Assignment 6

Name: Winton Gee

Q1[10] Write a program that finds the names of the students that have taken at least one of the courses with the greatest difficulty. In our example, John has taken such a course.

```
def firstPart(): Unit = {
  val conf = new SparkConf().setAppName("AppName").setMaster("local")
  val sc = new SparkContext(conf)
  val studentsRDD = sc.textFile(students).map(_.split(", "))
  val coursesRDD = sc.textFile(courses).map(_.split(", "))
  val gradesRDD = sc.textFile(grades).map(_.split(", "))

  // - Find the most difficult course (2nd file) : MOST_DIF_CLASS
  val maxDifficulty = coursesRDD.map {
    case Array(courseNum, difficulty) => difficulty.toInt
  }.max()

  val coursesKeyValue = coursesRDD.map {
    case Array(course, difficulty) => (course, difficulty.toInt)
  }

  val studentsKeyValue = studentsRDD.map {
    case Array(studentId, name, address, phoneNumber) => (studentId, name)
  }

  gradesRDD.map {
    case Array(studentId, course, grade) => (course, studentId)
  }
    // - Check which student has taken max difficulty (3rd file)
    .join(coursesKeyValue)
    .map {
      case (course, (studentId, difficulty)) => (studentId, (course, difficulty))
    }
    .filter {
      case (studentId, (course, difficulty)) => difficulty == maxDifficulty
    }
    // - Get the names of students in most difficult courses (1st file)
    .join(studentsKeyValue)
    .map {
      case (studentId, ((course, difficulty), name)) => name
    }
    .foreach(println)
 }
```

Q2[10] Write a program that prints the average course difficulty of the classes that are taken by each student.

```
def secondPart(): Unit = {
  val conf = new SparkConf().setAppName("AppName").setMaster("local")
  val sc = new SparkContext(conf)
  val studentsRDD = sc.textFile(students).map(_.split(", "))
  val coursesRDD = sc.textFile(courses).map(_.split(", "))
  val gradesRDD = sc.textFile(grades).map(_.split(", "))

  val coursesKeyValue = coursesRDD.map {
    case Array(course, difficulty) => (course, difficulty.toInt)
  }

  val studentsKeyValue = studentsRDD.map {
    case Array(studentId, name, address, phoneNumber) => (studentId, name)
  }

  // - Join grades with courses to figure out the average difficulty of courses student takes
  // - convert course to difficulty, group by student, find average
  gradesRDD.map {
    case Array(studentId, course, grade) => (course, studentId)
  }
    .join(coursesKeyValue)
    .map {
      // Note: 1 Is used as a Count
      case (course, (studentId, difficulty)) => (studentId, (difficulty, 1))
    }
    // Key -> (Sum Difficulty, Sum Count)
    .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
    .map {
      case (studentId, (difficulty, count)) => (studentId, difficulty.toDouble / count)
    }
    // Outer Join to get the names and include students without courses
    .rightOuterJoin(studentsKeyValue)
    .map {
      case (studentId, (averageDifficulty, name)) => name + ", " + averageDifficulty.getOrElse(0)
    }
    .foreach(println)
}
```

Q3[10]. Write a program that prints the top five most difficult classes.

```
def thirdPart(): Unit = {
  val conf = new SparkConf().setAppName("AppName").setMaster("local")
  val sc = new SparkContext(conf)
  val coursesRDD = sc.textFile(courses).map(_.split(", "))

  coursesRDD.map {
    case Array(course, difficulty) => (difficulty.toInt, course)
  }
    .sortByKey(ascending = false) // Higher Num = Higher Difficulty
    .take(5)
    .foreach(println)
}
```

Q4[10]. Write a program that prints the name of the students ordered by GPA in descending order (starting with the student with the highest GPA). The GPA of a student that has taken no courses should be 0 (use left outer or right outer join).

```
def fourthPart(): Unit = {
  val conf = new SparkConf().setAppName("AppName").setMaster("local")
  val sc = new SparkContext(conf)
  val studentsRDD = sc.textFile(students).map(_.split(", "))
  val gradesRDD = sc.textFile(grades).map(_.split(", "))

  val studentsKeyValue = studentsRDD.map {
    case Array(studentId, name, address, phoneNumber) => (studentId, name)
  }

  gradesRDD.map {
    // Key=StudentId : Value=(Grade, Count)
    case Array(studentId, course, grade) => (studentId, (getGrade(grade), 1))
  }
    // Key=StudentId : Value=(Sum Grade, Sum Count)
    .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
    // Calculate GPA
    .map {
      case (studentId, (grade, count)) => (studentId, grade / count)
    }
    .rightOuterJoin(studentsKeyValue)
    .map {
      case (studentId, (averageGrade, name)) => (averageGrade.getOrElse(0.0), name)
    }
    .sortByKey(ascending = false)
    .foreach(println)
}

def getGrade(grade: String): Double = grade match {
  case "A" => 4.0
  case "B" => 3.0
  case "C" => 2.0
  case "D" => 1.0
  case _ => 0.0
}
```