### Graph Traversal ( Due 23 Nov 2020 )

In this assignment you will be creating a graph from an input data file called [graph.txt](https://www.cs.utexas.edu/users/mitra/csFall2020/cs313/assgn/graph.txt). The first line in that file will be a single integer *v*. This number will denote the number of vertices to follow. The next *v* lines will be the labels for the vertices. There will be one label to a line. Assume that the labels are unique. The next line after the labels for vertices will be a single number *e*. This number will denote the number of edges to follow. There will be one edge per line. Each edge will be of the form - fromVertex, toVertex, and weight. If the weight is not given, assign a default weight of 1 to that edge. After the list of edges there will be a label for the starting vertex. This will be the starting vertex for both the Depth First Search and Breadth First Search. After that there will be two cities and you will have to delete the edges connecting the two cities and print the adjacency matrix. Then there will be a single city and you will delete this vertex and all edges from and to this vertex. You will print the list of vertices and the adjacency matrix showing all edges from it and all edges to it have been deleted. To delete a vertex from the graph - remove it from the vertex list and remove the corresponding row and column for this vertex.

Here is the outline of the [code](https://www.cs.utexas.edu/users/mitra/csFall2020/cs313/assgn/Graph.py) that we explained in class that you will be modifying. You can add an Edge class if you want to. You will be adding the following functions to the Graph class and the following test cases to your main program.

class Graph (object):

# check if a vertex is already in the graph

def has\_vertex (self, label):

# get the index from the vertex label

def get\_index (self, label):

# add a Vertex object with a given label to the graph

def add\_vertex (self, label):

# add weighted directed edge to graph

def add\_directed\_edge (self, start, finish, weight = 1):

# add weighted undirected edge to graph

def add\_undirected\_edge (self, start, finish, weight = 1):

# get edge weight between two vertices

# return -1 if edge does not exist

def get\_edge\_weight (self, fromVertexLabel, toVertexLabel):

# get a list of immediate neighbors that you can go to from a vertex

# return a list of indices or an empty list if there are none

def get\_neighbors (self, vertexLabel):

# return an index to an unvisited vertex adjacent to vertex v (index)

def get\_adj\_unvisited\_vertex (self, v):

# get a copy of the list of Vertex objects

def get\_vertices (self):

# do a depth first search in a graph starting at vertex v (index)

def dfs (self, v):

# do a breadth first search in a graph starting at vertex v (index)

def bfs (self, v):

# delete an edge from the adjacency matrix

# delete a single edge if the graph is directed

# delete two edges if the graph is undirected

def delete\_edge (self, fromVertexLabel, toVertexLabel):

# delete a vertex from the vertex list and all edges from and

# to it in the adjacency matrix

def delete\_vertex (self, vertexLabel):

def main():

# test depth first search

# test breadth first search

# test deletion of an edge

# test deletion of a vertex

if \_\_name\_\_ == "\_\_main\_\_":

main()

Here is sample output. For grading purpose, please follow this output format. Matching the provided sample prompt will expedite grading and prevent unexpected grading error.

Depth First Search

Houston

Atlanta

Kansas City

Los Angeles

San Francisco

Seattle

Denver

Chicago

Boston

New York

Dallas

Miami

Breadth First Search

Houston

Atlanta

Miami

Dallas

Kansas City

New York

Los Angeles

Denver

Chicago

Boston

San Francisco

Seattle

Deletion of an edge

Adjacency Matrix

0 1 0 1 0 1 0 0 0 0 0 0

1 0 1 1 0 0 0 0 0 0 0 0

0 1 0 1 1 0 0 0 0 0 1 0

1 1 1 0 1 1 0 0 0 0 0 0

0 0 1 1 0 1 0 1 1 0 1 0

1 0 0 1 1 0 1 1 0 0 0 0

0 0 0 0 0 1 0 1 0 0 0 0

0 0 0 0 1 1 1 0 1 0 0 0

0 0 0 0 1 0 0 1 0 1 0 1

0 0 0 0 0 0 0 0 1 0 0 1

0 0 1 0 1 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 1 1 1 0

Deletion of a vertex

List of Vertices

Seattle

San Francisco

Los Angeles

Kansas City

Chicago

Boston

New York

Atlanta

Miami

Dallas

Houston

Adjacency Matrix

0 1 0 0 1 0 0 0 0 0 0

1 0 1 0 0 0 0 0 0 0 0

0 1 0 1 0 0 0 0 0 1 0

0 0 1 0 1 0 1 1 0 1 0

1 0 0 1 0 1 1 0 0 0 0

0 0 0 0 1 0 1 0 0 0 0

0 0 0 1 1 1 0 1 0 0 0

0 0 0 1 0 0 1 0 1 0 1

0 0 0 0 0 0 0 1 0 0 1

0 0 1 1 0 0 0 0 0 0 1

0 0 0 0 0 0 0 1 1 1 0

For this assignment you may work with a partner. Both of you must read the paper on [Pair Programming.](https://www.cs.utexas.edu/users/mitra/csFall2020/cs313/assgn/PairProg-CACM-1999.pdf).

The file that you will be uploading will be called **Graph.py**. We are looking for clean and structured design. The file will have a header of the following form:

# File: Graph.py

# Description:

# Student Name:

# Student UT EID:

# Partner Name:

# Partner UT EID:

# Course Name: CS 313E

# Unique Number:

# Date Created:

# Date Last Modified:

If you are working with a partner only **one** of you will be submitting your program but make sure that you have your partner's name and eid in your program. If you are working alone, then remove the two lines that has the partner's name and eid in the header.

Use the [Canvas](http://canvas.utexas.edu/)system to submit your **Graph.py** file. We should receive your work by 11 PM on Monday, 23 Nov 2020. There will be substantial penalties if you do not adhere to the guidelines. Remember Python is case sensitive. The name of your file must match exactly what we have specified.

* Your Python program should have the proper header.
* Your code must run on the command line before submission.
* You should be submitting your file through the web based *Canvas* program. We will not accept files e-mailed to us.