

## CSC 369 – Assignment 3

### Question 1:

Consider two files:

One file has information about students (ID, name, address, phone number, courses taken).

*1, John, 123 Main 233 223 5566, (CSC365, CSC369, CSC469)*

The second file has information about courses and their difficulty.

*(CSC365, 1)*

*(CSC369, 1)*

*(CSC469, 2)*

Your goal is to print the name and addresses of students that have taken all the **top N** most difficult classes.

### Approach:

- 1) Implement a Map/Reduce job to associate each student with ONE class and that class's difficulty

- a) Use two mappers: (1) Student Mapper, (2) Class Difficulty Mapper:

- i. Student Mapper will output multiple pairs (1 for each class taken). This only applies if the student has taken all 3 classes.

```
if (!takenAllClasses)
    return;
for (i=0; i < numClasses; i++){
    context.write(className, (studentEntry-classes) +
        classes[i]);
}
```

- ii. Since we want to preserve the student data, the output of the Mapper will have a <key, value> pair of <Text className, Text allData> so that we can use all the information.
    - iii. The Class Difficulty Mapper will include "1" in the value so that it gets to the reducer before the Student information (which will have a "2").
    - iv. The phone number will be omitted from the output

- b) Use a partitioner & grouper to bring records with the same className to the same reducer node

- c) Use reducer to combine the records:

```
// The first record is guaranteed to be from the class file
if (!diffValueSet)
    diffValue = classDifficulty // this would be like tokens[1]

for (value : values){
    if (class record)
        skip
    else
        context.write(null, studentEntry + diffValue)
}
```

This leaves us with a student entry where the last entry is the class rating:

*1, John, 123 Main, CSC365, 1*

- 2) Implement a simpler second job to sum the ratings of each student
  - a) Map students by name
    - i. <Key, Value> = <Text stuName, Text allData>
  - b) Partition students by name only
  - c) Group students by name
  - d) Reducer sums the difficulty ratings of each student
 

```
for (value : values)
    sum += classDifficulty;
result = name + address + sum;
context.write(null, result);
```
  - e) Output shares similarities with former output, except there will be 1 record per student with an accumulated class rating and the class names will be omitted:
 

*1, John, 123 Main, 12*
- 3) Implement a third job to output the students taking the top N difficult classes
  - a) Implement Mapper with a TreeSet containing custom class **Records**.
    - i. This involves making a custom compareTo() for the **Record** class
    - ii. Override the setup method in the mapper to take N
    - iii. Override a cleanup method to actually write the records
  - b) Reducer makes its own TreeSet and then prints the top N
    - i. Make setup method to get N
    - ii. Write from the top of the TreeSet (to print in proper order)

#### Custom Classes –

- 1) Record Class: holds name, address, and difficulty field. Used for Tree Set in 3<sup>rd</sup> job.

```
// compareTo() logic
result = this.difficulty - other.difficulty
if (result == 0){
    result = this.name.compareTo(other.name)
}
if (result == 0){
    result = this.id - other.id
}
return result
```

**Question 2:**

Consider an input file that has information about students (ID, name, address, phone number, course taken, grade).

*1, John, 123 Main, 233 223 5566, ((CSC465 A) (CSC369 A) (CSC469 B))*

The problem is to **print the N students with the highest GPA**. You can assume that you get 4 points for A, 3 points for B, 2 points for C, 1 point for D, and 0 points for F. The average GPA will be a real number between 0 and 4.

**Approach:**

- 1) First job will associate each student with a GPA.
  - a) For each student, the Mapper will output a <key, value> pair for each class
    - i. <key, value> = <Text studentName, Text ID+className>
  - b) Partition and Group each student by their name
  - c) Reducer will take the average of the class grades and write them
 

```
double sum = 0;
int numClasses = 0;
```

```
for (record : records){
    if (grade == 'A')
        sum += 4;
    else if (grade == 'B')
        sum += 3;
    ...
}
context.write(ID+name, GPA);
```

- d) Output will be comprised of the student and their GPA:
 

*1, John, 3.2*

- 2) Using the new output, the next job will output the top N students based on their GPA.
  - a) Mapper will create a TreeSet to hold top N number of students
    - i. Implement a custom class **Student** to hold id, name, and GPA.
    - ii. Override setup and cleanup for extraction of N and context write
  - b) Reducer will write top N students
    - i. Has its own TreeSet<Student>
    - ii. Override setup to extract N
    - iii. Write N entries from the top of TreeSet.

- c) Output will be in the form:
 

*1, John, 3.2*

**Custom Classes –**

- 1) Student Class: holds ID, student name, and GPA. Used for TreeSet ordering
 

```
// compareTo() logic
result = this.gpa - other.gpa
if (result == 0)
    result = this.name.compareTo(other.name)
if (result == 0)
    result = this.id - other.id
return result
```

**Question 3:**

Consider an input file containing student ID and course name:

*1, CSC354 => John is enrolled in CSC354*

The problem is to **print the top N most popular classes** (i.e. classes with the highest enrollment).

**Approach:**

- 1) First job creates an output of class and number enrolled
  - a) Mapper outputs: <key, value> = <className, NullWritable>
    - i. **Assumes there are no duplicate entries.**
  - b) Partitioner and Group comparator make sure that entries with the same class name go to the same reducer node.
  - c) Reducer counts the number of entries and writes the class and its popularity
  - d) Output comprised of className followed by number of enrolled:
 

CSC 369, 32
- 2) Second job prints the top N popular classes
  - a) Mapper tracks the top N **Classes** and outputs the top N classes
    - i. Override setup method to obtain value for N
    - ii. Override cleanup method to do the actual writing
    - iii. Output: <key, value> = <NullWritable, Class>
  - b) Reducer tracks the top N **Classes** as well, and writes the top N
    - i. Override setup method to extract value for N.
  - c) Output is just the class name:
 

CSC 369

CSC 469