CMPSC-132: Programming and Computation II
Fall 2018

Homework 5

Due Date: 12/07/2018, 11:59PM
100 pts + Extra Credit

Instructions:

- The work in this assignment must be completed alone.
- Use the starter code provided on this CANVAS assignment. Do not change the function names.
- The file name must be HW5.py (incorrect name files will get a -10 point deduction)
- When any function returns an error, it must be a string containing "error"
- A doctest is provided as an example of code functionality. Getting the same result as the doctest does not guarantee full credit. You are responsible for debugging and testing your code with enough data.
- Do not include test code outside any function in the upload. Printing unwanted or ill-formatted data to output will cause the test cases to fail. Remove all your testing code before uploading your file. Do not include the input() function in your submission.

Goal:

In Module 6, we discussed the Graph data structure. Since the graph is a non-linear structure, there is no unique traversal. Nodes can be found using Breadth-First Search (visit the sibling nodes before visiting the child nodes) and Depth-first search (visit the child nodes before visiting the sibling nodes). Based on the implementation of the Graph data structure discussed during our lecture (provided in the starter code):

1. Create the method *bfs*(*start*). This method takes the key of the starting node and performs Breadth-first search in an instance of the class Graph. This method must **return** a list that contains the order in which the nodes where accessed during the search (following alphabetical order when discovering nodes). You **must** use your queue code from LAB 10 in order to produce your result. If you don't use the queue, your will not receive credit for the assignment
2. Create the method *dfs*(*start*). This method takes the key of the starting node and performs Depth-first search in an instance of the class Graph. This method must **return** a list that contains the order in which the nodes where accessed during the search (following alphabetical order when discovering nodes). You **must** use your stack code from LAB 9 in order to produce your result. If you don't use the stack, your will not receive credit for the assignment

Grading Notes:

- A random instance of the Graph class (directed or undirected, weighted or unweighted) will be created and the *bfs* and *dfs* methods will be called on 4 different starting nodes for 12.5 pts each. Make sure you use the Graph data structure provided in the starter code, otherwise, no credit will be given.
- Vertices and edges will be provided in random order (non-alphabetical order). The final result should be only the one provided by following alphabetical order.

EXAMPLE:
Note that this is only an example, the fact that you code produces the example's output does not ensure your code works properly. Test your code with several examples!

```
g1 = {'A': ['B','D','G'],
     'B': ['A','E','F'],
     'C': ['F'],
     'D': ['A','F'],
     'E': ['B','G'],
     'F': ['B','C','D'],
     'G': ['A','E']}
>>> g=Graph(g1)
>>> g.bfs('A')
['A', 'B', 'D', 'G', 'E', 'F', 'C']
>>> g.dfs('A')
['A', 'B', 'E', 'G', 'F', 'C', 'D']
```

```
g2 = {'F': ['D','C','B'],
     'A': ['G','D','B'],
     'B': ['F','A','E'],
     'E': ['G','B'],
     'C': ['F'],
     'D': ['F','A'],
     'G': ['A','E'],
     'F': ['D','C','B']}
>>> g=Graph(g2)
>>> g.bfs('A')
['A', 'B', 'D', 'G', 'E', 'F', 'C']
>>> g.dfs('A')
['A', 'B', 'E', 'G', 'F', 'C', 'D']
```

```
g3 = {'B': [('E',3),('C',5)],
     'F': [],
     'C': [('F',2)],
     'A': [('D',3),('B',2),],
     'D': [('C',1)],
     'E': [('F',4)]}
>>> g=Graph(g3)
>>> g.bfs('A')
['A', 'B', 'D', 'C', 'E', 'F']
>>> g.dfs('A')
['A', 'B', 'C', 'F', 'E', 'D']
```

EXTRA CREDIT: 40 pts, added regardless of the maximum 100 (no partial credit for incorrect answers)
Create the method *dijkstra*(*start*). This method takes the key of the starting node and runs Dijkstra's algorithm in an instance of the class Graph. The method returns a dictionary with the value of the shortest path from the starting node to every other node reachable from *start*.

```
g3 = {'B': [('E',3),('C',5)],
     'F': [],
    'C': [('F',2)],
    'A': [('D',3),('B',2),],
    'D': [('C',1)],
    'E': [('F',4)]}
>>> g=Graph(g3)
>>> g.dijkstra('A')
{'A': 0, 'B': 2, 'C': 4, 'D': 3, 'E': 5, 'F': 6}
```

Note: For this method, the dictionary does not have to be in alphabetical order, as long as the pair key, value is correct! i.e. {'A': 0, 'D': 3, 'B': 2, 'C': 4, 'E': 5, 'F': 6}, {'A': 0, 'B': 2, 'D': 3, 'C': 4, 'E': 5, 'F': 6}, etc., are also correct

Deliverables:

- Include all your code (including the stack and queue code) in your script named HW5.py. Submit it to the HW5 CANVAS assignment before the due date