Problem Solving Methodology in IT (COMP1001)

Assignment Seven

(Due at noon on 12 Nov 2018)

**(Instructions: see the instructions in Blackboard)**

Rocky K. C. Chang

1 November 2018

1. [Weight = 4] (The MCGW problem again) A COMP1001 student recently proposed another way of abstracting the problem. Instead of using 4 Boolean values to represent each state, he suggested to use three for the locations of the cabbage, goat and wolf. That is, the man's location is excluded from the states. Answer the questions below for this new abstraction.
2. [Weight = 1] Which states are legal and why?
3. [Weight = 1] Note that only one entity can be changed in each state transition. Draw the graph that consists of the legal nodes determined from (a).
4. [Weight = 1] Determine which links in (b) are legal and why?
5. [Weight = 1] Compare the solution obtained from this new abstraction with the old one. What are the differences?
6. [Weight = 3] (Generating a random directed graph) In a directed graph, each line is generally unidirectional (i.e., node *a* 🡪 node *b* does not imply node *b* 🡪 node *a*). Write a function that will accept *n* and *m* which are the number of nodes and the number of links each node has, respectively. This function then generates for each node *m* unidirectional links to a random selection of the other *n*−1 nodes.

***Function*** *genRandomDirectedGraph(n,m):*

***Input****: n is the number of nodes, and m is the number of links each node has, where n > m.*

***Output****: Return a random graph in which a node's m links to other nodes are selected randomly.*

Implement the graph using the same dictionary data type used for the MCGW problem. You may find *random.sample*() in the *random* module useful. Include the following test cases in your .*py* file. The *findShortestPath*() function will be given to you later.

*randomGraph = genRandomDirectedGraph(15,2)*

*print("The graph is:")*

*print(randomGraph)*

*print()*

*print("The shortest paths are:")*

*for node in range(1,15):*

*print(findShortestPath(randomGraph, node, 0, path=[]))*

*print()*

*randomGraph = genRandomDirectedGraph(25,2)*

*print("The graph is:")*

*print(randomGraph)*

*print()*

*print("The shortest paths are:")*

*for node in range(1,25):*

*print(findShortestPath(randomGraph, node, 0, path=[]))*

*print()*

*randomGraph = genRandomDirectedGraph(50,2)*

*print("The graph is:")*

*print(randomGraph)*

*print()*

*print("The shortest paths are:")*

*for node in range(1,50):*

*print(findShortestPath(randomGraph, node, 0, path=[]))*

1. [Weight = 6] (Finding your friends) Consider a group of 170 people (like our COMP1001 class). For convenience, we label them by 0, 1, 2, …, 169.
   1. [Weight = 3] Your first task is to create 800 "friendship links" between two randomly selected people in the group. You will use Python dictionary with integer-typed key (for the people) and set-typed values (for the set of friends) to model the friendship. Please include the following statements in your *.py* file for testing. You should have a total of 800 "friendship links", because a friendship link is bidirectional.

*# allPeople is a set of all people.*

*# friends is a dictionary for the friendship.*

*size = 0*

*for person in allPeople:*

*print(str(person)+":", friends[person])*

*size += len(friends[person])*

*print(size)*

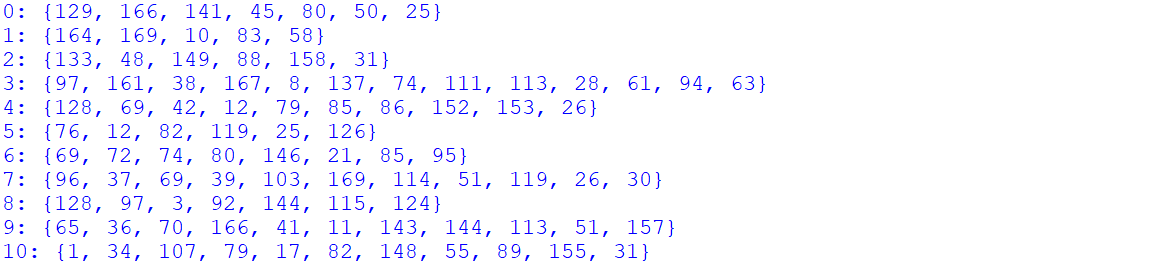
* 1. [Weight = 3] Your second task is to write a function to find out who is the most popular person in the group. If several people enjoy the same popularity, the function returns all of them.

***Function*** *mostPopular(dict):*

***Input****: dict is a dictionary of all people.*

***Output****: Return the people who are the most popular in the people in dict.*

Include statements that will print out the results according to the sample below.



**:**

**:**

