(a) Articulation + Bridge

```
Function DFS ()
clock = 1;
foreach $v$ in $V$ do
    parent[$v$] = $null$;
    visited[$v$] = $false$;
    is_articulate[$v$] = $false$;
    child[$v$] = 0;
    low[$v$] = 0;
    pre[$v$] = 0;
end
foreach $(u, v)$ in $E$ do
    is_bridge[$u$][$v$] = $false$;
end
foreach $v$ in $V$ do
    if visited[v]==$false$ then
        explore($v$);
        if child[v]>1 then
            is_articulate[$v$] = $true$;
        end
    end
end

Function explore ($z$);
pre[$z$] = low[$z$] = clock ++;
visited[$z$] = true;
foreach $(z, w)$ in $E$ do
    if w==parent(z) then
        continue;
    end
    if visited[w]==false then
        child[$z$] ++;
        parent[$w$] = $z$;
        explore($w$);
        low[$z$] = min(low[$z$], low[$w$]);
        if low[w]≥pre[z] and parent[z]!=null then
            is_articulate[$z$] = $true$;
        end
        if low[w]>pre[z] then
            is_bridge[$z$][$w$] = $true$;
        end
    else
        low[$z$]=min(low[$z$], pre[$w$]);
    end
end
```

(b) Articulation + Bridge + Biconnected components

```
Function DFS ()
clock = 1;
bcc_count = 0; // counter for biconnected components
foreach $v$ in $V$ do
    parent[$v$] = $null$;
    visited[$v$] = $false$;
    is_articulate[$v$] = $false$;
    child[$v$] = 0;
    low[$v$] = 0;
    pre[$v$] = 0;
end
foreach $(u, v)$ in $E$ do
    is_bridge[$u$][$v$] = false;
end
foreach $v$ in $V$ do
    if visited[v]==$false$ then
        explore($v$);
        if child[v]>1 then
            is_articulate[$v$] = $true$;
        else
            bcc_count ++;
        end
        while stack.empty()==$false$ do
            edge = stack.top();
            print(edge);
            stack.pop();
        end
    end
end
```

```
Function explore ($z$);
pre[$z$] = low[$z$] = clock ++;
visited[$z$] = true
foreach $(z, w)$ in $E$ do
    if $w$==parent(z) then
        continue;
    end
    if visited[w]==false then
        child[$z$] ++;
        parent[$w$] = $z$;
        stack.push(($z$,$w$));
        explore($w$);
        low[$z$] = min(low[$z$], low[$w$]);
        if low[w]≥pre[z] and parent[z]!=null then
            is_articulate[$z$] = true;
            bcc_count ++;
            while true do
                edge = stack.top();
                print(edge);
                stack.pop();
                if edge == $(z, w)$ then
                    break;
                end
            end
        end
        if low[w]>pre[z] then
            is_bridge[$z$][$w$] = true;
        end
    end
    else if pre[w]<pre[z] then
        stack.push(($z$,$w$));
        low[$z$]=min(low[$z$], pre[$w$]);
    end
end
```