University of Washington
Department of Computer Science and Engineering
CSE 417, Winter 2020
Yiliang Wang

January 29, 2020

Homework 3 Problem 5

Programming Problem 5 (10 points):

Implement an algorithm for the Connected Components Problem for undirected graphs. The algorithm should take as input an undirected graph, and identify the connected components. The vertices should be labeled by which component they belong to. If there are d connected components, you should name the components $1, 2, \ldots, d$. You will need to determine the size of each connected component.

Using the random graph generator from problem 4, experiment with the the connected component structure of random graphs. For a fixed n, look at what happens as you vary the value of p. You should use a moderately large value of n, at least $n = 1,000$ or even $n = 10,000$. (You should not choose an n so large that it takes a long time to generate the graphs, or you run into into overflow issues.) The two outputs that will be interesting to look at are the number of connected components and the size of the connected components. The "interesting" values of p will be small, for example, with $n = 10,000$ the range of interest for p is $0.0002 \leq p \leq 0.001$.

Answer:

```
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

public class Graph{
    public int n;
    public int m;
    public double p;
    public HashMap<Integer, ArrayList<Integer>> vertexMap = new HashMap<>();
    public ArrayList<int[]> edgeList = new ArrayList<>();

    public Graph(int n, double p) {
        this.n = n;
        this.p = p;
        m = (int) (n * (n - 1) * p / 2.0);
        getRandomGraph();
```

```
        }

        void getRandomGraph(){
        Random rand = new Random();
        for (int i = 0; i < n - 1; i++)
            for (int j = i + 1; j < n; j++)
                if (rand.nextDouble() < p){
                    AddEdge(i, j);
                }
        }

    void AddEdge(int i, int j) {
        if(!vertexMap.containsKey(i)){
            vertexMap.put(i, new ArrayList<>());
        }
        vertexMap.get(i).add(j);

        if(!vertexMap.containsKey(j)){
            vertexMap.put(j, new ArrayList<>());
        }
        vertexMap.get(j).add(i);
        edgeList.add(new int[] {i, j});
        edgeList.add(new int[] {j, i});
    }
}


import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Stack;

class calConnectComponent{
    public static void main(String[] args){
        for(double i = 0.0002; i <= 0.001; i += 0.00005){
            int sumConnectCompSize = 0;
            for(int j = 0; j <= 10; j ++){
                Graph g = new Graph(10000, i);
                List<Object> ret = connectComponent(g);
                HashMap<Integer, Integer> connectComp =
                (HashMap<Integer, Integer>) ret.get(0);
                sumConnectCompSize += connectComp.size();
            }
            System.out.printf("p:%.5f,componentN:%.2f,avgSize:%.2f\n",
            i, sumConnectCompSize/10.0, 10000.0*10.0/sumConnectCompSize);
```

```
        }
    }

    static List<Object> connectComponent(Graph g){
        int[] visited = new int[g.n];
        int[] ccLabel = new int[g.n];
        Stack<Integer> s = new Stack<>();
        HashMap<Integer, Integer> connectComp = new HashMap<>();
        int counter = 0;

        for (int i = 0; i < g.n; i++) {
            if (visited[i] == 0) {
                s.push(i);
                counter ++ ;
            }
            while(!s.empty()){
                int cur = s.pop();
                ccLabel[cur] = counter;
                if (visited[cur] == 0) {
                    if (g.vertexMap.containsKey(cur)) {
                        for (int j : g.vertexMap.get(cur)) {
                            if (visited[j] == 0) {
                                s.push(j);
                            }
                        }
                    }
                    if (!connectComp.containsKey(counter))
                        connectComp.put(counter, 0);
                    connectComp.put(counter, connectComp.get(counter) + 1);
                    visited[cur] ++;
                }
            }
        }
        return Arrays.asList(connectComp, ccLabel);
    }
}
```

Analysis:

p: 0.00020, componentN: 1768.60, avgSize: 5.65
p: 0.00025, componentN: 1018.20, avgSize: 9.82
p: 0.00030, componentN: 597.70, avgSize: 16.73
p: 0.00035, componentN: 360.40, avgSize: 27.75
p: 0.00040, componentN: 209.90, avgSize: 47.64
p: 0.00045, componentN: 127.10, avgSize: 78.68
p: 0.00050, componentN: 75.10, avgSize: 133.16

p: 0.00055, componentN: 48.60, avgSize: 205.76
p: 0.00060, componentN: 28.80, avgSize: 347.22
p: 0.00065, componentN: 16.60, avgSize: 602.41
p: 0.00070, componentN: 11.40, avgSize: 877.19
p: 0.00075, componentN: 7.40, avgSize: 1351.35
p: 0.00080, componentN: 4.80, avgSize: 2083.33
p: 0.00085, componentN: 3.10, avgSize: 3225.81
p: 0.00090, componentN: 2.50, avgSize: 4000.00
p: 0.00095, componentN: 1.80, avgSize: 5555.56

The result show that with the increase of p, the total number of connect component decrease follows a exponetial function, and the average size of connect component increase follows a exponetial function. And as p approaching to 0, the average size of component is approaching to 1 and as p approaching to 0.0001 the number of connect components is approaching to 1.