CIS 575. Introduction to Algorithm Analysis
Material for April 19, 2024

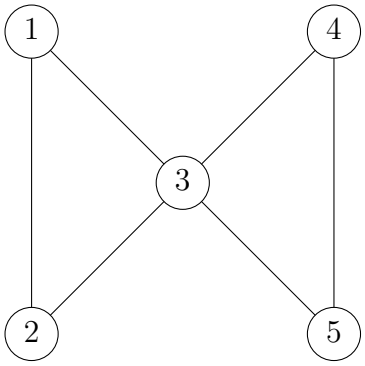
Depth-First Search on Undirected Graphs: An Application

The topic of this note is covered in *Cormen*'s Problem 20-2.

1 Articulation Points

For a **connected un**directed graph (V, E), an *articulation point* is a node such that if it is removed (together with the edges that touch it) the remaining graph is no longer connected. A real-world network that contains an articulation point would be very vulnerable!

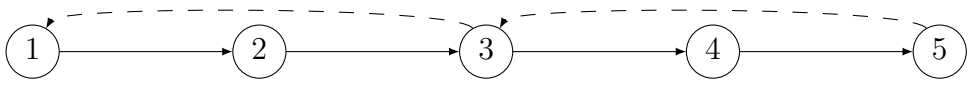
For example, consider the graph



where it is easy to see that node 3 is an articulation point (as removing it would leave an edge between 1 and 2 and an edge between 4 and 5 but nothing connecting these two components), and that no other articulation points exists.

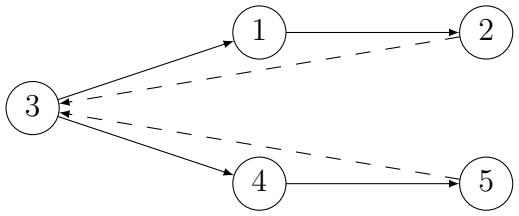
To guide us towards an algorithm for finding articulation points, let us recall how the depth-first-search algorithm from an undirected graph constructs a tree, together with back edges (but no other kind of edges). For our example, this can be done in several ways.

1. One possibility is that we start with node 1, and construct the tree



   where there are back edges from node 5 to node 3, and from node 3 to node 1. Since the root 1 has **only one child**, we can immediately infer that it is **not** an articulation point (since removing it would leave an almost intact tree that even without the back edges would connect all the remaining nodes).

2. Another possibility is that we start with node 3, and construct the tree



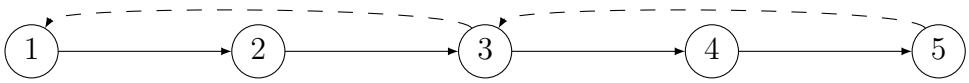
where there are back edges from node 2 to node 3, and from node 5 to node 3. Since the root 3 has **more than one child**, we can immediately infer that it **is** an articulation point (since removing it would result in the children being disconnected from each other, as there cannot be cross edges).

We have made an general observation:

> the root of the tree produced by the DFS algorithm is an articulation point iff it has at least two children

which we *could* use for an algorithm: run DFS once for each node! But this would have running time in $\Theta(|V| \cdot |E|)$; we would like to go for something that runs DFS only once and hence inherits (assuming all actions take constant time) its running time $\Theta(|E|)$.

For this purpose, we now show how an *arbitrary* tree produced by DFS can be used to find *all* the articulation points. It is convenient to assume that the nodes are numbered in the order of their discovery time. This is the case for the first tree which was



We shall now define a useful function, highest, such that for each node u, highest$[u]$ is the node highest[1] in the tree one can reach from u by

1. going down zero or more tree edges
2. going up *one* back edge, or staying put if no back edge exists.

With our assumption about nodes being numbered in order of their discovery time, the **"highest" node** is the one with the **lowest number**. In our example, highest will be given by the table

u	1	2	3	4	5
highest(u)	1	1	1	3	3

To see why highest$[4] = 3$, observe that from node 4 one can go down a tree edge to node 5 and then take a back edge to node 3 (but one cannot take a further back edge to node 1).

Now let us see how to detect if a node u is an articulation point. We have already seen how to do that if u is the root. If u is not the root, we claim that

> u is an articulation point iff u has a child w with highest$[w] \geq u$.

and shall now argue that this is always true:

"if": such a child is not able to reach up beyond u, and hence will be disconnected from the root if u is removed

[1] As usual in computer science, we assume the root is at the top of the tree.

"only if": if no such child exists, then all children of u can reach up beyond u (to a grandparent, great-grandparent, etc) and hence removing u will not cut the connection between the younger and the older generation.

Applying this fact to our example, we confirm that

- node 2 is *not* an articulation point, since its only child, node 3, satisfies $\mathsf{highest}[3] = 1 < 2$
- node 4 is *not* an articulation point, since its only child, node 5, satisfies $\mathsf{highest}[5] = 3 < 4$
- node 5 is *not* an articulation point, since it has no children
- node 3 *is* an articulation point, since it has a child 4 with $\mathsf{highest}[4] = 3$.

We can encode these considerations in the various actions taken during the DFS algorithm; with u the node and with w the target of the edge considered, we define

PreNode We must give $\mathsf{highest}[u]$ an initial value, which should be $d[u]$ (its discovery time) as u can certainly reach as high as itself

PreEdge No action is taken before the exploration of w

PostEdge After w has been explored, $\mathsf{highest}[w]$ has a value; if that value is $\geq d[u]$ (and u is not the root) we can declare that u is an articulation point.

In addition, if w can get somewhere by a sequence of tree edges possibly followed by a back edge, surely u can also get there. Therefore $\mathsf{highest}[u]$ is at least as high in the tree as $\mathsf{highest}[w]$, that is at least as low a number, and we must update $\mathsf{highest}[u]$ accordingly.

OtherEdge If $d[w] < d[u]$, and the edge is not a tree edge (in the other direction), the edge is a back edge which u could use to move up in the tree; hence we must update $\mathsf{highest}[u]$ accordingly.

PostNode If u is not the root, nothing happens. But if u is the root, we check if u has more than one child in T; if so, we can declare that u is an articulation point.

With these instantiations of the generic DFS algorithm, we end up with an algorithm for finding the articulation points in a graph:

```
ArticulationPoints(u)
    color[u] ← gray
    d[u] ← current_time
    highest[u] ← d[u]
    foreach (u, w) ∈ E
      if color[w] = white
        T ← T ∪ {(u, w)}
        ArticulationPoints(w)
        if highest[w] ≥ d[u] and u is not root
          u is articulation point
        highest[u] ← min(highest[w], highest[u])
      else if d[w] < d[u] and (w, u) ∉ T
        highest[u] ← min(d[w], highest[u])
    color[u] ← black
    if u is root and has at least two children in T
      u is articulation point
```

We may sketch the key steps taken by this algorithm on our example:

PreNode 1	highest[1] = 1	
PreNode 2	highest[2] = 2	
PreNode 3	highest[3] = 3	
OtherEdge 3→1	highest[3] = 1	
PreNode 4	highest[4] = 4	
PreNode 5	highest[5] = 5	
OtherEdge 5→3	highest[5] = 3	
PostEdge 4→5	highest[4] = 3	
PostEdge 3→4		3 AP as highest[4] = 3
PostEdge 2→3	highest[2] = 1	
PostEdge 1→2		
PostNode 1		1 not AP as only one child