
# Competitive Programming

From Problem 2 Solution in O(1)

## Graph Theory

## Articulation points using Tarjan

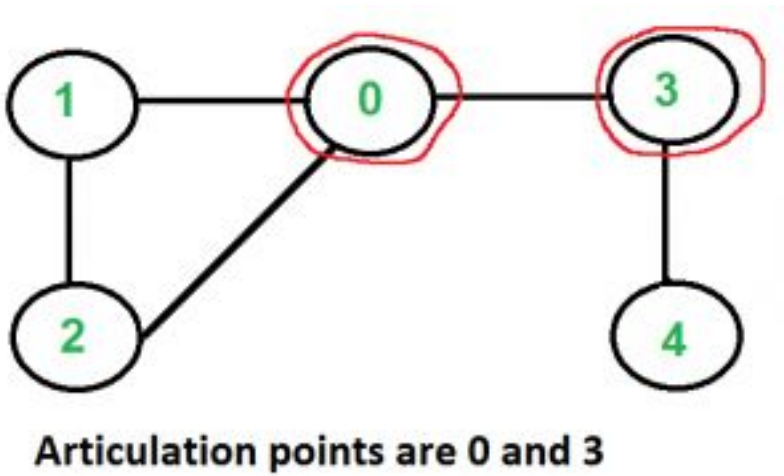
**Mostafa Saad Ibrahim**
PhD Student @ Simon Fraser University

# Articulation points

- In **undirected** graph, **removing** an articulation point (cut vertex/node):
  - if graph is fully connected ⇒ disconnects it
  - if not => disconnects a connected component
  - E.g. increases overall components
  - we can compute it using brute force: remove and test connectivity. But very slow
  - By definition: Leaf node is never an articulation point
- That is very similar to bridges, but node not edge

# Articulation points

node 0 is on a cycle
node 3 is not on a cycle
note: edge 0-3 and 3-4 are bridges




Src: http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/

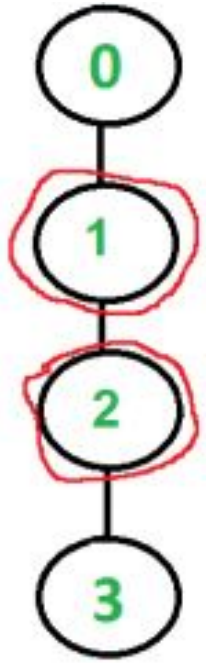
# Articulation points: Remove 0


Src: http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/

# Articulation points

leaves (0, 3) are not art-pts
regardless where dfs starts

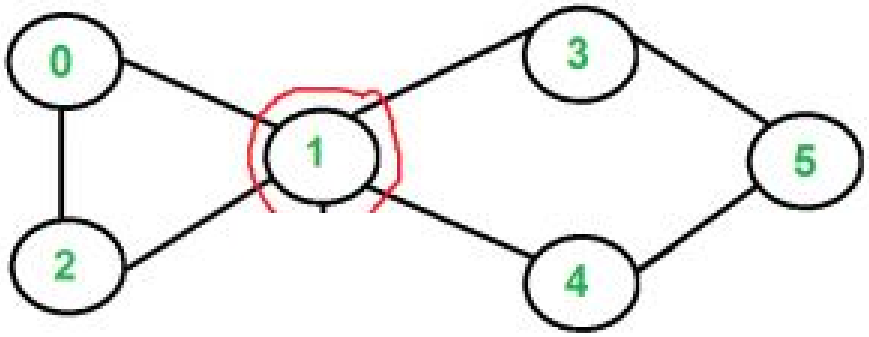
If removed them, other nodes
**remain** connected




Artiulation
Points are 1 & 2

Src: http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/

# Articulation points




Articulation Point is 1

There are **no bridges**...but point 1 is art-point.

Src: http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/

# Articulation points

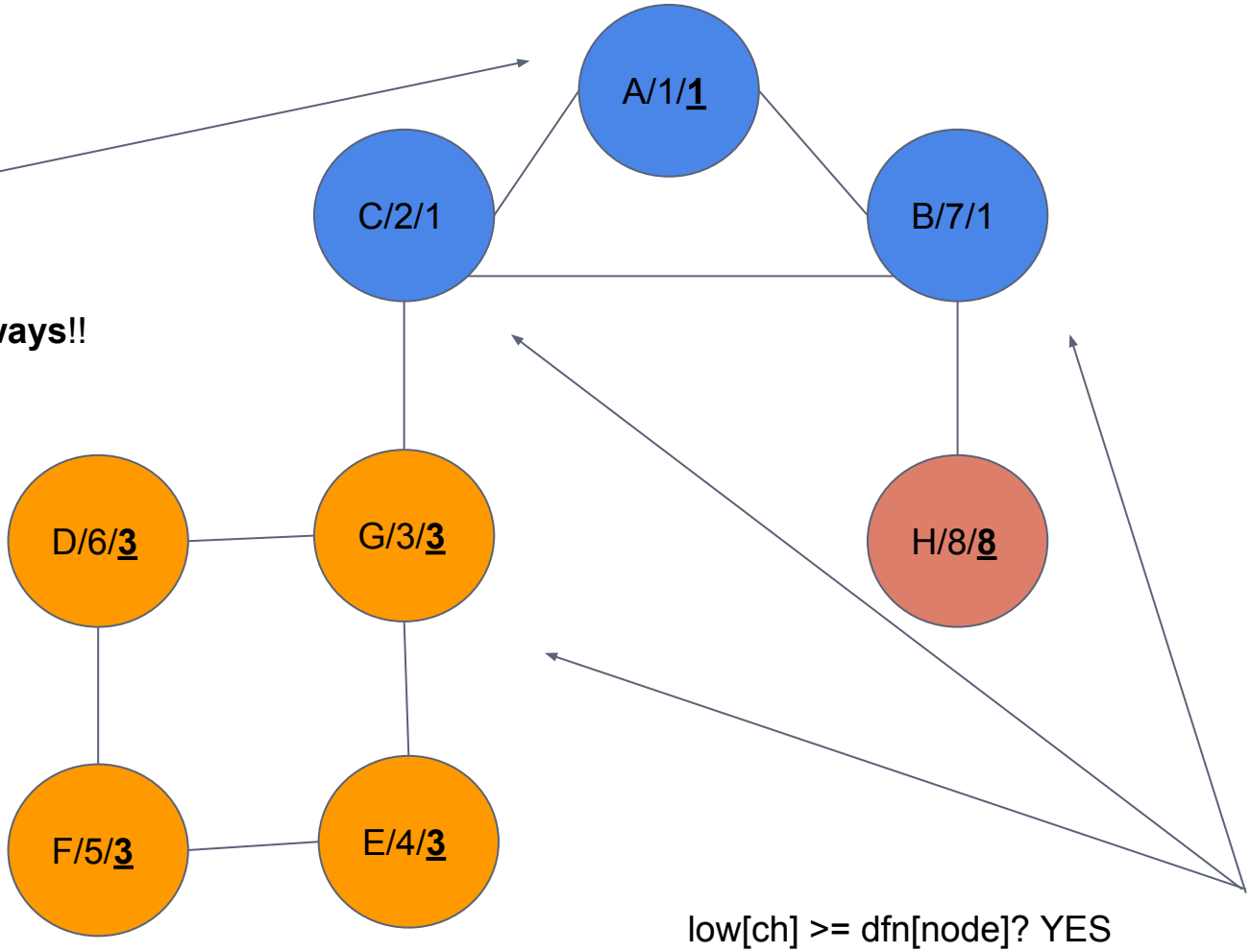
- Similar to bridges, why removing node **u** disconnects a subtree below it?
- Because every node **ch** under u can't reach a node upper **u**..hence remove u = disconnect
- In other words, if **lowlink[ch] >= dfn[u]**
  - Then **ch** node can't go beyon u
  - Hence u is an articulation point
  - Notice: node without child = leaf = NON art-pt
- Recall bridge condition: lowlink[**ch**] > dfn[u]

# Find art-pts

root dfn = low link = 1

Conditions applies for root **always**!!

Let's skip for now




low[ch] >= dfn[node]? YES
Articulation points

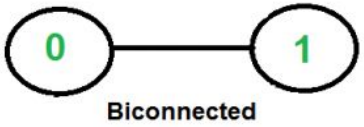
# Articulation points

```
void tarjan(int node, int par) {
    lowLink[node] = dfn[node] = ndfn++;

    rep(i, adjList[node]) {
        int ch = adjList[node][i];
        if (dfn[ch] == -1) {
            tarjan(ch, node);
            lowLink[node] = min(lowLink[node], lowLink[ch]);

            if (lowLink[ch] >= dfn[node])
                artpoints.insert(node);
        } else if (node != ch)
            lowLink[node] = min(lowLink[node], dfn[ch]);
    }
}
```

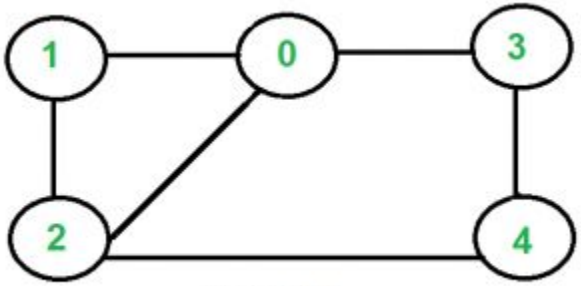
# Biconnected Graph

- Before handling root case, let's introduce
  - Biconnected Graph...Biconnected Component
- Connected Graph with no articulation points
  - Edge is a biconnected graph
  - There are two **vertex-disjoint** paths between any two vertices
  - Hence, There is a simple cycle through any two vertices.
- Using just 10 nodes, we can create 9,743,542 biconnected graphs

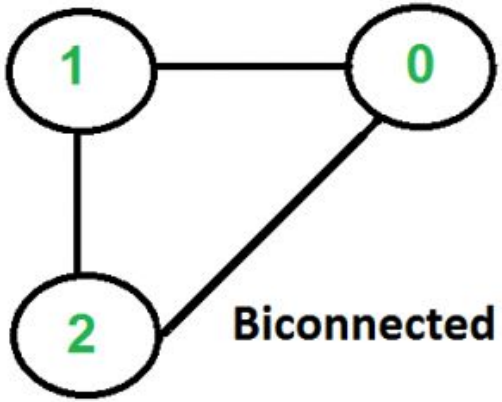
# Biconnected Graph



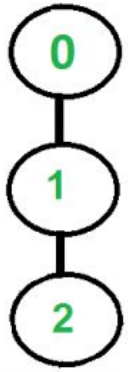
Biconnected



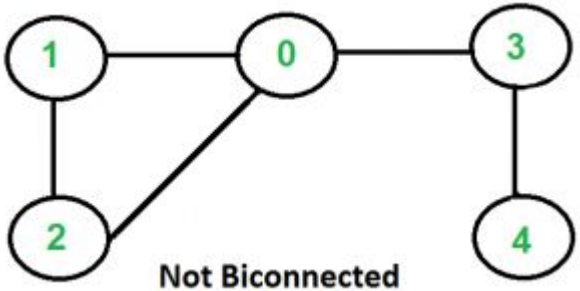
Biconnected



Biconnected



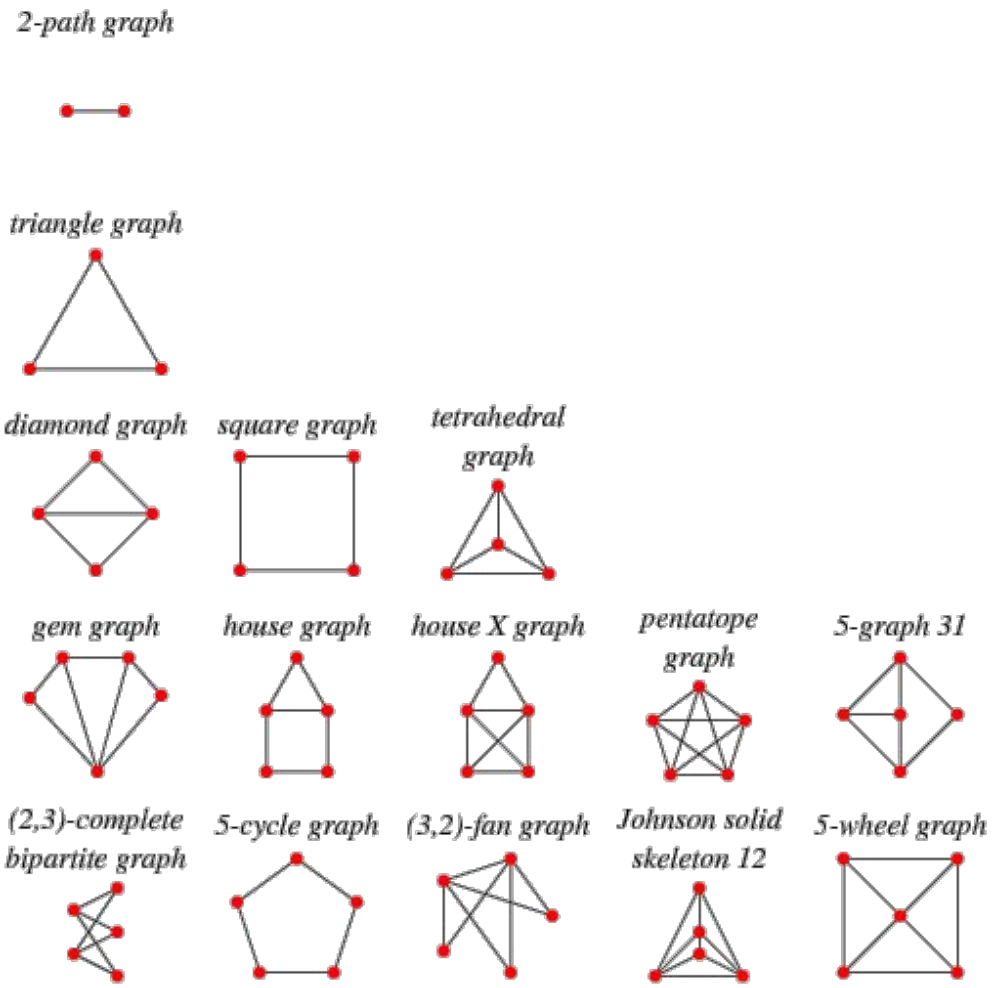
Not Biconnected



Not Biconnected

Src: http://www.gfgreader.info/post/551e0a12fd1aae1373effdb3

# Biconnected Graph

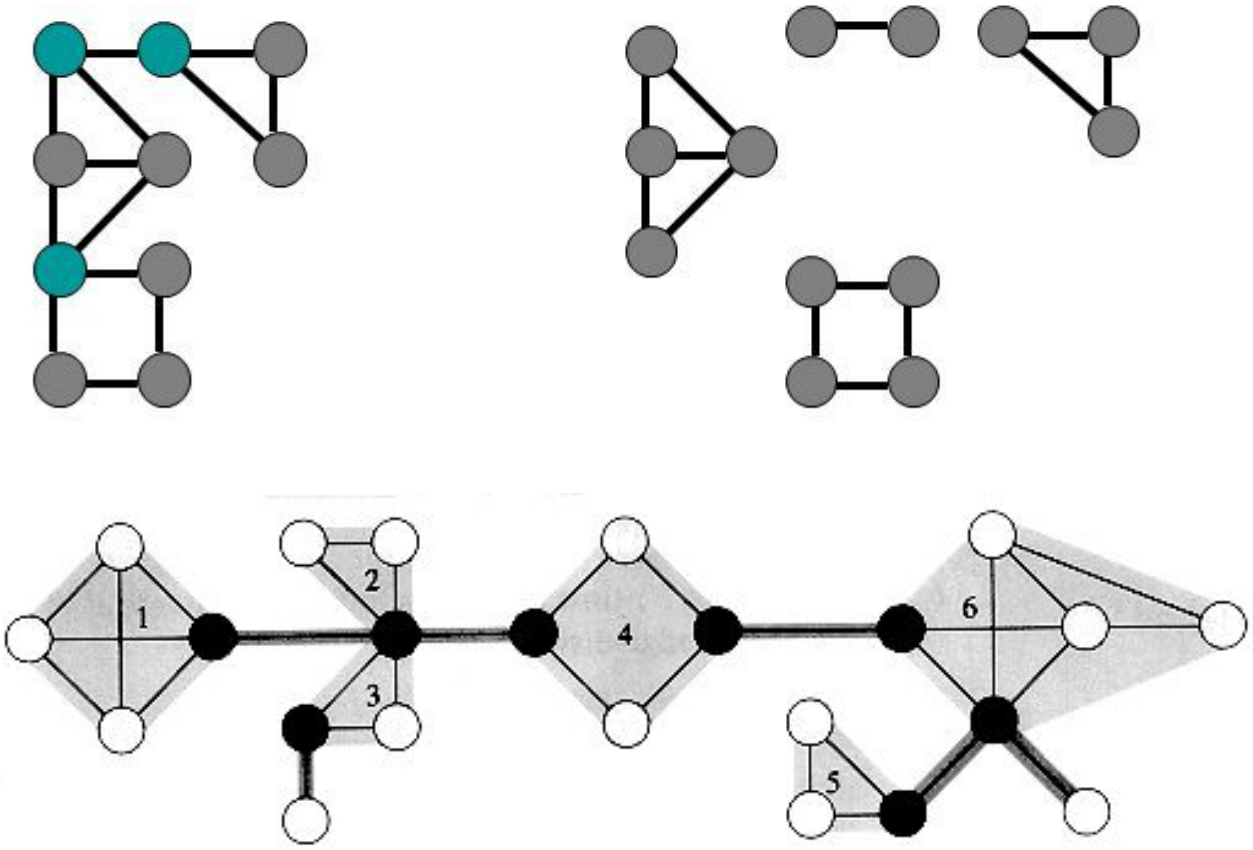



Src: http://mathworld.wolfram.com/BiconnectedGraph.html

# Biconnected Components (BCC)

- If graph has articulation points, split graph to components from these points
  - Each component is called a Biconnected component
  - Which satisfies Biconnected graph properties
- A graph with N articulation points has N+1 Biconnected components

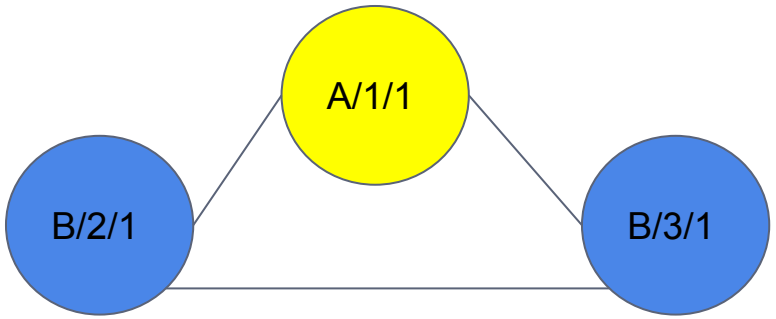
# Biconnected Components (BCC)



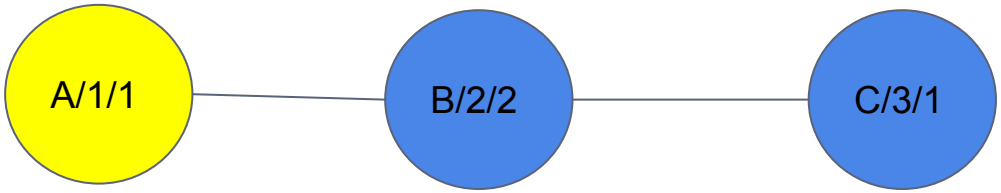

Src:
http://images.slideplayer.com/15/4656669/slides/slide_5.jpg
http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap23.htm

# Root case: Non articulation point

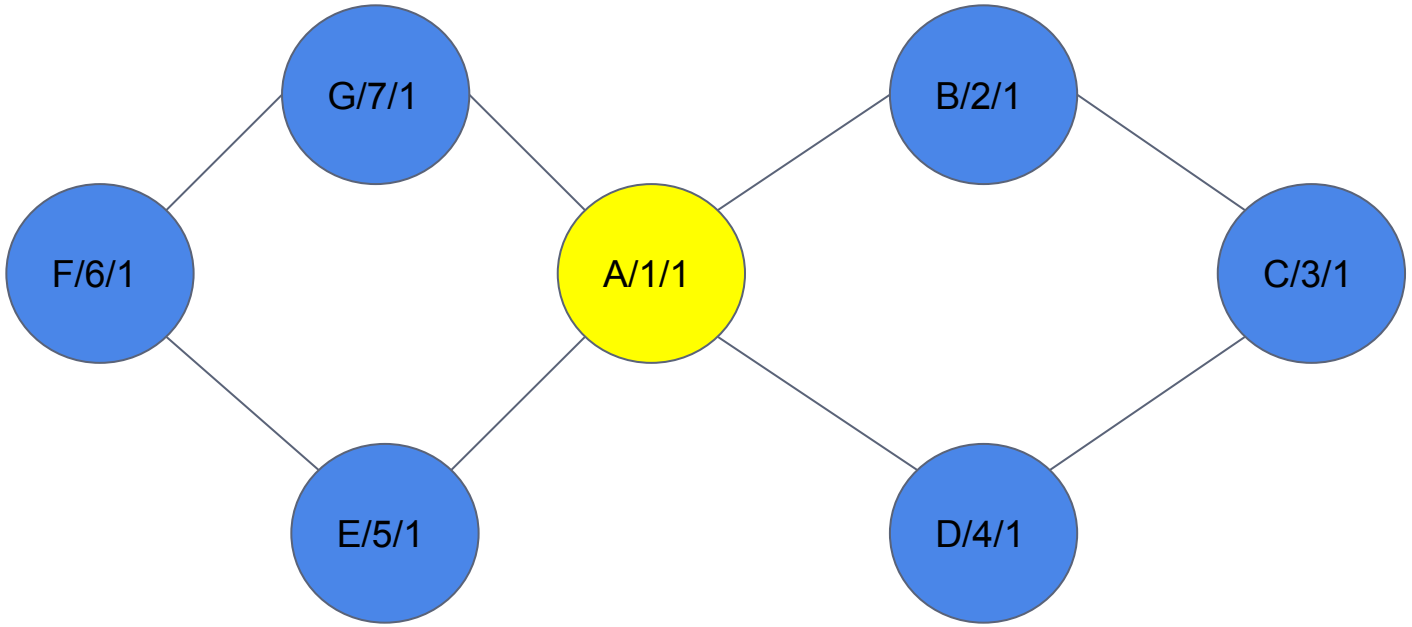
root is part of 1 biconnected component
⇒ NOT art-pt




root is actually a leaf
⇒ NOT art-pt



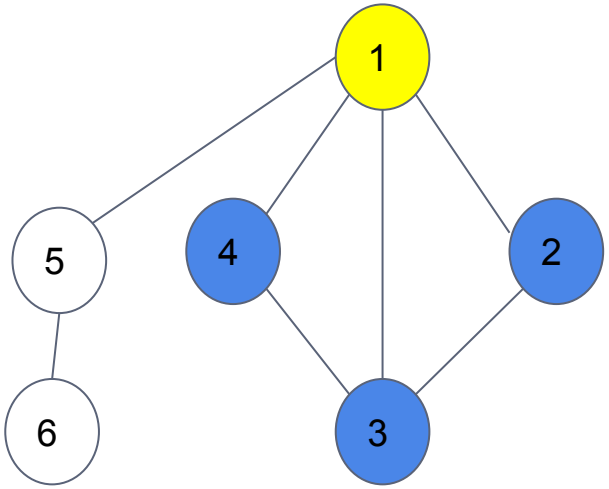
# Root case: Articulation point




root is part of 2+ biconnected components
⇒ Articulation point … removing it disconnects these BCCs

# DFS and moving between BCCs

```
void tarjan(int node, int par) {
    lowLink[node] = dfn[node] = ndfn++;

    rep(i, adjList[node]) {
        int ch = adjList[node][i];
        if (dfn[ch] == -1) {
```




If 1 is articulation point

- DFS goes through its children edges and back
- 2, 3, 4 **must** be marked visited too when back
- so only 1 unvisited condition ONLY is satisfied

- We can use such note to recognize moving from a BCC to another
- E.g. if root and moving to another BCC = Art-pt

# Handling Root

```
// Assumes whole graph is connected
bool root = false;

void tarjan(int node, int par) {
    lowLink[node] = dfn[node] = ndfn++;

    rep(i, adjList[node]) {
        int ch = adjList[node][i];
        if (dfn[ch] == -1) {
            tarjan(ch, node);
            lowLink[node] = min(lowLink[node], lowLink[ch]);

            if (lowLink[ch] >= dfn[node])
            {   // If first BCC, ignore it..otherwise art-pt
                if (dfn[u]==0 && root==false)
                    root=true;
                else
                    artpoints.insert(node);
            }
        } else if (node != ch)
            lowLink[node] = min(lowLink[node], dfn[ch]);
    }
}
```

# Finding Biconnected Components

```
stack< pair<int, int> > component;

void tarjan(int node, int par) {
    lowLink[node] = dfn[node] = ndfn++;

    rep(i, adjList[node]) {
        int ch = adjList[node][i];
        // Add Unvisited edge or non existing one (2 cases to avoid)
        if(node != ch && dfn[ch] < dfn[node])
            component.push( make_pair(u, w));

        if (dfn[ch] == -1) {
            tarjan(ch, node);
            lowLink[node] = min(lowLink[node], lowLink[ch]);

            if (lowLink[ch] >= dfn[node])
            {   // Get all edges till reach (node, ch) edge
                do
                {
                    edge = component.top(), component.pop();
                    cout<<edge.first+1<<" "<<edge.second+1<<"\n";
                }while(edge.first != node || edge.second != ch);
            }
        } else
```

# تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً