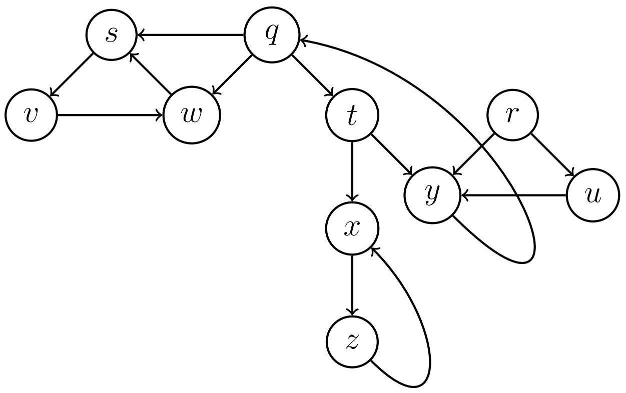
CS6033 Design and Analysis of Algorithms I
Homework Assignment 2 Part 2
10.02.2022

Group Members:
rk4305 (Sai Rajeev Koppuravuri)
sg7372 (Sriharsha Gaddipati)
vt2184 (Veeravenkata Raghavendra Naveenkumar Tata)
vt2182 (Venu Vardhan Reddy Tekula)

1. (5 points) Consider the given graph.



(a) (3 points) Create the adjacency list for the graph above.

Ans:

$\boxed{s} \rightarrow \boxed{v}$

$\boxed{v} \rightarrow \boxed{w}$

$\boxed{w} \rightarrow \boxed{s}$

$\boxed{x} \rightarrow \boxed{z}$

$\boxed{y} \rightarrow \boxed{q}$

$\boxed{z} \rightarrow \boxed{x}$

$\boxed{u} \rightarrow \boxed{y}$

$\boxed{t} \rightarrow \boxed{x}\boxed{y}$

$\boxed{r} \rightarrow \boxed{y}\boxed{u}$

$\boxed{q} \rightarrow \boxed{s}\boxed{w}\boxed{t}$

(b) (2 points) Create the adjacency matrix for the subgraph composed of nodes v, s, w, q and t.

Ans:

Matrix for the sub graph with nodes v, s, w, q and t

The total number of nodes = 5

The adjacency matrix of order 5 x 5 is

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. (8 points) Perform Breadth-first on the above graph given from Q1.

(a) (5 points) Start from vertex q and show elements of q (queue), the d (distance) and π (predecessor) values for each vertex in the queue that result for each iteration of the while loop. Also give the final distances and predecessor nodes for each vertex with q as source:

Ans:

From the given graph, let's take node q as the source point

Insert node q into the queue

queue q = $\begin{array}{|c|c|c|}\hline q & & \\ \hline\end{array}$

distance d = 0

predecessor values π = _

Explore all the children of node q which are not visited so far and insert them into the queue

q = $\begin{array}{|c|c|c|c|c|}\hline q & s & w & t & \\ \hline\end{array}$

d = 1

π = q

Now, remove node q and print it, store it in result list

q = $\begin{array}{|c|c|c|c|c|c|}\hline \cancel{q} & s & w & t & v & \\ \hline\end{array}$

$BFS = [q]$

Now, explore all the children of node s which are not visited so far and insert them into the queue

q = $\begin{array}{|c|c|c|c|c|}\hline \cancel{q} & s & w & t & \\ \hline\end{array}$

d = 1

π = q

Remove the node s from the queue, print it and store it in the result list

q = | ~~q~~ | ~~s~~ | w | t | v | |

$BFS = [q, s]$

Add all the un-visited children of node w

q = | ~~q~~ | ~~s~~ | w | t | v | |

d = 1

π = q

Dequeue node w, print it and store it in the result list

q = | ~~q~~ | ~~s~~ | ~~w~~ | t | v | |

$BFS = [q, s, w]$

Add all the un-visited children of node t

q = | ~~q~~ | ~~s~~ | ~~w~~ | t | v | x | y | |

d = 1

π = q

Dequeue node t, print it and store it in the result list

q = | ~~q~~ | ~~s~~ | ~~w~~ | ~~t~~ | v | x | y | |

$BFS = [q, s, w, t]$

Add all the un-visited children of node v

q = | ~~q~~ | ~~s~~ | ~~w~~ | ~~t~~ | v | x | y | |

d = 2

π = s

Dequeue node v, print it and store it in the result list

q = | ~~q~~ | ~~s~~ | ~~w~~ | ~~t~~ | ~~v~~ | x | y | |

$BFS = [q, s, w, t, v]$

Add all the un-visited children of node x

q = | ~~q~~ | ~~s~~ | ~~w~~ | ~~t~~ | ~~v~~ | x | y | z | |

d = 2

π = t

Dequeue node x, print it and store it in the result list

q = | ~~q~~ | ~~s~~ | ~~w~~ | ~~t~~ | ~~v~~ | ~~x~~ | y | z | |

$BFS = [q, s, w, t, v, x]$

Add all the un-visited children of node y

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	y	z	

d = 2

π = t

Dequeue node y, print it and store it in the result list

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	z	

$BFS = [q, s, w, t, v, x, y]$

Add all the un-visited children of node z

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	z	

d = 3

π = x

Dequeue node z, print it and store it in the result list

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	~~z~~	

$BFS = [q, s, w, t, v, x, y, z]$

There are still some nodes left that are un-visited. Let's now take node r as the source.

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	~~z~~	r	

d = 0

π = _ (there is no connection from the source q

Add all the un-visited children of node r

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	~~z~~	r	u	

d = 0

π = _ (there is no connection from the source q

Dequeue node r, print it and store it in the result list

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	~~z~~	~~r~~	u	

$BFS = [q, s, w, t, v, x, y, z, r]$

Add all the un-visited children of node u

q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	~~z~~	~~r~~	u	

d = 0

π = _ (there is no connection from the source q

Dequeue node u, print it and store it in the result list

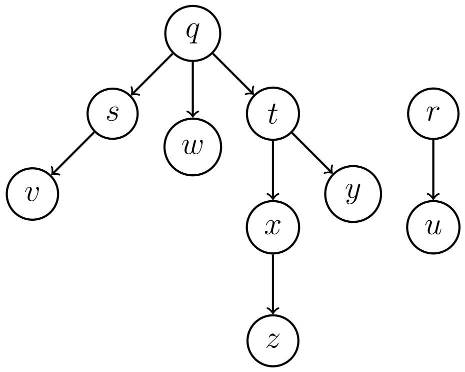
q =

~~q~~	~~s~~	~~w~~	~~t~~	~~v~~	~~x~~	~~y~~	~~z~~	~~r~~	~~u~~	

$BFS = [q, s, w, t, v, x, y, z, r, u]$

The final result is $BFS = [q, s, w, t, v, x, y, z, r, u]$

BFS Graph based on the final result

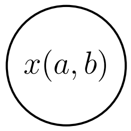


Summary
q is the source node.
The predecessor nodes for w, t and s is q and the distance is 1.
The predecessor nodes for v is s and the distance is 2.
The predecessor nodes for x and y is t and the distance is 2.
The predecessor nodes for z is x and the distance is 3.
The nodes r and u cannot be reached from the node q, so the distance can't be calculated if the source node is q. Predecessor of r is undefined and the predecessor of node u would be r

(b) (3 points) Draw the predecessor sub-graph.

Ans:

Note: For the below predecessor sub-graph notation, please see that x is the name of the node, a is the distance and b is the predecessor.



The predecessor sub-graph for the (a) solution

$$\begin{tikzcd}
 & s(1,q) \arrow[r] & v(2,s) & & & \\
q(0,\_) \arrow[ur] \arrow[r] \arrow[dr] & w(1,q) & & & r(\_,\_) \arrow[dr] & \\
 & t(1,q) \arrow[r] \arrow[dr] & x(2,t) \arrow[r] & z(3,x) & & u(\_,r) \\
 & & y(2,t) & & &
\end{tikzcd}$$

The nodes r and u cannot be reached from the source q.

3. (7 points) Perform DFS on the above graph given for Q1. Assume that the loop for the lines 5-7 of the DFS procedure considers the vertices in alphabetical order and assume the adjacency list is ordered alphabetically. Show the discovery and finish time for each vertex.

   Ans.

   From the given graph, let's take node q as the source point

   Visit node q and append it into the stack

   stack s = $\begin{array}{|c|c|c|}\hline q & & \\ \hline\end{array}$

   $DFS = [q]$

   Visit node s, append it

   stack s = $\begin{array}{|c|c|c|c|}\hline q & s & & \\ \hline\end{array}$

   $DFS = [q, s]$

   Visit node v, append it

   stack s = $\begin{array}{|c|c|c|c|c|}\hline q & s & v & & \\ \hline\end{array}$

   $DFS = [q, s, v]$

   Visit node w, append it

   stack s = $\begin{array}{|c|c|c|c|c|c|}\hline q & s & v & w & & \\ \hline\end{array}$

   $DFS = [q, s, v, w]$

The node w has no univisited adjacency nodes, so pop w
stack s = $\begin{array}{|c|c|c|c|c|}\hline \text{q} & \text{s} & \text{v} & & \\ \hline\end{array}$
$DFS = [q, s, v, w]$

No adjacency for v, pop it
stack s = $\begin{array}{|c|c|c|c|}\hline \text{q} & \text{s} & & \\ \hline\end{array}$
$DFS = [q, s, v, w]$

No adjacency for s, pop it
stack s = $\begin{array}{|c|c|c|}\hline \text{q} & & \\ \hline\end{array}$
$DFS = [q, s, v, w]$

Visit node t, append it
stack s = $\begin{array}{|c|c|c|c|}\hline \text{q} & \text{t} & & \\ \hline\end{array}$
$DFS = [q, s, v, w, t]$

Visit node x, append it
stack s = $\begin{array}{|c|c|c|c|c|}\hline \text{q} & \text{t} & \text{x} & & \\ \hline\end{array}$
$DFS = [q, s, v, w, t, x]$

Visit node z, append it
stack s = $\begin{array}{|c|c|c|c|c|c|}\hline \text{q} & \text{t} & \text{x} & \text{z} & & \\ \hline\end{array}$
$DFS = [q, s, v, w, t, x, z]$

No adjacency for z, pop it
stack s = $\begin{array}{|c|c|c|c|c|}\hline \text{q} & \text{t} & \text{x} & & \\ \hline\end{array}$
$DFS = [q, s, v, w, t, x, z]$

No adjacency for x, pop it
stack s = $\begin{array}{|c|c|c|c|}\hline \text{q} & \text{t} & & \\ \hline\end{array}$
$DFS = [q, s, v, w, t, x, z]$

Visit node y, append it
stack s = $\begin{array}{|c|c|c|c|c|}\hline \text{q} & \text{t} & \text{y} & & \\ \hline\end{array}$
$DFS = [q, s, v, w, t, x, z, y]$

No adjacency for y, pop it
stack s = $\begin{array}{|c|c|c|c|}\hline \text{q} & \text{t} & & \\ \hline\end{array}$

$DFS = [q, s, v, w, t, x, z, y]$

No adjacency for t, pop it
stack s = $\boxed{\text{q}}\boxed{}\boxed{}$
$DFS = [q, s, v, w, t, x, z, y]$

No adjacency for q, pop it
stack s = $\boxed{}\boxed{}$
$DFS = [q, s, v, w, t, x, z, y]$

Since there are more nodes that are un-visited. We will continue with taking source as r.

Visit node r, append it
stack s = $\boxed{\text{r}}\boxed{}\boxed{}$
$DFS = [q, s, v, w, t, x, z, y, r]$

Visit node u, append it
stack s = $\boxed{\text{r}}\boxed{\text{u}}\boxed{}\boxed{}$
$DFS = [q, s, v, w, t, x, z, y, r, u]$

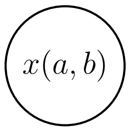
No adjacency for u, pop it
stack s = $\boxed{\text{r}}\boxed{}\boxed{}$
$DFS = [q, s, v, w, t, x, z, y, r, u]$

No adjacency for r, pop it
stack s = $\boxed{}\boxed{}$
$DFS = [q, s, v, w, t, x, z, y, r, u]$

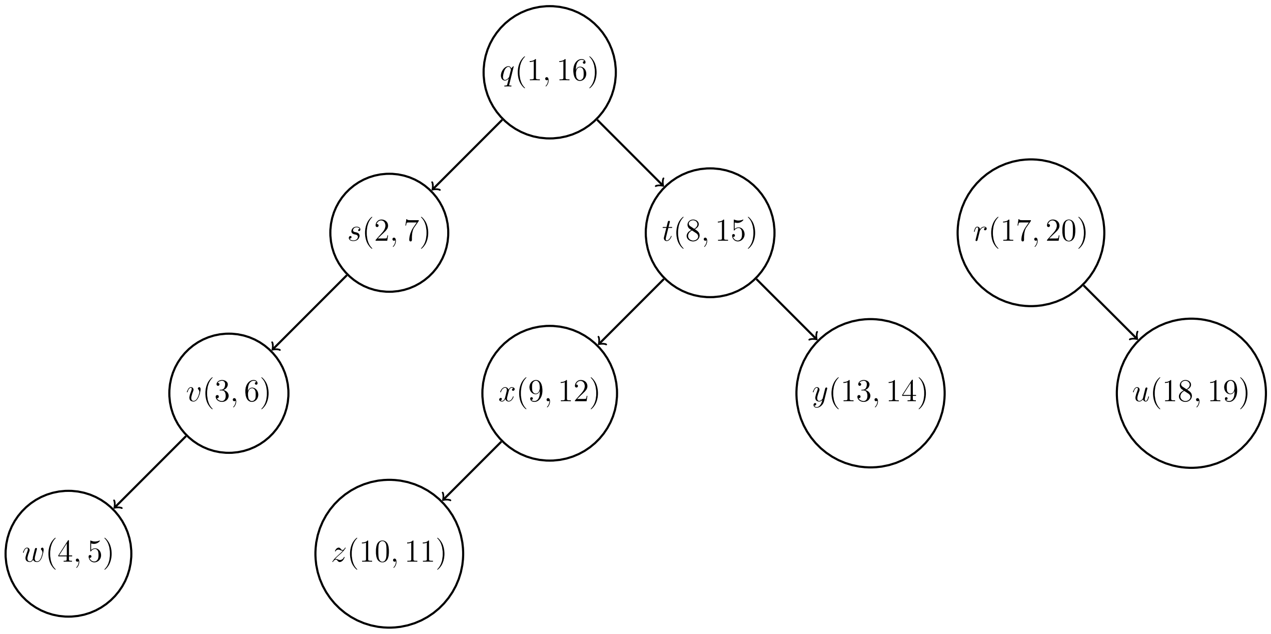
Final, DFS traversal for the given graph $= [q, s, v, w, t, x, z, y, r, u]$

The final result is $DFS = [q, s, v, w, t, x, z, y, r, u]$.

For the below graph notation, please see that x is the name of the node, a is the discovery time, and b is the finish time.



DFS Graph based on the final result



4. (15 points) You are a team bonding coach and have been hired by a company to conduct team bonding activity for its employees. Unfortunately, not all employees get along with each other. Before your session, the company has told you which employees do not get along with each other.

   (a) (12 points) Design an efficient algorithm to determine if/how you can separate the employees into two groups so that in each group the employees do not fight with each other.

   Ans:

   Using Bipartite graph algorithm, we can determine if we can divide people into two groups.

   Create a graph with edge between two persons who doesn't get along with each other.

   Now, Let's assign two different colours to all the nodes/persons in BFS traversal. Finally, person(s)/nodes with same colour belongs to same group. By this, we can get 2 groups with persons in same group never fight among themselves.

```
def isBipartite(graph, src):

    colorArr[src] = 1
    queue = []
    queue.append(src)

    while queue:
        u = queue.pop()
        if graph[u][u] == 1:
            return False;

        for v in range(V):
            if graph[u][v] == 1 and colorArr[v] == -1:
                colorArr[v] = 1 - colorArr[u]
                queue.append(v)

            elif graph[u][v] == 1 and colorArr[v] == colorArr[u]:
                return False
    return True




    if __name__ == '__main__':
        colorArr = [-1] * V
        group_1 = []
        group_2 = []
        if isBipartite(graph, src, colorArr)
            for i in range(len(colorArr)):
                if colorArr[i] == 1:
                    group_1.append(i)
                else:
                    group_2.append(i)
```

(b) (3 points) State your running time.

Ans:

$O(2V + E)$

5. (15 points) You are a well renowned choreographer and have been asked to make students stand in a straight line formation for a drone shot at the Global Citizen event. Some of the students you are dealing with have some animosity with certain other students and would not like to stand behind that particular student in the formation. The university gave you the list of which students don't want to be behind which other students.

(a) (12 points) Design an efficient algorithm to determine the order of the students for the formation. In-case there is a cyclic deadlock and you cannot form an order return saying ”Cycle Encountered”.

Ans:

We can solve this problem using Cycle Detection at first and then if there is no cycle detected in graph, then we are good to apply topological sort to get an ordering of students. We cannot achieve ordering of students if there is cycle detected in the given graph.

Initially let us formulate this problem into a graph. By assuming nodes as students, create an edge from student u to $\rightarrow$ student v such that v does not want to be behind u.

```
def IsCyclicDFS(v, vis, recStack):
    vis[v] = 1
    recStack[v] = 1

    for adj in graph[v]:
        if vis[adj] == 0:
            if isCyclicDFS(adj, vis, recStack) == True:
                return True

        elif recStack[adj] == 1:
            return True

    recStack[v] = 0
    return False

def isCyclic(graph, V):

    vis = [0] * (V + 1)
    recStack = [0] * (V + 1)
    for node in range(V):
        if vis[node] == 0:
            if isCyclicDFS(node,vis,recStack) == 1:
                return True
    return False

def topologicalSortDFS(v, vis, stack):
    vis[v] = 1
    for i in graph[v]:
        if vis[i] == 0:
            topologicalSortDFS(i, vis, stack)
    stack.append(v)
```

```
def topologicalSort(graph, V):
    vis = [0]*self.V
    stack = []

    for i in range(self.V):
        if vis[i] == 0:
            topologicalSortDFS(i, vis, stack)

    print(stack[::-1])

if __name__ == '__main__':
    if isCyclic(graph, V):
        print("Cycle Encountered")
    else:
        topologicalSort(graph, V)
```

(b) (3 points) State the running time of your algorithm.

Ans:

$O(V + E)$