How to find articulation points?

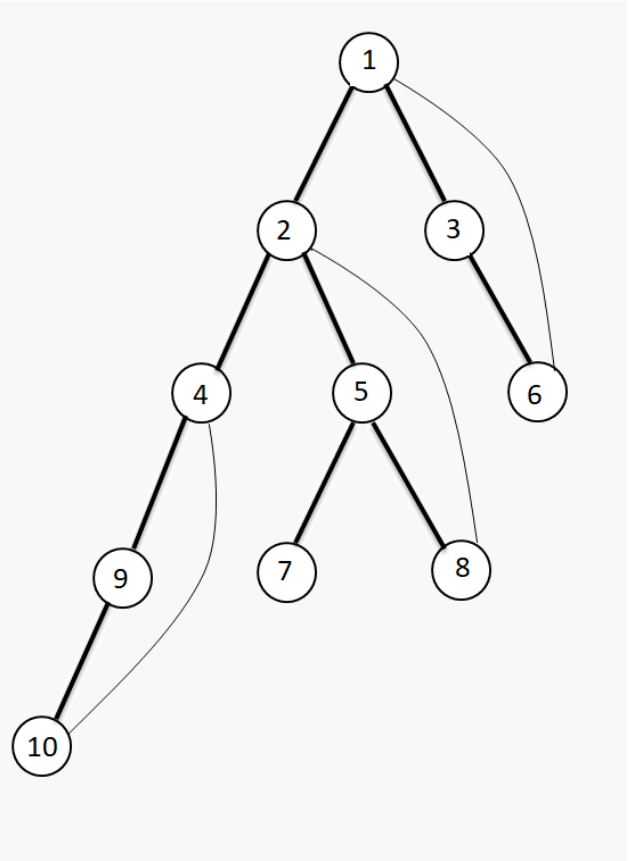
- **Brute Force Approach**
  A brute force approach to check if a vertex V is an articulation point or not is to remove the vertex V and check if there is an increase in the number of connected components of the graph.
  Hence to find all the articulation points in the graph, this approach will have a time complexity of O(V.(V + E)).

- **Tarjan's Algorithm**
  Tarjan's algorithm is based on the approach of DFS and takes O(V + E) time to find all the articulation points in the graph.
  Performing DFS on an undirected graph G always produces a DFS tree, so let us define what is a DFS tree.

  DFS Tree

  In the above diagram, we picked any arbitrary vertex, which in this case is node 1, and performed a DFS on the graph where the highlighted edges represent the edges traversed in the DFS traversal, so if we consider only these edges as a part of the graph it results in a tree, hence DFS traversal of an undirected graph always produces a DFS tree. The highlighted edges which are a part of the DFS traversal of the graph are known as **forward edges** or **tree edges**, while the remaining edges which do not become a part of the DFS traversal and connect a node to any of its descendants are known as **back edges**.

  For Example:



For the given graph we performed a DFS traversal starting from any arbitrary node as root, in this case, node **1** is the root and the edges visited by the DFS traversal of the graph are marked as **bold** edges while the unvisited ones are marked light basically representing the **back edges**.

In the above diagram:
Tree edges: 1-2, 2-4, 4-9, 9-10, 2-5, 5-7, 5-8, 1-3, 3-6
Back edges: 4-10, 2-8, 1-6

How to check if any vertex 'v' is an articulation point?

CASE 1: 'v' is not the root of the DFS tree:
It can be clearly observed that the back edges are the edges that connect a vertex to one of its ancestors in the DFS tree. Let's say we are at an **edge(v, to)** during the DFS such that **v != root** of the DFS tree and **to** is the child of **v** then **v** is said to be an articulation point if there is **no back edge connecting 'to' or any of its descendants to any of ancestors of 'v'**.

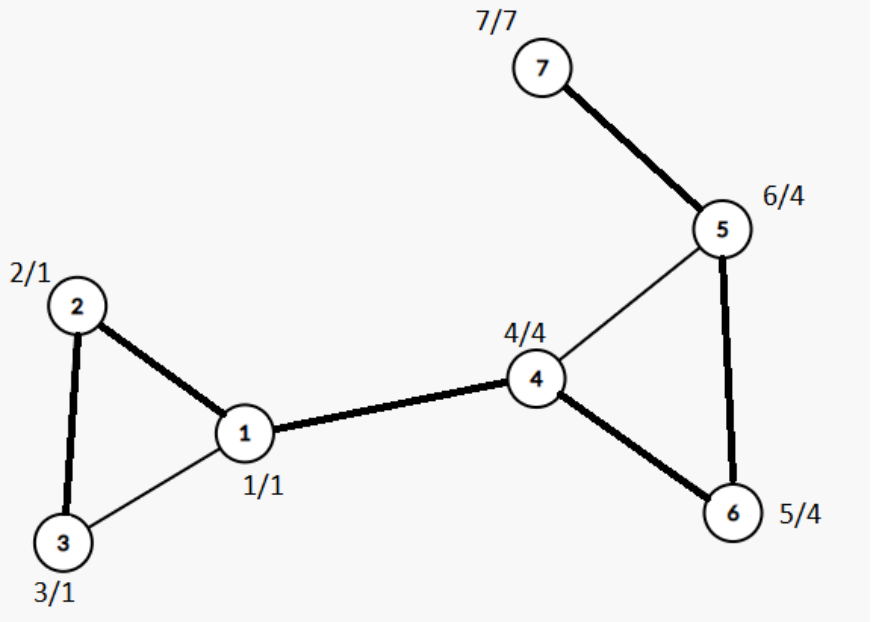
Now we need to devise a method to check this fact for each vertex v != root efficiently. Let us define **tin[v]** which denotes the entry time or the discovery time(also known as input time) for node v during DFS and also define **low[v]** as the **minimum**(tin[v], low[to]), where to is the direct descendant or the child of v, tin[p] for all vertices p such that there exists a back edge from v to p.

Hence **there is a back edge connecting 'to' or any of its descendants to any of the ancestors of 'v' iff low[to] < tin[v]**. If low[to] = tin[v] then there is a back edge connecting to and v. So, **v is an articulation point iff low[to] >= tin[v]**. The equality holds because if there is a back edge from 'to' to 'v', then removing v still disconnects the graph.

CASE 2: 'v' is the root of the DFS tree:
If the vertex **v** is the root of the DFS tree, then **v** is an articulation point iff **v** has more than one child in the DFS tree.

The above fact can be easily checked by the number of recursive calls made from v = root during DFS.

For Example:



Let us consider the above graph where we start the DFS from node **1**, which results in the formation of a DFS tree with the edges belonging to the DFS tree being marked as bold, we mark the discovery and update the low times of the nodes as we traverse the graph. Each node is labeled with **a/b**, where **a** represents the **discovery time or tin[v]**, whereas **b** represents the **low time or low[v]**, for a node v.
The DFS traversal is:
1->2->3->4->6->5->7

According to the algorithm, it can be clearly observed that due to the condition low[to] >= tin[v], nodes **1, 4** and **5** are articulation points where node 1 is the root and has greater than 1 child in the DFS tree.

Pseudocode:

```
/*
        The function takes an input of graph G(adjacencyList), tin, low and vis arrays,
        and the current node 'v' and parent 'par'.
        curtime is initialized to 0 used to fill the discovery times of nodes.
*/
function articulationPoints(G, v, par, tin, low, vis)

        vis[v] = true
        tin[v] = curtime
        low[v] = curtime
        curtime = curtime + 1

        //  Child variable keeps track of the number of children of node 'v'
        child = 0
        for to in G[v]
                //  If to == par, then the edge is back to the parent, we skip the iteration
                if to == par
                        continue
                //  If to is already visited, then we may have a back edge from 'v' to 'to'
                if vis[to] == true
                        low[v] = min(low[v], tin[to])
                else
                        child++
                        articulationPoints(G, to, v, tin, low, vis)
                        //  Checking for minimum value of low[v]
                        low[v] = min(low[v], low[to])
                        //  Checking if v is an articulation point
                        if low[to] >= tin[v] and par != -1
                                art_point[v] = true

        //  If the current node is the root node and has greater than 1 child
        if par == -1 and child > 1
                art_point[v] = true
```

Time Complexity: O(V + E), as we perform a DFS traversal of the given graph, where V is the number of vertices and E is the number of edges in the graph.