1. **Prove or Disprove: An undirected graph is a forest if and only if every edge is a bridge.**

The statement is true. We use the lemma: "An edge is a bridge iff it belongs to no cycle."

(=>) If the undirected graph is a forest, then every edge is a bridge.

A forest is acyclic, so no edge belongs to a cycle. By the lemma, every edge is a bridge.

(<=) If every edge is a bridge, then the undirected graph is a forest.

Suppose, for contradiction, that the graph has a cycle. Then any edge in that cycle is not a bridge, which contradicts our lemma. Therefore, if there are no cycles, then the graph is a forest.

2. **Prove or Disprove: In an undirected connected graph, each endpoint of a bridge is either of degree 1 or an articulation point.**

The statement is true.

Let e be a bridge connecting nodes x and y. We show that x either has degree 1, or has an articulation point.

Suppose, for contradiction, that x does not have degree one and is not an articulation point. Since it does not have a degree 1, and x has an incident edge e, deg(x) >= 2. x must have another incident edge (x,z) where z != x, and possibly (z = y).

Since x is not an articulation point, removing x leaves the graph connected, there exists a path from z to y avoiding x. This path, z to y, together with edges (x,z) and (x,y) forms a cycle containing (x,y), contradicting that (x,y) is a bridge since bridges do not belong to any cycle.

Therefore, the assumption is false. x must have degree 1 or it is an articulation point.

3. **Recall that in Kosaraju's algorithm, one needs to "reverse" the graph. Another term for the graph with all the edge orientations reversed is the graph's transpose. Let M be the adjacency matrix of a graph and let N be the adjacency matrix of its transpose. How are M and N related?**

N is the transpose of M, in matrix terms, it is M flipped along the main diagonal.

4. **Consider an undirected graph.**

4.1. Prove that a back edge can never be a bridge.

Let e = (x,y) be a back edge. Thus, y is an ancestor of x in the DFS forest, so there is a path from y to x using tree edges. These tree edges, combined with the back edge, forms a cycle. Therefore, e is not a bridge.

4.2. Suppose i is the parent of j in the DFS forest. Prove that (i,j) is a bridge iff low[i] > disc[i].

True. We prove both directions of the statement:

[=>] If (i,j) is a bridge, then low[j] > disc[i].

- (i,j) being a bridge means there's no other path from j to any ancestor of i except (i,j). Hence, every path from j stays within the subtree rooted at j.
- Therefore, the earliest discovery time reachable from j, including via back edges, is at least disc[j] > disc[i].
- Thus, low[j] > disc[i].

[<=] If low[j] > disc[i], then (i,j) is a bridge.

- low[j] > disc[i] means there is no back edge from j's subtree that reaches i or any ancestor of i.
- Hence, the only connection from j's subtree to the rest of the graph is through the tree edge (i,j).
- Removing (i,j) would disconnect j's subtree from the rest, therefore, (i,j) is a bridge.

4.3. Suppose i is not a root in the DFS forest. Prove that i is an articulation point if and only if i has a child j such that low[j] >= disc[i].

We prove both sides of the statement:

[=>] "If i is an articulation point, then i has a child j such that low[j] >= disc[i]."

- Removing i disconnects the graph, so some pair of nodes in different subtrees, or one in a subtree and one outside, lose connection.
- Take child j whose subtree contains a node disconnected from outside.
- If low[j] < disc[i], then j's subtree could reach an ancestor of i via a back edge, remaining connected after removing i, which is a contradiction. Therefore, low[j] >= disc[i].

[<=] “If i has a child j such that low[j] >= disc[i], then i is an articulation point.

- low[j] >= disc[i] means that there is no back edge from j’s subtree that reaches any ancestor of i.
- Thus, the only connection from j’s subtree to the rest of the graph, including i’s parent, is through i.
- Removing i would disconnect j’s subtree from i’s parent and the rest of the graph.
- Therefore, i is an articulation point.

4.4. Suppose i is a root in the DFS forest. Prove that i is an articulation point if and only if i has at least two children.

We prove both sides of the statement.
[=>] “If i is an articulation point, then i has at least two children.”
If i had only one child, removing i would leave its single subtree still connected via tree edges, so the graph remains connected. This would contradict i being an articulation point, hence i must have at least two children.

[<=] “If i has at least two children, then i is an articulation point.”

- Let j and k be two distinct children of i.
- Any path from j to k must go through i because:
  - tree edges only go downward within each child’s subtree
  - back edges go upward but cannot jump directly between different subtrees without passing through i (as i is the root and has no ancestors above it).
- So, removing i disconnects j and k, which means that i is an articulation point.

5. Consider a connected undirected graph.

5.1. Prove or Disprove: removing a bridge doesn’t turn any edge from a non-bridge to a bridge.

True.

- An edge is a bridge if and only if it does not belong to any cycle.
- Consider a non-bridge e in some cycle C.
- Remove a bridge f that is not part of the cycle C, then cycle C remains intact.
- e is still in a cycle, and e is still a non-bridge.

5.2. Prove or Disprove: removing a bridge doesn’t turn any edge from a bridge to a non-bridge.

True. Consider bridge e, which is not in a cycle. Recall that an edge is a bridge if and only if it does not belong to any cycle. Removing a bridge, or any edges for that matter, cannot create new cycles. After removal, e would still not be in a cycle, and e remains a bridge.

5.3. Prove or Disprove: removing a non-bridge doesn't turn any edge from a non-bridge to a bridge.

False. Consider a simple cycle of 3 nodes: a-b-c-a. Initially, no edges are bridges because they're all in a fucking cycle. Remove any edge, and it becomes a path, which means every edge remaining is a bridge. Therefore, removing a non-bridge can lead to turning other non-bridges into a bridge.

5.4. Prove or Disprove: removing a non-bridge doesn't turn any edge from a bridge to a non-bridge.

True. Consider bridge e, which is not in a cycle. Removing edges cannot create new cycles. After removing a non-bridge, e would still not be in a cycle, and e remains a bridge. Therefore, removing a non-bridge does not turn any edge from a bridge to a non-bridge.

6. **Given an undirected graph G, we say two nodes x and y are 2-edge-connected if there is a path from x to y that doesn't pass through a bridge.**

A 2-edge-connected component is a maximal 2-edge-connected subgraph; that is, a subgraph such that every pair of nodes is 2-edge-connected, and it is maximal with this property: no additional nodes or edges can be added without breaking the property.

6.1. Prove that if one removes a bridge, then the 2-edge-connected-components remain the same.

True. Let G be a graph and f be a bridge in G. Let G' = G - f.

We use the following key lemmas:

1. 2-edge-connected components are equivalence classes under relation $x \sim y$ iff there exists a path from x to y containing no bridge, which is exactly how 2ECCs are defined.
2. For any nodes x,y, $x \sim y$ in G iff $x \sim y$ in G'.

We prove the second lemma:

- If $x \sim y$ in G, there is a path P with no bridges. Bridge f is not part of P since f is a bridge. So, removing f doesn't affect P, and by 5.1., no edge in P becomes a bridge after removal. So P remains a bridge-free path in G', so $x \sim y$ in G'.

- If x ~ y in G', then in G either the same path works, or if it used f, that would be impossible since f is removed, but also f is a bridge in G so any path using f contains a bridge anyway and would thus, not be a part of the 2ECC. Thus, x ~ y in G.

Since the equivalence relation is preserved, the equivalence classes (2-edge-connected components) remain the same after removing a bridge.

6.2. Prove that the 2-edge-connected components are precisely the connected components if one removes all bridges from the graph.

True. Let G be a graph and G' be G with all bridges removed.

By 6.1, removing bridges one by one preserves the 2-edge-connected components. After removing all bridges, consider any connected component H of G':

- Any path between two nodes in H contains no bridges, since all bridges have been removed.
- Hence every pair of nodes in H is 2-edge-connected, in which H is contained in some 2ECC.
- Conversely, nodes in the same 2ECC of G are connected in G' since their connecting bridge-free path remains after removing all bridges. Thus, they lie in the same connected component of G'.

Thus, the 2ECCs of G are exactly the connected components of G'.

6.3. Let C[i] be the 2-edge-connected component containing node i. Consider a graph B(G) whose node set is the set of 2-edge-connected components of G, and for each bridge (x,y) in G, we add an edge (C[i], C[j]) in B(G). Prove that B(G) is a bridge forest of G.

We show that B(G) contains no cycles.

Suppose, for contradiction, that B(G) has a cycle: X_0-X_1-X_2-...-X_{k-1}-X_0 with k >= 1. Each edge (X_p, X_{p+1}) in B(G) corresponds to a bridge (a_p, b_{p+1}) in G where $a_p \in X_p$,$b_{p+1} \in X_{p+1}$. Since X_p is a 2ECC, there exists a bridge-free path between b_p and a_p within X_p. Concatenating these bridges and bridge-free paths gives a cycle in G:

$$a_0 \text{ -> } b_1 \text{ -> } a_1 \text{ -> } b_2 \text{ -> } a_2 \text{ -> } \dots \text{ -> } a_{k-1} \text{ -> } b_0 \text{ -> } a_0$$

This cycle contains bridge (a_0, b_1) or any such (a_p, b_{p+1}), contradicting that bridges belong to no cycle. Hence, no cycle in B(G) means that B(G) is a forest.

6.4. Find an O(n+e) algorithm that constructs the bridge forest of a graph.

We use the following algorithm:

1. Find all bridges in G using DFS
    a. Run DFS from any node, computing disc[u] and low[u] for each node.
    b. Edge (u,v) is a bridge iff low[v] > disc[u], where u is a parent of v.
    c. This runs in O(n + e).
2. Next, we remove all bridges from G to get G'.
    a. Each connected component of G' is a 2-edge-connected component (2ECC).
    b. Assign each node a component ID via BFS/DFS on G'.
    c. This runs in O(n + e).
3. Next, we build B(G).
    a. Let nodes = 2ECCs, with one per component ID.
    b. For each bridge (x,y) in original G:
        i. Let C_x = component IDof x, C_y = component ID of y.
        ii. Add edge (C_x, C_y) to B(G) if not already present.
    c. This takes O(e), processing every bridge once.

The time complexity is therefore O(n + e) + O(n + e) + O(e) = O(2(n + e)) + O(e) = O(n + e).

7. Consider the following problem:

> **Problem 4.** Mr. Juan Cho owns a computer shop in the rural town Sacramen-Cho. The computers in Cho's shop are connected to each other via LAN cables. Each LAN cable connects two distinct computers together. There are $n \geq 2$ computers and $c \geq 0$ LAN cables, and it is guaranteed that every pair of computers are connected via the LAN cables—that is, for every pair of computers x and y, there is a path consisting of a sequence of LAN cables from x to y.
>
> Unfortunately, the LAN cables are not very reliable. A LAN cable sometimes breaks. If one breaks, it takes a while to replace it, and it might temporarily disconnect the network!
>
> Mr. Juan Cho is tired of this. He wants to add a single new LAN cable that will fix things. To do this, he will choose two computers and add a new LAN cable between them.
>
> Is it possible to add at most one new LAN cable so that in the resulting network, the computers will still be connected to each other even if any single cable breaks? If it is possible, identify a pair of computers where you can add the new LAN cable to satisfy the above.

7.1. Find an algorithm that solves Problem 4 in $O(n^2(n+c))$ time.

For every unordered pair of distinct nodes (u,v):

1. We temporarily add an edge (u,v) to G to get G'.
2. Find all bridges in G' using DFS (disc[], low[]).
    a. Run DFS, computing discovery times and low values.
    b. Edge (x,y) is a bridge iff low[y] > disc[x], where x is a parent of y.
    c. So the time is $O(n + c)$ per DFS on $O(n^2)$ pairs.
3. If no bridges are found in G', return (u,v) as solution.

If no pair works after checking all $O(n^2)$ pairs, return None. $O(n+c)$ DFS * $O(n^2)$ pairs = $O(n^2(n+c))$.

7.2. Find an algorithm that solves Problem 4 in O(n+c) time.

1. First compute the bridge forest B(G):
    a. find all bridges via DFS $O(n + c)$.
    b. Remove bridges to get 2ECCs.
    c. Build B(G), with nodes = 2ECCs, edges = bridges,
2. Analyze B(G):
    a. B(G) is a tree, a connected forest.
    b. Count its leaves, nodes with degree 1 in B(G).
3. Now, we decide based on leaf count:
    a. Case 1: B(G) has 1 node: G already has no bridges, so any pair works.

b. Case 2: B(G) has exactly 2 leaves: X and Y so B(G) is a path.
    i. pick any x in X and any y in Y and add edge (x,y).
    ii. this turns the path into a cycle, so all bridges are now nonbridges. thus solving the problem.
c. Case 3: B(G) has >= 3 leaves, then the task is impossible, since any new edge connects at most two leaves. In the best case (3 leaves), the third leaf would remain degree 1 in B(G) and its incident bridge remains a bridge, leaving the problem unsolvable.

Time complexity: O(n+c) for BFS + O(n) for counting leaves = O(n+c).