assignment6

December 3, 2024

1 Assignment 6 - Ishaan Sathaye

Solve the following problems using Spark/Scala. Do not create any map data structures. Try to avoid using traditional loops when possible.

Consider three files. One file has information about students (ID, name, address, phone number):

```
1, John, 123 Main, 233 223 5566 // is an example
```

Consider a second file that has information about courses and their difficulty:

```
CSC365, 1
CSC369, 1
CSC469, 2
// is an example
```

Consider a third file that contains the student ID, course, and grade. For example:

```
1, CSC365, A
1, CSC369, A
1, CSC469, B
```

It contains information about the student taking a class and earning a grade.

1. 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.

Solution:

```scala
object App {
    def main(args: Array[String]): Unit = {
        Logger.getLogger("org").setLevel(Level.OFF)
        Logger.getLogger("akka").setLevel(Level.OFF)

        val conf = new SparkConf().setAppName("assignment6").setMaster("local[*]")
        val sc = new SparkContext(conf)

        val coursesText = sc.textFile("/user/isathaye/input/courses.txt").persist()
        val studentsText = sc.textFile("/user/isathaye/input/students.txt").persist()
        val gradesText = sc.textFile("/user/isathaye/input/grades.txt").persist()

        val courses = coursesText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), cols(1).trim().toInt)
        }).persist()
        val students = studentsText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), (cols(1), cols(2), cols(3)))
        }).persist()
        val grades = gradesText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), (cols(1).trim(), cols(2).trim()))
        }).persist()

        // Find the courses with the greatest difficulty level
        // course, difficulty
        val maxDifficulty = courses.map(_._2).max()
        val hardestCourses = courses.filter(_._2 == maxDifficulty)

        // Join students with grades
        val studentGrades = students.join(grades).map {
            case (studentID, ((name, address, phone), (course, grade))) =>
            (course, (studentID, name, address, phone, grade))
        }

        // Filter students that have taken at least one of the hard courses
        val studentsTakingHardestCourses = studentGrades.join(hardestCourses).map {
            case (course, ((studentID, name, address, phone, grade), difficulty)) =>
            (studentID, (name))
        }.distinct()

        // Print only the name
        studentsTakingHardestCourses.map(_._2).collect().foreach(println)

        sc.stop()
    }
}
```

2. Write a program that prints the average course difficulty of the classes that are taken by each student. For example, the program will print **John, 1.33**. Make sure to print average course difficulty as 0 if the student doesn't take any classes (hint, use left outer or right outer join).

Solution:

```
object App {
    def main(args: Array[String]): Unit = {
        Logger.getLogger("org").setLevel(Level.OFF)
        Logger.getLogger("akka").setLevel(Level.OFF)

        val conf = new SparkConf().setAppName("assignment6").setMaster("local[*]")
        val sc = new SparkContext(conf)
        val coursesText = sc.textFile("/user/isathaye/input/courses.txt").persist()
        val studentsText = sc.textFile("/user/isathaye/input/students.txt").persist()
        val gradesText = sc.textFile("/user/isathaye/input/grades.txt").persist()

        val courses = coursesText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), cols(1).trim().toInt)
        }).persist()
        val students = studentsText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), (cols(1), cols(2), cols(3)))
        }).persist()
        val grades = gradesText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), (cols(1).trim(), cols(2).trim()))
        }).persist()

        // Rearrange grades to have course as key
        val gradesCourse = grades.map({
            case (studentId, (course, grade)) => (course, (studentId, grade))
        })

        // Do a left outer join on gradesCourse and courses
        val courseDifficulty = gradesCourse.leftOuterJoin(courses).map({
            case (course, ((studentId, grade), Some(difficulty))) => (studentId, difficulty)
            case (course, ((studentId, grade), None)) => (studentId, 0)
        })

        // Group by studentId and calculate average difficulty
        val studentDifficulty = courseDifficulty.groupByKey().map({
            case (studentId, difficulty) => {
                val avg = difficulty.sum / difficulty.size.toDouble
                (studentId, avg)
            }
        })

        // Join with students to get student name
        val studentDifficultyWithName = studentDifficulty.join(students).map({
            case (studentId, (avg, (name, _, _))) => (name, avg)
        })

        studentDifficultyWithName.collect().foreach(println)

        sc.stop()
    }
}
```

3. Write a program that prints the top 5 most difficult classes.

Solution:

```scala
object App {
    def main(args: Array[String]): Unit = {
        Logger.getLogger("org").setLevel(Level.OFF)
        Logger.getLogger("akka").setLevel(Level.OFF)

        val conf = new SparkConf().setAppName("assignment6").setMaster("local[*]")
        val sc = new SparkContext(conf)

        val coursesText = sc.textFile("/user/isathaye/input/courses.txt").persist()
        val studentsText = sc.textFile("/user/isathaye/input/students.txt").persist()
        val gradesText = sc.textFile("/user/isathaye/input/grades.txt").persist()

        val courses = coursesText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), cols(1).trim().toInt)
        }).persist()
        val students = studentsText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), (cols(1), cols(2), cols(3)))
        }).persist()
        val grades = gradesText.map(line => {
            val cols = line.split(",")
            (cols(0).trim(), (cols(1).trim(), cols(2).trim()))
        }).persist()

        val top5 = courses.map(x => (x._2, x._1)).sortByKey(false).take(5)
        println("Top 5 most difficult classes:")
        top5.map(x => println(x._2))

        sc.stop()
    }
}
```

4. 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).

Solution:

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

```
def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.OFF)
    Logger.getLogger("akka").setLevel(Level.OFF)

    val conf = new SparkConf().setAppName("assignment6").setMaster("local[*]")
    val sc = new SparkContext(conf)

    val coursesText = sc.textFile("/user/isathaye/input/courses.txt").persist()
    val studentsText = sc.textFile("/user/isathaye/input/students.txt").persist()
    val gradesText = sc.textFile("/user/isathaye/input/grades.txt").persist()

    val courses = coursesText.map(line => {
        val cols = line.split(",")
        (cols(0).trim(), cols(1).trim().toInt)
    }).persist()
    val students = studentsText.map(line => {
        val cols = line.split(",")
        (cols(0).trim(), (cols(1), cols(2), cols(3)))
    }).persist()
    val grades = gradesText.map(line => {
        val cols = line.split(",")
        (cols(0).trim(), (cols(1).trim(), cols(2).trim()))
    }).persist()

    // Convert courses to key value pair and replace grade with GPA num
    val gpaGrades = grades.map(x => (x._1, (x._2._1, convertGPA(x._2._2))))

    // Join studentGPA with students and group by student ID
    val studentGPA = students.leftOuterJoin(gpaGrades).map({
        case (id, ((name, address, phone), Some((course, gpa)))) => (id, (name, address, p
        case (id, ((name, address, phone), None)) => (id, (name, address, phone, 0.0))
    })

    // Group then aggregate by student ID and calculate average GPA
    val studentGrouped = studentGPA.groupByKey().mapValues(values => {
        val gpaList = values.map(_._4)
        val gpaSum = gpaList.sum
        val gpaCount = gpaList.size
        val gpaAvg = gpaSum / gpaCount
        gpaAvg
    })

    // Join studentGrouped with students and sort by GPA in descending order
    val studentGPAOrdered = studentGrouped.join(students).sortBy(_._2._1, false)

    // Print student name, gpa
    studentGPAOrdered.collect().foreach(x => println(x._2._2._1 + ", " + x._2._1))
```

```
        sc.stop()
    }
}
```