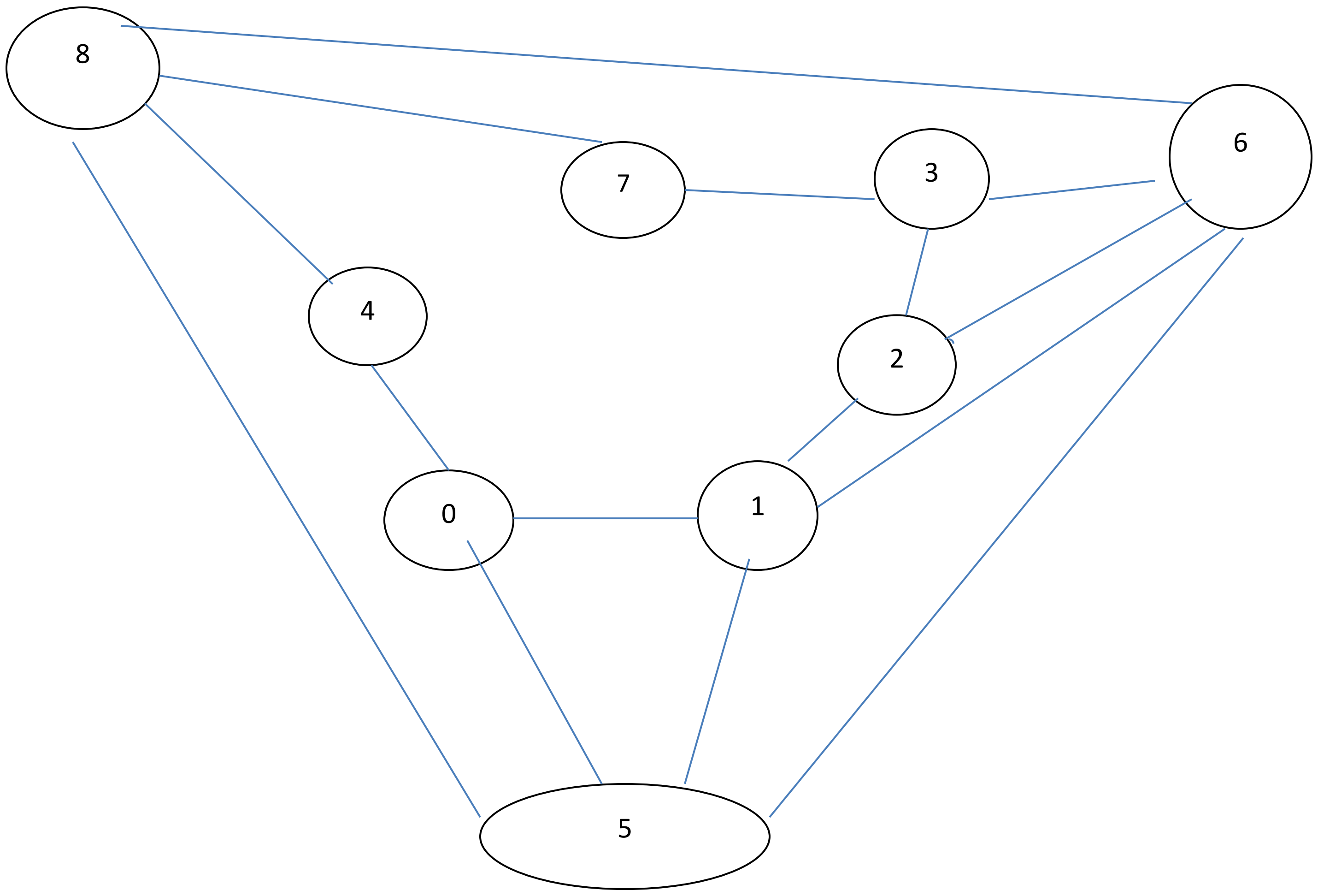
Mamoutou

Sangare

CSC 225

V00010526

Question 1-a



b- An adjacency list representation of G:

Vertex	List containing Adjacent Vertices
0	1 -> 4 ->5
1	0 -> 2 -> 5 -> 6
2	1-> 3 -> 6
3	2->6 ->7
4	0->8
5	0 -> 1-> 6-> 8
6	1 ->2 ->3 ->5 ->8
7	3-> 8
8	4 -> 5 ->6 ->7

C - The adjacency matrix representation of G is symmetry since G is an indirect graph

	0	1	2	3	4	5	6	7	8
0	0	1	0	0	1	1	0	0	0
1	1	0	1	0	0	1	1	0	0
2	0	1	0	1	0	0	1	0	0
3	0	0	1	0	0	0	1	1	0
4	1	0	0	0	0	0	0	0	1
5	1	1	0	0	0	0	1	0	1
6	0	1	1	1	0	1	0	0	1
7	0	0	0	1	0	0	0	0	1
8	0	0	0	0	1	1	1	1	0

d - I –Ordering the vertices as they are visiting using DFS starting at 0:

0 1 2 3 6 5 8 4 7

II – Ordering the vertices as they are visiting using BFS starting at 0

0 1 4 5 2 6 8 3 7

Question 2

We know that a forest a union of tree.

Let T be a tree with n vertices. We know a tree with n vertices has n-1 edges (m = n - 1; where m being edges and n being vertices). Since a forest is a union of trees, then

$F = T_1 \cup T_2 \cup T_3 \cup \ldots \cup T_k = \bigcup_{i=1}^{k} T_i$

Now let n_i be the number of vertices in each tree then $| V(F) | = \sum_{i=1}^{k} n_i = n$

Since each component of the forest is a tree, then we have $n_i - 1$ edges for T_i.

Therefore $|E(F)| = m = \sum_{i=1}^{k} n_i - 1 = n - k$ as desired.

Question 3

a –Algorithm TopologicalSort(G)

input: digraph G with n vertices

output: topological ordering of G if there is a unique or indication of no unique topological order exists or no topological exists

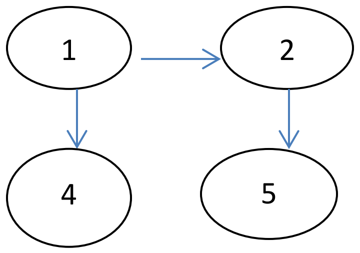
```
S ← Empty stack
count1 ← 0
count2  ← 0
for u ϵ G.vertices() do
    incounter (u) ← indeg (u)
    if incounter(u) = 0 then
        S.push(u)
        count1 ← count1 + 1  // if we push more than once than there is not unique topological ordering
i←1
while not S.isEmpty() do
    u ← S.pop()
    number u as vertex v(i)
    i ← i + 1
    if count2 = 1
        count2 ←0
    for all e u ϵ G.outIncidentEdges(u)
        w ← opposite(u,e)
        incounter(w) ← incounter(w) -1
        if incounter(w) = 0 then
            S.push(w)
           count2 ← count2 + 1   // if we push more than once than one then unique topological ordering

 if i <= n then
    return '' G has a direct cycle, no topological sort exits”
if count1 > 1 || count2 > 1
   return  “There is not a unique topological sort''
```

return $v_1, v_2, \dots, v_n$

b – i) True. Since the definition of a post-order traversal is to visit all of the children before visiting the children's parent which is precisely the definition the property of the topological ordering. An example to show that by assuming the direct graph as defined in the question,

let G be a directed graph with vertices 2,1,4,5



then any post order traversal is a topological; for instance ,1,2,4,5 and 1,4,2,5 are a post order traversal and a topological ordering.

II- True. We will prove by contradiction. Suppose a graph has a topological ordering, and a depth- first traversal of the same directed graph has at least a back edge. Since the graph has a back edge, the graph will not be a direct acyclic graph (DAG), it will have a cycle; which is a contradiction since the topological ordering required the graph to be acyclic (no cycle). Therefore, if a graph has a topological ordering, then a depth first traversal of the same directed graph will not see any back edges.

Question 4

Input: A group of people (undirected graph)

Output: a subgroup of people who are all friends with each other (subgraph)

We can represent the problem as a graph by representing people as vertices and the friendship relation among them as edges. We will end up with a forest which may contain multiple trees. And the tree in the forest with the most vertices is the output subgroup that we are looking for, so we can return that tree.

We take a person (A) from the input group and become building our forest. We take a second person (B) and ask if A is friend with B. If the answer is yes, then put an edge between A and B (A-B), if the answer is not, we will have two connected components. In other words, there will not any edge between A and B. We take a third person (C) in the input group and ask if C is friend with A. if C is friend with A, we put an edge between A and C (A-C). If the answer is not (C is not friend with A) we ask if C if is friend with B. If yes, we put an edge between B and B(C-B). If not, we will three connected components. We do this with the remaining people in the input group. When we finish, the tree with the most vertices in the forest will be the group of people who are all friends with each other. We return that group.