CS102L Data Structures & Algorithms
Lab Week 12

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

Spring 2024

Contents

Introduction

1.1 Instructions

- This lab will contribute 1% towards your final grade.
- The deadline for submission of this lab is at the end of lab time.
- The lab must be submitted online via CANVAS. You are required to submit a .zip file that contains all the *.py* files.
- The .zip file should be named as *Lab_12_aa1234.zip* where *aa1234* will be replaced with your student id.
- **Files that don't follow the appropriate naming convention will not be graded.**
- **Your final grade will comprise of both your submission and your lab performance.**

1.2 Marking scheme

This lab will be marked out of 100.

- 50 Marks are for the completion of the lab.
- 50 Marks are for progress and attendance during the lab.

1.3 Lab Objectives

The objective of this lab is:

- To learn how to create graph traversal algorithms in Python.

1.4 Late submission policy

There is no late submission policy for the lab.

1.5 Use of AI

Taking help from any AI-based tools such as ChatGPT is strictly prohibited and will be considered plagiarism.

1.6 Viva

Course staff may call any student for Viva to provide an explanation for their submission.

Prerequisites

2.1 Pytest

Pytest is a popular testing framework in Python known for its simplicity and power in facilitating efficient and scalable testing for applications and software. In order to use Pytest, one first needs to install it and create test functions prefixed with *test_* in a Python file. To install Pytest, you can write the following command on the terminal:

```
pip install pytest
```

After installing Pytest, one can execute all the tests by typing the following command on the terminal:

```
pytest
```

If you want to test any particular file, then you will have to specify the filename as follows:

```
pytest <filename>.py
```

In case the above mentioned pytest commands do not work on your system, even though pytest is installed, try using the following commands:

```
py -m pytest
```

```
py -m pytest <filename>.py
```

If all the tests pass successfully, then you should receive a similar output.

```
PS C:\Work\Spring2024\DSA\Lab1> pytest test_Question1.py
============================= test session starts ==============================
platform win32 -- Python 3.11.2, pytest-7.4.4, pluggy-1.3.0
rootdir: C:\Work\Spring2024\DSA\Lab1
collected 10 items

test_Question1.py ..........

============================== 10 passed in 0.06s ==============================
```

However, if all the tests fail, then you should receive a similar output.

```
PS C:\Work\Spring2024\DSA\Lab1> pytest test_Question1.py
============================= test session starts ==============================
platform win32 -- Python 3.11.2, pytest-7.4.4, pluggy-1.3.0
rootdir: C:\Work\Spring2024\DSA\Lab1
collected 10 items

test_Question1.py FFFFFFFFFF

================================== FAILURES ===================================
_____________________________ test_question1[1st10-1st20-e54dc15ccafa608142ffa6340b035c327f972e24882052cfc3345f240f4ee237] ___
```

Lab Exercises

3.1 Depth First Search

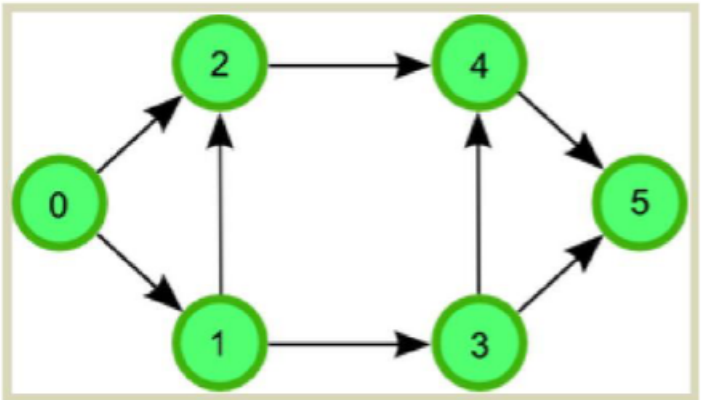
Algorithm 1 dfs(G, s)

Require: G: Graph, s: Starting node
Ensure: List of visited nodes

```
1:  stack ← Initialize Stack
2:  push(stack, s)
3:  visited ← Empty List
4:  visited.append(s)
5:  while stack is not empty do
6:      a ← pop(stack)
7:      if a ∉ visited then
8:          visited.append(a)
9:      end if
10:     for i in G[a] do
11:         if i[0] ∉ visited then
12:             push(stack, i[0])
13:         end if
14:     end for
15: end while
16: return visited
```

Implement the Depth-first Search (DFS) algorithm in the function `dfs` that takes a directed graph, `G`, and a starting node, `s`, and returns a list of nodes in the order that they were visited.

3.1.1 Example




```
>>> G = {0: [(1, 1), (2, 1)], 1: [(2, 1), (3, 1)], 2: [(4,
   1)], 3: [(4, 1), (5, 1)], 4: [(5, 1)], 5: []}
>>> print(dfs(G, 0))
[0, 2, 4, 5, 1, 3]
```

This function should be implemented in the file **Question1.py**, accessible within the Lab 12 module on CANVAS.

3.1.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question1.py
```

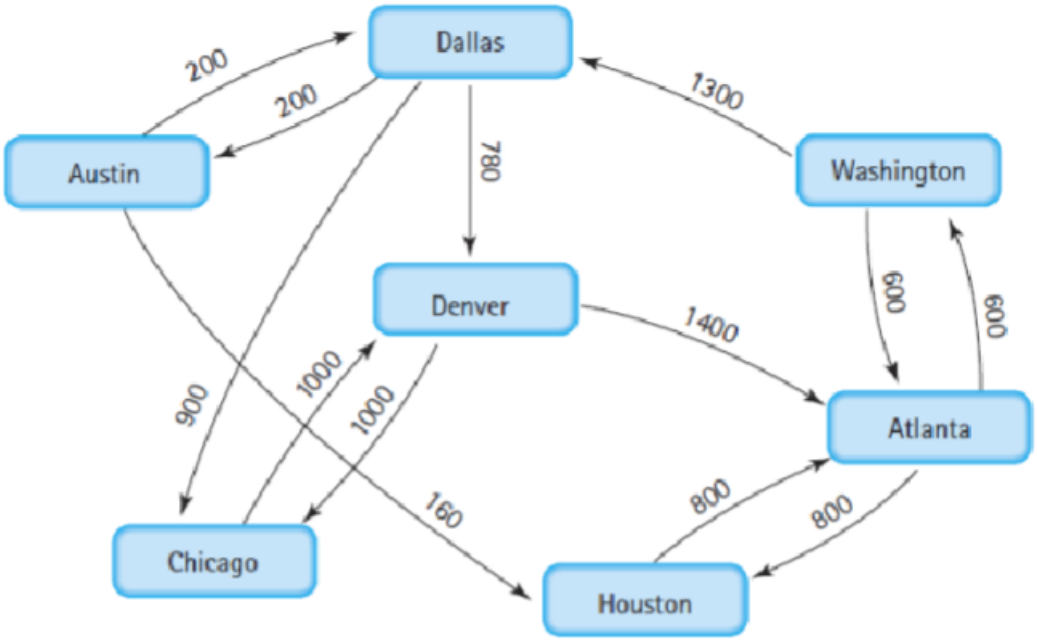
```
py -m pytest test_Question1.py
```

3.2 Detecting Cycles in Graphs

Write a function, `check_cycles` that takes a graph, `G`, list of nodes, `lst` and checks if they form a cycle, returning the appropriate boolean value. Make sure that all the nodes in the given list are visited in the same order. A cycle is a directed path that starts and ends at the same vertex. Before writing code, make sure you can identify cycles yourself.

3.2.1 Example

The following graph is an example from Rosen (2011). It shows the flights and distances between some of the major airports in the United States.




```
>>> G = {'Dallas': [('Austin', 200), ('Denver', 780), ('
    Chicago', 900)], 'Austin': [('Dallas', 200), ('Houston',
    160)], 'Washington': [('Dallas', 1300), ('Atlanta', 600)
    ], 'Denver': [('Atlanta', 1400), ('Chicago', 1000)], '
    Atlanta': [('Washington', 600), ('Houston', 800)], '
    Chicago': [('Denver', 1000)], 'Houston': [('Atlanta',
    800)]}
>>> print(check_cycles(G, ['Austin','Houston','Atlanta','
    Washington']))
False
>>> print(check_cycles(G, ['Dallas','Denver','Atlanta','
    Washington']))
True
```

This function should be implemented in the file **Question2.py**, accessible within the Lab 12 module on CANVAS.

3.2.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question2.py
```

```
py -m pytest test_Question2.py
```

3.3 Breadth First Search

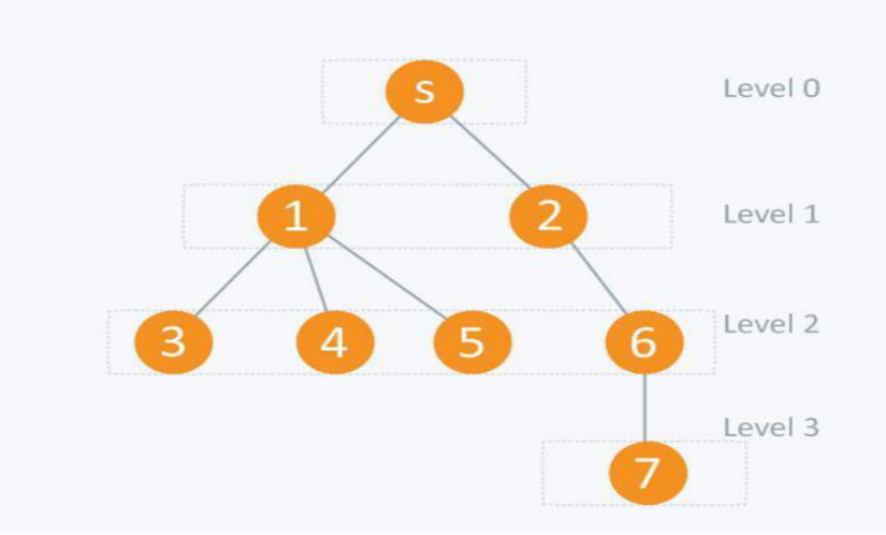
Algorithm 2 bfs(G, s)

Require: G: Graph, s: Starting node
Ensure: List of visited nodes

```
queue ← Initialize Queue
enQueue(queue, s)
visited ← Empty List
visited.append(s)
while queue is not empty do
    x ← deQueue(queue)
    for i in G[x] do
        if i[0] ∉ visited then
            enQueue(queue, i[0])
            visited.append(i[0])
        end if
    end for
end while
return visited
```

Implement the Breadth-First Search (BFS) algorithm (traversing left to right in each level) in the function `bfs` that takes a graph, `G`, and a root node, `s`, and returns a list of nodes in the order that they were visited.

3.3.1 Example




```
>>> G = {'s': [(1, 1), (2, 1)], 1: [(3, 1), (4, 1), (5, 1)],
    2: [(6, 1)], 3: [], 4: [], 5: [], 6: [(7, 1)], 7: []}
>>> print(bfs(G,'s'))
['s', 1, 2, 3, 4, 5, 6, 7]
```

This function should be implemented in the file **Question3.py**, accessible within the Lab 12 module on CANVAS.

3.3.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question3.py
```

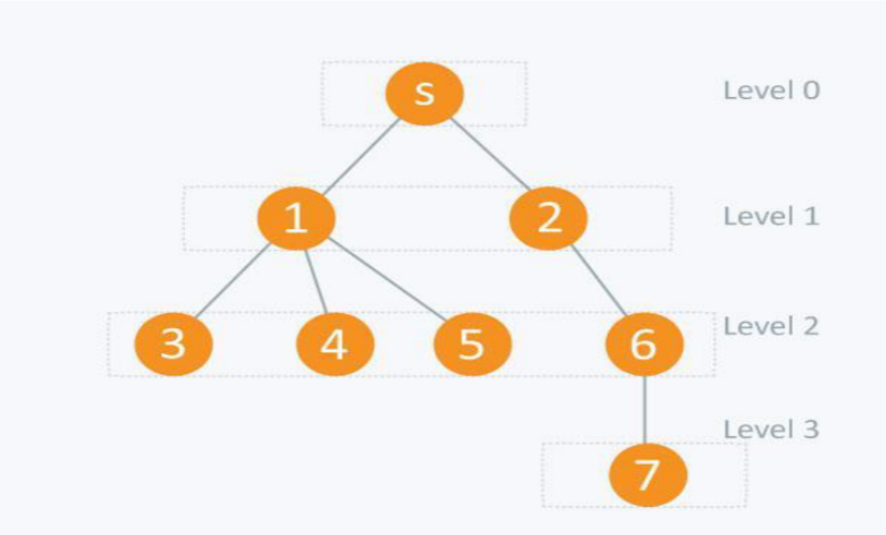
```
py -m pytest test_Question3.py
```

3.4 Nodes of a level in the graph

How to determine the node on a certain level in the given graph? As you know, BFS involves a level-order traversal of a graph. Hence, you can use BFS to determine nodes on a given level.

Write a function `nodes_of_level(G, level)` that takes a graph and level number as input and returns a list of all the nodes that are on the given level (the first node of the graph is considered the starting node or level 0).

3.4.1 Example




```
>>> G = {'s': [(1, 1), (2, 1)], 1: [(3, 1), (4, 1), (5, 1)],
    2: [(6, 1)], 3: [], 4: [], 5: [], 6: [(7, 1)], 7: []}
>>> print(nodes_of_level(G,3))
[7]
>>> print(nodes_of_level(G,1))
[1, 2]
```

This function should be implemented in the file **Question4.py**, accessible within the Lab 12 module on CANVAS.

3.4.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question4.py
```

```
py -m pytest test_Question4.py
```