Algorithms and Data Structures

COMP261

Articulation Points 2: Implementation

Yi Mei

yi.mei@ecs.vuw.ac.nz

Outline

- Idea revisited
- Implementation of the AP algorithm
  - Recursive version
  - Iterative version

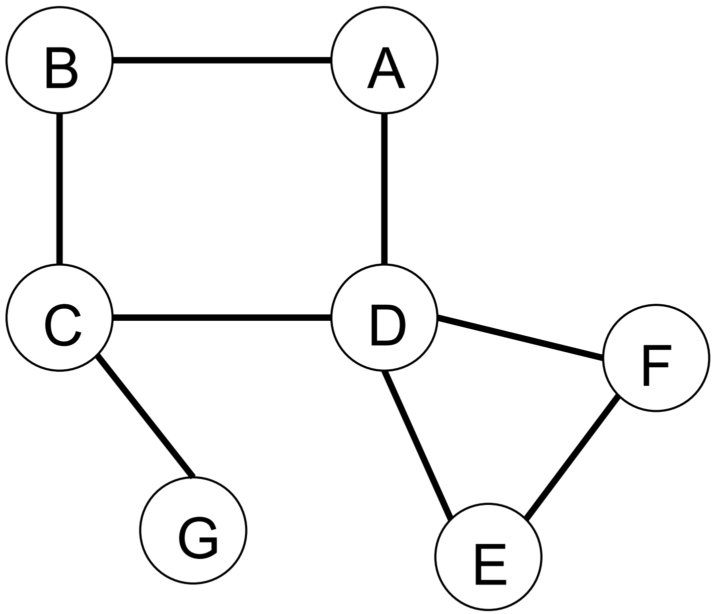
Idea Revisited

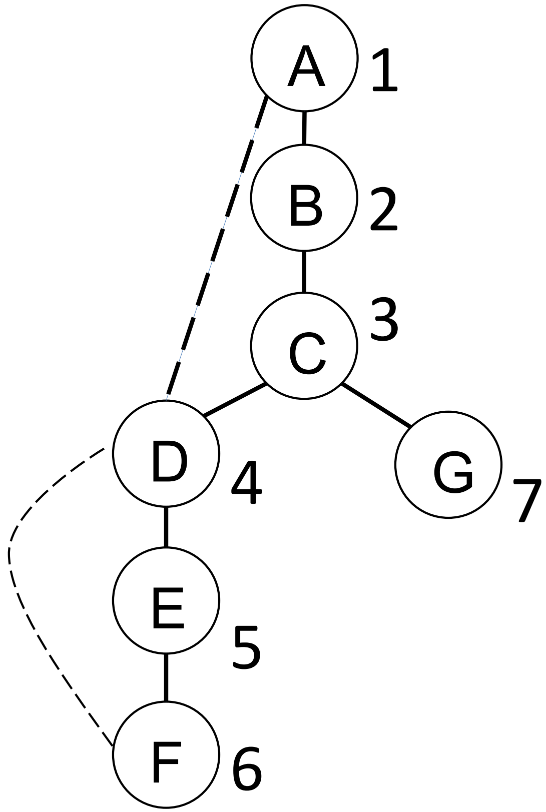
- Articulation point: a node whose removal will disconnect the graph
- Brute force search
  - For each node, run a DFS starting from one neighbour (one sub-tree)
  - If all the nodes are visited (one sub-tree), then the node is not an articulation point. Otherwise, it is an articulation point.
  - Complexity: O(n*e): n is number of nodes, e is number of edges
  - Very inefficient, many checks are unnecessarily repeated

- A new efficient algorithm with a single DFS
  - Assign count numbers to each node during DFS (larger count numbers are children, smaller are parents)
  - Check for root node: number of sub-trees
  - Check for other nodes: alternative path from children to parents

Example of Idea

- A single DFS rooted from node A
  - One single DFS to assign count numbers, record the edges that are in the DFS and not in the DFS: (A, D), and (D, F)
  - A is not articulation point: root node, one sub-tree
  - B is not articulation point: all the children can reach parents via (A, D)
  - C is articulation point: no alternative path from G to parents
  - D is articulation point: no alternative path from E, F to parents
  - E, F, G?





Implementation

- DFS can be implemented by recursion and iteration
  - **Recursion**: easy to design/write, hard to debug
  - **Iteration**: easy to debug, hard to design/write
- We also design these two versions of the AP algorithm

```
RecDFS(node) {
   if (node is unvisited) {
      set node as visited;
      for (each neighbour of node) {
         RecDFS(neighbour);
      }
   }
}
```

```
IterDFS(node) {
   Initialise fringe as an empty stack;
   Add node into fringe;
   while (fringe is not empty) {
      get&remove n* from the fringe;
      set n* as visited;
      add all unvisited neighbour of n* into fringe;
   }
}
```

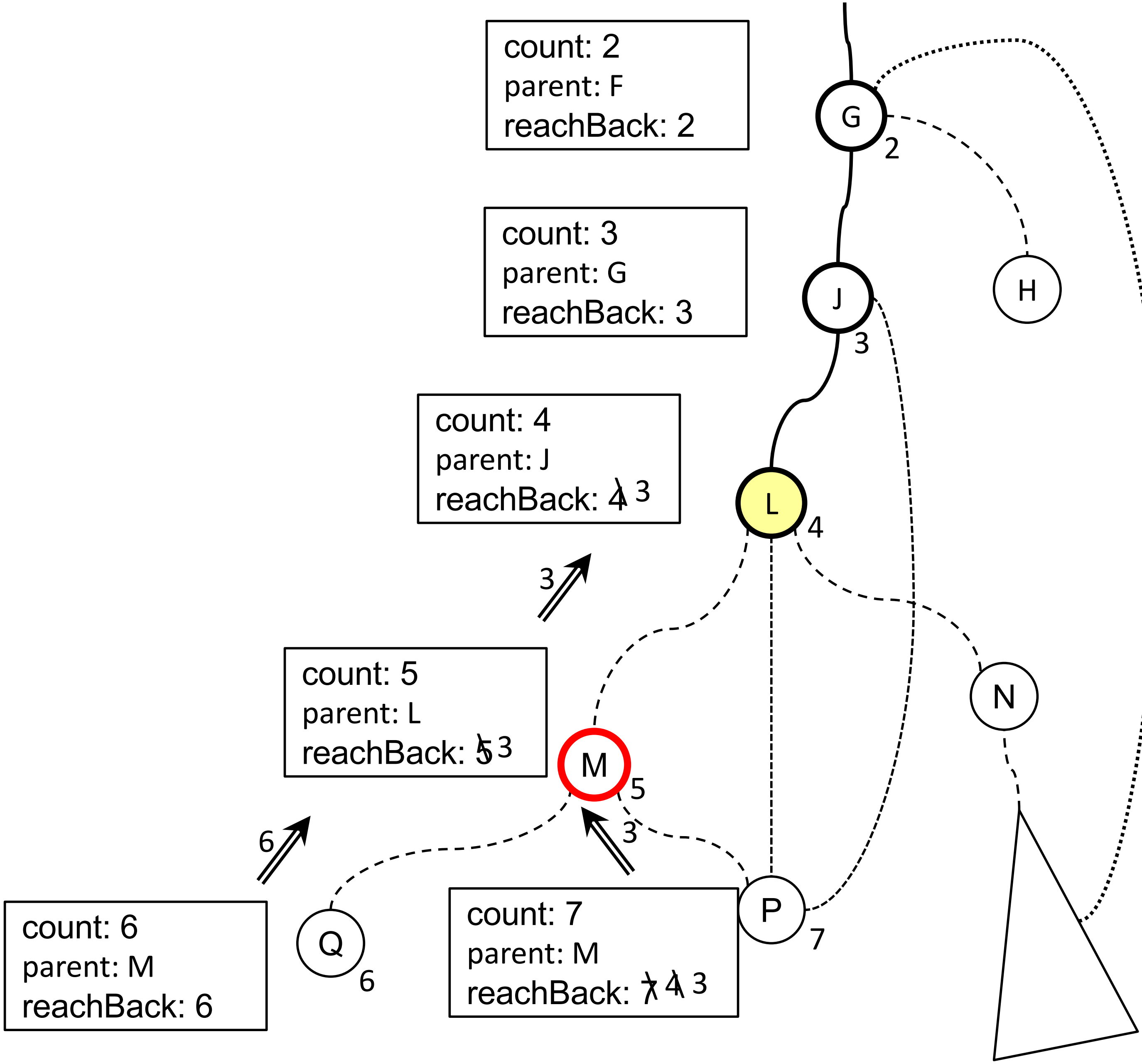
Recursive AP Algorithm

```
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;
}
```

Example of Recursive AP Algorithm



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

- **Goal**: set/update reachBack of each node in the sub-tree
- For each node, reachBack is set/updated in the following cases:
  - When first visited: reachBack = count
  - When one of its children's reachBack is updated: reachBack = min(childReach, reachBack)
  - When all the children have been calculated, update the reachBack of its parent
- Need to update the node information in different stages of DFS

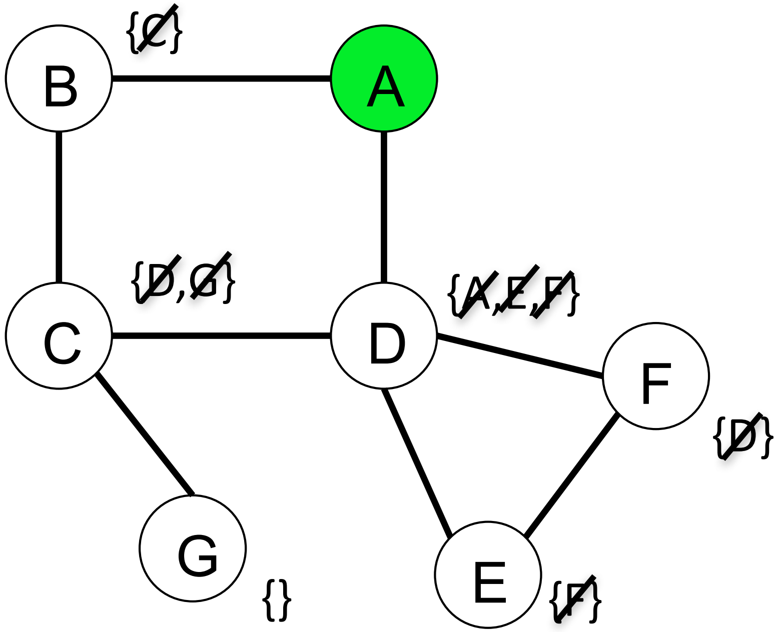
- However, traditional DFS updates node information only once (when visiting the node), and never revisits the node again
- Need a new data structure (modified stack)
  - Last-In-First-Out
  - Traditional stack: pop = get and remove
  - Modified stack: peek = get but not remove

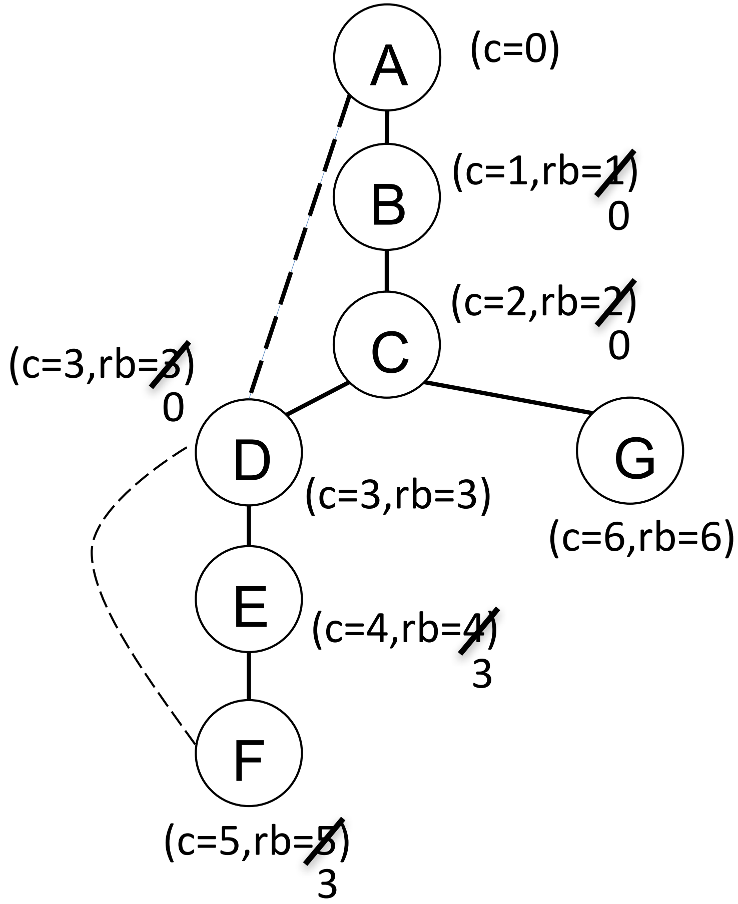
Iterative AP Algorithm

```
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 of Iterative AP Algorithm

<B,1,A>
<C,2,B>
<D,3,C>
<E,4,D>
<F,5,E>
<G,6,C>





APs				
	D	**C**		

Summary

- Two versions of implementing the AP algorithm
  - Recursive
  - Iterative

- DFS can be implemented in two ways
- Recursive is more straightforward to design
- Iterative is more tricky
  - Need to process nodes in different stages
  - A new operation to the fringe: **peek** - process without removing