Algorithms and Data Structures

COMP261
Tutorial 4

Yi Mei

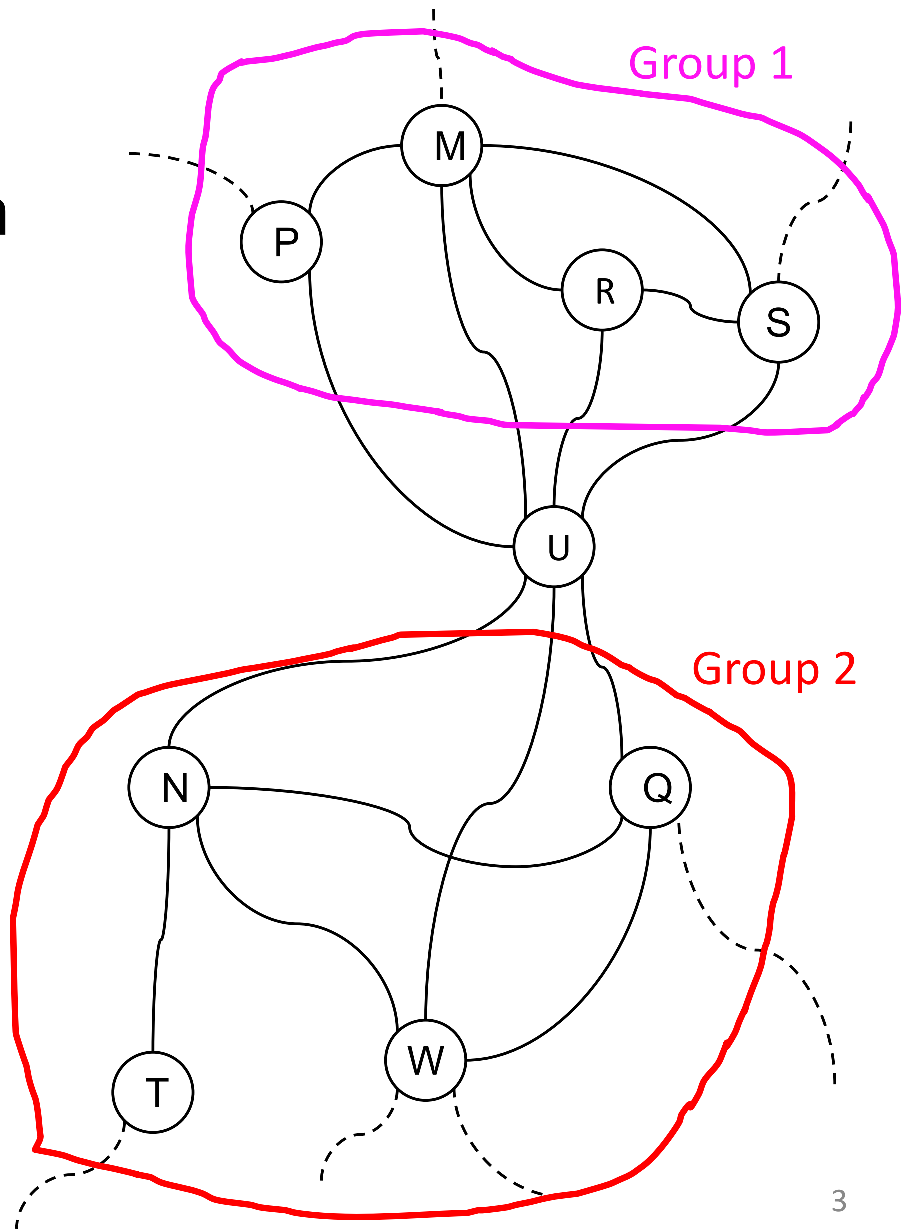
yi.mei@ecs.vuw.ac.nz

Outline

- Finding all articulation points
  - Idea
  - Implementation: recursive and iterative

Finding All Articulation Points Efficiently

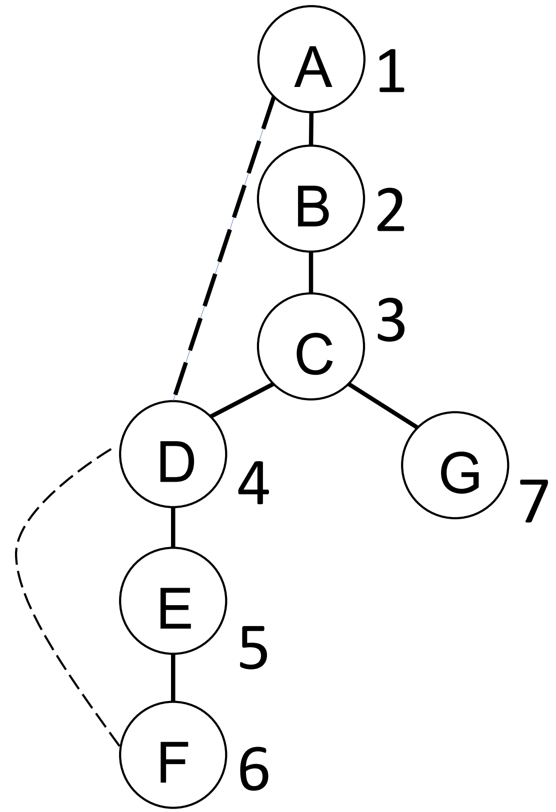
- **Idea**: an articulation point separates the graph into two groups, so that all paths from nodes in one group to nodes in the other group MUST go through the node.



- Example:
  - node U is an articulation point, since all paths between any node in group 1 and any node in group 2 must go through U.

Two Sets of a Node

- In the search tree, each node separates the nodes into two subsets
  - **Children set**: Nodes in its subtree
  - **Parents set**: Nodes not in its subtree



- Check if **Children** an **Parents** are separated after removing the node
  - The node is an articulation point if at least one **child** node and **parents** are separated after removing it
  - There is no alternative path

Articulation Points Algorithm

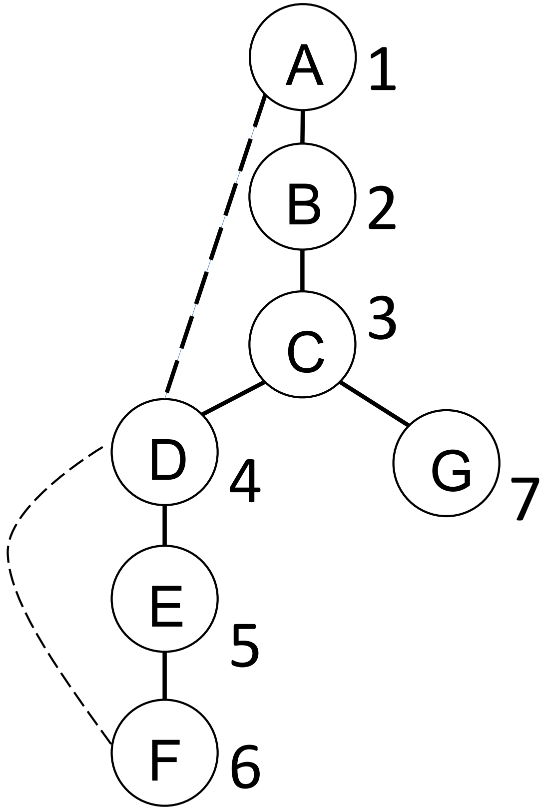
- **Theorem**: a non-root node A is an articulation point, **if and only if** there exists a child node B, for which there is no alternative path from B to any of the **parents**
  - Removing node A will separate B and the parents
  - This is independent of the root node of the DFS and order that the neighbours are checked

- Checking alternative paths for a child node B of node A
  - An edge in the graph, but not in the DFS tree
  - Directly link node B to a parent node
  - Link another child node C in the same subtree of B to a parent node
    - All the child node in the same subtree are connected without node A

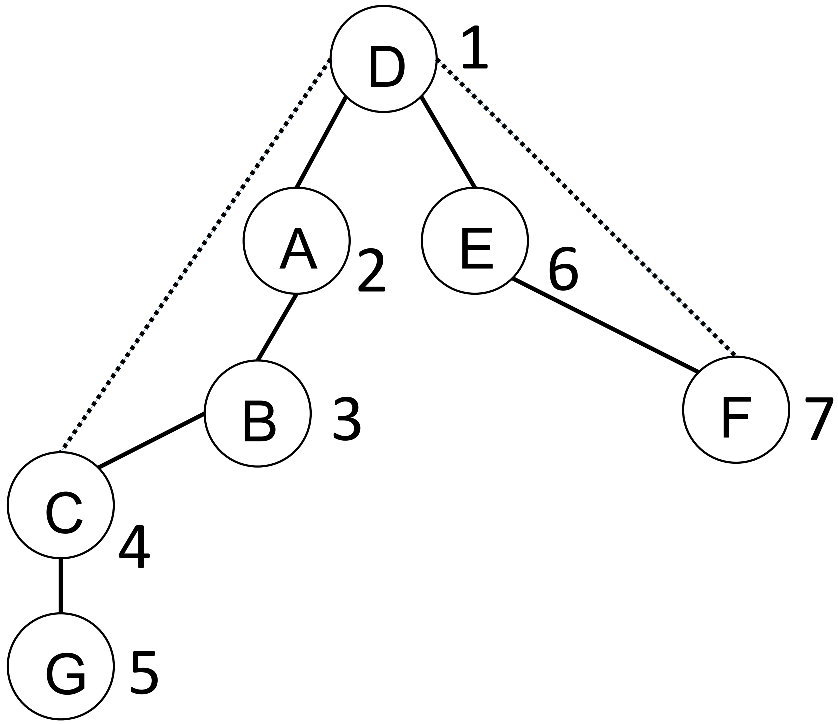
Articulation Points Algorithm

- **Theorem**: a **root node** is an articulation point **if and only if** it has multiple sub-trees in the DFS
  - Proof is easy (no alternative path between sub-trees)

A is not an articulation point (one sub-tree)



D is an articulation point (two sub-trees)



Recursive AP Algorithm

```
Initialise count number of all nodes as count(node) = ∞, meaning all nodes are unvisited;
Initially, APs = {}, that is, no articulation point is found;
Randomly select a node as the root node, set count(root) = 0, numSubTrees = 0;

for (each neighbour of root) {
   if (count(neighbour) = ∞) {
      recArtPts(neighbour, 1, root);  // recursive DFS for the neighbour
      numSubTrees ++;
   }

   if (numSubTrees > 1) then add root into APs;
}

------------------------------------------------------------------------------------------

recArtPts(node, count, parent) {
   …
}
```

Recursive AP Algorithm

```
recArtPts(node, count, parent) {
    count(node) = count;
    // store the minimum count number the node can reach back via alternative path
    reachBack = count;
    for (each neighbour of node other than parent) {
        // case 1: direct alternative path: neighbour is visited before
        if (count(neighbour) < ∞)
            reachBack = min(count(neighbour), reachBack);
        // case 2: indirect alternative path: neighbour is an unvisited child in the same sub-tree
        else {
            // calculate alternative paths of the child, which can also be reached by itself
            childReach = recArtPts(neighbour, count+1, node);
            reachBack = min(childReach, reachBack);
            // no alternative path from neighbour to any parent
            if (childReach >= count) then add node into APs;
        }
    }
    return reachBack;
}
```

Iterative AP Algorithm

```
Initialise count(node) = ∞, APs = {};
Randomly select a node as the root node, set count(root) = 0, numSubTrees = 0;

for (each neighbour of root) {
   if (count(neighbour) = ∞) {
      iterArtPts(neighbour, 1, root);
      numSubTrees ++;
   }

   if (numSubTrees > 1) then add root into APs;
}

---------------------------------------------------------------------------------------

iterArtPts(node, count, parent) {
   …
}
```

The only difference from the recursive version

Iterative AP Algorithm

```
iterArtPts(firstNode, count, root) {
   Initialise stack as a single element <firstNode, count, root>;
   repeat until (stack is empty) {
      peek <n*, count*, parent*> from stack;
      if (count(n*) = ∞) {
         count(n*) = count, reachBack(n*) = count;
         children(n*) = all the neighbours of n* except parent*;
      }
      else if (children(n*) is not empty) {
         get a child from children(n*) and remove it from children(n*);
         if (count(child) < ∞) then reachBack(n*) = min(count(child), reachBack(n*));
         else push <child, count+1, n*> into stack;
      }
      else {
         if (n* is not firstNode) {
            reachBack(parent*) = min(reachBack(n*), reachBack(parent*));
            if (reachBack(n*) >= count(parent*) then add parent* into APs;
         }
         remove <n*, count*, parent*> from stack;
}}}
```

Example