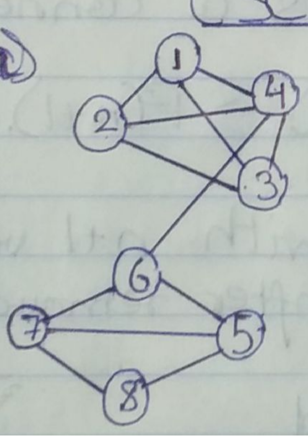
15/03/16 CSC Assignment 4. Dwoop Banerjee V00837868

1. a)



b) 1, 2, 3, 4, 6, 7, 8, 5.

c) 1, 2, 3, 4, 6, 5, 7, 8.

2. Adjacency lists -

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

Adjacency matrix -

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

3. $P(n)$ = A connected graph with n vertices has 2 distinct vertices, each of which can be removed individually (including all adjacent vertex) without disconnecting the remaining graph.

Base Case:- $P(2)$: Either one of the vertices can be removed. The remaining graph is a single vertex in both cases, which is a connected graph.

Inductive Case:- $P(2) | \cdots | P(n) \to P(n+1)$.

G = Connected graph with n+1 vertices.
G′ = " " after removal of vertex v

Case 1:- G′ is connected

- Since $P(n)$ is true (Inductive Hypothesis), there are 2 distinct vertices w & u that can be removed one by one from G′ without disconnecting the graph.
- If w was removed from G rather than v, G′ would still be connected.

Scanned by CamScanner

- Thereby, 2 distinct vertices can be removed from G, give us G' (still connected).

Case 2:- G' is disconnected

Then G' is made up of k connected component subgraphs $\{g_1, g_2, \dots, g_k\}$. $k \geq 2$ because with atleast $n+1$ vertices in G with $n \geq 2$, the remaining n vertices ($n \geq 2$) must be in separate components if G' is disconnected. Each connected component, g_i, has a vertex, v_i, that was adjacent to v in G.

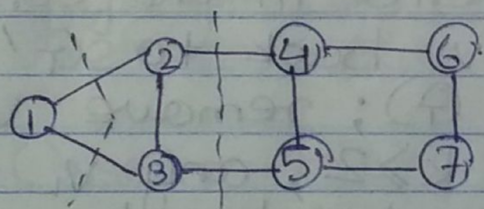
↳ Case 2.1 – For every g_i, v_i is the only vertex in the connected component.
- Add v back to G' (this gives us back G); remove v_1 instead.
- Since $k \geq 2$, any v_k could be removed
- A vertex has been removed from G in 2 ways, and both result in connected graphs.

↳ Case 2.2 – Atleast one of g_i (called g_x) has 2 or more vertices.
- The Strong Inductive Hypothesis applies to g_x, so there are 2 distinct vertices that can be removed from g_x without disconnecting it.
- Since there are two, and they are distinct, atleast one of them is not v_x. Call the other one x.
- Add v back to G' to get back G.
- Already identified one vertex x, which can be removed without disconn-ecting G.

- If g_p is just one vertex, it can be removed as in Case 2.1.
- If g_p is more than one vertex then Induction Hypothesis enables removal of one vertex V_p as in case 2.2.
- The vertex removed from g_p accounts for the second vertex that could be removed from G without disconnecting it.

∴ The proof above, P(n+1) allows the removal of 2 distinct vertices from G with disconnecting it, ie:- it can be applied to one vertex and its adjacent vertices.

eg:-



G

Node ① or even nodes ①, ② & ③ can be removed without disconnecting G.

Algorithm-
The graph can be traversed in postorder. Since post order follows the idea of Left child-Right child-Root node, it will visit the root node last. The root node usually holds the least value (depending on the tree). Once the node with the least value is reached, it can be removed. Since the root node lies on the outer ring/circle of the tree, removing it would not disconnect the graph.

```
Traverse through the graph using DFS, starting at vertex v{
    Create an empty stack S;
    For each vertex u, set visited at vertex u = false;
    Push v on S;
    while(S is not empty){
        u = pop S;
        if(vertex at position u is not visited){
        set visited at position u = true;
        for(each unvisited neighbour, n, of vertex u){
            Push n onto S;
        }
        }
    }
}
Check if the vertex can be removed without disconnecting G{
    Find root node;
    Check if root has one child;
    if(root node has one child){
        Print "found vertex at position u";
    }
    if(vertex u has no children){
        Print "found vertex at position u";
    }
    if(there is a non-tree edge going above u from a sub-tree
                                   below a child of u){
        Print "found vertex at position u";
    }
}
```

4. Each component of F is a tree, hence it has $F_1, F_2, F_3, \ldots, F_k$ components, each of which is a tree. Since trees have no cycles, and has $n-1$ edges, for each $i \in [k]$, component F_i has $|F_i| - 1$ edges. Thus F has

$\sum_{i=1}^{k} (|F_i| - 1) = n-k$ edges.

$\therefore$ F = n-k number of edges

5. Define an empty list L;
Create a stack, S;
Push all vertices with no incoming edges onto S;
while (S is not empty){
    pop element off S and store in variable x;
    add x to L;

for(all the vertices w with an edge e from x to w){
    remove e from graph;
}
if(w has no other incoming edges){
    push w onto S;
}
if(graph has edges left or w = x){
    Print "Has no unique topological order";
}else{
    Print "Has unique topological ordering";
}
}
}