Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

HALİL ONUR ÇEÇEN
161044057

Course Assistant: Fatma Nur Esirci

1 Q1

This part about Question1 in HW7

1.1 Problem Solution Approach

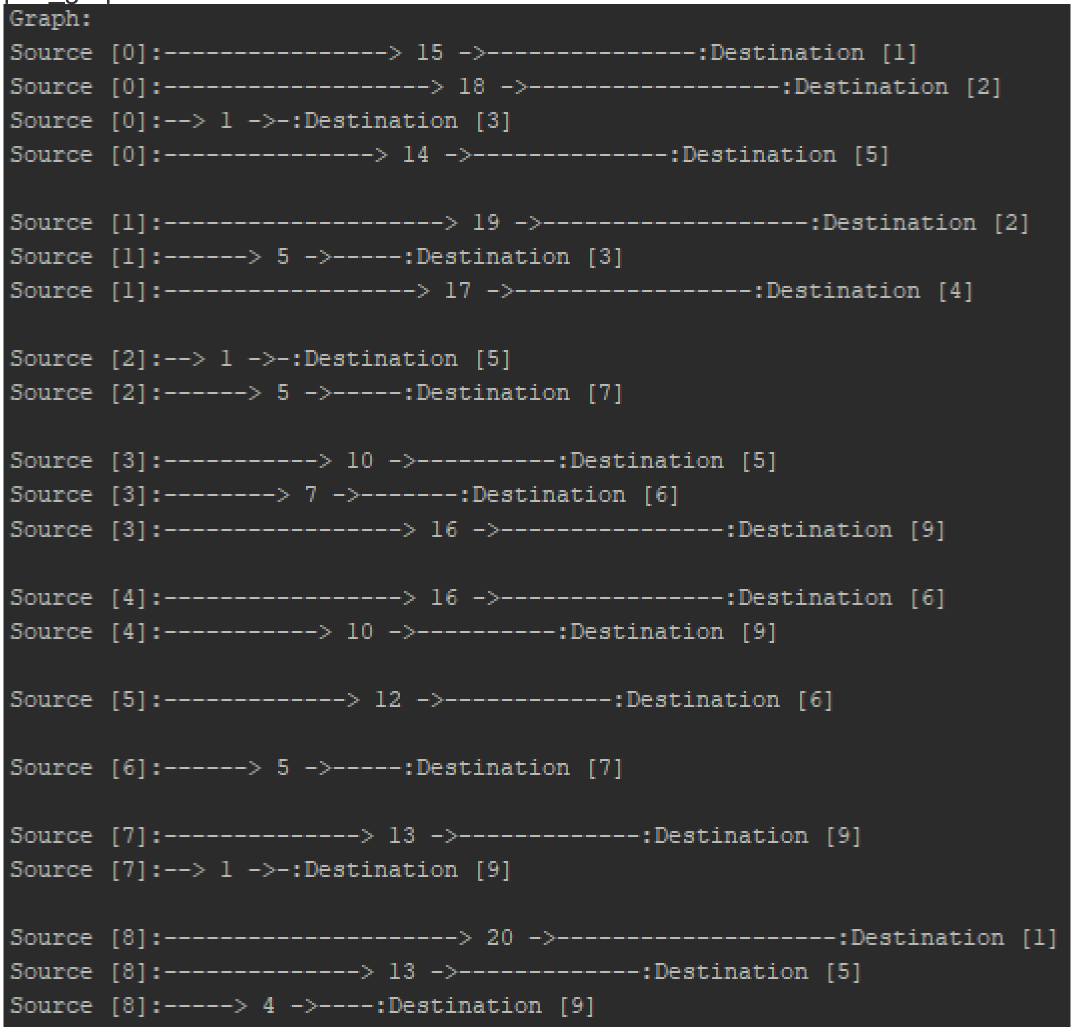
I’ve created an undirected graph with 10 vertices and 20 edges with random weight within 1 to 20.

- To plot the grap, function prints all the edges with their weights in a graph.
- To check if its directed or not I checked if an edge has a complement of itself or not.
- To check the graph is cyclic or not, I tried to use visited nodes recurrence times but couldn’t succeed while doing that. So, my is_acyclic_graph function doesn’t work properly.
- To check shortest path, I used djikstra’s algorithm to create given source vertexs distance from the other vertices and the parents array of themselves. After that using parents array I created the path.

1.2 Test Cases

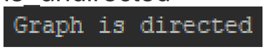
Show that this func results ->

- plot_graph

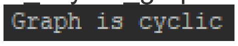
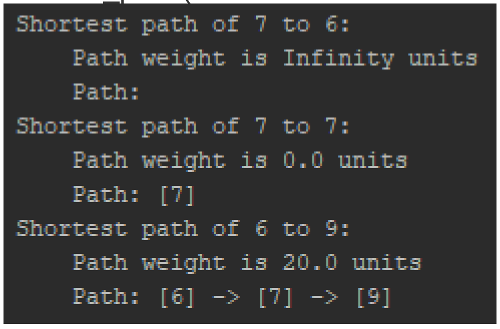


```
Graph:
Source [0]:----------------> 15 ->---------------:Destination [1]
Source [0]:-------------------> 18 ->------------------:Destination [2]
Source [0]:--> 1 ->-:Destination [3]
Source [0]:---------------> 14 ->--------------:Destination [5]

Source [1]:--------------------> 19 ->-------------------:Destination [2]
Source [1]:------> 5 ->-----:Destination [3]
Source [1]:------------------> 17 ->-----------------:Destination [4]

Source [2]:--> 1 ->-:Destination [5]
Source [2]:------> 5 ->-----:Destination [7]

Source [3]:-----------> 10 ->----------:Destination [5]
Source [3]:--------> 7 ->-------:Destination [6]
Source [3]:-----------------> 16 ->----------------:Destination [9]

Source [4]:-----------------> 16 ->----------------:Destination [6]
Source [4]:-----------> 10 ->----------:Destination [9]

Source [5]:-------------> 12 ->------------:Destination [6]

Source [6]:------> 5 ->-----:Destination [7]

Source [7]:--------------> 13 ->-------------:Destination [9]
Source [7]:--> 1 ->-:Destination [9]

Source [8]:---------------------> 20 ->--------------------:Destination [1]
Source [8]:--------------> 13 ->-------------:Destination [5]
Source [8]:-----> 4 ->----:Destination [9]
```

- is_undirected



```
Graph is directed
```

- is_acyclic_graph
  
  ```
  Graph is cyclic
  ```
  (Altough its not)
- shortest_path (use least 3 different label pair)
  
  ```
  Shortest path of 7 to 6:
      Path weight is Infinity units
      Path:
  Shortest path of 7 to 7:
      Path weight is 0.0 units
      Path: [7]
  Shortest path of 6 to 9:
      Path weight is 20.0 units
      Path: [6] -> [7] -> [9]
  ```
  There is no path from 7 to 6.

2 Q2

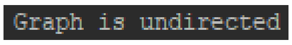
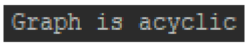
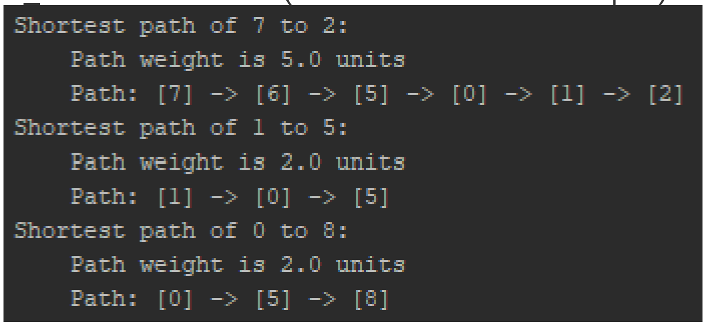
This part about Question2 in HW7

2.1 Problem Solution Approach

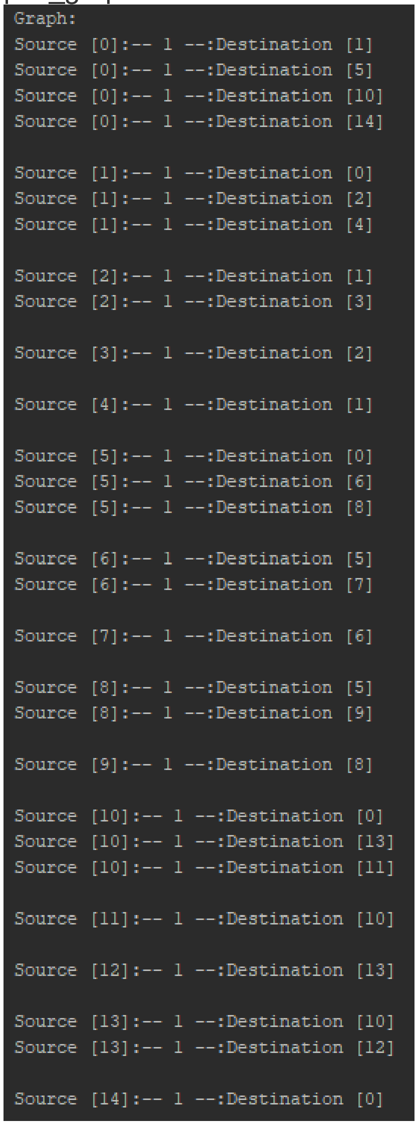
I've created an undirected graph with 15 vertices and 13 edges without weights.
NOTE: Since functions are same I didn't explain them again here. Their explanation can be found in Section 1.1.

2.2 Test Cases

Show that this func results ->

- is_undirected 
  ```
  Graph is undirected
  ```
- is_acyclic_graph 
  ```
  Graph is acyclic
  ```
- is_connected function (use least 3 different label pair)
  
  ```
  Shortest path of 7 to 2:
      Path weight is 5.0 units
      Path: [7] -> [6] -> [5] -> [0] -> [1] -> [2]
  Shortest path of 1 to 5:
      Path weight is 2.0 units
      Path: [1] -> [0] -> [5]
  Shortest path of 0 to 8:
      Path weight is 2.0 units
      Path: [0] -> [5] -> [8]
  ```

- plot_graph



```
Graph:
Source [0]:-- 1 --:Destination [1]
Source [0]:-- 1 --:Destination [5]
Source [0]:-- 1 --:Destination [10]
Source [0]:-- 1 --:Destination [14]

Source [1]:-- 1 --:Destination [0]
Source [1]:-- 1 --:Destination [2]
Source [1]:-- 1 --:Destination [4]

Source [2]:-- 1 --:Destination [1]
Source [2]:-- 1 --:Destination [3]

Source [3]:-- 1 --:Destination [2]

Source [4]:-- 1 --:Destination [1]

Source [5]:-- 1 --:Destination [0]
Source [5]:-- 1 --:Destination [6]
Source [5]:-- 1 --:Destination [8]

Source [6]:-- 1 --:Destination [5]
Source [6]:-- 1 --:Destination [7]

Source [7]:-- 1 --:Destination [6]

Source [8]:-- 1 --:Destination [5]
Source [8]:-- 1 --:Destination [9]

Source [9]:-- 1 --:Destination [8]

Source [10]:-- 1 --:Destination [0]
Source [10]:-- 1 --:Destination [13]
Source [10]:-- 1 --:Destination [11]

Source [11]:-- 1 --:Destination [10]

Source [12]:-- 1 --:Destination [13]

Source [13]:-- 1 --:Destination [10]
Source [13]:-- 1 --:Destination [12]

Source [14]:-- 1 --:Destination [0]
```

3 Q3

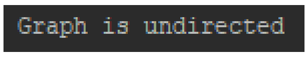
This part about Question3 in HW7

3.1 Problem Solution Approach

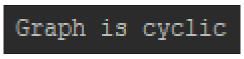
I’ve created an undirected graph with 15 vertices and 13 edges without weights.
NOTE: Since functions are same I didn’t explain them again here. Their explanation can be found in Section 1.1.

3.2 Test Cases

Show that this func results ->

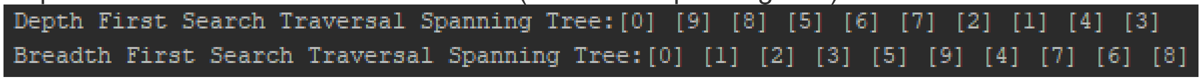
- is_undirected 

```
Graph is undirected
```

- is_acyclic_graph 

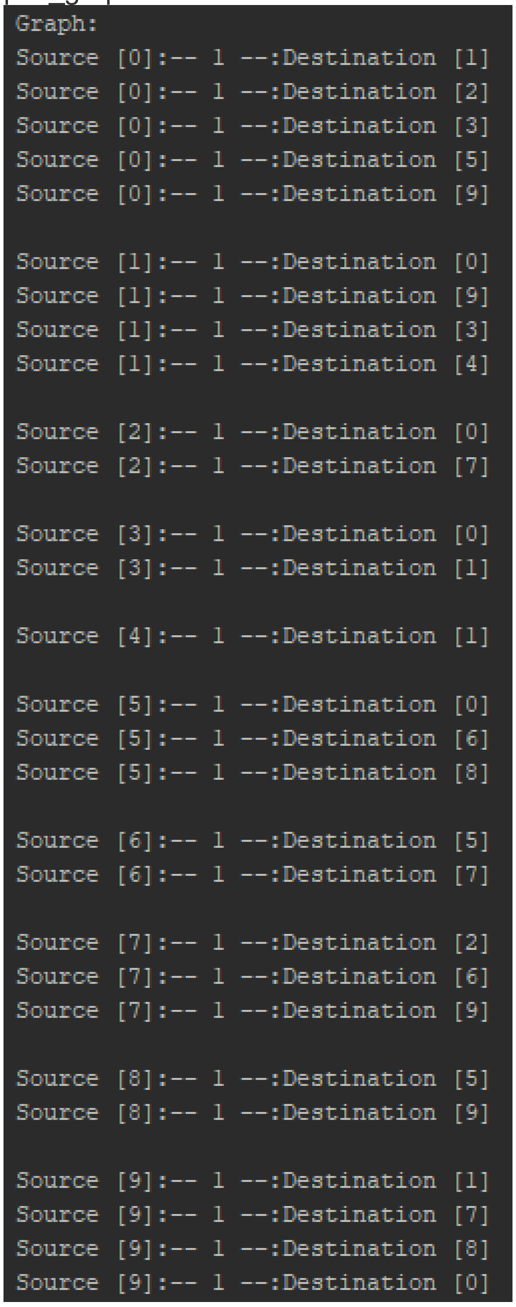
```
Graph is cyclic
```

- DepthFirstSearch and BreathFirstSearch (Show that spanning tree)



```
Depth First Search Traversal Spanning Tree:[0] [9] [8] [5] [6] [7] [2] [1] [4] [3]
Breadth First Search Traversal Spanning Tree:[0] [1] [2] [3] [5] [9] [4] [7] [6] [8]
```

- plot_graph



```
Graph:
Source [0]:-- 1 --:Destination [1]
Source [0]:-- 1 --:Destination [2]
Source [0]:-- 1 --:Destination [3]
Source [0]:-- 1 --:Destination [5]
Source [0]:-- 1 --:Destination [9]

Source [1]:-- 1 --:Destination [0]
Source [1]:-- 1 --:Destination [9]
Source [1]:-- 1 --:Destination [3]
Source [1]:-- 1 --:Destination [4]

Source [2]:-- 1 --:Destination [0]
Source [2]:-- 1 --:Destination [7]

Source [3]:-- 1 --:Destination [0]
Source [3]:-- 1 --:Destination [1]

Source [4]:-- 1 --:Destination [1]

Source [5]:-- 1 --:Destination [0]
Source [5]:-- 1 --:Destination [6]
Source [5]:-- 1 --:Destination [8]

Source [6]:-- 1 --:Destination [5]
Source [6]:-- 1 --:Destination [7]

Source [7]:-- 1 --:Destination [2]
Source [7]:-- 1 --:Destination [6]
Source [7]:-- 1 --:Destination [9]

Source [8]:-- 1 --:Destination [5]
Source [8]:-- 1 --:Destination [9]

Source [9]:-- 1 --:Destination [1]
Source [9]:-- 1 --:Destination [7]
Source [9]:-- 1 --:Destination [8]
Source [9]:-- 1 --:Destination [0]
```

4 Q4

The biggest difference between BFS and DFS is their traversal order. BFS is like level order for trees. It visits first vertex and its adjacent vertices in order. Then it continues to their adjacent vertices. But DFS visits through one adjacent of first vertex until there aren't any vertex left from that vertex. After that it continues to other adjacent vertices of first vertex.
Advantages:

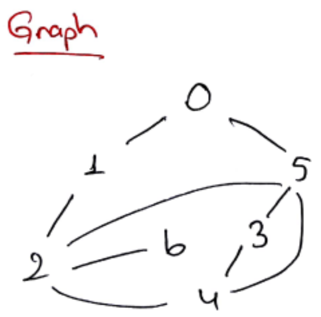
- DFS basically searches by possible paths its more convenient to search a path between a and b points.
- BFS is more convenient if you plan on searching something closer to given vertex.
- If a graph has lots of edges from a single vertex BFS might use much more space than DFS since it queues every sibling.

Usage Areas:

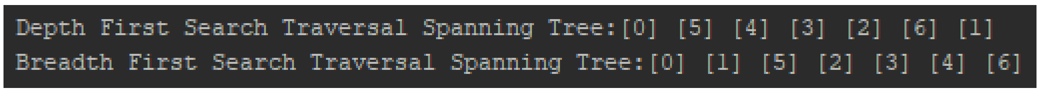
- To find all connected vertices BFS would be more meaningful since it starts from siblings.
- To find a path between two vertices DFS is used because of its path checking.

4.1 Spanning Trees

Assuming vertex 1 is the first vertex of the graph.

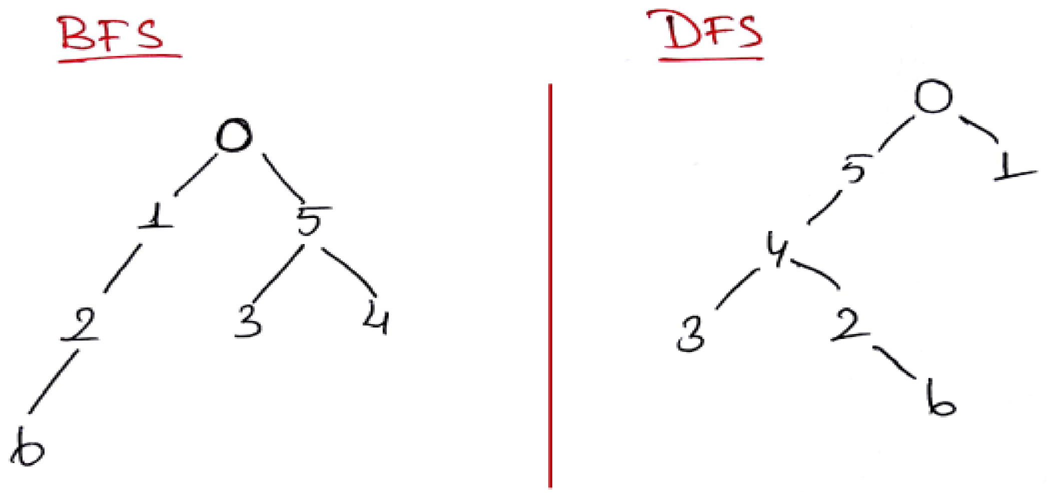


Algorithm run results:



```
Depth First Search Traversal Spanning Tree:[0] [5] [4] [3] [2] [6] [1]
Breadth First Search Traversal Spanning Tree:[0] [1] [5] [2] [3] [4] [6]
```

Tree representations:


BFS
0
1
5
2
3
4
6
DFS
0
5
1
4
3
2
6