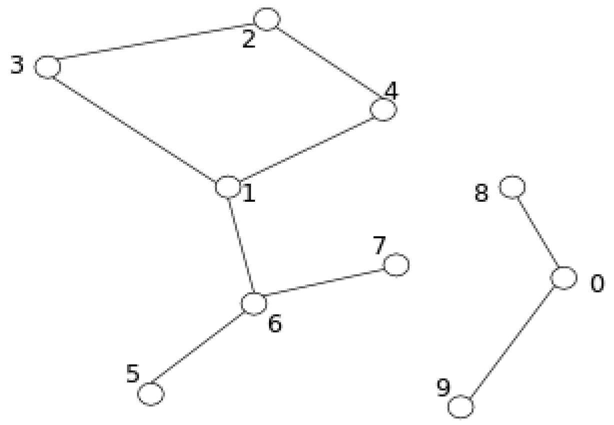
An undirected graph is represented in the input text file using one line per graph vertex. For example, the line

```
1,2,3,4,5,6,7
```

represents the vertex with ID 1, which is connected to the vertices with IDs 2, 3, 4, 5, 6, and 7. For example, the following graph:



is represented in the input file as follows:

```
3,2,1
2,4,3
1,3,4,6
5,6
6,5,7,1
0,8,9
4,2,1
8,0
9,0
7,6
```

The following pseudo-code finds the connected components. It assigns a unique group number to each vertex (we are using the vertex ID as the group number), and for each graph edge between Vi and Vj, it changes the group number of these vertices to the minimum group number of Vi and Vj. That way, vertices connected together will eventually get the same minimum group number, which is the minimum vertex ID among all vertices in the connected component. First you need a class to represent a vertex:

```
class Vertex extends Writable {
  short tag;                 // 0 for a graph vertex, 1 for a group number
  long group;                // the group where this vertex belongs to
  long VID;                  // the vertex ID
  Vector adjacent;      // the vertex neighbors
  ...
}
```

Class Vertex must have two constructors: Vertex(tag,group,VID,adjacent) and Vertex(tag,group).

First Map-Reduce job:

```
map ( key, line ) =
  parse the line to get the vertex VID and the adjacent vector
  emit( VID, new Vertex(0,VID,VID,adjacent) )
```

Second Map-Reduce job:

```
map ( key, vertex ) =
  emit( vertex.VID, vertex )   // pass the graph topology
  for n in vertex.adjacent:
     emit( n, new Vertex(1,vertex.group) )  // send the group # to the adjacent
vertices

reduce ( vid, values ) =
  m = Long.MAX_VALUE;
  for v in values {
     if v.tag == 0
        then adj = v.adjacent.clone()     // found the vertex with vid
     m = min(m,v.group)
  }
  emit( m, new Vertex(0,m,vid,adj) )      // new group #
```

Final Map-Reduce job:

```
map ( group, value ) =
   emit(group,1)

reduce ( group, values ) =
   m = 0
   for v in values
       m = m+v
   emit(group,m)
```

The second map-reduce job must be repeated multiple times. For your project, repeat it 5 times. You can repeat a job, by putting the job in a for-loop. The args vector in your main program has the path names: args[0] is the input graph, args[1] is the intermediate directory, and args[2] is the output. The first Map-Reduce job writes on the directory `args[1]+"/f0"`. The second Map-Reduce job reads from the directory `args[1]+"/f"+i` and writes in the directory `args[1]+"/f"+(i+1)`, where `i` is the for-loop index you use to repeat the second Map-Reduce job. The final Map-Reduce job reads from `args[1]+"/f5"` and writes on `args[2]`. The intermediate results between Map-Reduce jobs must be stored using SequenceFileOutputFormat.

To compile and run project1:

```
cd project3

mvn install

rm -rf output

~/hadoop-2.6.5/bin/hadoop jar target/*.jar Graph small-graph.txt intermediat output
```