Biconnected Components Algorithm

To output the biconnected components of an undirected graph G, we modify the algorithm for finding articulation vertices so that whenever it encounters a new edge in G, it pushes it on a *stack*. Also after we finish the recursive search from a child v of vertex u, we check if u is an articulation vertex for v. If so, we output all edges on the stack upto and including (u, v). Notice that when we return to the root we need to output a biconnected component even if the root is not a cut-vertex. (For example, the graph may not have *any* cut-vertices.)

```
proc BiconnectedComponents;
    count ← 0;
    Initialize(stack);
    for each u ∈ V[G] do visited[u] ← false;
    for each u ∈ V[G] do parent[u] ← nil;
    for each u ∈ V[G] do
        if not visited[u] then DFS-Visit[u];
end proc;


proc DFS-Visit(u);
    visited[u] ← true;
    count ← count+1;
    d[u] ← count;
    low[u] ← d[u];
    for each v ∈ Adj[u] do
        if not visited[v] then
            Push(stack, (u,v));
            parent[v] ← u;
            DFS-Visit(v);
            if (low[v] ≥ d[u]) then OutputComp(u,v);
            low[u] ← min(low[u],low[v]);
        endif
        else if (not(parent[u]=v) and (d[v] < d[u]) then
            (* (u,v) is a back edge from u to its ancestor v *)
            Push(stack, (u,v));
            low[u] ← min(low[u],d[v]);
        endif;
    end for;
end proc;


proc OutputComp(u,v);
    Print(" New Biconnected Component Found");
    repeat
        Pop(stack, e);
        Print(e);
    until (e = (u,v));
end proc;
```