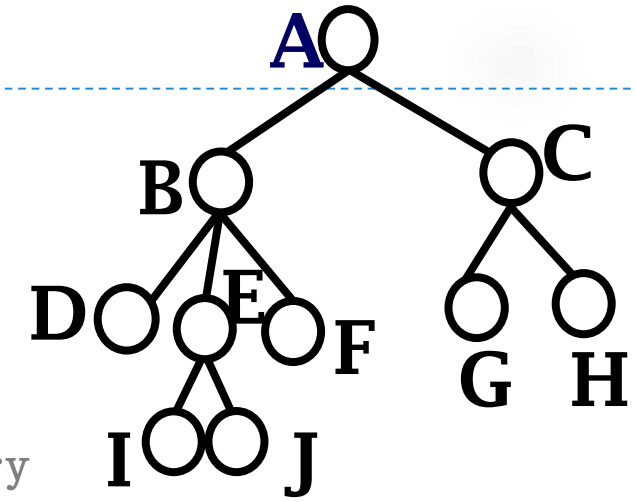
Data Structures and Algorithms (6)

Instructor: Ming Zhang
Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Chapter 6 Trees



- General Definitions and Terminology of Tree
  - Trees and Forest
  - Equivalence Transformation between a Forest and a Binary Tree
  - Abstract Data Type of Tree
  - General Tree Traversals
- Linked Storage Structure of Tree
- Sequential Storage Structure of Tree
- K-ary Trees

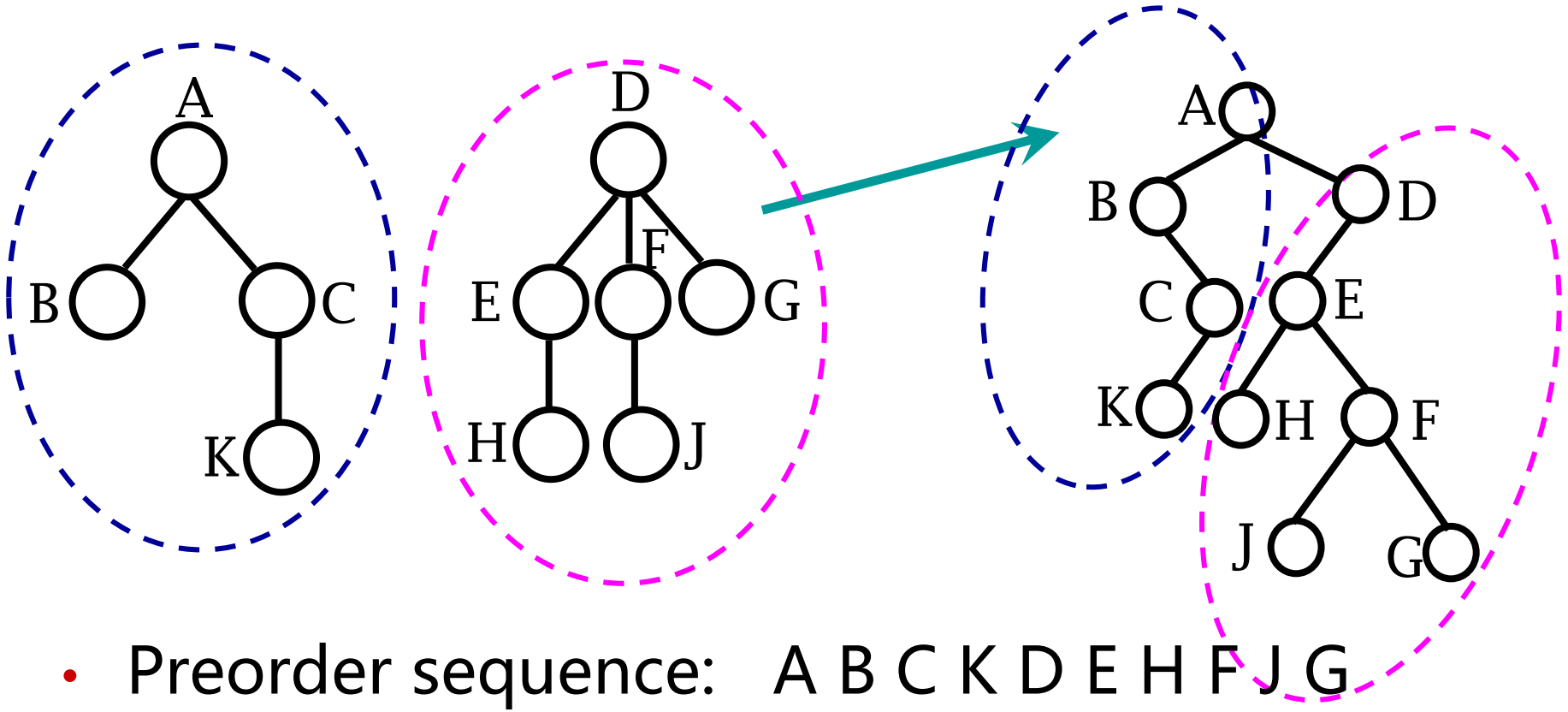
Abstract Data Type of Tree

```
template<class T>
class TreeNode  {                                  // The ADT of the tree node
public:
  TreeNode(const T& value);                        // Constructor
  virtual ~TreeNode() {};                          // Destructor
  bool isLeaf();                                   // Check whether the current node is the
                                                   //    leaf node or not
  T Value();                                       // Return the value of the node
  TreeNode<T> *LeftMostChild();                    // Return the left-most (first) child
  TreeNode<T> *RightSibling();                     // Return the right sibling
  void setValue(const T& value);                   // Set the value of the current node
  void setChild(TreeNode<T> *pointer);             // Set the left child
  void setSibling(TreeNode<T> *pointer);           // Set the right sibling
  void InsertFirst(TreeNode<T> *node);             // Insert a node as the left child
  void InsertNext(TreeNode<T> *node);              // Insert a node as the right sibling
};
```

Abstract Data Type of Tree

```
template<class T>
class Tree  {
public:
    Tree();                                          // Constructor
    virtual ~Tree();                                 // Destructor
    TreeNode<T>* getRoot();                          // Return the root node
    void CreateRoot(const T& rootValue);             // Create a root node whose value is rootValue
    bool isEmpty();                                  // Check whether it is an empty tree
    TreeNode<T>* Parent(TreeNode<T> *current);       // Return parent node
    TreeNode<T>* PrevSibling(TreeNode<T> *current);  // Return the previous sibling
    void DeleteSubTree(TreeNode<T> *subroot);        // Delete the subtree rooted at "subroot"
    void RootFirstTraverse(TreeNode<T> *root);       // Depth-first preorder traversal
    void RootLastTraverse(TreeNode<T> *root);        // Depth-first postorder traversal
    void WidthTraverse(TreeNode<T> *root);           // Breath-first traversal
};
```

Traversal of a Forest



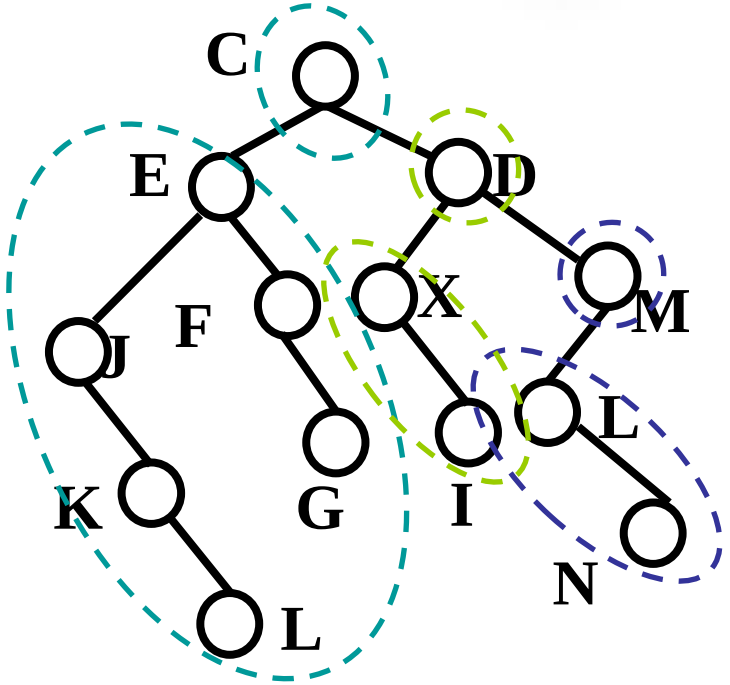
- Preorder sequence: A B C K D E H F J G
- Postorder sequence: B K C A H E J F G D

Forest Traversal vs. Binary Tree Traversal

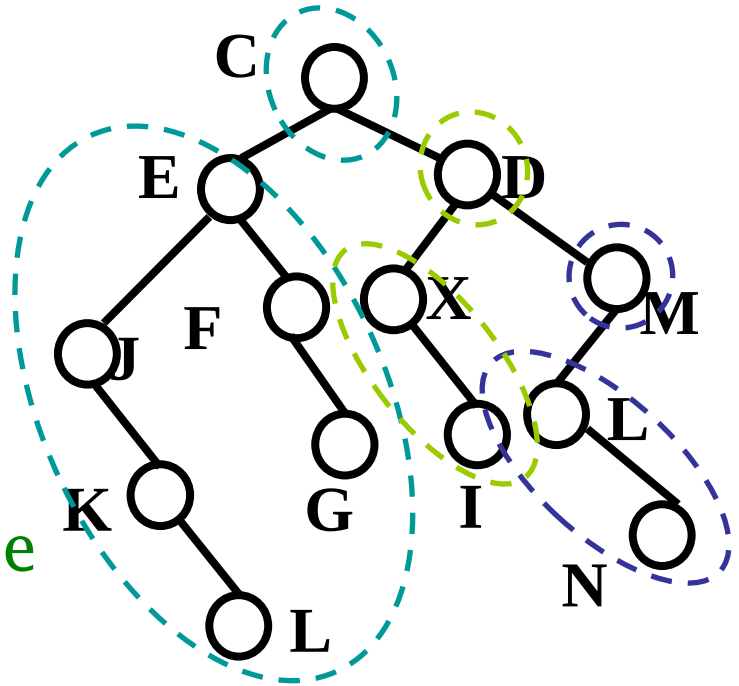
- Preorder forest traversal
  - Preorder binary tree traversal
- Postorder forest traversal
  - Inorder binary tree traversal
- Inorder forest traversal?
  - We cannot define between which two child nodes should the root locate.

Depth-First Preorder Forest Traversal

```
template<class T>
void Tree<T>::RootFirstTraverse(
        TreeNode<T> * root)  {
    while (root != NULL) {
        Visit(root->Value());     // Visit the current node
        // Traverse the subtree forest of the root node of
        // the first tree (except the root node)
        RootFirstTraverse(root->LeftMostChild());
        root = root->RightSibling();     // Traverse other trees
    }
}
```



Depth-First Postorder Traversal



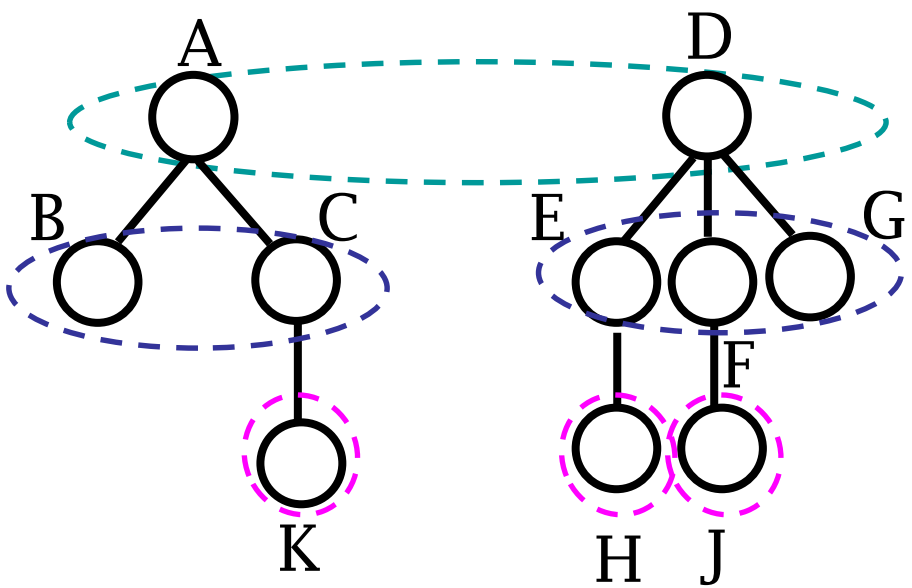

```
template<class T>
void Tree<T>::RootLastTraverse(
        TreeNode<T> * root)  {
    while (root != NULL) {
      // Traverse the subtree forest of the root node
      // of the first tree
      RootLastTraverse(root->LeftMostChild());
      Visit(root->Value());              // Visit the current node
      root = root->RightSibling();  // Traverse other trees
    }
}
```

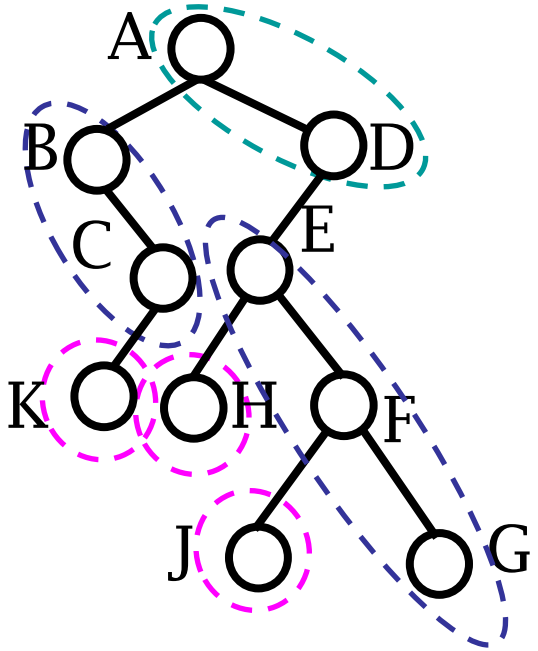
Breadth-First Forest Traversal

- Breadth-first forest traversal
  - Also be called level-order traversal

- a) First, visit the nodes who are at level 0
- b) Second, visit the nodes who are at level 1
- c) Continue until all the nodes at the deepest level are visited

Breadth-First Forest Traversal





- Breadth-first forest traversal sequence: A D B C E F G K H J
- **Look at the right diagonal of the binary tree storage structure**

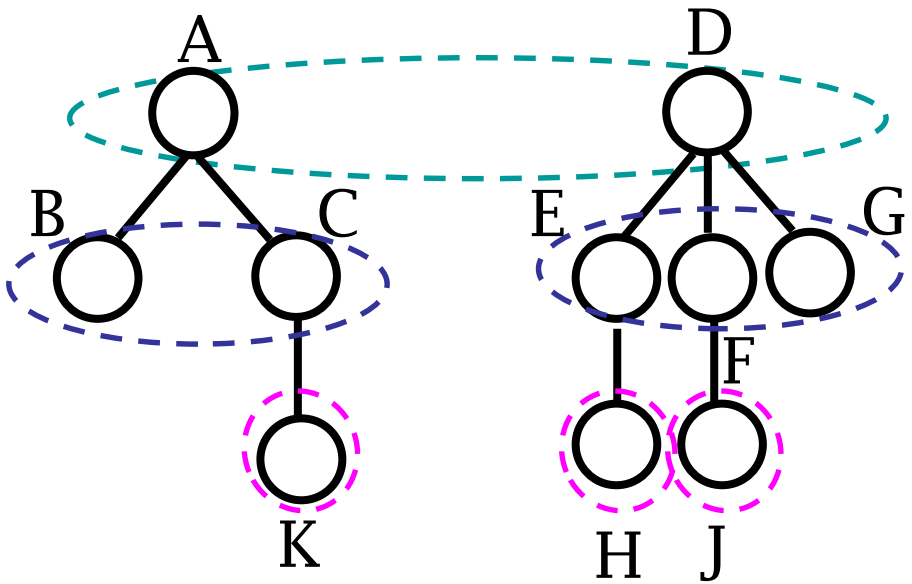
Breadth-first forest traversal

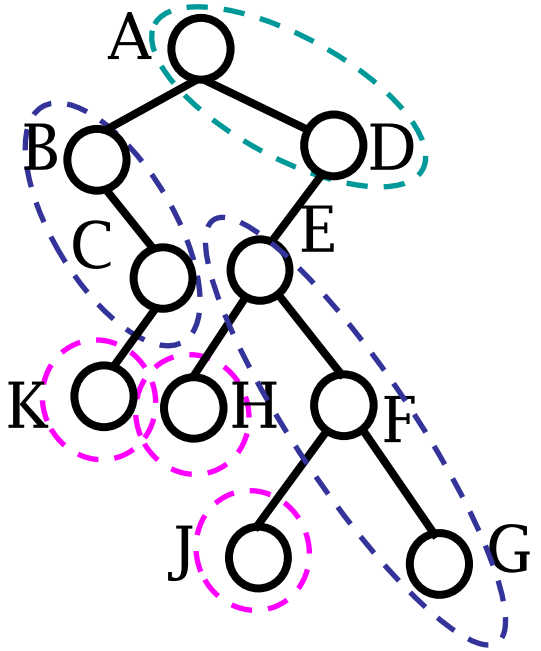
```
template<class T>
void Tree<T>::WidthTraverse(TreeNode<T> * root)  {
   using std::queue;                              // Use STL queue
   queue<TreeNode<T>*> aQueue;
   TreeNode<T> * pointer = root;
   while (pointer != NULL) {
      aQueue.push(pointer);                          // Put the current node go into the queue
      pointer = pointer->RightSibling();     // pointer pointing to right sibling
    }
 while (!aQueue.empty()) {
      pointer = aQueue.front();                 // Get the first element of the queue
      aQueue.pop();                          // Pop the current element out of the queue
      Visit(pointer->Value());                       // Visit the current node
      pointer = pointer-> LeftMostChild(); // pointer pointing to the first child
      while (pointer != NULL) {                 // Put the child nodes of the current node
                                    // into the queue
            aQueue.push(pointer);
            pointer = pointer->RightSibling();
    }  }  }
```

Questions

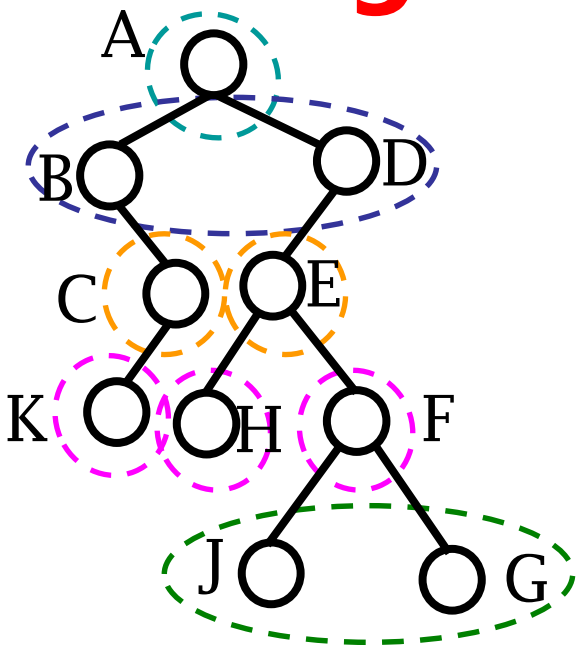
1. Can we use the template of preorder binary tree traversal to accomplish the preorder forest traversal?
2. Can we use the template of inorder binary tree traversal to accomplish the postorder forest traversal?
3. How to accomplish non-recursive depth-first forest search?

different opinions of breadth-first search





wrong



- Cannot use the template of breadth-first binary tree traversal. For example ,
  - In the left figure, the breadth-first forest traversal: A D B C E F G K H J
    - **Look at the tilt dotted circles in the middle**
  - In the right figure, the breadth-first binary tree traversal: A B D C E K H F J G
    - **Look at the parallel dotted circles on the right**

Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao

Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)