Assignment 3

Graph

Ehu Shubham Shaw
Joe Johnson
March 1, 2025

Question 1 ***Solution*****:** To determine the number of topological orderings, here we examine the many methods to arrange vertices while maintaining interdependence. Looking at the graph, there we find edges connecting vertices a, b, c, d, e, and f. A → B → D → C → F → E → F.

Starting with the single source vertex 'a', I can select between 'b' and 'd' for the following vertex. Following through on all possible valid orders:

If I choose 'b' after 'a', I must put 'c' before 'f', but I can arrange 'd' and 'e' in several ways if I select 'd' after 'a', I have to place 'e' before 'f', but I can arrange 'b' and 'c' in numerous ways sown below

counting all valid orderings: a, b, c, d, e, f -> a, b, d, c, e, f -> a, b, d, e, c, f -> a, d, b, c, e, f -> a, d, e, b, c, f -> a, d, b, e, c, f,

Therefore, this graph has 6 possible topological orderings.

Question 2 ***Solution*****:** I'll use a depth-first search approach to detect cycles in an undirected graph we can consider the below pseudocode I am giving below

```
DetectCycle(G):
   Mark all vertices as unvisited
   Initialize empty stack as lets say S for tracking the cycle path
   for each vertex v in G:
      if v is unvisited:
         if DFSCycleCheck(G, v, NULL, S) returns true:
            Output cycle from stack S
            return true;
   return false

DFSCycleCheck(G, current, parent, S):
   Mark current as visited
   Push current onto stack S
   for each neighbor u of current:
      if u is unvisited:
         if DFSCycleCheck(G, u, current, S) returns true:
            return true
      else if u is not the parent of current:
         Push u onto stack S
         return true
         Pop current from stack S
  return false
```

Because each vertex is processed once, the time complexity is O(n), and each edge is O(m), resulting in a total complexity of O(m+n). The procedure operates by executing DFS on each unvisited vertices. When we reach an already visited vertex that is not the current vertex's parent, we have discovered a cycle. The stack S monitors the current path, which might be output as a cycle when one is identified.

Question 3 Solution: So let us consider 3 cace n2 = number of nodes with two children, n1 = number of nodes with one child N0 = number of leaves > nodes with zero children here so need to prove that n2 = n0 - 1.

Base case**:** For a binary tree with just one node, we have n0 = 1 (one leaf node), n2 = 0 (no nodes with two children) Therefore, n2 = 0 = 1 - 1 = n0 - 1, which holds.
Inductive hypothesis**:** Assume that for any binary tree with k nodes, the relation n2 = n0 - 1 holds.
lets consider adding one new node to this tree this new node must become a child of an existing node here two possible cases are possible
Case 1: Adding to a leaf node:

- The leaf becomes a node with 1 child
- The new node becomes a leaf
- n0 remains unchanged (lost one leaf, gained one leaf)
- N2 remains unchanged
- The relation n2 = n0 - 1 still holds

Case 2: Adding to a node with 1 child:

- This node becomes a node with 2 children
- The new node becomes a leaf
- n0 increases by 1
- N2 increases by 1
- If previously n2 = n0 - 1, now (n2 + 1) = (n0 + 1) - 1, which simplifies to n2 = n0 - 1

By mathematical induction, for any binary tree, n2 = n0 - 1.

Question 4 Solution: so by using contradiction lets assume G !=(not equal) T. This means there exists at least one edge (v, w) in G that is not in T since T spans all vertices of G, both v and w are in T let's consider the unique path between v and w in T.
If edge (v, w) exists in G but not in T, there are two possibilities regarding the positions of v and w in T:

- If w is an ancestor of v in T, then:
  - In BFS, v would be discovered from its parent in T
  - In DFS, v could be discovered directly from w if the DFS visits w before v's parent
  - This creates different trees for DFS and BFS, contradicting our assumption
- If w is at the same level as v in T, then:
  - In BFS, both v and w would be discovered at the same level
  - But when edge (v, w) exists, either v would discover w or w would discover v in DFS
  - This again creates different trees, contradicting our assumption

If v and w belong to distinct branches of T, BFS will find them through their parents, but DFS will find them directly by visiting one first.
Again, producing diverse trees.
Because all alternatives result in contradictions, there cannot be such an edge (v, w), thus G must equal T.

Question 5 Solution: So the claim is **true**. because of the contradiction: Assume G is not connected. Then G contains at least two components, let them be C_1 and C_2. So Let $|C_1| = k$ represent the number of vertices in the first component. Then $|C_2| \leq n - k$ (since there could be more than two components).
Consider a vertex v in C_1 that has a degree of at least n/2.
Since G is not connected, all edges from v must connect to other vertices in C_1. This means that C_1 must have at least n/2 + 1 vertices, including v. So $k \geq n/2 + 1$
Similarly, any vertex w in C_2 must have at least n/2 degree.
Since all edges from w connect to other vertices in C_2, there must be at least n/2 + 1 vertices.

This indicates $|C_1| + |C_2| \geq (n/2 + 1) + (n/2 + 1) = n + 2$, which is greater than the total number of vertices, n. This inconsistency demonstrates that G must be related. As a result, the property that each device is within 500 meters of at least n/2 other devices ensures that the network remains connected, assuming n is even.