Part(a)

Suppose our input data is:

117	51	1
194	51	1
299	51	3
230	151	51
194	151	79
51	130	10

At first I used map() method to implement the mapper. In this method, one line of string is split into three parts: source, target, weight. The source is ignored because we only care about the inbound edges. The target and weight are recorded as a key and a value, respectively. Suppose this dataset isn't divided into multiple parts. The mapper emits: <51,1>, <51,1>, <51,3>, <151,51>, <151,79>, <130,10>.

I wrote the reduce() method to implement the reducer. For a given key (e.g., 51), it compared all the corresponding values (i.e., 1, 1, 3) and selected the largest one as the final value. The output of the reducer is:

51	3
151	79
130	10

Part(b)

Our input dataset is:

Student, Alice, 1234
Student, Bob, 1234
Department, 1123, CSE
Department, 1234, CS
Student, Joe, 1123

First I wrote a class that implemented the WritableComparable interface that will be used to wrap our key:

```
public class TaggedKey implements Writable, WritableComparable<TaggedKey> {
    private Text joinKey = new Text();
    private IntWritable tag = new IntWritable(); //tag=1 if the value has "Department" and 2 otherwise
    public int compareTo(TaggedKey taggedKey) {
        int compareValue = this.joinKey.compareTo(taggedKey.getJoinKey());
        if(compareValue == 0 ){
            compareValue = this.tag.compareTo(taggedKey.getTag());
        }
       return compareValue;
    }
    //Details like setter and getter methods left out for clarity
 }
```

I divided the dataset into two parts based on the first column (i.e., Student or Department). The records starting with "Department" has a tag=1 and the records starting with "Student" has a tag=2. The joinkey is Department_ID. The compareTo() method ensures that the output records are sorted in ascending order of Department_ID. Keys with the same joinKey value will have a secondary sort on the value of the tag field. For a given key, the first value is the deparment name, followed by the students information.

Then I used a partitioner that only considered the join key when determining which reducer the composite key and data were sent to:

```
public class TaggedJoiningPartitioner extends Partitioner<TaggedKey,Text> {
    public int getPartition(TaggedKey taggedKey, Text text, int numPartitions) {
        return taggedKey.getJoinKey().hashCode() % numPartitions;
    }
}
```

The map method is presented below.

```
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    Text data = new Text()
    TaggedKey taggedKey = new TaggedKey();
    String[] valueItems = value.toString().split(",");  //split one line into three words
    if(valueItems[0].equals("Department")){
        taggedKey.setTag(1);
        taggedKey.setJoinKey(valueItems[2]);
        data.set(valueItems[1]);
    }else{
        taggedKey.setTag(2);
        taggedKey.setJoinKey(valueItems[1]);
        data.set(valueItems[2]);
    }
    context.write(taggedKey, data);
}
```

This method splits one line of the dataset into three words and set a tag based on the first word of this record. If it's "Department", then tag=1, the joinKey is the second word and the value is the third word. If it's "Student", then tag=2, the joinKey is the third word and the value is the second word.
The output of the mapper is:
<1123, CSE>, <1123, Joe>
<1234, CS>, <1234, Alice>, <1234, Bob>

Then data is joined in the reducer:

```
protected void reduce(TaggedKey key, Iterable<Text> values, Context context) throws IOException{
    NullWritable nullKey = NullWritable.get();
    String departName;
    Text joinedText = new Text();
    int i=0;
    for (Text value : values) {
        if(i==0){
            departName=value.toString();
        }else{
            joinedText.set(key.getJoinKey()+","+value.toString()+","+departName);
            context.write(nullKey, joinedText);
        }
        i++;
    }
}
```

For each key, this reducer records the first value as the department name. For each of the remaining values of this key (i.e., student names), the reducer writes one record. The output format is <department_id, student_name, department_name>. In this case, the output is:

```
1123,Joe,CSE
1234,Alice,CS
1234,Bob,CS
```