Bevin Tang
1/29/18

CSC 369 -- Assignment 2

Consider an input file with the following example input:

John Back, 23, B, CSC366
Bob Wilson, 11, B, CS201
John Back, 23, A, CSC369

In general, the input file will contain the student name, the student ID number, grade, and course. You need to write a Map/Reduce program that prints the student name, student id, and the list of classes for that student. The output should be **sorted by** name and then sorted by grade for each name. Here is example output.

Bob Wilson, 11, (B, CS201)
John Back, 23, (A, CSC369), (B, CSC366) **// sorted by name and then by grade**

a) What is the natural and the composite key?

Natural: <Student Name>
Composite: <Student Name, Student ID, Grade-Class Pair>

b) Show the composite key class (must implement WritableComparable and have compareTo method)

```
public class StudentKey implements Writable,
      WritableComparable<StudentKey> {

    private final Text name = new Text();
    private final IntWritable id = new IntWritable();
    private final Text grade = new Text();

    public StudentKey() {}

    public StudentKey(String name, int id, String grade,
      String className) {
        this.name.set(name);
        this.id.set(id);

        // create format (grade, class)
        String gradeClass = "(" + grade + ", " + className + ")";
        this.grade.set(grade);
    }

    public Text getName(){
        return name;
    }

    public IntWritable getId(){
        return id;
    }

    public Text getGrade(){
        return grade;
    }

    @Override
    public void write(DataOutput out) throws IOException {
        name.write(out);
        id.write(out);
        grade.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        name.readFields(in);
        id.readFields(in);
        grade.readFields(in);
    }

    @Override
    public int compareTo(StudentKey other) {
        int nameComp = name.compareTo(other.name);
        // If same name, compare grades
        if (nameComp == 0)
            return grade.compareTo(other.grade);
        return nameComp;    // Otherwise, compare names
    }
}
```

c) Show the mapper class

```
public class StudentMapper extends Mapper <LongWritable, Text,
      StudentKey, Text> {

    public void map(LongWritable key, Text value, Context context) throws
            IOException, InterruptedException {

        String line = value.toString();
        String[] tokens = line.split(",");.trim();
        if (tokens.length < 4)
            return;

        // Grab information from line
        String name = tokens[0];
        int id = Integer.parseInt(tokens[1]);
        String grade = tokens[2];
        String className = tokens[3];
        StudentKey stuKey = new StudentKey(name, id, grade, className);

        // Output: <CompositeKey, (grade, class)>
        context.write(stuKey, stuKey.getGrade());
    }
}
```

d) Show the partitioner class

```
public class StudentPartitioner extends
      Partitioner<StudentKey, IntWritable> {

    @Override
    public int getPartition(StudentKey stuKey, Text stuGrade,
       int numPartitions) {

        // Partition based on Student Name
        return Math.abs(stuKey.getName().hashCode % numPartitions);
    }
}
```

e) Show the group comparator class

```
public class StudentGroupComparator extends WritableComparator {
    public StudentGroupComparator() {
        super(StudentKey.class, true);
    }

    @Override
    public int compare(WritableComparable o1, WritableComparable o2) {
        StudentKey stu1 = (StudentKey) o1;
        StudentKey stu2 = (StudentKey) o2;

        // Group nodes based on Student Name
        return o1.getName().compareTo(o2.getName);
    }
}
```

f) Show the reducer class

```
public class StudentReducer extends Reducer <StudentKey, Text,
      Text, Text> {

    @Override
    protected void reduce (StudentKey stukey, Iterable<Text> grades,
            Context context) throws IOException, InterruptException {

        // Make key include both name and id
        String key = stukey.getName() + ", " + stukey.getId() + ", ";

        // Construct value associated with key
        String result = "";
        for (Text grade : grades) {
            result += grade + ", ";
        }
        // Remove ", " from the end
        result = result.substring(0, result.length-2);

        // Output in format: student, ID, (grade, class), ...
        context.write(key, new Text(result));
    }
}
```