Programming Assignment 1

Daniel Quiroga

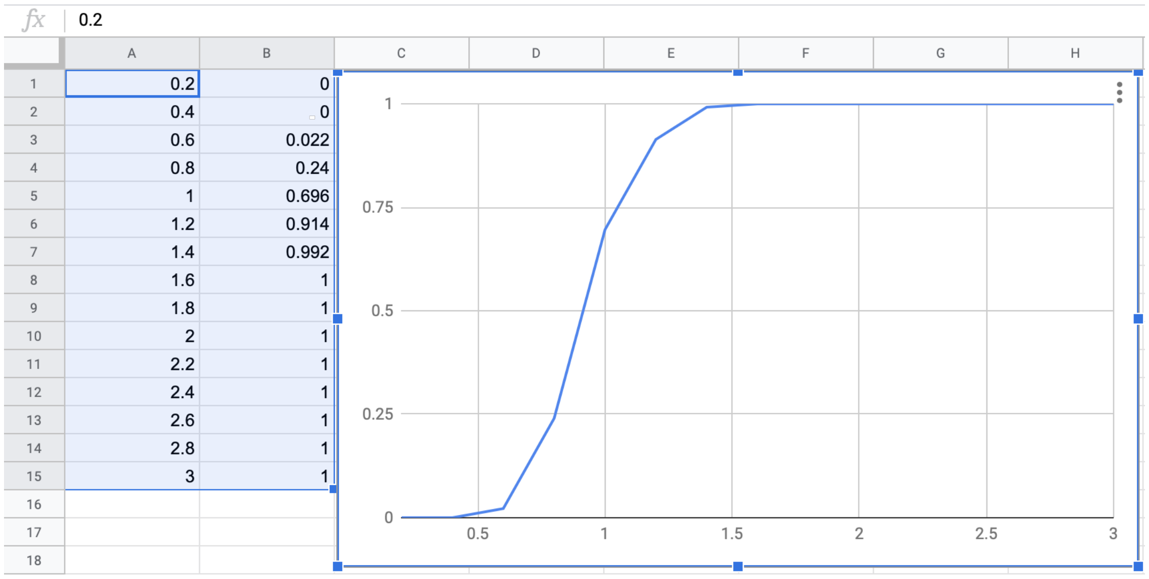
February 28, 2019

1 Hello World!

```
import java.util.*;
import java.text.*;

// used JAVA SE 11

public class PA1 {
        public static void main(String[] args){
                Scanner n = new Scanner(System.in);
                System.out.println("What is your n? ");
                int nodes = (int) n.nextDouble();
                while (nodes < 2) {
                        System.out.println("Sorry choose a n > 1
                         ↪  and a whole number. What is your n?
                         ↪  ");
                        nodes = (int) n.nextDouble();
                }

                Scanner p = new Scanner(System.in);
                System.out.println("Chose a p between [0,1]. What
                 ↪  is your p? ");
                Double prob = p.nextDouble();
                while ((prob < 0) || (prob > 1)) {
                        System.out.println(" Sorry choose a p
                         ↪  greater inside [0,1]. What is your p?
                         ↪  ");
                        prob = p.nextDouble();
                }

                Scanner t = new Scanner(System.in);
                System.out.println("Pick a whole number t greater
                 ↪  than 1. What is your t? ");
                int t_val = (int) t.nextDouble();
                while(t_val < 1){
```

```
                System.out.println("Sorry pick a whole
                ↪  number t greater than 1. What is your
                ↪  t? ");
                t_val = (int) t.nextDouble();
        }

        System.out.println(main(nodes, prob, t_val));
        mainTest();
}

public static int main(int nodes, Double p, int t_val){
        List visited = new ArrayList();
        List vis_search;
        int n = nodes;
        int[][] adj_list = generate_graph(n, p);
        for(int k = 0; k < n; k++){
                if(visited.contains(k) == false){
                        vis_search = BFS(k, adj_list);
                        for(int m = 0; m <
                        ↪  vis_search.size(); m++){
                                // add the error message
                                visited.add(vis_search.get(m));
                        }
                        if(vis_search.size() >= t_val){
                                return 1;
                        }
                }
        }
        return 0;
}

public static void mainTest(){
        System.out.println("Testing.. for c [.2,3]
        ↪  increment by .2. 500 random graphs for each
        ↪  c");
        Double c = .2;
        DecimalFormat df = new DecimalFormat("#.#");
        while(c <= 3.199999){
                Double percentage = testing(c);
                System.out.println("c-value: "+
                ↪  df.format(c) + "  percentage: " +
                ↪  percentage + " of conncections out of
                ↪  500 graphs");
                c += .2;
        }
        System.out.println("Ran 7500 randomly generated
         ↪  graphs. 500 gaphs for each c");
}

public static Double testing(Double c){
        Double n = 40.0000;
        Double p = c/n;
        int i = 1;
        int count = 0;
        while(i <= 500){
                int ret_value = main(40, p, 30);
                if(ret_value == 1){
                        count += 1;
                }
                i++;
        }
        return count/500.00;
}

public static List BFS(int s, int[][] adj){
        HashMap level = new HashMap();
        HashMap parent = new HashMap();
        List vis = new ArrayList();
        List frontier = new ArrayList();
        level.put(s, 0);
        parent.put(s, -1); // did this because there is
         ↪  no null and there will never be a node that
         ↪  is less than 0
        vis.add(s);
        frontier.add(s);
        int i = 1;
        while (frontier.size() != 0) {
                List next_up = new ArrayList();
                for (Object ob :  frontier){
                        int u = (int) ob;
                        for (int v : adj[u]){
                                if (level.containsKey(v)
                                 ↪  == false){
                                        level.put(v, i);
                                        parent.put(v, u);
                                        next_up.add(v);
                                        vis.add(v);
                                }
                        }
                }
                frontier = next_up;
                i++;
            }
            return vis;
        }

        public static int[][] generate_graph(int n, Double p) {
            Random rand = new Random();
            Double num;
            ArrayList<Integer>[] connect = new ArrayList[n];
            for(int index = 0; index < n; index++){
                connect[index] = new
                 ↪  ArrayList<Integer>();
            }
            int[][] graph = new int[n][];
            for(int x = 0; x < n; x++){
                for(int y = 0; y < n; y++){
                    num = rand.nextDouble();
                    if((num < p) && (x != y) &&
                     ↪  (connect[x].contains(y) ==
                     ↪  false)){
                        connect[x].add(y);
                        connect[y].add(x);
                    }
                }
            }
            for(int index = 0; index < n; index++){
                int size = connect[index].size();
                graph[index] = new int[size];
                for(int i = 0; i < size; i++){
                    graph[index][i] =
                     ↪  connect[index].get(i);
                }
            }

        return graph;
        }
}
```

2 REPORT: Graph and implementation



	A	B
1	0.2	0
2	0.4	0
3	0.6	0.022
4	0.8	0.24
5	1	0.696
6	1.2	0.914
7	1.4	0.992
8	1.6	1
9	1.8	1
10	2	1
11	2.2	1
12	2.4	1
13	2.6	1
14	2.8	1
15	3	1
16		
17		
18		

To begin, I first made a function that would create an adjacency list $(generate_graph)$. Inside this function I had an object rand to generate my randomness and num that would keep track of what rand gave me. I then created connect with is an arraylist that will be used to keep track of each vertex and its connected vertices. I then had my two for loops that would go through each vertex and assign if there was a connection between two vertices or not. To finish off this function, I made it so that my int[][] object graph had all the proper indexes with its corresponding vertex connections. This simplified my solution when I would use the BFS function.
In my BFS function I used the pseudocode shown in class but added a list that would keep track of every node visited in the algorithm. then at the end I would return this visited object. This made it easy to check what nodes were visited with the search.
My main function is what preforms my program. It has an ArrayList object of visited that will keep a global track of all the nodes visited in the graph. after every bfs, those nodes returned from it are added to visited so that they I do not run a bfs on nodes are already looked at since we would have two bfs's of the same size. After each run of the bfs, i check to see if the size of the list returned is equal to t, if it is i returned 1 if i continue going through each node and repeating. If by the end of each node there was not a bfs that returned a bfs list of greater than t, i just returned 0. When running a bfs, this gives a list of all the nodes that are connected to one another. That is why I don't not have to run a bfs on a node that has already been visited since the length of that connected components has already been taken into account.
The beginning of the class just gets user input as specified in the project specs

and will not accept some inputs. Also, the test and maintest functions is what I used to create my graph and also to get a visual perspective to see if my code was working as described in the specs of the project.
The graph is essentially showing up that when the c gets to about 1.4, meaning we get a p-value of .035, it begins to give us a graph who has 30 of the 40 nodes connected. This is a way to check to see that the random graph was created using the proper methodology. The y-axis is in terms of decimals from 0 to 1 which is to measured as percentage of graphs with at least 30 connected componenets. X-axis is the c value I used that would get me a p-value after dividing by the number of nodes (40)

Collaborated with: Ashley Robinson