

Assignment-3

(Total Points: 100)

Instructions:

- Download the attached python file assignment_3.py (make sure to **NOT** change the name of this file).
- Follow the instructions and **replace** all TODO comments in the scaffolding code.
- Do **NOT** change the function signature provided in the scaffolding code, but you can add as many functions as you want to help you complete the task.
- Test your code as much as you can to make certain it is correct.
- Run flake8 in addition to testing your code; I expect professional and clear code with minimal flake8 warnings (<5) and having McCabe complexity (<10) from all of you.
- Create a write up with formatted code and screenshots of your output, running the McCabe complexity command and error free console. The code must be formatted as text and not as an image.
- Save the write-up as a PDF and submit it along with your python code (file name assignment_3.py) as separate attachments before the due date on canvas.
- Failure to follow these instructions can disrupt the grading process, and so it may result in substantial deductions to the overall score of the assignment of up to 25 points for each instruction.

Note: Running flake 8

flake8 path/to/your/file (for warnings and errors)

flake8 --max-complexity 10 path/to/your/file (for complexity)

Problem 1 (Max points 15):

Part A In your program.

Given an adjacency-matrix representation of a directed graph, implement the **'in_degree'**, and **'out_degree'** method in the scaffolding code. You must invoke the **'print_degree'** to output the degree in a certain format, so the grading script can judge the correctness of your answer.

Part B In your write-up.

How long does it take (in big-O notation) to compute the out-degree of every vertex? How long does it take (in big-O notation) to compute the in-degrees? Justify your answer with proper reasoning.

Problem 2 (Max points 15):

The transpose of a directed graph $G = (V, E)$ is the graph $G^T = (V, E^T)$, where $E^T = \{(v, u) \mid (u, v) \in E\}$. Thus, G^T is G with all its edges reversed. Implement an efficient algorithm for the **'transpose'** method in the scaffolding code for computing G^T from G for the adjacency-matrix representation of G . Provide the running time of your algorithm in big-O notation.

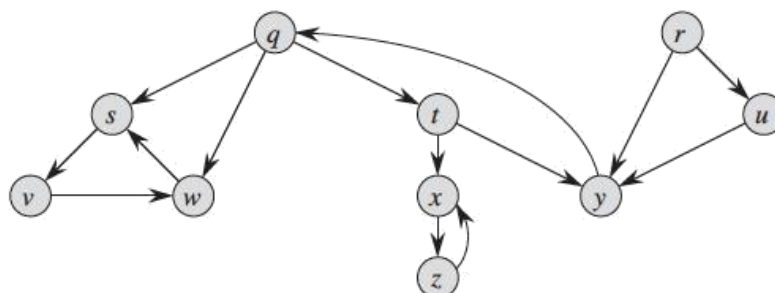
Problem 3 (Max points 15):

Part A In your code

Show how depth-first search works on the graph below by implementing the **'dfs_on_graph'** method in the scaffolding code. Assume that the vertices are explored in alphabetical order (at any point, if it must choose between q and r , then q will be explored before r), and that each vertex in the adjacency matrix is ordered alphabetically. Print the discovery and finishing times for each vertex using **'print_discover_and_finish_time'** method.

Part B In your write-up

Show the classification of each edge in your write-up (Tree/Forward/Back/Cross).



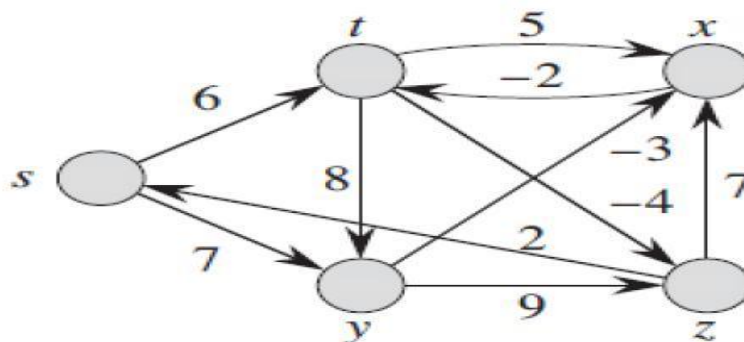
Problem 4 (Max points 15):

Suppose that we represent the graph $G = (V, E)$ as an adjacency matrix. Give a simple (using linear search) implementation of Prim's algorithm ($O(V^2)$ time) by implementing the '**prim**' method in the scaffolding code.

Problem 5 (Max points 20):

Implement '**bellman_ford**' method in the scaffolding code and run the Bellman-Ford algorithm on the directed graph given below, using vertex z as the source. In each pass, relax edges in the order $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$ and invoke the '**print_d_and_pi**' method to print out the d and π values after each pass. (For reference: if (u,v) is an edge which is relaxed in the pass, then $v = u$)

Now, change the weight of edge (z,x) to 4 and run the algorithm again, using s as the source.



Problem 6 (Max points 20):

Dijkstra's Algorithm

```
DIJKSTRA( $G, w, s$ )
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2   $S = \emptyset$ 
3   $Q = G.V$ 
4  while  $Q \neq \emptyset$ 
5       $u = \text{EXTRACT-MIN}(Q)$ 
6       $S = S \cup \{u\}$ 
7      for each vertex  $v \in G.Adj[u]$ 
8          RELAX( $u, v, w$ )
```

Implement the '**dijkstra**' method in the scaffolding code and run Dijkstra's algorithm on the directed graph given below, using vertex s as the source. Invoke '**print_d_and_pi**' method to print out the d and π values and the vertices in set S after each iteration of the while loop in the algorithm given above.

