Trees - Introduction

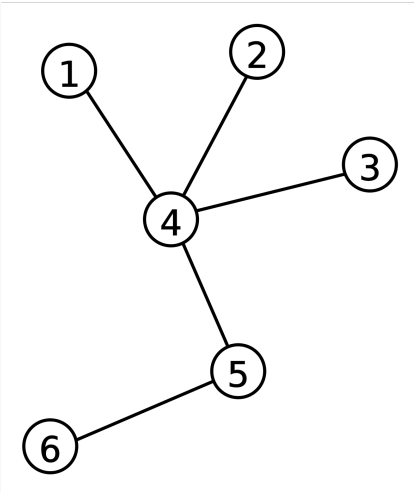
Soumil Aggarwal

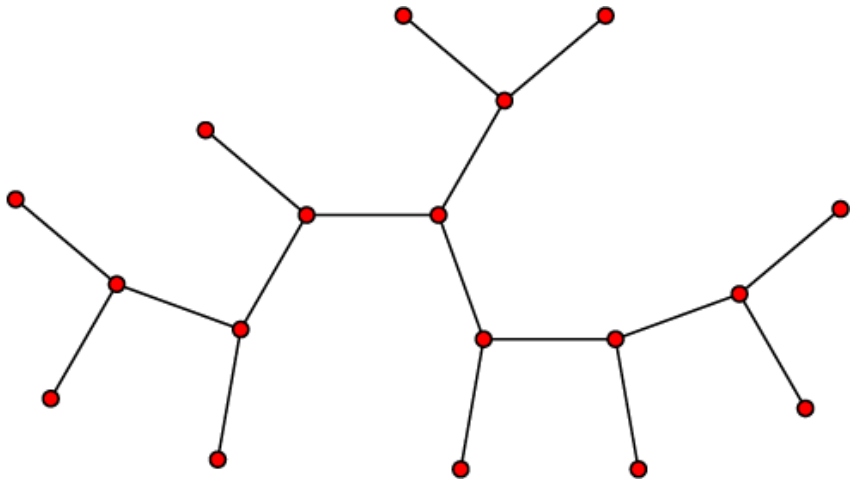
Algorithms and Coding Club
Indian Institute of Technology Delhi

6 December 2021

Definitions

Tree - A connected, acyclic graph





Leaves of a tree - vertices with degree 1. The leaves of the first tree are $1, 2, 3, 6$.

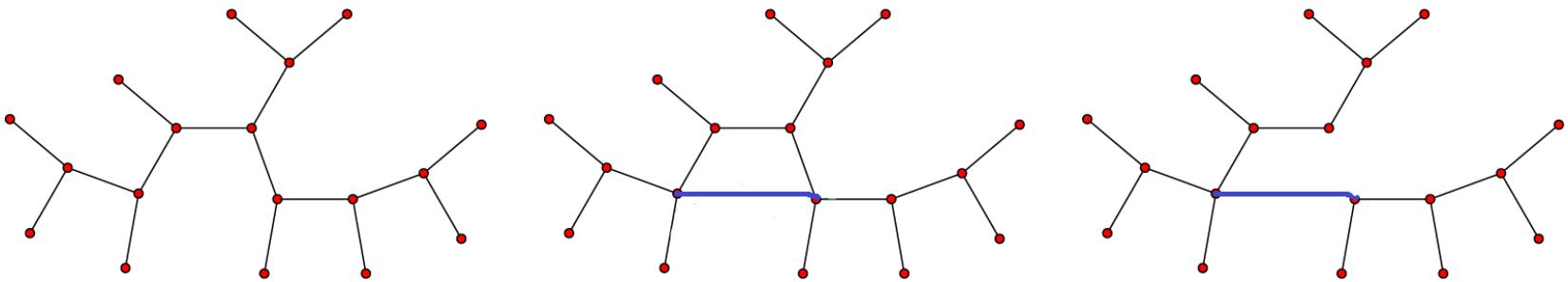
Some basic facts

- Let G be a graph with n vertices and m edges. Then any two of the following statements together imply the third.
  - G is connected.
  - G is acyclic.
  - $m = n - 1$.

  G is a tree iff the above three conditions are true.
- Every tree with at least 2 vertices has at least 2 leaves.
- There exists a unique path between any two vertices of a tree.

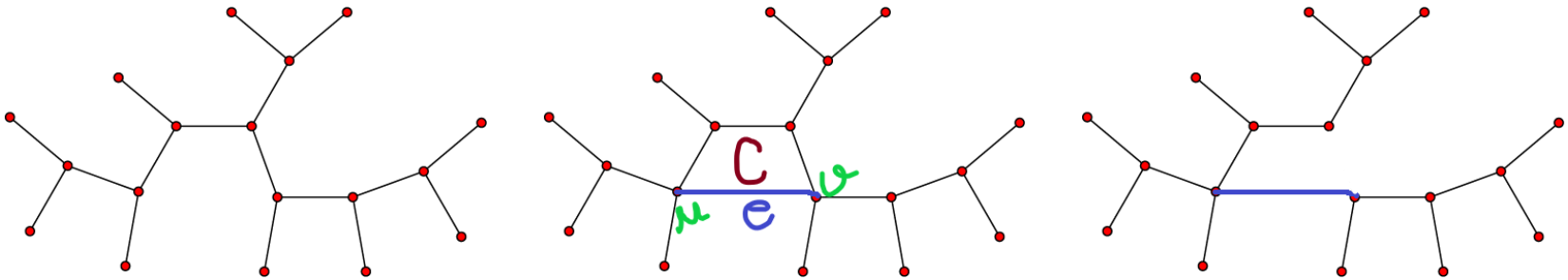
Cycle Property

Adding an edge to a tree gives you exactly one cycle. Removing an edge from this cycle again gives you a tree.



Cycle Property

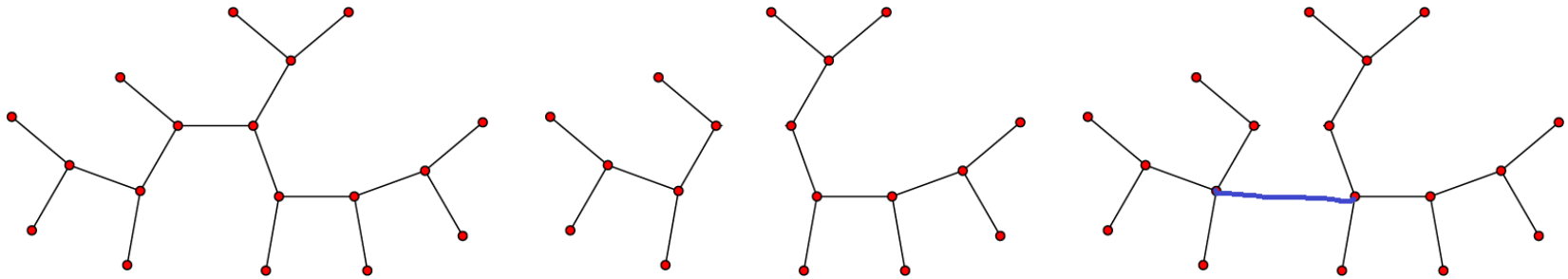
Proof



- Suppose the edge added was $e = \{u, v\}$. Since $m = n \neq n - 1$, and the graph remains connected, it must have a cycle. Any such cycle must contain e.
- Suppose C is a cycle. Then $C - e$ is the path between u and v in the original tree, which is unique. Hence C is unique.
- Removing an edge from C makes the graph acyclic again, and $m = n - 1$, hence we again get a tree.

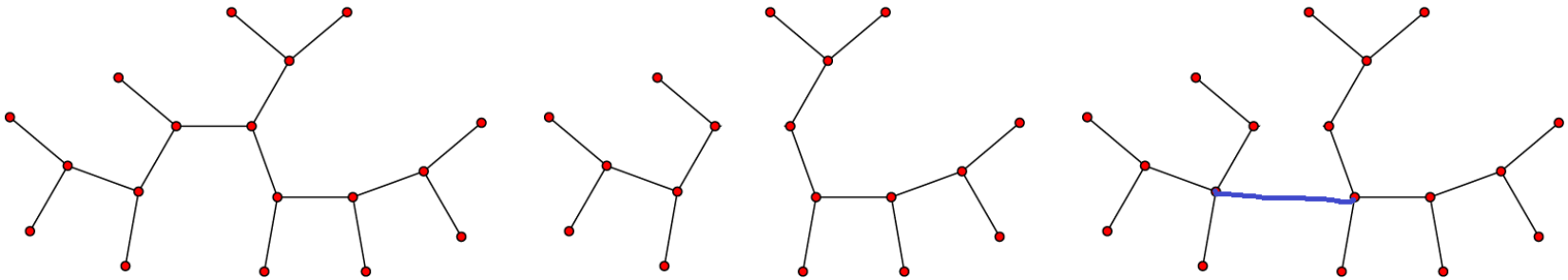
Cut Property

Removing an edge from a tree gives you two connected components. Adding an edge between two vertices from the two different connected components again gives you a tree.



Cut Property

Proof



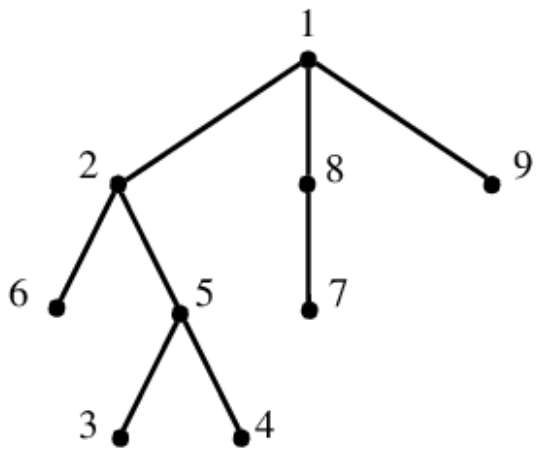
- Note that the graph remains acyclic on removing an edge. So its connected components must be trees. Since $m = n - 2$, there must be exactly 2 connected components.
- Adding an edge between the two connected components makes the graph connected again. Since $m = n - 1$ now, we again got a tree.

DFS on Trees

Since trees are acyclic, and the path between any two vertices is unique, DFS is much easier to implement for trees than general graphs. More specifically, we don't need to maintain an array to keep track of which nodes have already been visited. We only need to ensure that we don't traverse an edge back and forth.

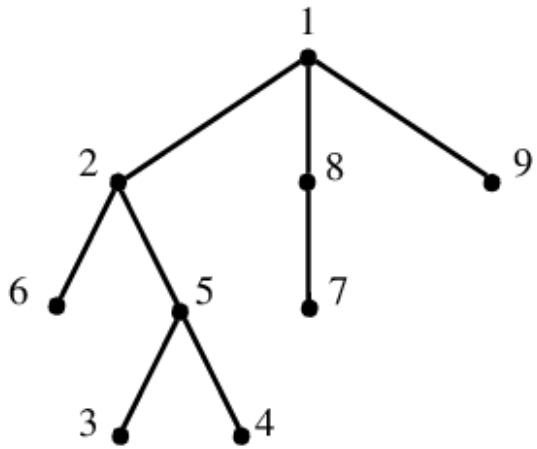
```
1  void dfs(int v, int p) {
2      // parent[v] = p;
3      // process node v
4      for (auto u : adj[v]) {
5          if (u != p) dfs(u, v);
6      }
7  }
8  dfs(x, 0); // Basically means dfs(x, null)
```

Rooted trees - 1



- Choose a vertex r as the root. Run dfs(r, 0), with line 2 uncommented. The tree along with the parent array now forms a tree rooted at r.
- Every node has one parent (except r which has no parent).
- If v is the parent of u, we call u a child of v. A node may have multiple children. Nodes without children are exactly the leaves of the rooted tree (the root isn't called a leaf even if it has degree 1).

Rooted trees - 2



- Drawing a rooted tree so that the children lie below the parent gives a natural visual representation.
- Rooted trees have a nice recursive structure. Each node is the root of a subtree which consists of the node itself and the subtrees of its children.
- This recursive structure is why treating a tree as a rooted tree is very useful. It helps us define dp and recursion on the tree, as we'll see in the next part.