assignment2

October 15, 2024

1 Assignment 2 - Ishaan Sathaye

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 an example output.

Bob Wilson, 11, (B, CS201)

John Back, 23, (A, CSC369), (B, CSC366) // sorted by name and then by grade for each name

a. What is the natural and the composite key?

- The natural key is the student name and the composite key is the student name, student id, and a pair of grade and course.

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

```java
public class CompositeKey implements Writeable, WritableComparable<CompositeKey> {

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

    public CompositeKey() {}

    public CompositeKey(String name, int id, String grade, String course) {
        this.name.set(name);
        this.id.set(id);
        this.grade.set(grade);
        this.course.set(course);
    }
    public Text getName() {
        return name;
    }

    public IntWritable getId() {
        return id;
    }

    public Text getGrade() {
        return grade;
    }

    public Text getCourse() {
        return course;
    }

    public Text getCourseGradePair() {
        return new Text("(" + grade + ", " + course + ")");
    }

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

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

    @Override
    public int compareTo(CompositeKey other) {
        int cmp = name.compareTo(other.name);
        if (cmp == 0) {
            // if names are same then compare by grade
            return grade.compareTo(other.grade);
        }
        // if names are different then compare by name
        return cmp;
    }
}
```

c. Show the mapper class.

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

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, Interru
        String line = value.toString();
        String[] tokens = line.split(",");
        if (tokens.length != 4) {
            return;
        }
        String name = tokens[0].trim();
        int id = Integer.parseInt(tokens[1].trim());
        String grade = tokens[2].trim();
        String course = tokens[3].trim();
        CompositeKey compositeKey = new CompositeKey(name, id, grade, course);

        context.write(compositeKey, new Text(compositeKey.getCourseGradePair()));
}
```

d. Show the partitioner class.

```java
public class StudentPartitioner extends Partitioner<CompositeKey, Text> {

    @Override
    public int getPartition(CompositeKey key, Text value, int numPartitions) {
        return Math.abs(key.getName().hashCode() % numPartitions);
    }
}
```

e. Shows the group comparator class.

```java
public class GroupingComparator extends WritableComparator {
    protected GroupingComparator() {
        super(CompositeKey.class, true);
    }

    @Override
    public int compare(WritableComparable w1, WritableComparable w2) {
        CompositeKey key1 = (CompositeKey) w1;
        CompositeKey key2 = (CompositeKey) w2;
        int nameComparison = key1.getName().compareTo(key2.getName());
        if (nameComparison != 0) {
            return nameComparison;
        }
        // if names are same then compare by id
        return Integer.compare(key1.getId().get(), key2.getId().get());
    }
}
```

f. Show the reducer class.

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

    @Override
    public void reduce(CompositeKey key, Iterable<Text> values, Context context) throws IOExce
        String finalKey = key.getName() + ", " + key.getId() + ", ";
        String res = "";
        for (Text value : values) {
            res += value.toString() + ", ";
        }
        // remove the last comma
        res = res.substring(0, res.length() - 2);
        context.write(finalKey, new Text(res));
}
```