# Finding articulation points in a graph in $O(N + M)$

**Table of Contents**

We are given an undirected graph. An articulation point (or cut vertex) is defined as a vertex which, when removed along with associated edges, makes the graph disconnected (or more precisely, increases the number of connected components in the graph). The task is to find all articulation points in the given graph.

The algorithm described here is based on depth first search and has $O(N + M)$ complexity, where $N$ is the number of vertices and $M$ is the number of edges in the graph.

## Algorithm

Pick an arbitrary vertex of the graph $root$ and run depth first search from it. Note the following fact (which is easy to prove):

- Let's say we are in the DFS, looking through the edges starting from vertex $v \neq root$. If the current edge $(v, to)$ is such that none of the vertices $to$ or its descendants in the DFS traversal tree has a back-edge to any of ancestors of $v$, then $v$ is an articulation point. Otherwise, $v$ is not an articulation point.

- Let's consider the remaining case of $v = root$. This vertex will be the point of articulation if and only if this vertex has more than one child in the DFS tree.

Now we have to learn to check this fact for each vertex efficiently. We'll use "time of entry into node" computed by the depth first search.

So, let $tin[v]$ denote entry time for node $v$. We introduce an array $low[v]$ which will let us check the fact for each vertex $v$. $low[v]$ is the minimum of $tin[v]$, the entry times $tin[p]$ for each node $p$ that is connected to node $v$ via a back-edge $(v, p)$ and the values of $low[to]$ for each vertex $to$ which is a direct descendant of $v$ in the DFS tree:

$$low[v] = \min \begin{cases} tin[v] & \\ tin[p] & \text{for all } p \text{ for which } (v, p) \text{ is a bac} \\ low[to] & \text{for all } to \text{ for which } (v, to) \text{ is a tr} \end{cases}$$

Now, there is a back edge from vertex $v$ or one of its descendants to one of its ancestors if and only if vertex $v$ has a child $to$ for which $low[to] < tin[v]$. If $low[to] = tin[v]$, the back edge comes directly to $v$, otherwise it comes to one of the ancestors of $v$.

Thus, the vertex $v$ in the DFS tree is an articulation point if and only if $low[to] \geq tin[v]$.

## Implementation

The implementation needs to distinguish three cases: when we go down the edge in DFS tree, when we find a back edge to an ancestor of the vertex and when we return to a parent of the vertex. These are the cases:

- $visited[to] = false$ - the edge is part of DFS tree;
- $visited[to] = true$ && $to \neq parent$ - the edge is back edge to one of the ancestors;

- $to = parent$ - the edge leads back to parent in DFS tree.

To implement this, we need a depth first search function which accepts the parent vertex of the current node.

C++ implementation Show/Hide

Main function is `find_cutpoints`; it performs necessary initialization and starts depth first search in each connected component of the graph.

Function `IS_CUTPOINT(a)` is some function that will process the fact that vertex $a$ is an articulation point, for example, print it (Caution that this can be called multiple times for a vertex).

## Practice Problems

- UVA #10199 "Tourist Guide" [difficulty: low]
- UVA #315 "Network" [difficulty: low]
- SPOJ - Submerging Islands