Zachary Hutchinson
zachary.s.hutchinson@maine.edu

HOMEWORK 6

LISTS of LISTS
COS125 - Fall 2022
Due: Oct 28th at 5:00 PM on Brightspace

2022-10-21

Instructions

Submit a Python program that solve the following problem. Your answers **must** be Python3 code. To get full credit, your programs must be correct and the code must be neat. Each code file must contain the usual header comment (filled out) at the top of the file. Failure to include and fill out the header comment will result in a deduction. The collaboration line should contain the names of anyone in the course with whom you discussed the assignment. You do not need to include the names of staff members. If you did not collaborate, your collaboration should say: “I didn’t collaborate with anyone”.

DO NOT FORGET THE FULL HEADER COMMENT!

IMPORTANT: Only .py files will be graded. Do not submit code in any other file format. You do not need to submit output. If we have doubts about your code’s correctness, we will run it ourselves.

Goal

The goal of this homework is to give you practice creating and using lists.

Task A - Adjacency Matrix to Adjacency List

BACKGROUND

A graph is comprised of a set of vertices and edges. A directed graph is a graph in which the edges are directed. A directed edge can only be traversed in one direction (*follow the arrow!*). Figure 1 is an example of a directed graph:

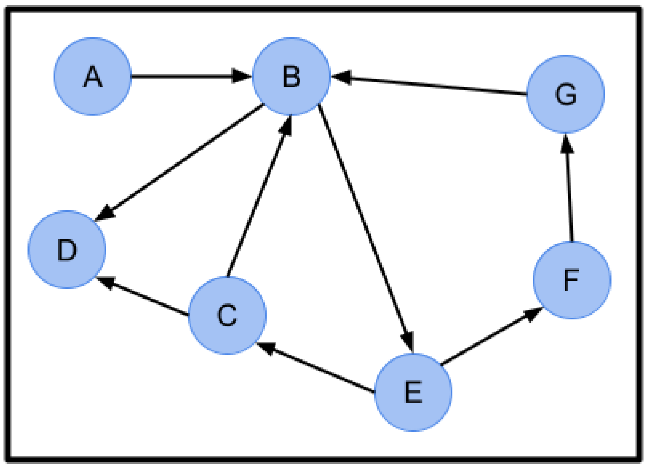

Figure 1: A directed graph. Blue circles are vertices. Black lines with arrows are directed edges.

In the above example, there is an edge from vertex A to vertex B but not vice versa because the arrow points from A to B. One vertex is **adjacent to** another if there is a directed edge from it to the other. For example, A is adjacent to B because there is an edge from A to B. However, B is **not** adjacent to A.

Graphs can be represented in several ways. Two ways graphs can be represented are by an *adjacency matrix* and an *adjacency list*.

ADJACENCY MATRIX: An adjacency matrix is a square matrix of 0's and 1's. A 1 in the matrix signifies that there is an edge between two vertices. A 0 signifies there is not an edge. Here is the adjacency matrix for the graph in Figure 1.

	A	B	C	D	E	F	G
A	0	1	0	0	0	0	0
B	0	0	0	1	1	0	0
C	0	1	0	1	0	0	0
D	0	0	0	0	0	0	0
E	0	0	1	0	0	1	0
F	0	0	0	0	0	0	1
G	0	1	0	0	0	0	0

Table 1: An adjacency matrix representation of Figure 1. The graph should be read: from row to column. For example, the 1 in second row should be read as: there is an edge from vertex A to vertex B.

ADJACENCY LIST: An adjacency list represents a matrix in a more compact form. Edges are represented as a list of adjacent vertices. Here is the adjacency list representation of the graph in Figure 1 and the adjacency matrix above. These three things (the graph, the matrix and the list) are equivalent in that they represent the same graph.

```
A: B
B: D, E
C: B, D
D:
E: C, F
F: G
G: B
```

In the above adjacency list, we can see that vertex *A* is adjacent to *B* because *B* appears in *A*'s list.

TASK REQUIREMENTS

You are to write a program that does the following:

- Ask the user for the number of vertices in the graph.
- Ask the user for the probability that there is an edge from one node to another. The probability is in the range $[0, 1]$.
- Using the two values entered by the user, the program should generate a random adjacency matrix.
  - In your code, the matrix should be stored as a two dimensional list.
  - An edge connects from one vertex to another if a random floating point number drawn from 0 to 1 is less than the probability entered by the user.
- Next the program should create an adjacency list which is equivalent to the randomly generated adjacency matrix.
  - The adjacency list should be created in code using a list of lists.
- Print both the matrix and the list per the example below. The adjacency matrix should be neat with columns neatly aligned. **NOTE**: You can expect the user to ask for 10 or less vertices for the sake of matrix formatting. For 10 or less vertices, all columns are one character wide which can be formatted without any fancy formatting tools.
- The program should consist of five functions:
  - **GenerateAdjMatrix**: GenerateAdjMatrix creates and returns the adjacency matrix. The function takes two parameters: the two inputs provided by the user.
  - **CreateAdjList**: CreateAdjList creates and returns the adjacency list. The function takes the adjacency matrix as a parameter.
  - **PrintAdjMatrix**: This function prints the adjacency matrix in the format specified in the example below.

  - **PrintAdjList**: This function prints the adjacency list in the format specified in the example below.
  - **main**: All code not in one of the other four functions should be in main (e.g., user input) except for import statements and constants.

Unlike the opening example, your vertices are labeled using natural numbers starting from 0.

Example run of the program for Task A (NOTE: The adjacency matrix row and column headers are not part of the adjacency matrix itself. They are just labels created during printing):

```
NUMBER OF VERTICES: 10
PROBABILITY OF AN EDGE BETWEEN VERTICES: 0.25

ADJACENCY MATRIX:

  0 1 2 3 4 5 6 7 8 9
0 0 0 0 1 0 0 0 0 1 0
1 0 0 0 0 0 0 0 1 1 1
2 1 0 1 0 1 0 0 0 0 1
3 0 0 1 1 0 0 0 0 0 0
4 0 0 0 1 1 0 0 0 0 0
5 0 0 0 0 0 0 0 1 0 0
6 1 1 0 1 0 0 0 0 0 0
7 0 0 0 0 0 0 0 1 0 1
8 0 0 1 0 0 0 0 0 1 1
9 0 0 0 0 0 1 1 1 0 0

ADJACENCY LIST:

0: 3 8
1: 7 8 9
2: 0 2 4 9
3: 2 3
4: 3 4
5: 7
6: 0 1 3
7: 7 9
8: 2 8 9
9: 5 6 7
```