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

January 29, 2020

Homework 3 Problem 4

Programming Problem 4 (10 points):

The purpose of this problem is to construct a random graph generator for use in other programming problems and to demonstrate an implementation of graphs using adjacency lists.. A random graph generator, given an input parameter n, picks "at random" a graph with n vertices. There are multiple different models of random graphs. We will consider the *edge density* model, where a parameter p gives the probability of each edge being present. This model is referred to as $\mathcal{G}_p^n$. The undirected random graph generation problem is: given an integer n and a real number p with $0 \leq p \leq 1$, construct an undirected graph on n vertices where each edge $\{u, v\}$ has probability p of being in the set of edges E, and the probability of each edge is independent. Write a generator for $\mathcal{G}_p^n$ which given inputs n and p constructs a random undirected graph in adjacency list representation.

For this problem, write the graph generator and a routine to print the edges and vertices of a graph. Print the results a creating two different random graphs using $n = 10$ and $p = 0.2$.

Answer:

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

public class Graph{
    public int n;
    private int m;
    private 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);
        //System.out.printf("n:%d,␣m:%d,␣p:%.4f\n", n, m, p);
        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});
    }

    public static void main(String[] args){
        Graph g1 = new Graph(10, 0.2);
        Graph g2 = new Graph(10, 0.2);
        System.out.println(”g1:␣” + g1.vertexMap);
        for (int[] edge : g1.edgeList){
            System.out.print(Arrays.toString(edge) + ’␣’);
        }
        System.out.println();
        System.out.println(”g2:␣” + g2.vertexMap);

        for (int[] edge : g2.edgeList){
            System.out.print(Arrays.toString(edge) + ’␣’);
        }
    }
}
```

```
g1: 0=[6, 9], 1=[7], 2=[6], 3=[4, 8], 4=[3, 9], 5=[9], 6=[0, 2, 7, 9], 7=[1, 6], 8=[3], 9=[0, 4, 5, 6]
[0,6][6,0][0,9][9,0][1,7][7,1][2,6][6,2][3,4][4,3][3,8][8,3][4,9][9,4][5,9][9,5][6,7][7,6][6,9][9,6]

g2: 0=[5], 1=[4, 5, 6], 2=[9], 3=[6], 4=[1, 8], 5=[0, 1], 6=[1, 3, 7], 7=[6, 8], 8=[4, 7], 9=[2]
[0,5][5,0][1,4][4,1][1,5][5,1][1,6][6,1][2,9][9,2][3,6][6,3][4,8][8,4][6,7][7,6][7,8][8,7]
```