CS-202

Data Structures

Assignment 5

Topics: Graphs and Graph Traversals

Due Date: April 24, 2022

You have been hired by a new company called **4Run**. 4Run delivers packages from one warehouse to the other. You are hired to make sure that all packages are delivered the most efficient way and running costs are reduced. The company is being operated in an underdeveloped area where road connectivity is low.

Warehouses are connected by roads. Whenever there is a package to be delivered, you have to take into account which will be the shortest path from one warehouse to the other. Thanks to all the Data Structures you've been studying you know how to design a system that takes in the locations and their connections (path), and outputs the best possible way to deliver these packages 4Run :)

In this assignment you are given one file *Graph.py* that contains 5 functions. Throughout each task you will be implementing the functions on the same file and testing your code. Throughout the assignment you can create helper functions as you deem fit. You are NOT allowed to use external libraries to implement your graph algorithms.

There are 4 tasks in this assignment. **Task 1, Task 2 and Task 3 and mandatory**. **Task 4 is optional** and will count towards extra credit for those that do it.

Task 1: Dataset Parsing 15 Marks

You first need to determine how exactly will you be representing your graph in terms of the underlying data structure.

You can opt for either of the two given data structures to store your Graph:

1. Adjacency Matrix
2. Adjacency List

In your *Graph.py* mention which data structure you used and why.

For loading the graph from the file (inside the constructor), you must use the add_edge function (which will make things easier for you). It will be graded.

The graph can be directed or undirected. The constructor of Graph takes a flag - False for undirected and True for directed. Your implementation should work for both types of graphs.

Please note that your code for parsing should be generalised enough to work on all datasets based on the same format as that provided to you. Corresponding to this task you are required to build a display function that returns a string, meaning that the entire graph will be displayed as a single string. For reference check the example below:

For a graph having 4 locations and 5 connections each with format:

[Location1] [Location2] [Weight]

Input:

n4 c5

A B 60

A C 10

B D 25

C B 5

D A 12

Note: n4 c5 corresponds to 4 locations and 5 connections

Output of the Display Function:

(A,B,60) (A,C,10) (A,D,12) (B,A,60) (B,C,5) (B,D,25) (C,A,10) (C,B,5) (D,A,12) (D,B,25)

Format: Single string line that contains edges sorted alphabetically (first node then second) and separated by a single space.

Note: All edges (connections) are undirected which means for A B 60, you will store both (A,B,60) and (B,A,60).

Deliverables:

def __init__(self, filename, flag):

- Constructor of your Graph class that takes in a file with *filename* and flag and populates the graph.
- Filenames: graph1.txt, graph2.txt, graph3.txt, graph4.txt, graph5.txt

- Flag is True for directed graphs and False for undirected graphs.
- Does not require a return value.

def add_edge(self, start, end, weight, flag):

- Inserts an edge to your graph. Edge will start from *char start* and end at *char end.* Edge will have a weight of *int weight.*
- Does not require a return value.

def display(self):

- Displays your graph in the format above.
- Returns a string.

Explanation for which Data structure was chosen to implement your graph and why.

Task 2: Connectivity Check 30 Marks

Before you deliver packages from one warehouse to the other, you need to check whether you can reach the destination warehouse. You do not want to waste money on a warehouse that you cannot even reach.

Once your graph is ready, it needs to check whether two warehouses are connected or not i.e. if a path exists from one location to another location through a series of connections.

You are, therefore, required to fill a Reachable Function which checks, if in a given graph, one location can be reached from another. The function should return a Boolean meaning that it only shows if such a path exists or not.

There are several ways to check reachability. You need to perform a simple traversal of the graph. As a hint, there is a traversal that checks depth and another for breadth. You need to implement either one. There you go, we just did half of the work for you.

Note: Do NOT use Dijkstra's algorithm for reachability. You will score 0, if you do so. Choose another way – we gave you a hint above.

Deliverables:

def reachable(self, start, end):

- Determines if node *end* is reachable by node *start*.
- Returns a boolean, True for reachable and False for unreachable.

Task 3: Shortest Path Between Warehouses 30 Marks

4Run delivers packages to multiple warehouses across the city. 4Run wants to improve their company's delivery services as compared to competitor delivery services in the same region by devising routes which minimise their delivery times and maximise efficiency. Hence, 4Run has assigned you the task of implementing an algorithm to calculate the shortest possible routes between the warehouse locations.

From your data structures class, you remember your instructor talking about the **Dijkstra's shortest path algorithm** and decide to use that.

Dijkstra's shortest path algorithm is one of the most famous algorithms for finding the shortest path between two nodes, and in your case, two warehouses. If your supervisor tries to find a path between two nodes which are not connected, the system should cater to it. The algorithm only needs to return the weight of the shortest path, and not the path itself. If no path exists, return -1.

Note: You can assume that the graph provided to you in the test file for this part will be an undirected graph and that all weights are non-negative.

Deliverables:

def dijkstra(self, start, end):

- Finds the shortest path between node *start* and node *end*.
- Returns the weight of the shortest path (int).

Task 4: Ordering Warehouses in Delivery Routes 15 Marks

Optional

Certain delivery routes require a specific order of warehouses to stop at temporarily during the journey. You are required to arrange the warehouses in a given directed graph while preserving their order relative to the overall delivery routes. Topological sorting seems the appropriate algorithm to use here, and so you decide to solve the given task by implementing it.

You need to implement the topo_sort function, which needs to sort the warehouses topologically and return the sorted warehouse locations. Since there are multiple possible correct orderings, the test files will check your output against all possible orderings for that graph. Your output only needs to match one possible ordering in order to pass the tests.

Note: You can assume that the graphs provided to you in the test file for this part will be directed graphs and have no directed cycles.

Deliverables:

def topo_sort(self):

- Sorts the graph using the toposort algorithm.
- Returns a string. Example: ABCDEF

Test files

Use py test.py to compile and run the test file.

Caution

4Run does not appreciate instances of malpractice such as hardcoding your way through tasks. Ensure that you spend more time on completing the assignment than exploring a risky way out of it. All graph algorithms should be implemented by you and no external libraries should be used.

Submission Guidelines

You will be required to submit **only** the Graph.py file with the following naming convention: “rollnumber.py”.

Good Luck!