Artificial Intelligence (Fall 2024)

Problem Set 1

Topic: Dive into the Graphs

Reading Content for Assignment: Chapter 22 of Introduction to Algorithms by CLRS

Instructions:
Implementation language will be Python No other language is allowed.

Tasks:

Task 1:

Write a program for representation of graph.

- Read a text file. Take name of the input from the user.
- Format of file is as follow

```
4_0

A B C D

4

A B

B C

C A

D C
```

First line contain number of vertices. Then after the underscore whether the graph is directed or not(1 for Directed and 0 for Undirected)
Second line contains their names. (space delimited)
Third line contains the number of edges.
Next lines contain the edges

Follow the following structure for Task1 functions.

```
class Graph:
    def __init__(self):
        self.vertices = []  # List to store vertex names
        self.edges = []     # List to store edges as tuples
(start, end)
        self.is_directed = False  # To store whether the graph is
directed
        self.adjacency_list = {}  # Dictionary to store adjacency
list
```

```
    def read_graph_from_file(self, filename):
        """
        Reads the graph from a file with the specified format.
        Input: filename - name of the file containing the graph.
        """
        # Stub: implement file reading logic
        pass

    def get_vertex_count(self):
        """
        Returns the total number of vertices in the graph.
        Output: int - number of vertices.
        """
        # Stub: return the count of vertices
        return len(self.vertices)

    def get_edge_count(self):
        """
        Returns the total number of edges in the graph.
        Output: int - number of edges.
        """
        # Stub: return the count of edges
        return len(self.edges)

    def is_graph_directed(self):
        """
        Returns whether the graph is directed or not.
        Output: bool - True if the graph is directed, False
otherwise.
        """
        # Stub: return the directed/undirected status
        return self.is_directed

    def get_neighbors(self, vertex):
        """
        Returns the neighbors of the given vertex.
        Input: vertex - the vertex whose neighbors are to be
returned.
        Output: list - list of neighboring vertices.
        """
        # Stub: return the list of neighbors
        return self.adjacency_list.get(vertex, [])
```

- Answer the following questions using Console UI. (UI is upto you. How good you make it)
    - How many Vertices are there?
    - How many Edges are there?
    - Is the graph Directed?
    - Which are the neighbors of any vertex (lets say for A).

Task 2:

Run the DFS on the graph and output the following information.

- What is the order of nodes visited.
- Draw the tree programmatically. (Bonus)

```
def dfs(graph, start_vertex): #return the tuple of node name visited in order.
def dfs_draw_tree(graph, start_vertex):
```

Task 3: Run the BFS on the graph and output the following information.

- What is the order of nodes visited.
- What is the distance of any node from the start node.
- How many levels are there in the tree?
- Draw the tree programmatically. (Bonus)

```
def bfs(graph, start_vertex): #return the tuple of node name visited in order.
def bfs_distance(graph, start_vertex,end_vertex): #return distance as integer
def bfs_number_of_levels(graph, start_vertex,end_vertex): #return integer
def bfs_draw_tree(graph, start_vertex):
```

Task 4:

- Is the graph Acyclic?
- If not, how many cycles are there.
- Output the path which contains the cycles.

```
def is_acyclic(graph, start_vertex): #returns True or False
```

Task 5:

- What is the shortest point between two vertices.

```
def distance_dijxtra(graph, start_vertex,end_vertex): #return distance and path as following,
(5, [(A,B,2),(B,C,3)])
```

- What if the graph contains the negative edge values?

```
def distance_bellmanford (graph, start_vertex,end_vertex): #return distance and path as following,
   (5, [(A,B,2),(B,C,3)])
```

Things to Remember (Not following the instruction will result in ZERO MARKS):

- Properly commented code and clean code will get more marks.
- Plagiarism will never be tolerated.

What to Submit:

You are required to submit only 4 files zipped in a folder. No file other than zip will be accepted.

- Task1.py
- Taks2.py
- Task3.py
- Task4.py
- Task5.py