of lines and the other array being the array of ints in each line representing whether the value at each index was an empty string or not. Then, the case aggregates the number of nulls in each line, zips it up with the line, and then reduces everything to one array of ints that count how many nulls were in each line. The second function achieves the same thing because it uses the case class to add up the number of null values using pattern matching.

Program Outputs

1. shortestPath in Graph
   println(g.shortestPath("x", "y")) → ListBuffer(x, j, r, y)
   println(g.shortestPath("b", "f")) → ListBuffer(b, f)
   println(g.shortestPath("c", "y")) → ListBuffer(c, r, y)
   println(g.shortestPath("r", "r")) → ListBuffer(r)
   println(g.shortestPath("x", "Not a Node")) → ListBuffer(No path found.)

2. moved function in Partition
   var pc = new PartitionClass()
   println(pc.moved(1000000, 100, 107)) → 0.0094

3. weight in Binary
   println(toBinary(37, 8)) → 00100101
   println(toBinary(1234567890, 32)) → 01001001100101100000001011010010
   println(weight("00100101")) → 3
   println(weight("01001001100101100000001011010010")) → 12