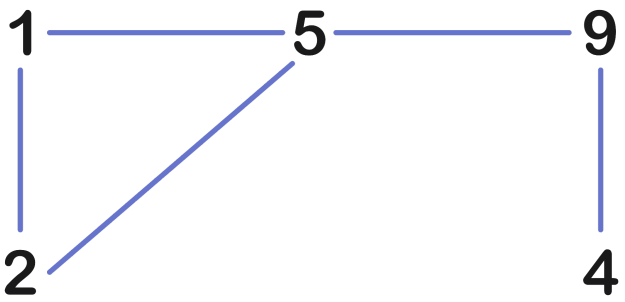
Articulation Point

An articulation point (or cut vertex) is a vertex in an undirected connected graph if removing it (and edges through it) disconnects the graph. Articulation points are single points in a connected network that if they fail, split the network into two or more components. They are useful in the design of dependable networks. An articulation point in a disconnected undirected graph is a vertex removal that increases the number of connected components..



In this 5 and 9 are articulation points.

1 and 2 Articulation Ponits

Articulation Points in a Graph using Tarjan's Algorithm:

The idea is to use DFS (Depth First Search). In DFS, follow vertices in a tree form called the DFS tree. In the DFS tree, a vertex u is the parent of another vertex v, if v is discovered by u.

In DFS tree, a vertex u is an articulation point if one of the following two conditions is true.

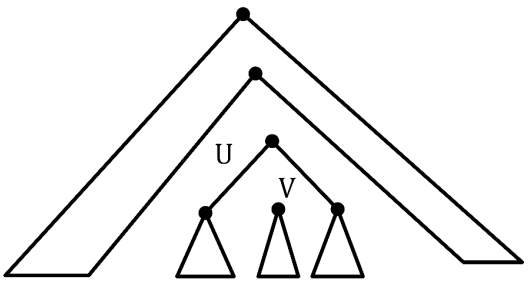
a. u is the root of the DFS tree and it has at least two children.
b. u is not the root of the DFS tree and it has a child v such that no vertex in the subtree rooted with v has a back edge to one of the ancestors in DFS tree of u.

The following figure shows the same points as above with one additional point that a leaf in DFS Tree can never be an articulation point.

Follow the below steps to Implement the idea: Do DFS traversal of the given graph

- In DFS traversal, maintain a parent[] array where parent[u] stores the parent of vertex u.
- To check if u is the root of the DFS tree and it has at least two children. For every vertex, count children. If the currently visited vertex u is root (parent[u] is NULL) and has more than two children, print it.
- To handle a second case where u is not the root of the DFS tree and it has a child v such that no vertex in the subtree rooted with v has a back edge to one of the ancestors in DFS tree of u maintain an array disc[] to store the discovery time of vertices.
- For every node u, find out the earliest visited vertex (the vertex with minimum discovery time) that can be reached from the subtree rooted with u. So we maintain an additional array low[] such that:

  low[u] = min(disc[u], disc[w]) , Here w is an ancestor of u and there is a back edge from some descendant of u to w.



If remove u does not seprate V, there must be an exit from v's subtree Via back edge

Root node is an articulationpoint iff it has more than one child

Leaf is never an articulation point

non-leaf, non-root node u is an articulation point

no non-tree edge goes above u from a sub-tree below some child of u.

import java.util.*;

Code Implementation :

```
class Solution {
    static int time;
    static void addEdge(ArrayList<ArrayList<Integer> > adj, int u, int v){
        adj.get(u).add(v);
        adj.get(v).add(u);
    }

    static void APUtil(ArrayList<ArrayList<Integer> > adj, int u,
                       boolean visited[], int disc[], int low[],
                       int parent, boolean isAP[]){
        int children = 0;
        visited[u] = true;
        disc[u] = low[u] = ++time;
        for (Integer v : adj.get(u)) {
            if (!visited[v]) {
                children++;
                APUtil(adj, v, visited, disc, low, u, isAP);
                low[u] = Math.min(low[u], low[v]);
                if (parent != -1 && low[v] >= disc[u])
                    isAP[u] = true;
            }
            else if (v != parent)
                low[u] = Math.min(low[u], disc[v]);
        }

        if (parent == -1 && children > 1)
            isAP[u] = true;
    }

    static void AP(ArrayList<ArrayList<Integer> > adj, int V){
        boolean[] visited = new boolean[V];
        int[] disc = new int[V];
        int[] low = new int[V];
        boolean[] isAP = new boolean[V];
        int time = 0, par = -1;

        for (int u = 0; u < V; u++)
            if (visited[u] == false)
                APUtil(adj, u, visited, disc, low, par, isAP);
```

```
        for (int u = 0; u < V; u++)
                if (isAP[u] == true)
                        System.out.print(u + " ");
        System.out.println();
}

public static void main(String[] args){
        int V = 5;
        ArrayList<ArrayList<Integer> > adj1 =
                                        new ArrayList<ArrayList<Integer> >(V);
        for (int i = 0; i < V; i++)
                adj1.add(new ArrayList<Integer>());
        addEdge(adj1, 1, 0);
        addEdge(adj1, 0, 2);
        addEdge(adj1, 2, 1);
        addEdge(adj1, 0, 3);
        addEdge(adj1, 3, 4);
        System.out.println("Articulation points in first graph");
        AP(adj1, V);
        V = 4;
        ArrayList<ArrayList<Integer> > adj2 =
                                        new ArrayList<ArrayList<Integer> >(V);
        for (int i = 0; i < V; i++)
                adj2.add(new ArrayList<Integer>());

        addEdge(adj2, 0, 1);
        addEdge(adj2, 1, 2);
        addEdge(adj2, 2, 3);

        System.out.println("Articulation points in second graph");
        AP(adj2, V);

        V = 7;
        ArrayList<ArrayList<Integer> > adj3 =
                                                new ArrayList<ArrayList<Integer> >(V);
        for (int i = 0; i < V; i++)
                adj3.add(new ArrayList<Integer>());

        addEdge(adj3, 0, 1);
```

```
                addEdge(adj3, 1, 2);
                addEdge(adj3, 2, 0);
                addEdge(adj3, 1, 3);
                addEdge(adj3, 1, 4);
                addEdge(adj3, 1, 6);
                addEdge(adj3, 3, 5);
                addEdge(adj3, 4, 5);

                System.out.println("Articulation points in third graph");

                AP(adj3, V);
        }
}
```

I hope you liked the article, Thanks for reading !