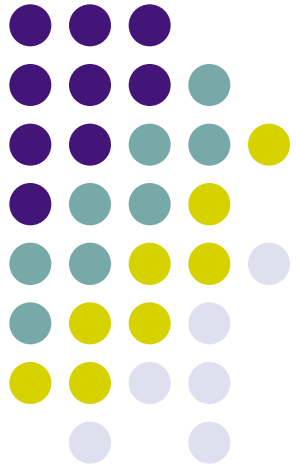
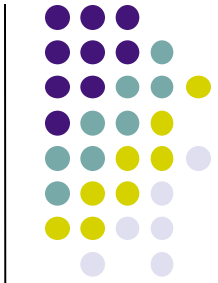
COMP20003
Algorithms and Data Structures
Deletion from BST

Nir Lipovetzky
Department of Computing and Information Systems
University of Melbourne
Semester 2

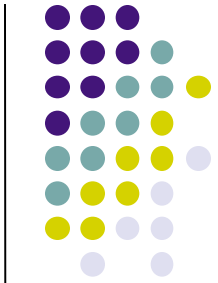


Binary search trees: Deletion



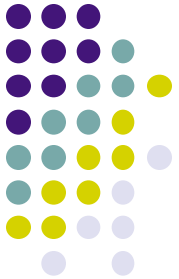
- Deletion?
- Deletion from a bst involves;
  - the in-order predecessor; or
  - the in-order successor.
- In-order successor and in-order predecessor can be obtained from in-order traversal.

Traverse



- Visit every node once
- Do something during the visit:
  - Print node value, or
  - Mark node as visited
  - Check some property of node
- Use in any linked data structure
  - Tree
  - Graph
  - List

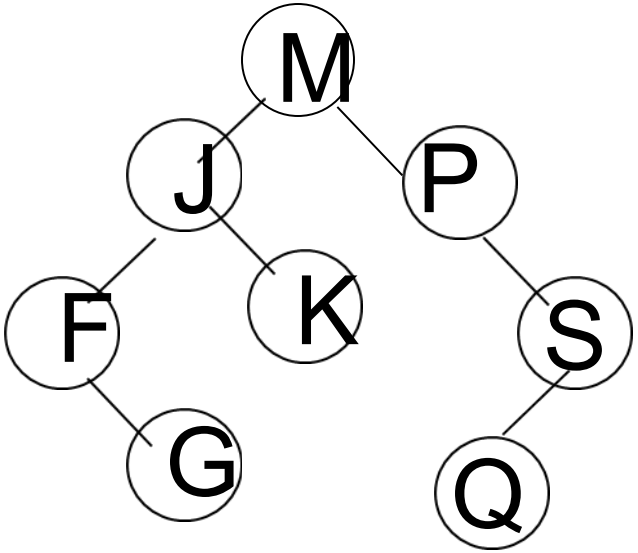
Traversal: recursive
In-order traversal, tree



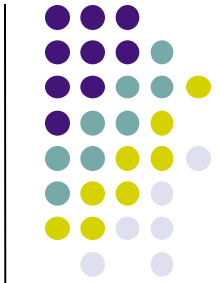
```
traverse(struct node *t)
{
   if(t!=NULL)
   {
       traverse(t->left);
     visit(t);
     traverse(t->right);
   }
 }
```

Exercize

- Trace recursive in-order tree traversal on the following tree, with visit(t) as print.

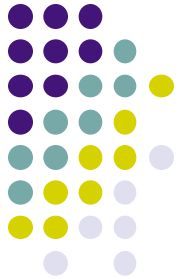


In-order traversal: Application:



- For a binary search tree, an in-order traversal prints all nodes in key-order.

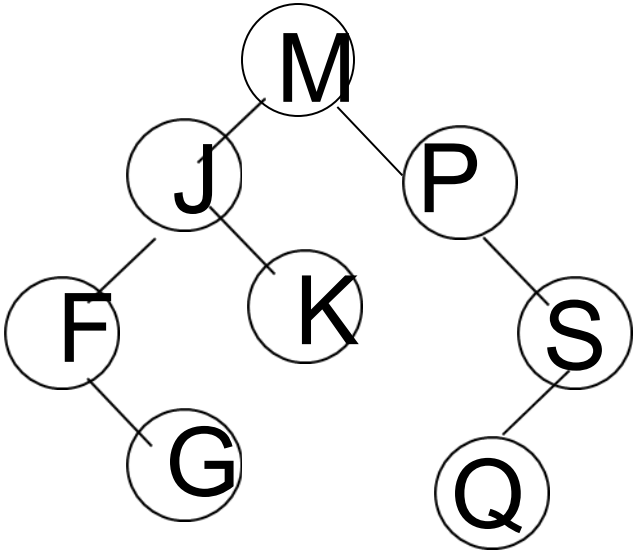
Post-order Traversal



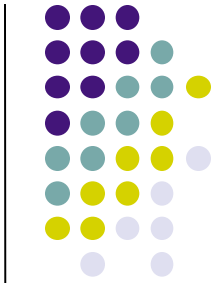
```
traverse(struct node *t)
{
   if(t!=NULL)
   {
       traverse(t->left);
     traverse(t->right);
       visit(t);
   }
 }
```

Exercize

- Trace recursive post-order tree traversal on the following tree, with visit(t) as print.

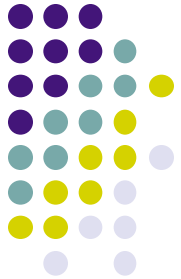


Post-order traversal: Application:



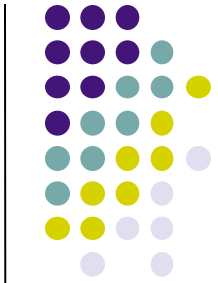
- Free all nodes in tree (free left and right nodes before freeing current node)
- Note: can’t free a tree by just freeing the root!

Pre-order Traversal



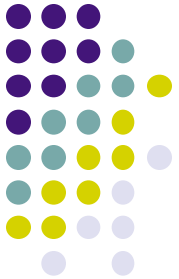
```
traverse(struct node *t)
{
   if(t!=NULL)
   {
      visit(t);
        traverse(t->left);
      traverse(t->right);
    }
 }
```

Pre-order traversal: Application:

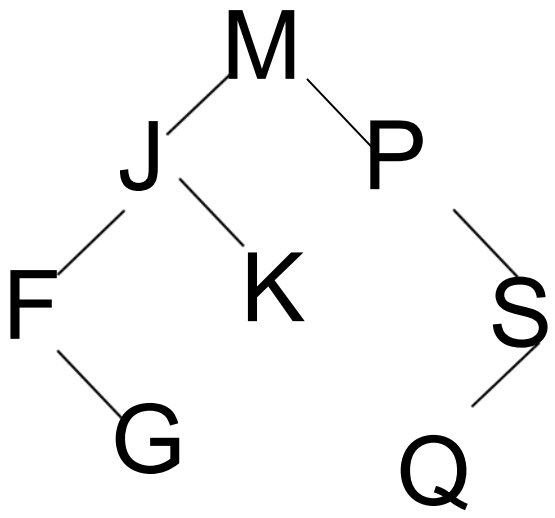


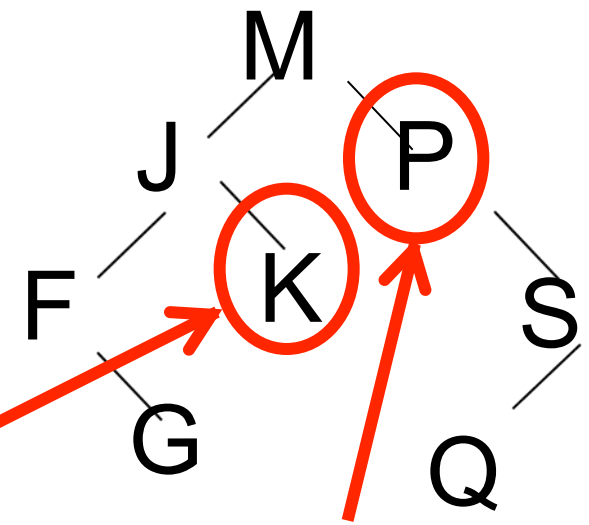
- Can be used to Copy a tree

In-order traversal: Application:



- For a binary search tree, an in-order traversal prints all nodes in key-order.





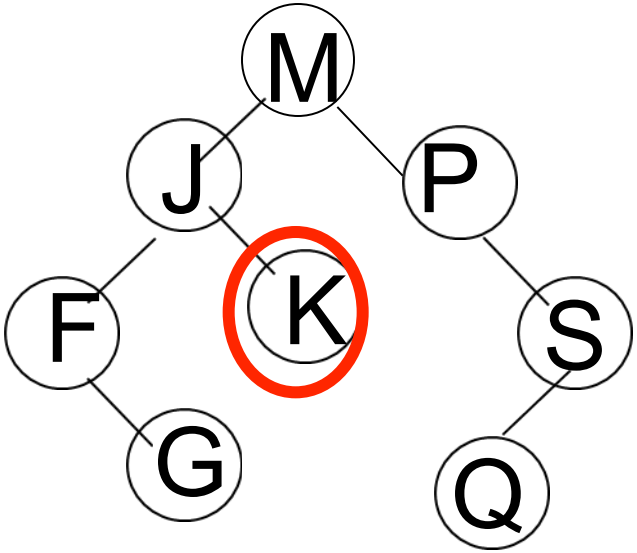
In-order predecessor of root

In-order successor of root

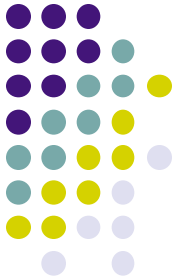
In-order successor and in-order predecessor

In-order: FGJKMPQS

In-order **predecessor** of root M is **rightmost** node of **left** subtree.

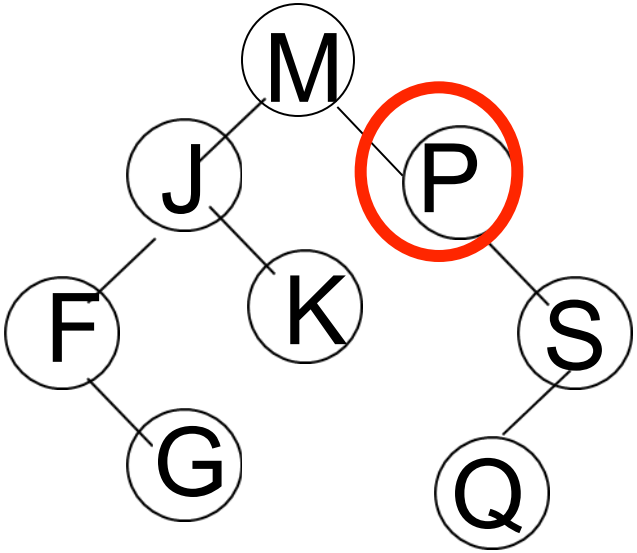


In-order successor and in-order predecessor



In-order: FGJKMPQS

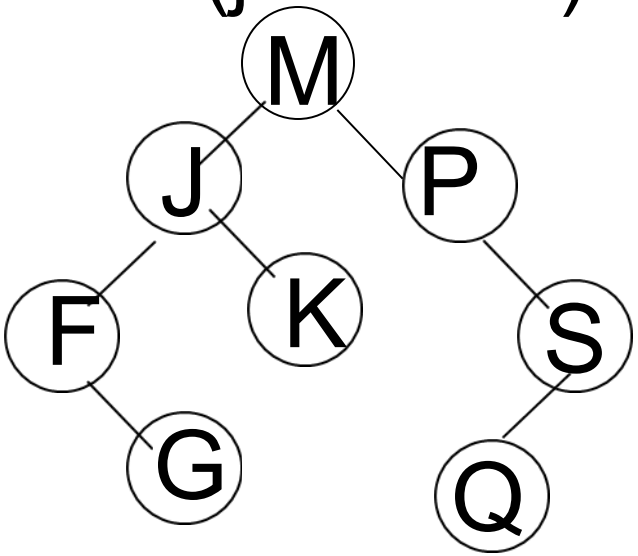
In-order **successor** of root M is **leftmost** node of **right** subtree.



In-order successor and in-order predecessor

In-order: FGJKMPQS

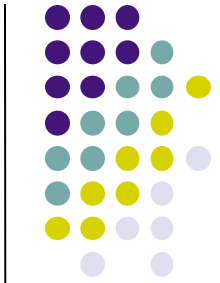
Every node has a predecessor (just before) and a successor (just after):



What are in-order predecessor and successor of node J?

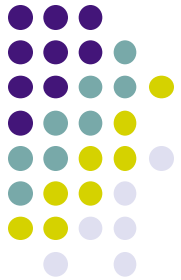
What are in-order predecessor and successor of node P?

In-order predecessor and in-order successor



- Just before (or after) in in-order traversal
  - Rightmost node in the left subtree; or
  - Leftmost node in the right subtree

Deletion from bst (finally)

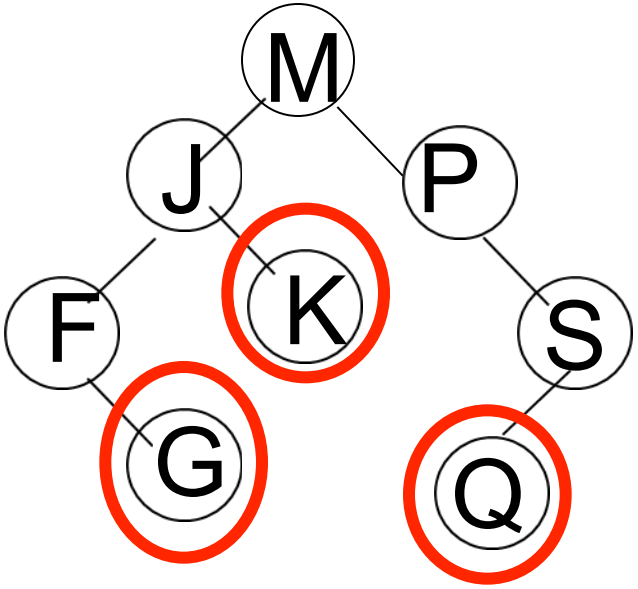


- Step 1: find the node to be deleted.
- Step 2: delete it!
- Three cases for deletion:
  - Case 1: Node is a leaf.
  - Case 2: Node has *either* a left child *or* a right child, but *not both*.
  - Case 3: Node has *both* a left child *and* a right child.

Case 1: Node is a leaf

In this example: G, K, Q

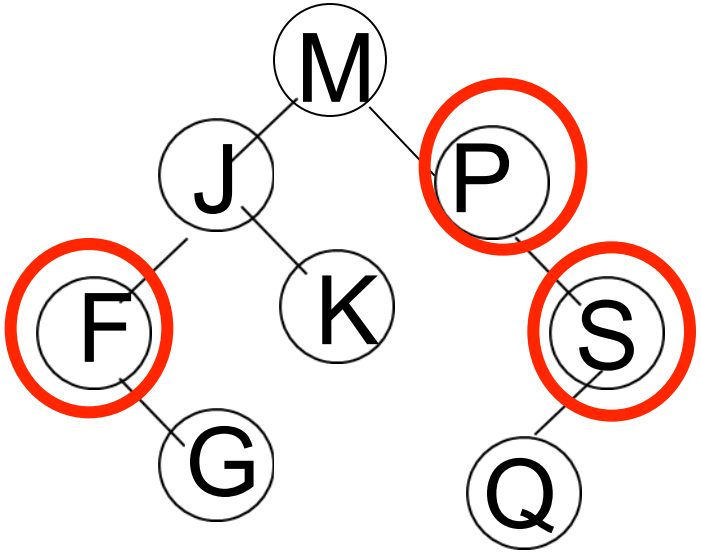
Just delete the node.



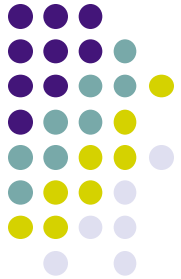
Case 2: Node has *one* child

In this example: F, S, P

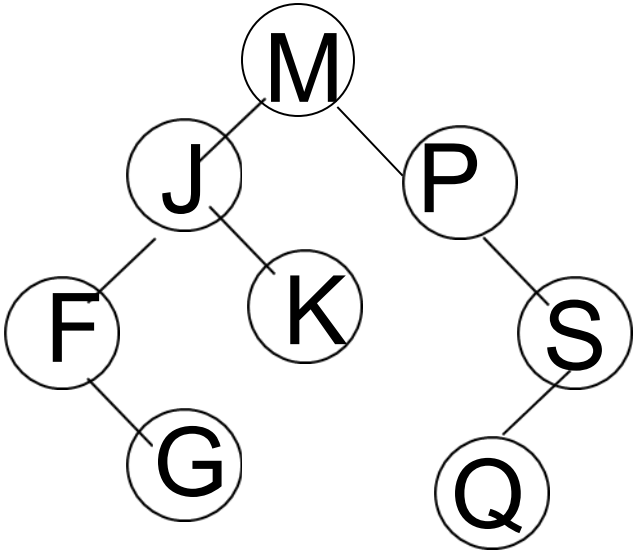
Replace node with the child.



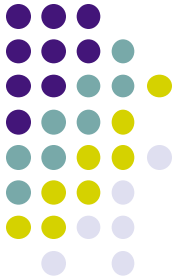
Case 3: Node has *two* children



In this example: M, J



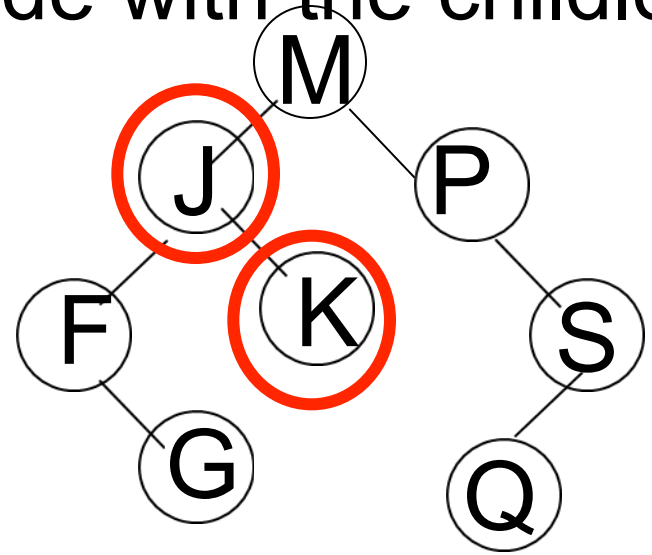
Case 3a: Node has *two* children, but…



… one of these children has no children.

In this example: J

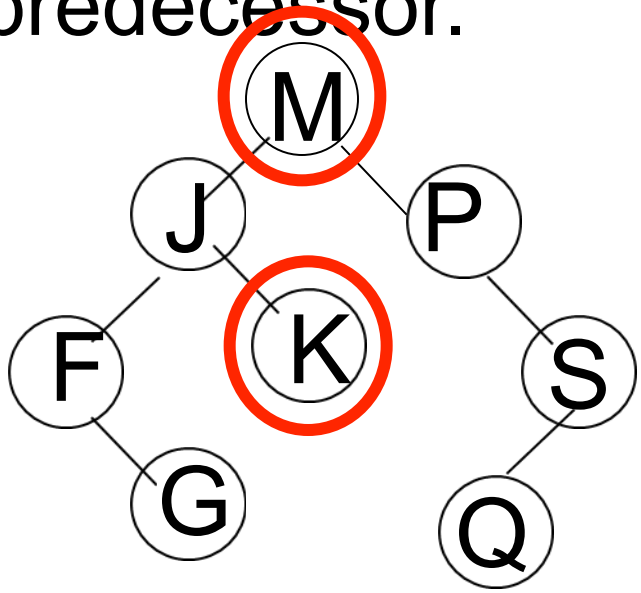
Replace node with the childless child (K).



Case 3b: Node has *two* children, both have children

In this example: M

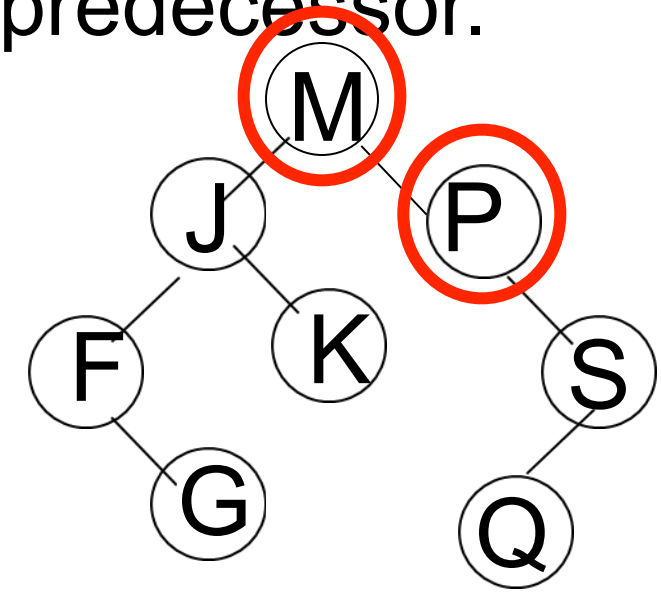
Replace node with *either* in-order successor or in-order predecessor.



Case 3b: Node has *two* children, both have children

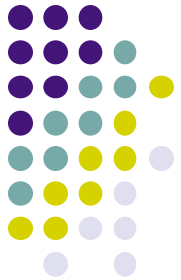
In this example: M

Replace node with *either* in-order successor or in-order predecessor.



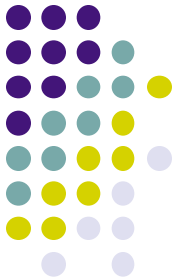
(and if P had a left child, that node would be the In-order successor of M, so would replace M with that node)

Deletion from bst:



- Step 1: find the node to be deleted.
- Step 2: delete it!
- Replace the deleted node with:
  - Case 1: Node is a leaf: nothing.
  - Case 2: Node has *either* a left child *or* a right child, but *not both*: the single child
  - Case 3: Node has *both* a left child *and* a right child: in-order predecessor or successor.

Deletion from bst: Analysis



- Worst case:
  - Time to find the node: O( )
  - Time to find the in-order predecessor or successor: O( )
  - Total time:
- Average case:
  - Time to find the node: O( )
  - Time to find the in-order predecessor or successor: O( )
  - Total time: