COT 6405 - Intro to Theory of Algorithms

Spring 2023

Project 2: Graph Algorithms

Due: April 24, 11:59 PM

❖ Project Description

In this project you need to implement graph algorithms. You will be building a graph network. In addition, you will need to build a profession and title dictionary. After that you will need to code and implement test cases for graph algorithms like BFS, DFS, Dijkstra's and strongly connected components. More details about these will be given below.

❖ Project Package

For this project, you will be given a few files that you need to work with. Those files include the one dataset files, two skeleton code files, and one python code file to run the test cases. The testing python script tells you how many test cases your code passed and failed.

- ### Dataset

  For this project you will be using the **imdb_network.csv**. The required data is extracted and provided as CSV file.

(1) ***Imdb_network.csv*** file
The dataset file includes the following attributes as columns.

tconst – a unique identifier for each movie
nconst – a unique identifier for each person
primaryTitle – popular title that is used for promotions
primaryName – name by which a person is often credited
primaryProfession – top 3 professions of a person

- Skeleton codes

(1) ***Graph_creation.py***
This python skeleton code file has the functions defined for **creating title, profession dictionary, adding nodes, edges and graphs**. Detailed comments are provided on what you need to code and where to write the code

(2) ***graph_algorithms.py***
This python skeleton code file needs to be used to implement **the bfs, dfs, Dijkstra and kosaraju algorithms.** Comments on what you need to do and "Need to Code" comments are provided. Read the comments carefully and complete the code.

- Testing scripts

A testing script named **'graph_testcases.py'** is provided. You do not have to write any code in this file. This file has all the test case functions called. And you can run this file to check and test your code as running this file gives you how many test cases your code passed or what testcases your code failed.

❖ Project Instructions

Graph_testcases.py:

This Python file will execute your codes and check whether the provided test cases are passed or failed.

Graph_creation.py

- Start implementing the `graph_creation`.py file.
  This file will be used for creating two dictionaries and adding nodes, edges and creating graph.
  This file contains a function **_create_title_dict ()** where you will need to create a dictionary. The key values will be the nconst (of actor or director) and the values field will be a list of movies (primaryTitle's) the actor or director is involved in.

- In the **_create_profession_dict ()** you will need to create a dictionary where keys will be nconst and the values will be corresponding actor or director. An example dictionary is mentioned as a comment in the code.

- In the movie network class of the same file you will need to implement 3 functions

- In the **add_node ()** function you will need to add nodes to the graph. The nodes will be nconst values. It takes node as argument.

- In the **add_edge ()** function you will need to add edges to the graph. It takes node1, node 2, nconst_ar_dr, and weight as arguments. Node 1 and Node 2 are the two end nodes for the edge, nconst_ar_dr is profession dictionary you constructed above, and weight is the number of common movie titles between the two nodes. The result should be a dictionary where key is nconst (Node 1) and value is again a dictionary with key representing other nconst (Node 2) and weight as the value.

- **Before adding Edge weights you must follow the below Instructions:**
  1) The weight of the edges should be greater than 2
  2) Node 1-> Node 2 relation can be that of a director -> actor, but not that of an actor -> director. Again, detailed example of this is provided as a comment in the code of get_graph function.
  3) Bi-directional edges should be added if both Node1 and Node2 are of same category either both are actors, or both are directors. For example: If Node 1 is of actor, then Node 2 should also be of actor and if Node 1 is of director, then Node 2 should also be of director. More details are provided as comments in the code.

- In **create_graph ()** function you need to create a graph by following the proper edge weights and node conventions as mentioned. More details are provided as comments in the code.

Graph_algorithm.py

- This file has 4 functions you need to code.

- The **bfs ()** function takes a graph which is a dictionary representing the graph, start_node which represents the node where the transversal needs to start, search_node which is the node to be searched (optional) as arguments.

- This bfs function needs to implement the breadth first search algorithm and should return the list of visited nodes in order during the search. But if the search_node is mentioned then you need to return 1 if the search node is found and 0 if the search node is not found.

- **Note:** **In BFS** If the given start_node belongs to one strongly connected component then the other nodes belong to that particular component can only be traversed. But the nodes belonging to other components must not be traversed if those nodes were not reachable from the given start_node.

- The **dfs ()** function takes a graph which is a dictionary representing the graph, start_node, which is the starting node of transversal, visited which is the set of visited nodes, path which is list of nodes in the current path and search_node which is the node to be searched (optional) as arguments. And this function needs to implement the depth first search algorithm.

- The function should return a list of nodes visited in the order they area visited during the search. But if the search_node is given then you need to return 1 if the node is found, 0 if the node is not found.

- **Note1:** The optional parameters "**visited**" and "**path**" are initially not required to be passed as inputs but needs to be updated recursively during the search implementation. If not required for your implementation purposes they can be ignored and can be removed from the parameters.

- **Note2: In DFS** If the given start_node belongs to one strongly connected component then the other nodes belong to that particular component can only be traversed. But the nodes belonging to other components must not be traversed if those nodes were not reachable from the given start_node.

- Next you need to implement Dijkstra algorithm in **Dijkstra ()** function. This function takes graph, start_node and end_node as arguments. Graph is a dictionary representing the graph with keys as nodes and values as dictionaries with edges and their weights, start_node is the node where search should start, end_node is the node we want to reach.

- The function returns 0 if the end node is not reachable from the start node. If the end node is reachable then the function needs to return a list containing 3 elements in the order mentioned below -
  1) Shorted path from start_node to end_node. Meaning it is again a list of nconst values in the visited order.
  2) Total distance of the shortest path – which is the summation of edge weights between minimum visited nodes.
  3) Hop count between start and end nodes.

- Next you need to implement the **Kosaraju ()** function. This calculated the strongly connected components. It takes graph as an input argument and needs to return -
  1) List of strongly connected components. Each component is again a list of nconst values.

❖ Test Cases

dictionary_test ()	Checks whether the dictionary contains all actors and directors (nconst id's) as keys and their corresponding values as list of movies(primaryTitle) they are involved in.
movie_network_test ()	Checks whether the graph is constructed correctly or not.
testcase_1_1()	Checks whether all the nodes (nconst id's) are visited from the given start node (nconst id) using BFS.
testcase_1_2(), testcase_1_3()	Given the start node and search node, checks whether the search node exists while traversing from the start node using BFS.
testcase_2_1	Checks whether all the nodes (nconst id's) are visited from the given start node (nconst id) using DFS .
testcase_2_2(), testcase_2_3()	Given the start node and search node, checks whether the search node exists while traversing from the start node using DFS.
testcase_3()	Checks: • Shortest Path (list of nconst id's in the order they are traversed) • Summation of edge weights in the shortest path(integer) • Hop count(integer)
testcase_4()	Checks: • The total number of strongly connected components

❖ Regarding Project queries

- Post your queries/ questions on piazza.
- Note: Queries through email or outlook or Microsoft Teams are not encouraged.
- If you want to Discuss your queries with TA's, you can meet in Microsoft Teams During our office Hours.

❖Submission

All the coding should be done only in ***python*** language. Once done, please zip your completed python files and submit the zip file.

Discussion about the project is totally encouraged. However, all the submissions will be checked for plagiarism and any instance of copying or reusing of other's code is highly discouraged and will have serious consequences such as getting a 0 for the project and getting a grade 'F' for the course.

❖ Grading

The submitted code will be tested on a few different test cases and points will be assigned based on the correctness.
Late submissions for up to a week will be accepting with a penalty of **10%** reduction per late day (including week ends).