```java
/* Implement a directed graph using linked list, and
 * compute the page rank after k iterations.
 *
 * Created on 12/05/2015
 * Author: Seung-A Jang
 *
 */

import java.util.Arrays;
import java.util.LinkedList;

class Pair implements Comparable<Pair>{
   private int index;
   private double value;

   public Pair(int i, double v){
        index = i;
        value = v;
   }

   @Override
   public int compareTo(Pair other) {
        return Double.valueOf(other.value).compareTo(this.value);
   }

   public String toString(){
        return index +" " + value;
   }

}

public class DirectedGraph {
   //array of a linkedlist that stores edge(v->u)
   private LinkedList<Integer>[] nodeList;

   // constructor initialize an directed graph, n is the number of nodes
   public DirectedGraph(int n){
        nodeList = (LinkedList<Integer>[]) new LinkedList[n];

        //initialize each node linkedlist to null
        for(int i = 0; i < nodeList.length; i++){
               nodeList[i] = new LinkedList<>();
        }
   }

   // check if the given node id is out of bounds
   private boolean outOfBounds(int nidx){
        return (nidx < 0 || nidx >= nodeList.length);
   }

   // set an edge (n1,n2)
   // beware of repeatingly setting a same edge and out-of-bound node ids
   public void setEdge(int n1, int n2){
        if(outOfBounds(n1) || outOfBounds(n2)) return;
        //Check if there's a repeating edge. If not, add it to the list.
        if(nodeList[n1].contains(n2))
               return;
        else
               nodeList[n1].add(n2);
   }
```

```
   // compute page rank after num_iters iterations
   // then print them in a monotonically decreasing order
   void computePageRank(int num_iters){
         double[] oldPR = new double[nodeList.length];     //values of old PageRank
         double[] newPR = new double[nodeList.length];     //values of new(updated) PageRank

         //array of outdegree of each node
         int[] outDegree = new int[nodeList.length];
         for(int i=0; i<outDegree.length; i++)
                outDegree[i] = nodeList[i].size();

         //set initial pagerank of each node to 1.
         for(int i = 0; i<oldPR.length; i++){
                oldPR[i] = 1;
         }

        //calculate k number of pagerank
         for(int k = 0; k < num_iters; k++){
                //nodes to be calculated for pagerank (i.e. old pagerank[] index)
                for(int p = 0; p < oldPR.length; p++){
                       //nodes from 1 to n-1
                       for(int n = 0; n < nodeList.length; n++){
                              //iterating element of each index(i.e. linkedlist)
                              for(int i = 0; i < nodeList[n].size(); i++){
                                     if(nodeList[n].get(i) == p)
                                            newPR[p] += oldPR[n]/outDegree[n];
                              }
                       }
                }

                oldPR = newPR;
                newPR = new double[nodeList.length];
         }

         //sort pagerank in descending order
         Pair[] sortedPR = new Pair[oldPR.length];
         for(int i = 0; i < oldPR.length; i++){
                sortedPR[i] = new Pair(i, oldPR[i]);
         }

         Arrays.sort(sortedPR);
         for(int i = 0; i < sortedPR.length;i++)
                System.out.println(sortedPR[i]);

   }//computePR

}//end
```