CP312: Algorithm Design & Analysis

Assignment #5

CP312, WLU, 2022

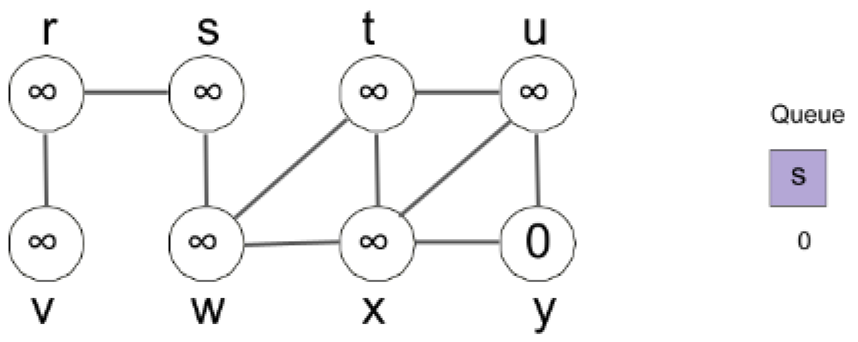
Instructor: Masoomeh Rudafshani

Due: Friday, March 18th, 2022

Declan Hollingworth
190765210
holl5210@mylaurier.ca

Nishant Tewari
190684430
tewa4430@mylaurier.ca

Q1. [10 points] Run the BFS algorithm on the following graph and show the pred and distance arrays as the algorithm proceeds. Start from node y. Show the values on queue at each step of the algorithm.



BFS Array

Node	**Pred**	**Dist**	**Queue**
y	null	0	u, x
u	y	1	x, t
x	y	1	t, w
t	x	2	w
w	x	2	s
s	w	3	r
r	s	4	v
v	r	5	∅

Q2. [10 bonus points] For the same graph in question one, run DFS algorithm and show the pred and discover and finish arrays as the algorithm proceeds.

Step 1

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y		
x	u		
t	x		
w	t		
s	w		
r	s		
v	r		

Step 2

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u		
t	x		
w	t		
s	w		
r	s		
v	r		

Step 3

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x		
w	t		
s	w		
r	s		
v	r		

Step 4

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t		
s	w		
r	s		
v	r		

Step 5

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	
s	w		
r	s		
v	r		

Step 6

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	
s	w	6	
r	s		
v	r		

Step 7

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	
s	w	6	
r	s	7	
v	r	8	

Step 8

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	
s	w	6	
r	s	7	
v	r	8	

Step 9

Node	Pred	Discover	Finish
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	
s	w	6	
r	s	7	
v	r	8	9

Step 10

Node	Pred	Discover	Finish
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	
s	w	6	
r	s	7	10
v	r	8	9

Step 11

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	
s	w	6	11
r	s	7	10
v	r	8	9

Step 12

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	
w	t	5	12
s	w	6	11
r	s	7	10
v	r	8	9

Step 13

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	
t	x	4	13
w	t	5	12
s	w	6	11
r	s	7	10
v	r	8	9

Step 14

Node	**Pred**	**Discover**	**Finish**
y	null	1	
u	y	2	
x	u	3	14
t	x	4	13
w	t	5	12
s	w	6	11
r	s	7	10
v	r	8	9

Step 15

Node	Pred	Discover	Finish
y	null	1	
u	y	2	15
x	u	3	14
t	x	4	13
w	t	5	12
s	w	6	11
r	s	7	10
v	r	8	9

Step 16

Node	Pred	Discover	Finish
y	null	1	16
u	y	2	15
x	u	3	14
t	x	4	13
w	t	5	12
s	w	6	11
r	s	7	10
v	r	8	9

Therefore, there are 16 steps in order to **fully explore** this depth first algorithm.

Q3. [10 points] You are given an undirected graph G=(V, E). Determine whether G is a tree or not? The runtime of your algorithm should be O(V). Write the pseudocode of your algorithm and analyze its time complexity.

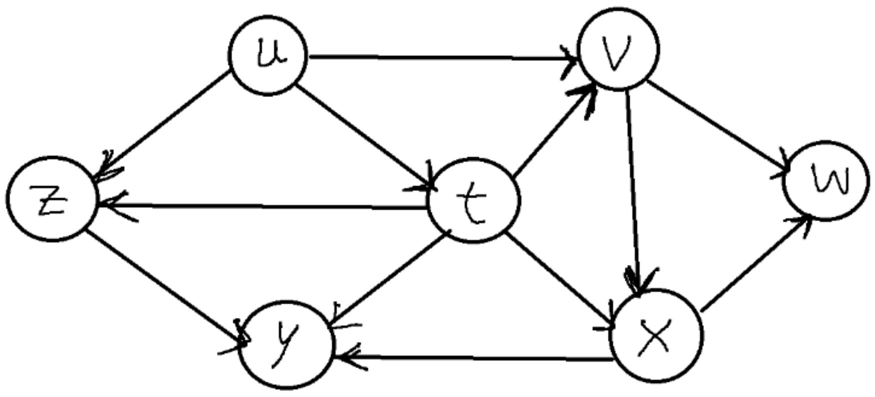
In order for an undirected graph to be a tree, each node after the root after have exactly one parent and ensure no cycles exist. In order to confirm that, we must create to functions, one to go through the graph, and another function that we can call to see if a cycle exists among a set of vertices.

```
# Create algorithm to check for cycles
def checkCycle(u, visited, parent): #O(V)
    Visited[u] = True #mark vertex u as visited
    for all v adjacent with u:
        if visited[v] = true
            if isCycle(v, visited, u) == true:
                return true
        else if v ≠ parent:
            return true
    return false

def treeOrNoTree(graph):
    visited = [False] * V  #mark all node as unvisited
    if checkCycle(0, visited, null) is true #O(V)
        return false
    if the graph is not connected:
        return false
    else
        return true
```

The runtime for the pseudocode above is O(V).

Q4. [5 points] How many topological ordering does the following graph have? Write down all possible topological ordering.



7 possible topological orderings:

- [u, t, v, x, w, z, y]
- [u, t, v, x, z, w, y]
- [u, t, v, x, z, y, w]
- [u, t, z, v, x, y, w]
- [u, t, z, v, x, w, y]
- [u, t, v, z, x, y, w]
- [u, t, v, z, x, w, y]

Q5. [10 points] Prove that a directed acyclic graph must have at least one node u which has no incoming edge, i.e., indegree (u) = 0

In order for a graph to be classified as a directed acyclic graph, it cannot contain a closed loop/cycle within the graph.

Assume that the graph in figure 5.1 is a DAG where each vertex has an indegree(u) >= 1

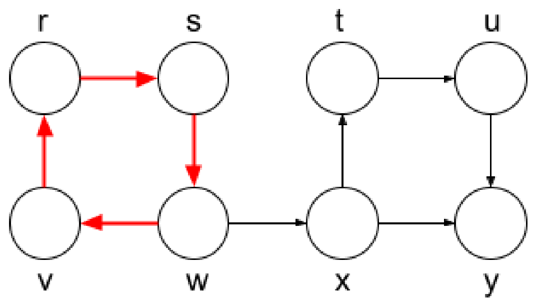


figure 5.1

When we look at figure 5.1, because each vertex has an indegree(u) >= 1, a contained loop is created meaning the graph is not a valid directed acrylic graph. Therefore we prove by contraction that a direct acrylic graph must have a least one vertice where indegree(u) = 0.

We can analyze this further in figure 5.2 by removing the incoming edge from one of the vertices to create the valid direct acrylic graph.

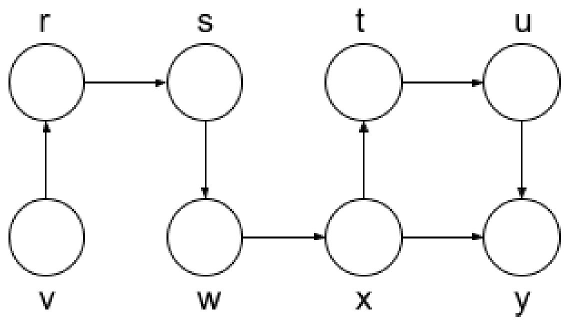


figure 5.2

Q6. [15 points] Given graph G = (V, E), the graph G'= (V', E') is defined such that (u,v) is in E' if and only if G contains a path with at most two edges between u and v. Describe an efficient algorithm for computing G' from G. Once develop the algorithm assuming G has an adjacency-matrix representation and once develop the algorithm assuming adjacency-list representation for the graph. Analyze the running times of your algorithms.

Q7. [10 points] Let G = (V, E) be an undirected graph. Let v be a vertex in G. Give an O(V+E) time algorithm that finds the shortest cycle in G which contains the vertex v.

```
def FindShortestCycle(G):
    V,E = G
    queue = QUEUE()
    shortest_Cycle = ∞
    visit[len(V)]
    range[len(V)]

    for x in V:
        visit[x] = False
        range[x] = 0
        queue.append(x)

        while len(queue) > 0:
            vertex = queue[0]
            visit[vertex] = True
            queue.dequeue()

            for adjacent in E[vertex]:
                if visit[adjacent] == False:
                    range[adjacent] = range[vertex] + 1
                    queue.append(adjacent)
                else:
                    newRange = range[vertex] + range[adjacent]
                    shortest_Cycle = min (shortest_Cycle, newRange + 1)


    if shortest_Cycle is infinity:
         return 0

    return shortest_Cycle
```

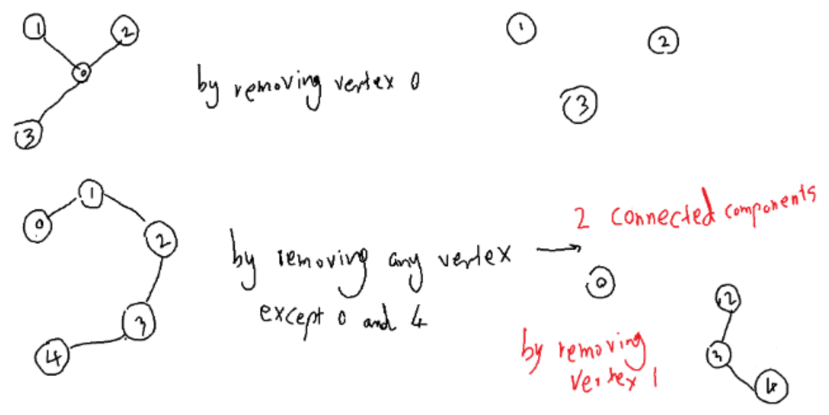
The runtime of the following algorithm is an O(V + E) time complexity. The algorithm iterates through every vertex where it will be added to a queue. From there, a while loop will be used to iterate through the queue.

Q8. [15 bonus points] You are given a tree with |V| vertices and |E| edges. Remember for a tree we have:
|V| = |E|-1

By removing one vertex and all the edges that go from that vertex to some other vertices, the tree will become disconnected and you end up with two or more connected components.

Design an algorithm to return the largest number of components you can get by removing a single vertex of the tree. Develop the pseudocode and analyze the time complexity of your algorithm.

Example: The first one is a tree with four vertices and the second one is a tree with five vertices.


by removing vertex 0
by removing any vertex except 0 and 4
2 connected components
by removing vertex 1