(a) Map function (MAPPER)

The input of mapper is the (key, value) pairs supplied by the Record Reader. The keys are generated based on bit offset and values is the string value of each lines of the input file.

The output of mapper is another (key, value) pair wherein the key is the *src* and value is *weight*. The algorithm would be to tokenize the input value string and store the first token as output key and last token as output value. The Java code of this mapper is as follows:

```
public static class TokenizerMapper
   extends Mapper<LongWritable, Text, Text, IntWritable>{


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


           String line = value.toString();
    String lasttoken = null;
    StringTokenizer s = new StringTokenizer(line,"\t");
    String node = s.nextToken();


    while(s.hasMoreTokens())
    {
      lasttoken=s.nextToken();
    }


    int weight = Integer.parseInt(lasttoken);
    context.write(new Text(node),  new IntWritable(weight));
  }
 }
```

The combiner takes the output of mapper as an input and group them by key and sort the out by keys.

Reduce function (REDUCER)

The input of reducer is the (key, value) pairs supplied by combiner. The keys the output keys of mapper and also sorted and value is an array of all the output value of mapper associated with that particular key.

The output of reducer is another (key, value) pair wherein the key is the input key whereas value is the maximum value in input value array. The Java code of this reducer is as follows:

```
public static class IntSumReducer
   extends Reducer<Text,IntWritable,Text,IntWritable> {


 private IntWritable result = new IntWritable();
 public void reduce(Text key, Iterable<IntWritable> values,
                    Context context
                    ) throws IOException, InterruptedException {
  int max = 0;
  for (IntWritable val : values) {
    if(val.get()>max)
            max=val.get();
  }
  result.set(max);
  context.write(key, result);
 }
}
```

(b) The MapReduce algorithm can be given as follows:

Map function (MAPPER)

The input of mapper is the (key, value) pairs supplied by the Record Reader.

The output of mapper is another (key, value) pair wherein the key is the *src* and value is *tgt*. If we set the record reader input method as KeyValueTextInput format then we can have it as an *identity mapper*.

The combiner takes the output of mapper as an input and group them by key and sort the output by keys as well.

Reduce function (REDUCER)

The input of reducer is the (key, value) pairs supplied by combiner.

The output of reducer is another (key, value) pairs wherein the key represent a node and value represents the node's out neighbors' out neighbors. The pseudo code can be given as

---

```
Function Reducer(key,Value){
        For each V in Value{
                vList = V.values()              %%Obtain the list of values associated with the key V
                For each v in vList{
                        If  (v 'not equal' key)
                                Context.Write(key,v)  %%Store the output
                  }
           }
}
```

Consider the following toy graph:

Input of your algorithm:

src	*tgt*
4	3
1	2
2	3
4	2
2	1
3	2

The output of Mapper would be:

(4, 3) (1, 2) (2,3) (4, 2) (2, 1) (3, 2)

The output of combiner would be:

(1,[2]) (2,[1,3]) (3,[2]) (4, [2,3])

The output of reducer would be

1	3
3	1
4	1
4	3
4	2

Detailed Description:

Mapper is just and identity mapper and combiner is just groupByKey and sort by Key. We give details of reducer:

For key 1:

The Value=[2]

Iterating over this list we get

V=2

vList=[1,3]

Now iterating over this list we get

1 != 3 ➔ (1,3) as reducer output

For key 2:

The Value=[1,3]

Iterating over this list we get

V=1

vList=[2]

Now iterating over this list we get nothing

V=3

vList=[2]

Now iterating over this list we get nothing

Reducer output is nill for this

For key 3:

The Value=[2]

Iterating over this list we get

V=2

vList=[1,3]

Now iterating over this list we get

3 != 1 ➔ (3,1) as reducer output

For key 4:

The Value=[2,3]

Iterating over this list we get

V=2

vList=[1,3]

Now iterating over this list we get

4 != 1 ➔ (4,1) as reducer output

4 != 3 ➔ (4,3) as reducer output

V=3

vList=[2]

Now iterating over this list we get

4 != 2 ➔ (4,2) as reducer output