



Tree

Ja-Hee Kim



Agenda

01 Introduction

Terminology, size

02 Binary Tree

Definition, examples

03 Tree Traversal

Implementing a binary tree using an array

04 Implementation of Tree

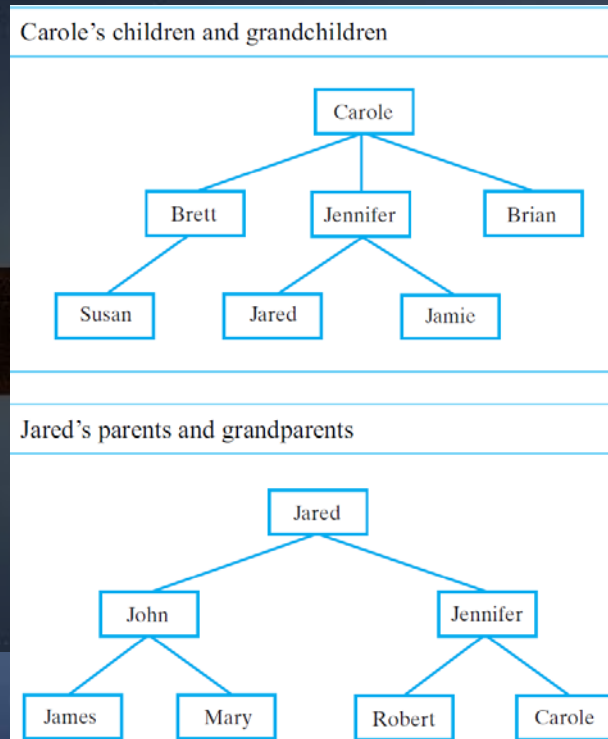
Array Tree and Linked Tree



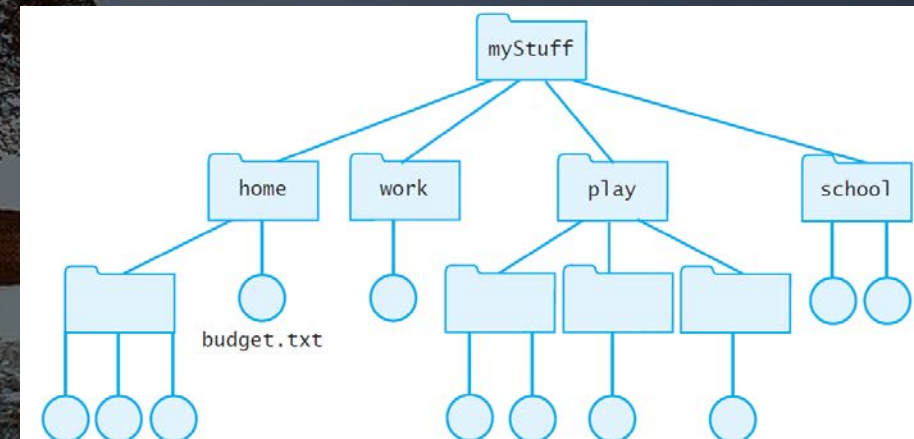
Introduction of Tree

Introduction of a Tree

- Definition
 - a non-linear structure based on nodes and links
- Example
 - Family tree



Files in folders



Rooted Tree

- Empty tree is a tree.
- If S is a set of trees
 - any trees of S do not share a node.
 - $T = (r, S)$ is a tree
 - r is a root
 - a tree in S is a sub-tree of T





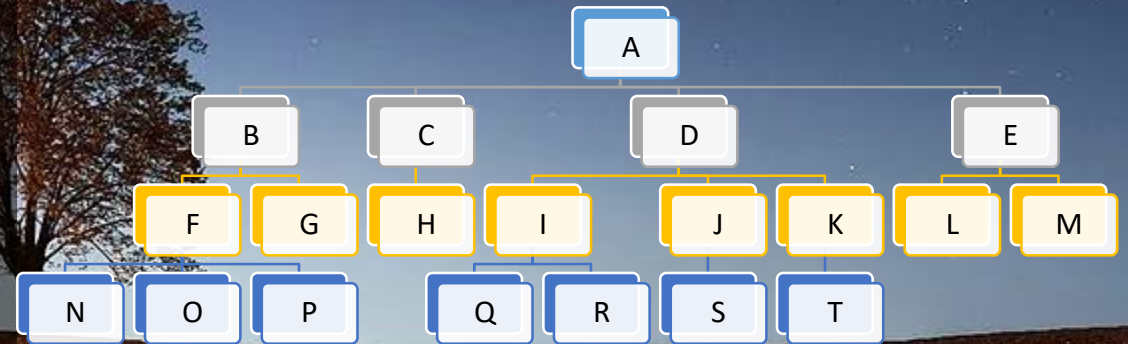
Terminologies of a Tree

- Node
- Edge
- Level
- Root
- Parent
- Children
- Ancestor
- Descendant
- Subtree
- Degree
- Leaf
- Interior node
- Path
- Height
- Depth
- Full
- Size



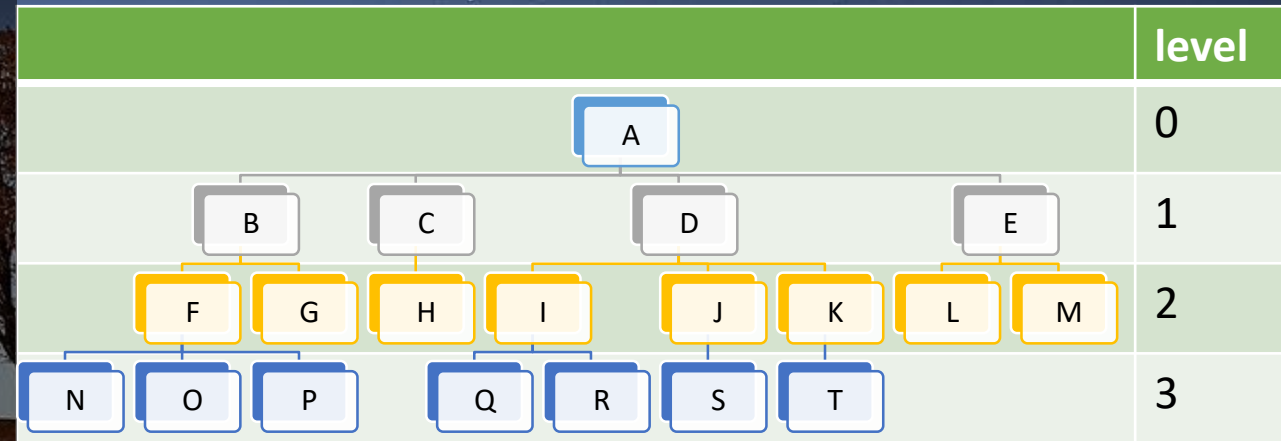
Node and edges

- A tree = nodes + edges
- Node = element
 - It usually contains data.
- Edge
 - the connecting link between any two nodes



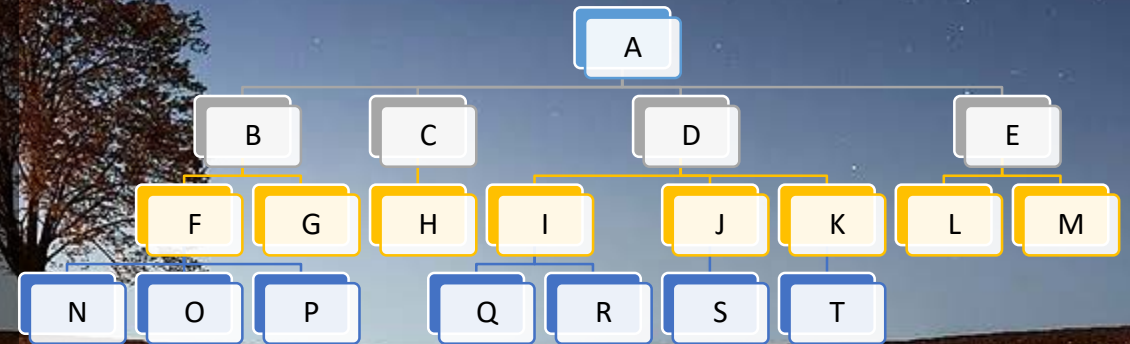
Level and Root

- Level
 - Node's hierarchy
- Root
 - A node in the top level
 - A single node in a tree



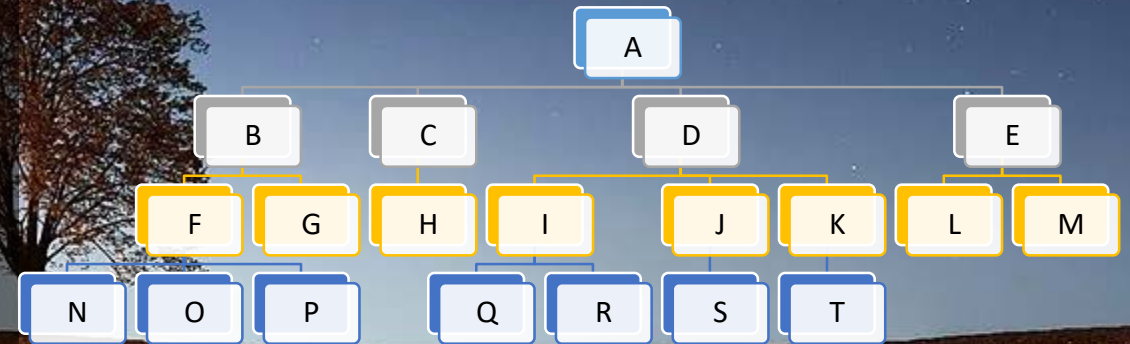
Parent & Children

- A node which has branch from it to any other node
- Parent node
 - A node which has child / children
 - If two nodes are connected, the closer to the root is a parent.
- Children node
 - If two nodes are connected, the closer to the root is a parent.
- Sibling
 - Nodes with the same parent.



Ancestor vs descendant

- descendant
 - either one of its children or a child of another descendant.
- Ancestor
 - If a node is an ancestor of an element x is an element of whom x is a descendant.



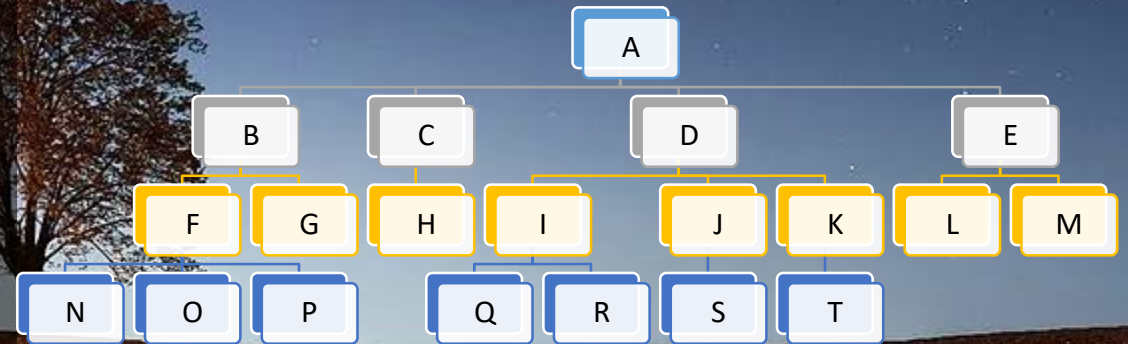
Subtree of a tree

- Subtree
 - Any node and its descendants form a subtree of the original tree
 - a tree rooted at a child of that node.



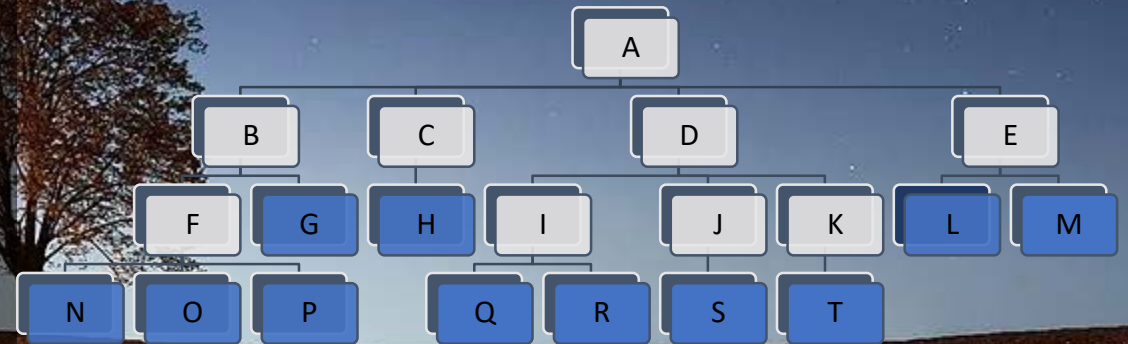
Degree

- The degree
 - the size of its subtree set
 - its number of children.
 - Ex: $\deg(D) = 3$
- The degree of a tree
 - the largest degree of its elements.
 - Ex: the degree of the right tree is 4 because of A.



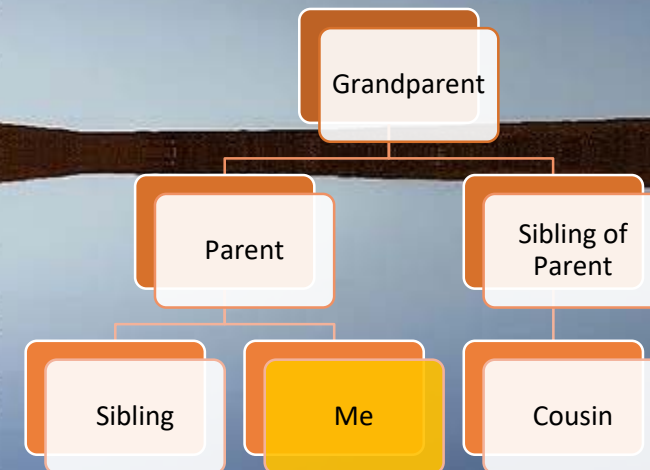
Leaf vs Interior node

- Leaf
 - A node of degree 0
 - A node with no children.
- Interior node
 - A node is not a leaf.



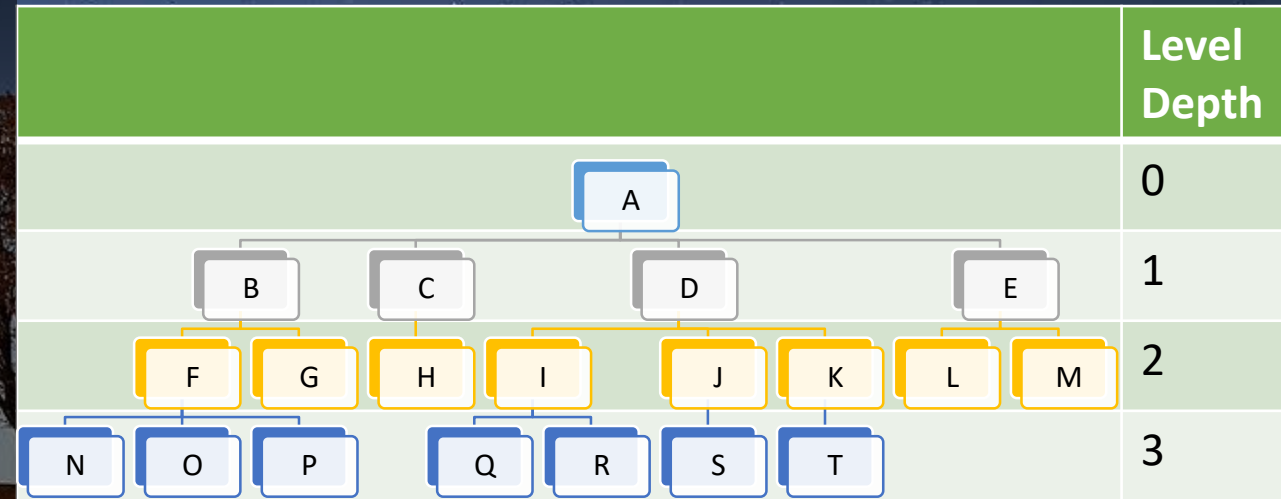
Path

- Path
 - a sequence of elements, each the parent of the next element in the sequence.
- root path
 - a path that begins at the root.
- root-to-leaf path
 - a root path that ends at a leaf.
- The length of a path
 - the number of adjacent pairs in the path; i.e., the number of steps in the path.




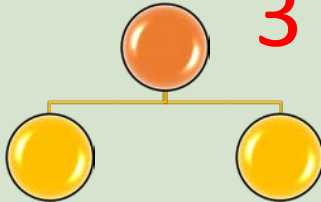
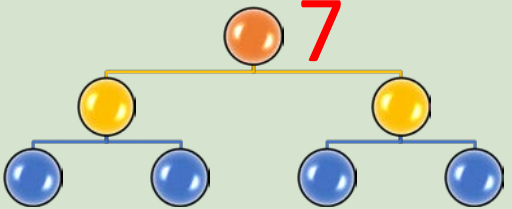

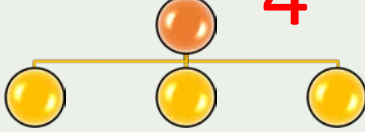

Depth and Height

- The height of a tree
 - its greatest level; the length of its longest root-to-leaf path.
- The depth
 - = Level
 - the length of its root path.



Full Tree

- A tree is full if all of its leaves are at the same level and all of its other nodes have the same degree.

Height Degree	0	1	2
2	 1	 3	 7
3	 1	 4	 13

$$n = \frac{d^{h+1} - 1}{d - 1} \quad \frac{3^{2+1} - 1}{3 - 1} = \frac{27 - 1}{2} = 13$$

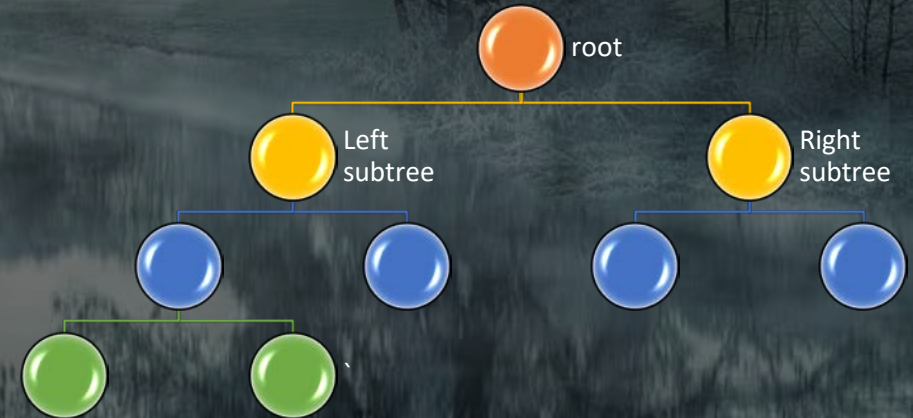
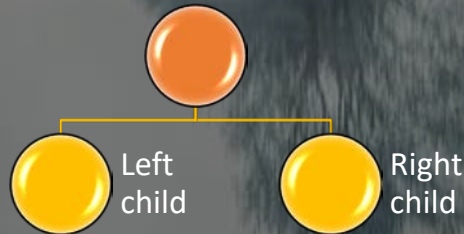




Binary Tree

Binary Tree

- A tree of which the maximum degree is two
≠ a tree of 2 degree
- Recursive definition of a binary tree
 - Empty tree is a binary tree.
 - Each tree has two subtrees whose root nodes are the nodes pointed to by the leftLink and rightLink of the root node.



Ordered Tree

- An ordered tree is an oriented tree in which the children of a node are somehow "ordered."



Ordered tree \neq Unordered tree

Terminologies

- Full binary tree
- Complete binary tree
- Skewed binary tree



Full Binary Tree

- Every node in the tree has either 0 or 2 children.
- The number of node of full binary tree $2^{h+1} - 1$
 - Where h = the height of the tree



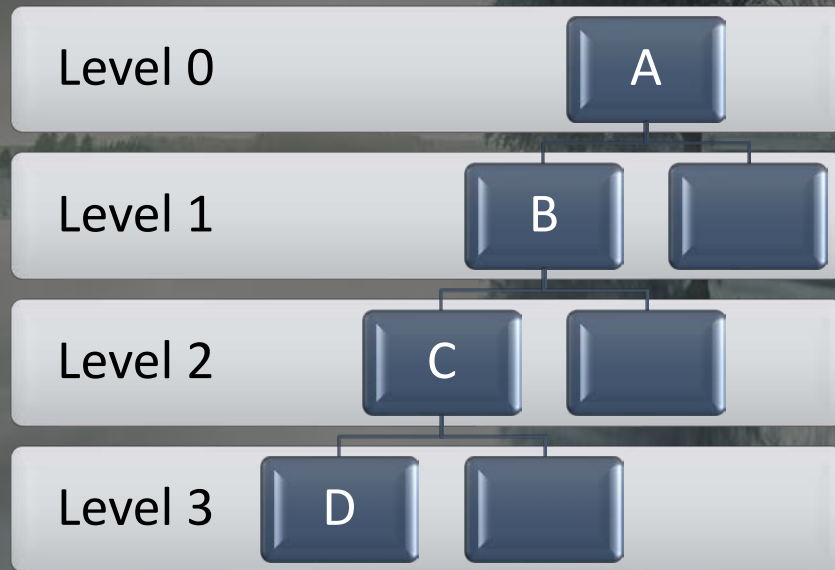
Complete Binary Tree

- In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible.



Skewed Binary Tree

- Every node has only left (or right) sub



Left Skewed Binary Tree



Right Skewed Binary Tree

The number of nodes

- The number of nodes of which tree (n)
 - Where the height is h
 - Is more than or equal to a skewed tree
 - Is less than or equal to a full tree

$$h + 1 \leq n \leq 2^{h+1} - 1$$

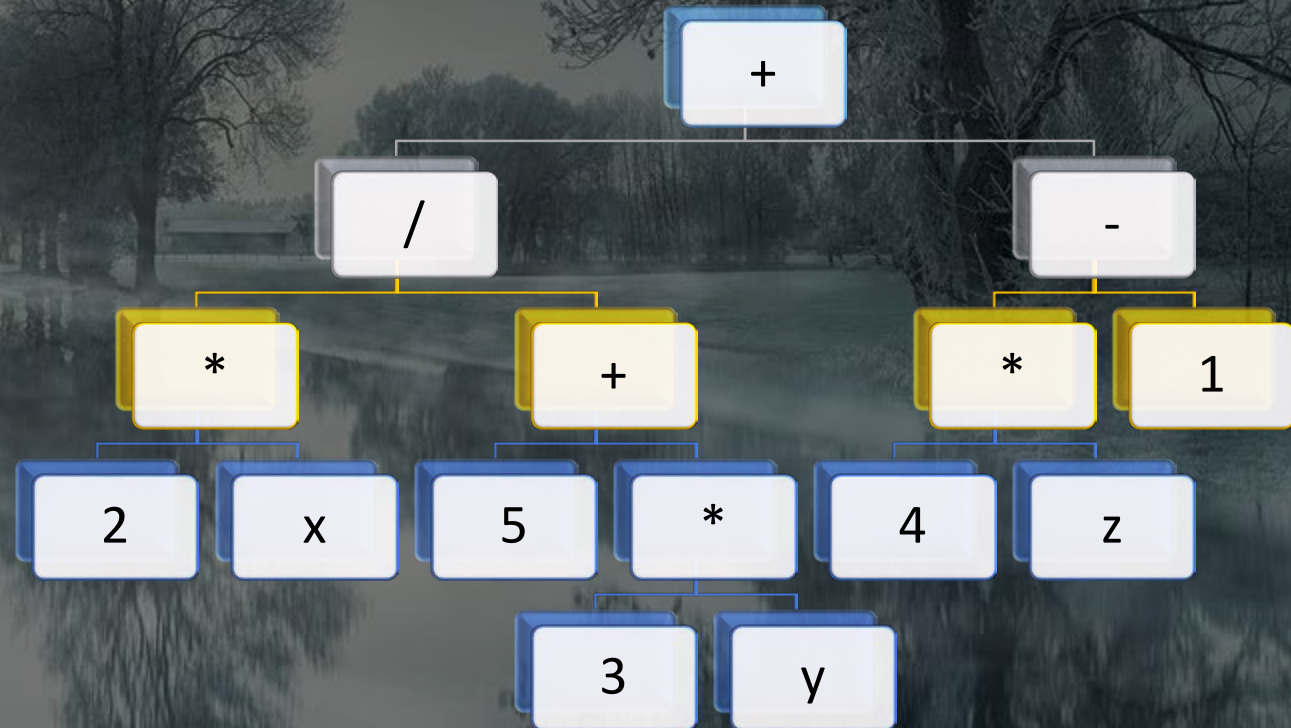


$h=3$



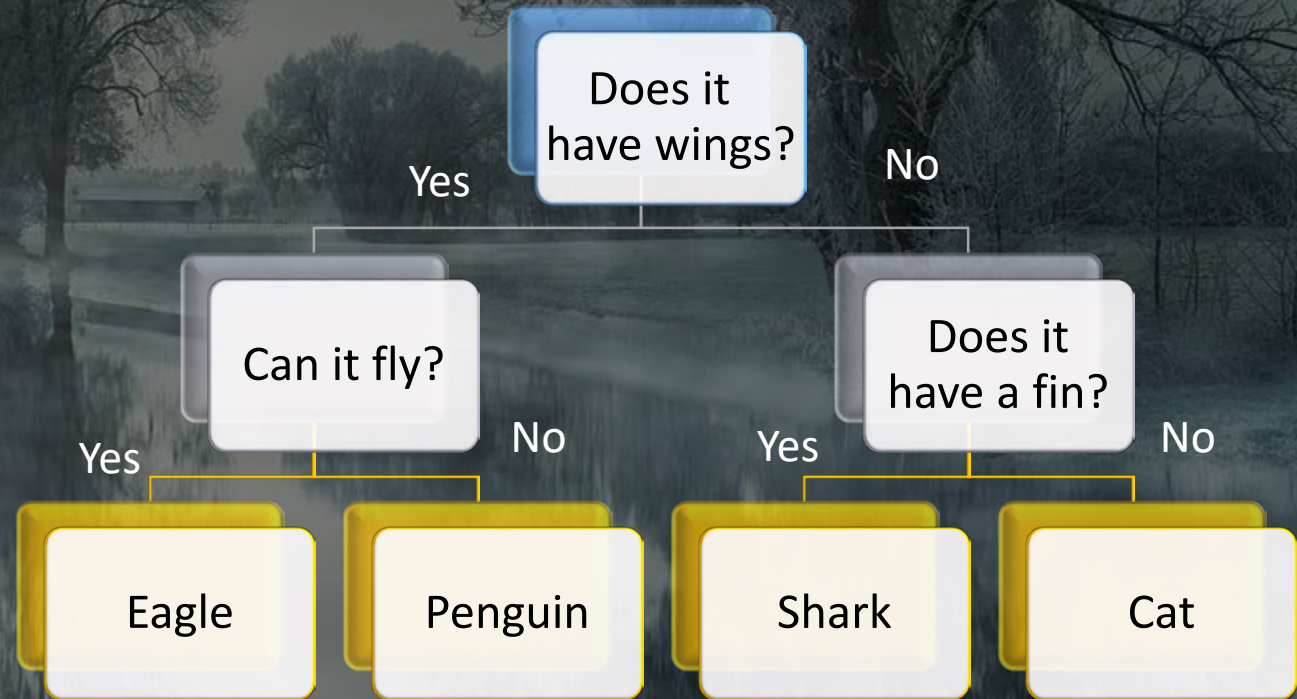
Example1: Expression Tree

- an arithmetic expression such that Each operand is in a distinct leaf.
- Each binary operator is in a distinct internal node whose two subtrees represent the two values upon which it operates.
- Ex: $\frac{2x}{5+3y} + (4z - 1)$



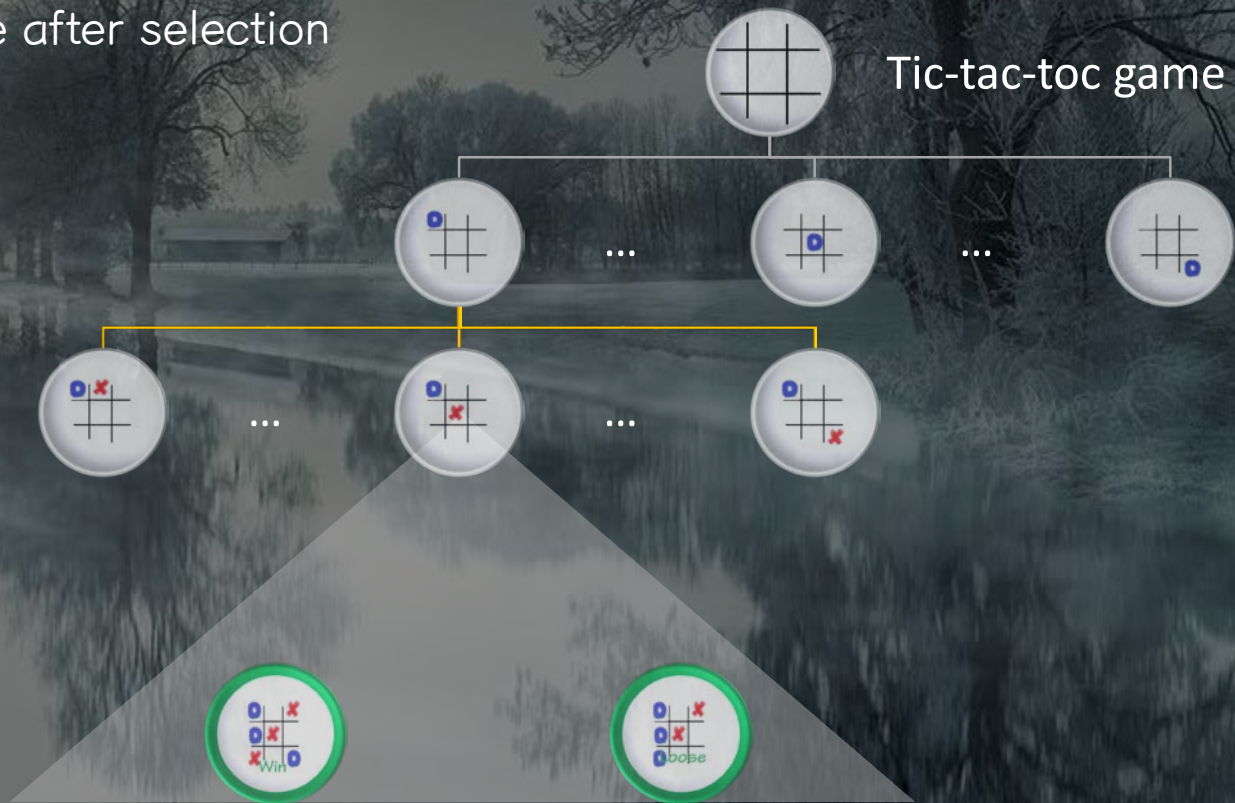
Example2: Decision Tree

- Expert System
 - System for helping its user to solve problems or make decisions
- Decision Tree
 - The basis of an expert system
 - Interior node: question
 - Leaf: conclusion



Example3: Game Tree

- A kind of a decision tree
- Interpretation
 - Node: a state of the game
 - Children: possible next state after selection
 - Leaf: win or lose







Tree Traversal

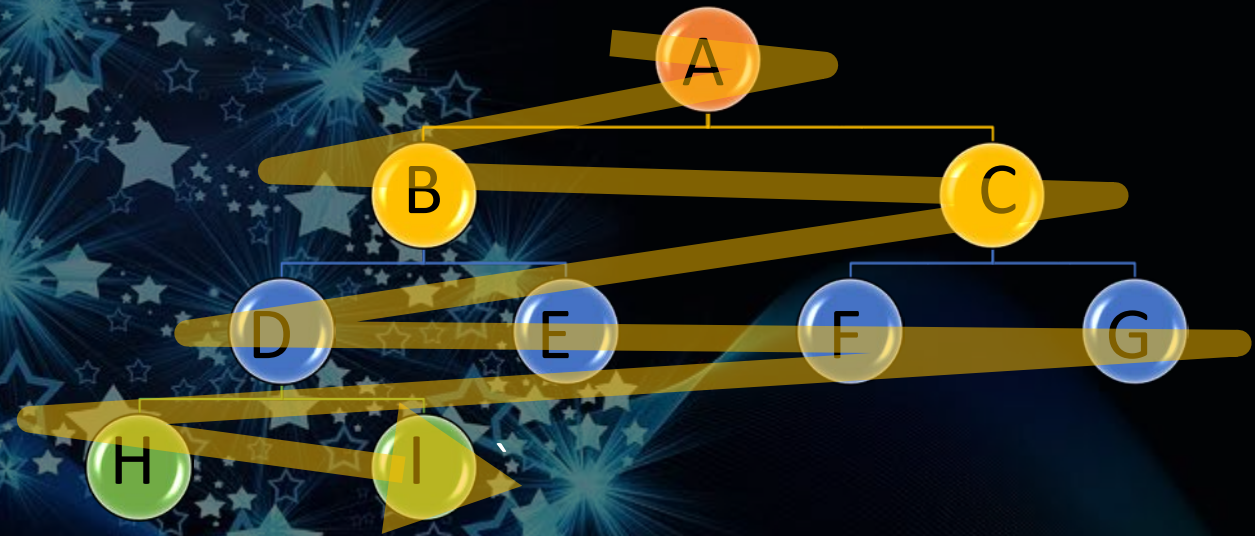
Traversals for Ordered Tree

- BFS (Breadth First Search)
 - Level Order Traversal
- DFS (Depth First Search)
 - Pre-order Traversal
 - In-order Traversal (Only for a binary tree)
 - Post-order Traversal



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

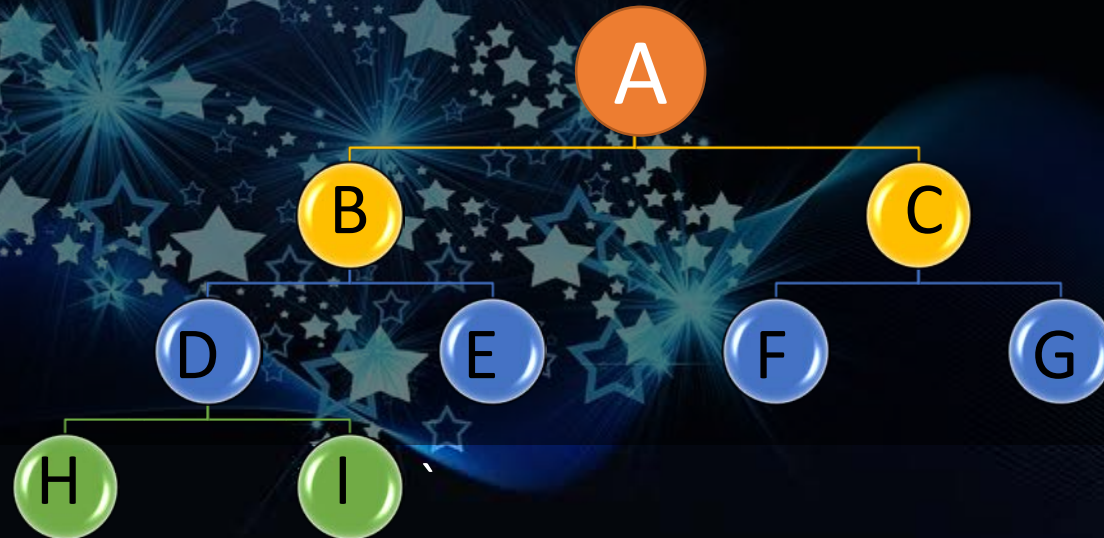
queue



BFS

queue

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

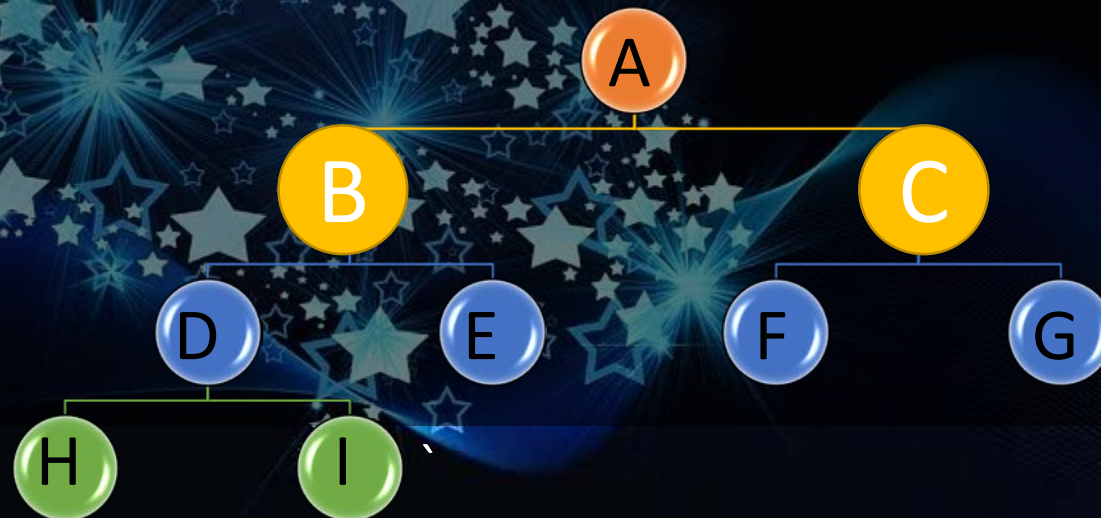


BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

queue

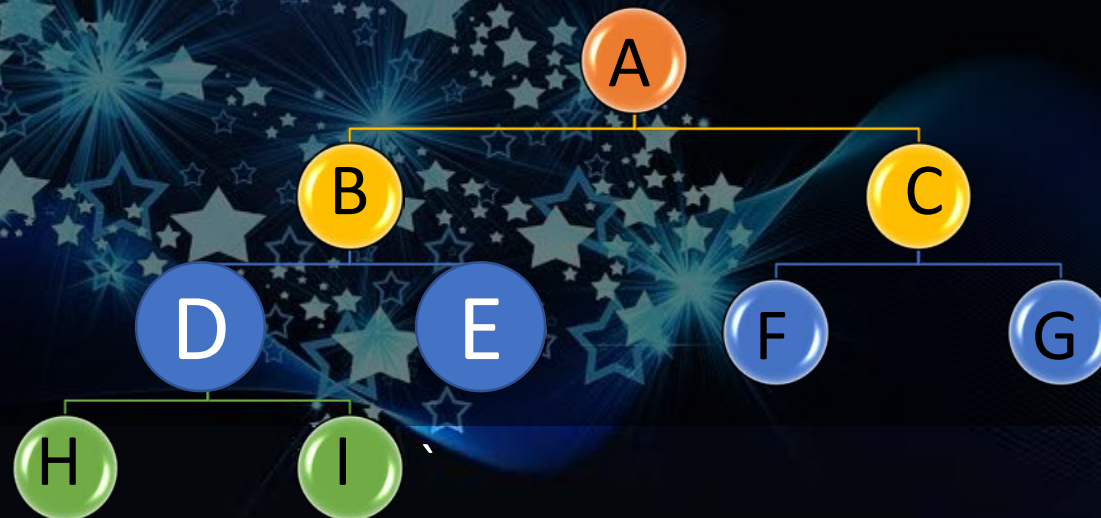
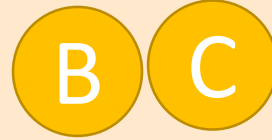
A



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

queue



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

queue

C

D

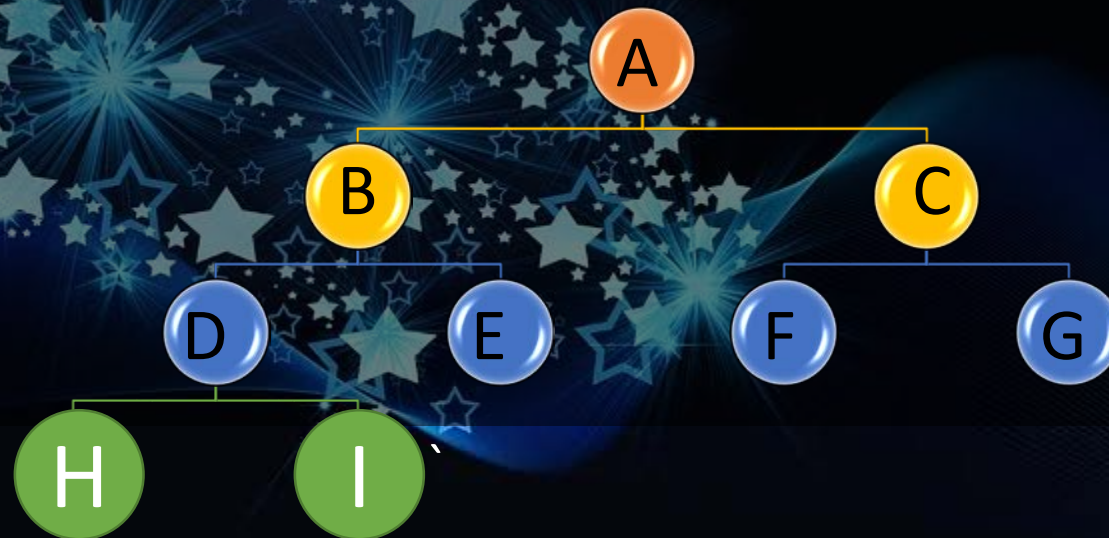
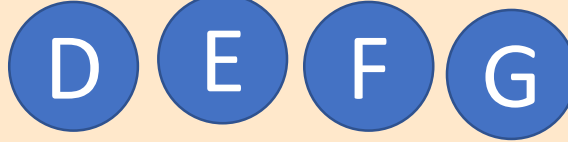
E



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

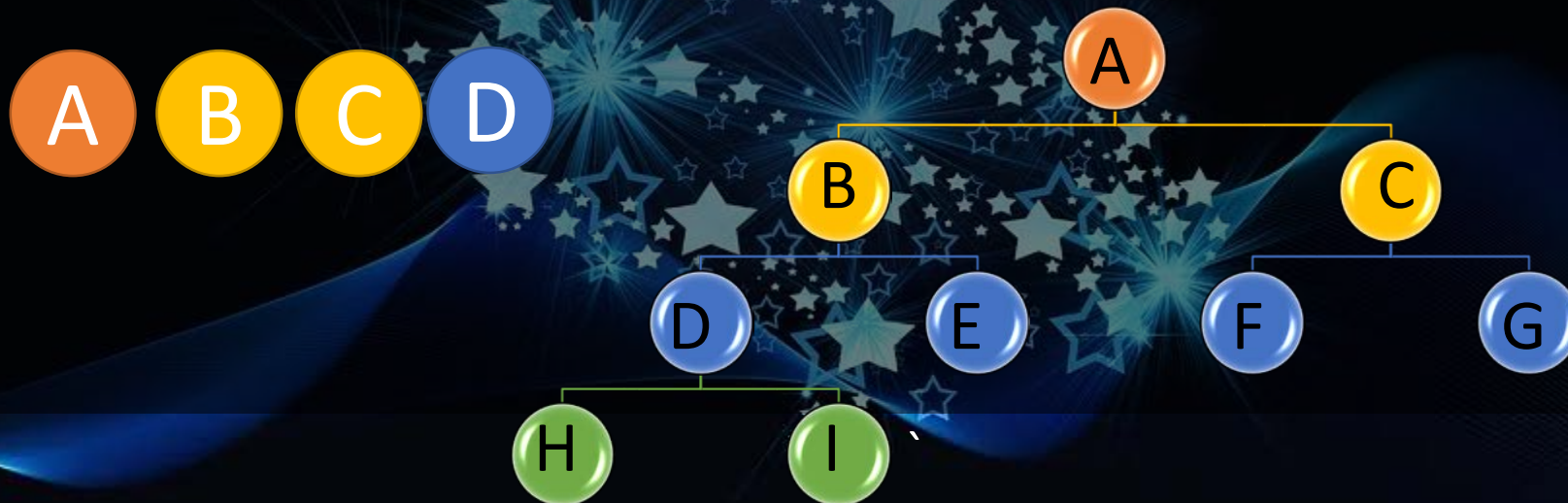
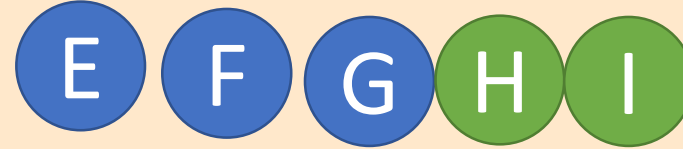
queue



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

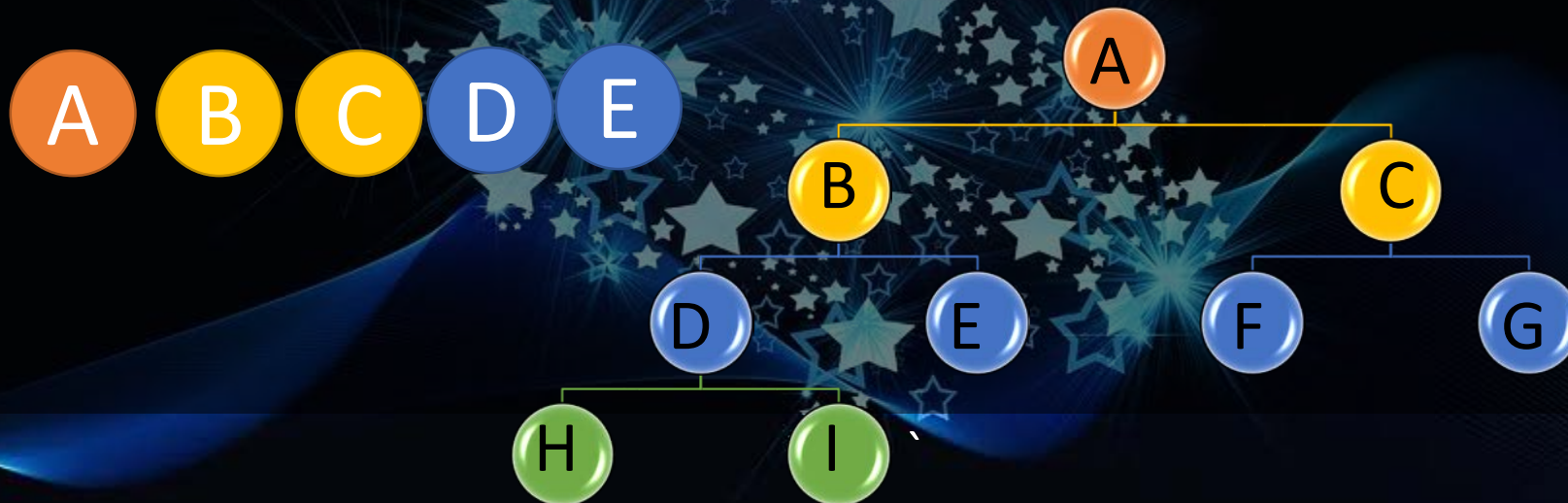
queue



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

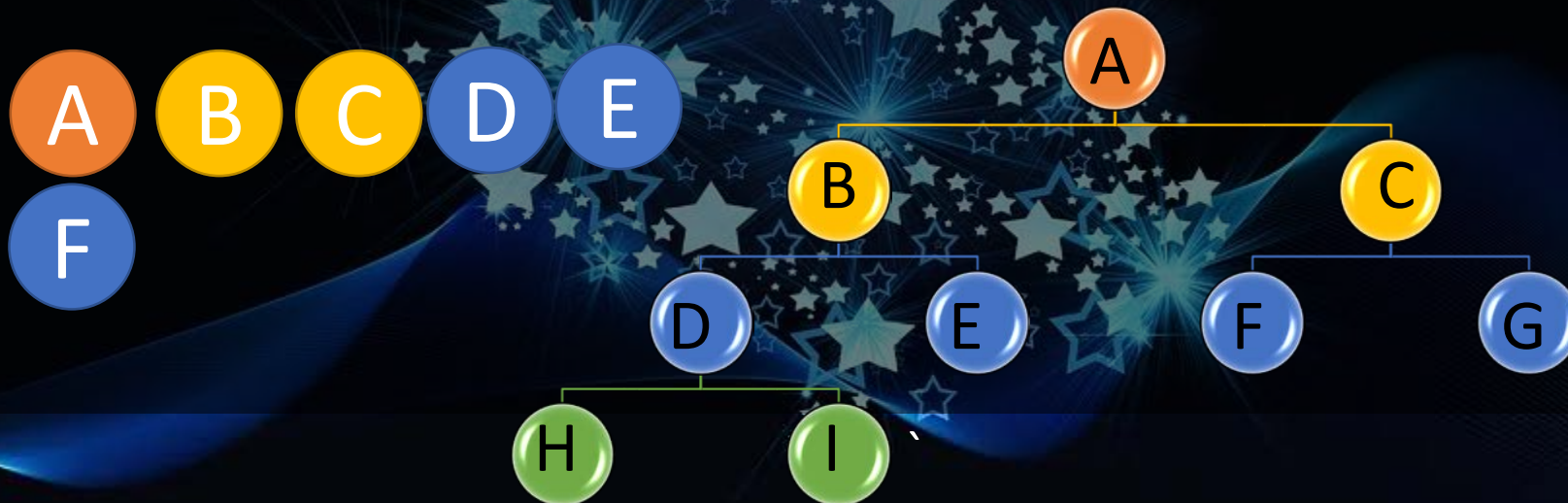
queue



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

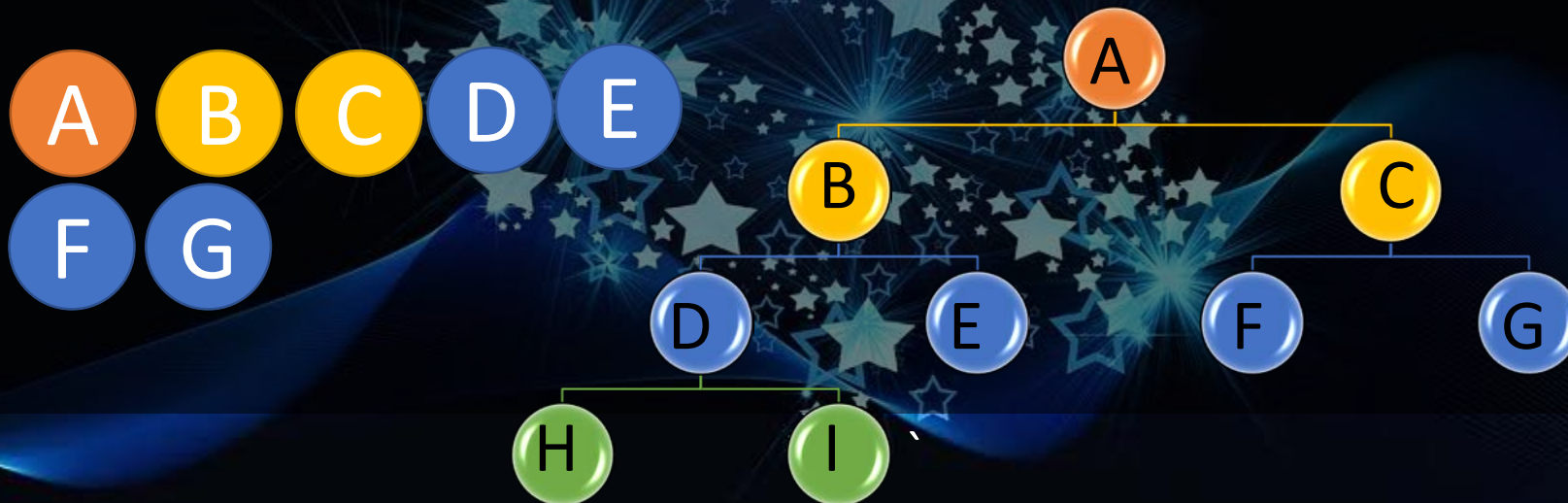
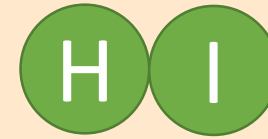
queue



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

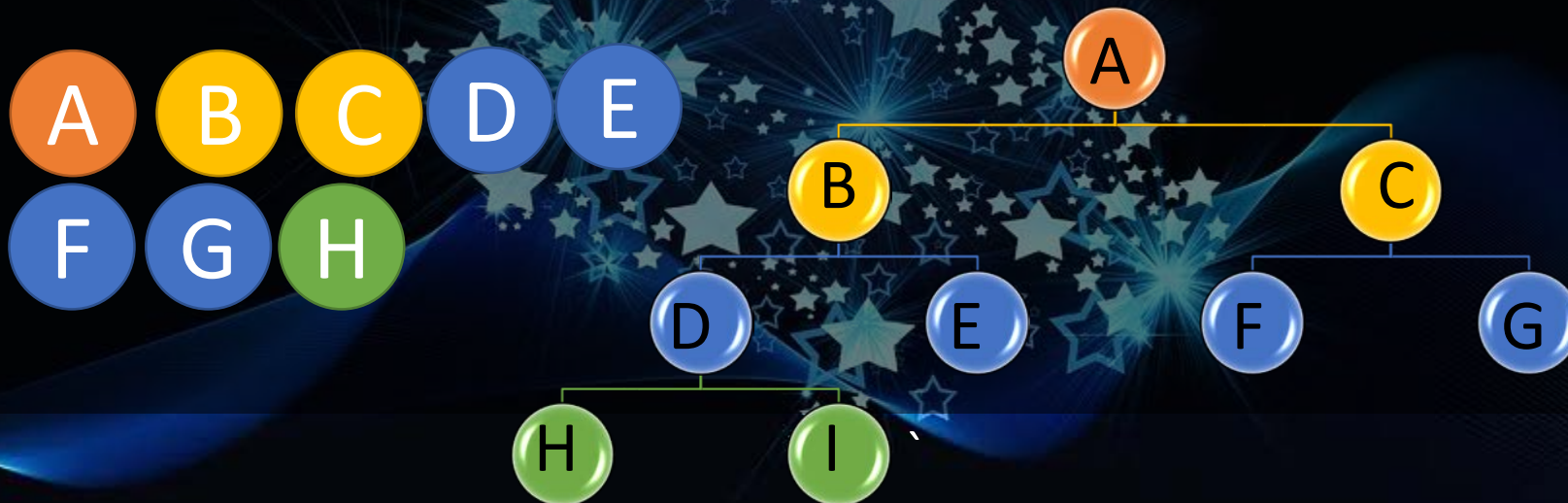
queue



BFS

1. Initialize a queue
2. Add the root to the queue
3. Repeat until the queue is empty
 1. Remove x from the queue
 2. Visit x
 3. Add its children

queue

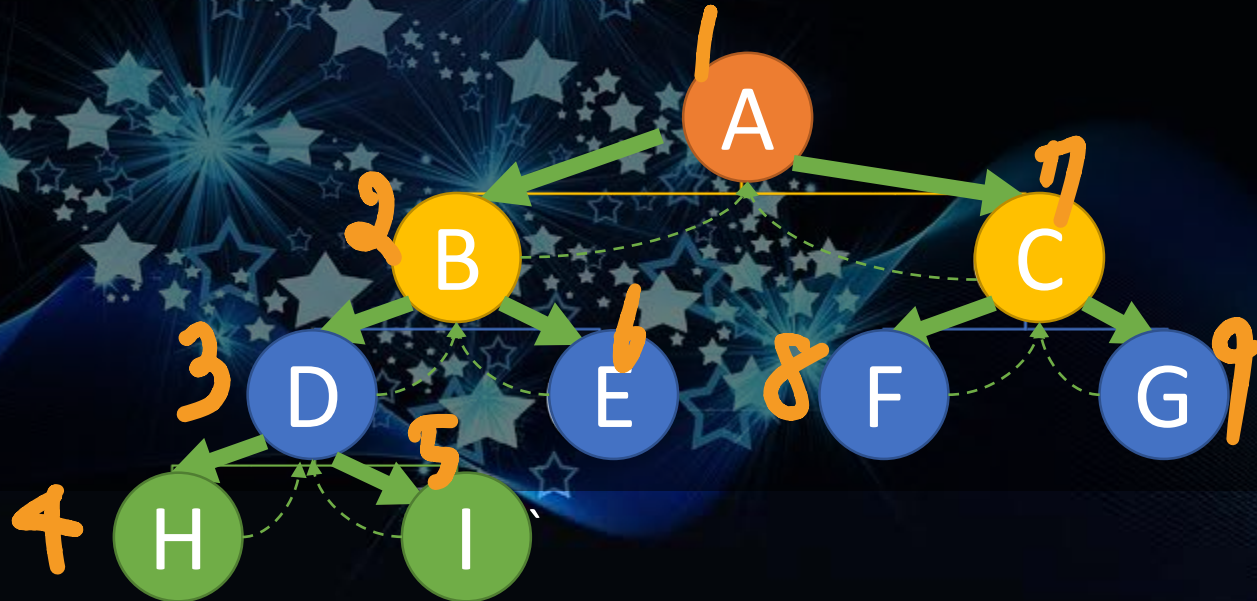


DFS(Pre-order)

1. Visit me
2. Pre-order left subtree
3. Pre-order right subtree

ABDHI ECFG

root에서 출발
순서대로



DFS(in-order)

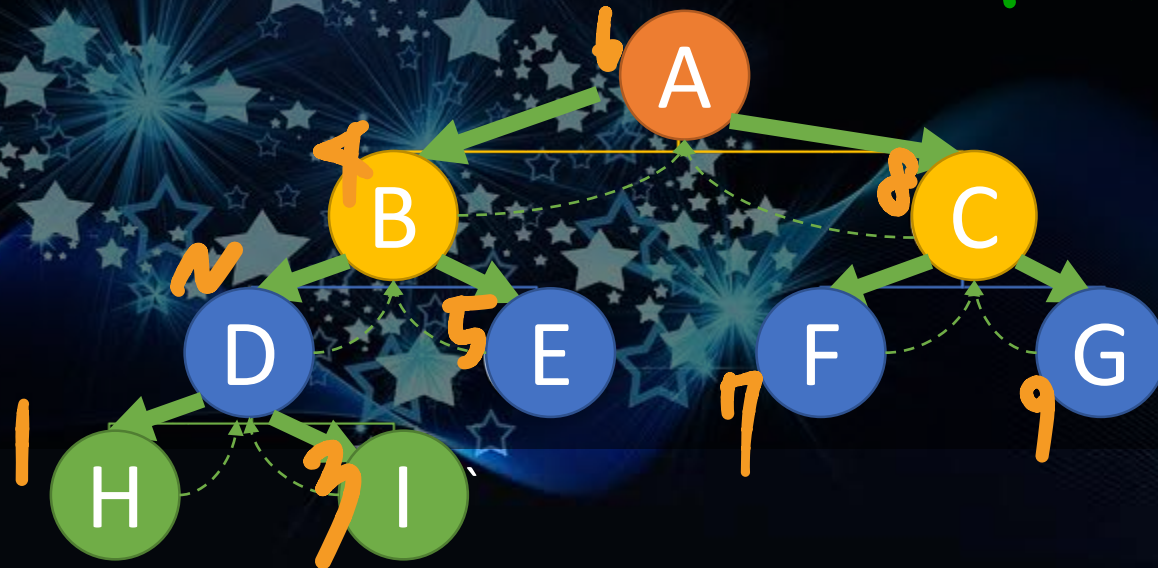
1. in-order left
2. Visit me
3. in-order right

맨 왼쪽 끝까지

부모 먼저

말바꾸기

H D I B E A F C G



DFS(Post-order)

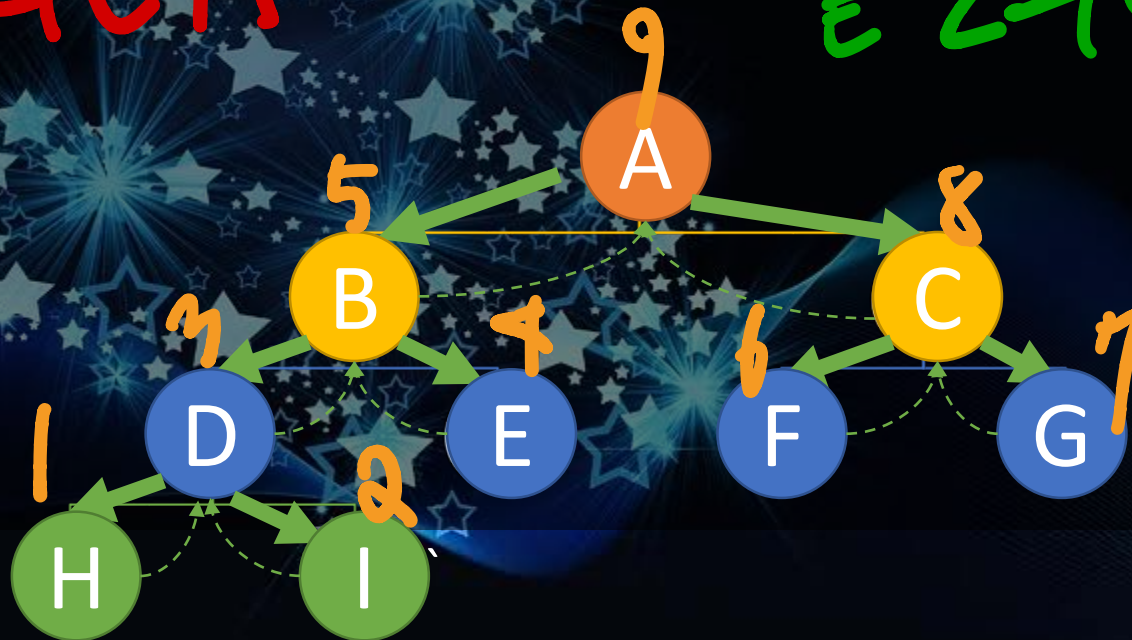
1. Post-order left
2. Post-order right
3. Visit me

H I D E B F G C A

맨 왼쪽 출발

처음 먼저

끝부분







Array Tree

Two ways of Implementing a Tree

- Assume that the tree is a binary tree.
- Static memory
 - Array
- Dynamic memory
 - Linked Tree



Interface of a Tree

- Generic interface
- Get Information
 - getRootData
 - getHeight
 - size
 - isEmpty
- Clear
- Traversal of tree
 - bfs: level order traversal
 - preorder: pre-order traversal
 - inorder: in-order traversal
 - postorder: post-order traversal

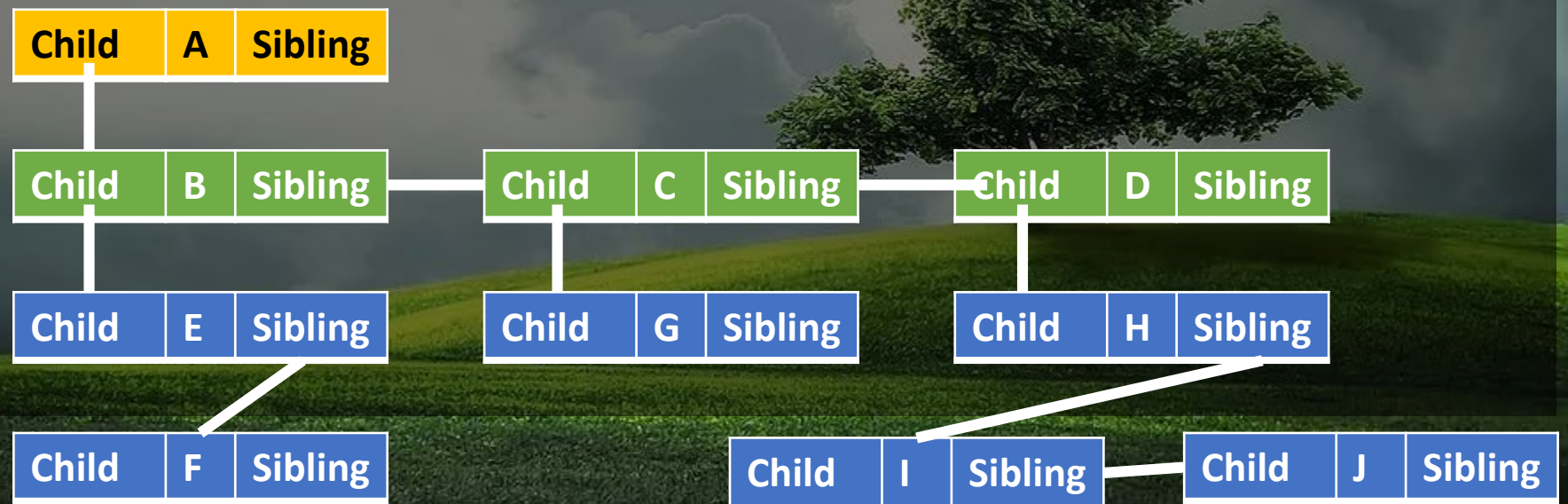
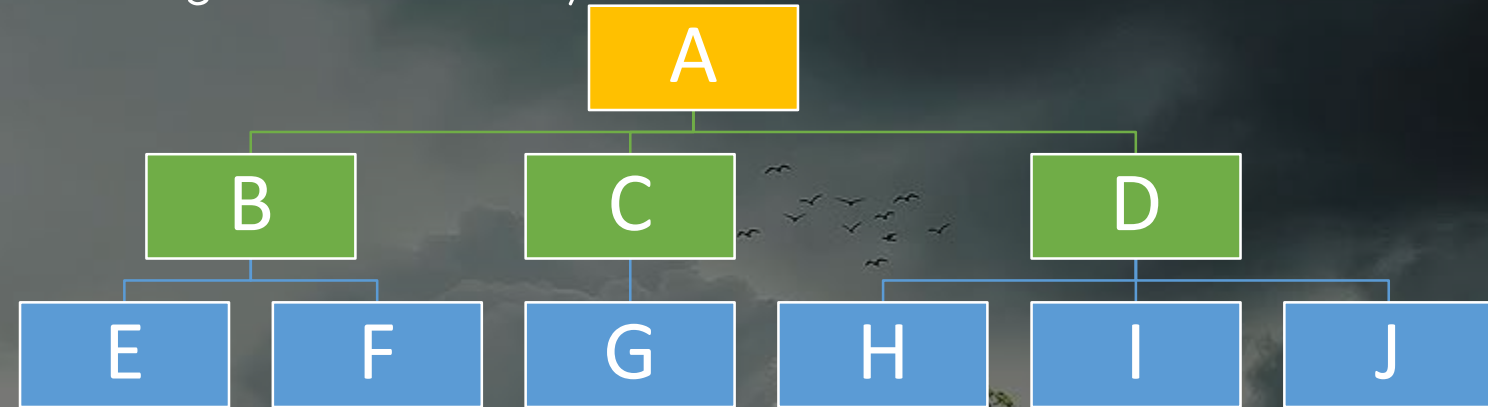
```
<<Java Interface>>  
TreeInterface<T>  
(default package)  
  
• getRootData()  
• getHeight():int  
• size():int  
• isEmpty():boolean  
• clear():void  
• bfs():String  
• preorder():String  
• inorder():String  
• postorder():String
```


General tree → binary tree

- Every general tree can be changed into a binary tree.

- Rule

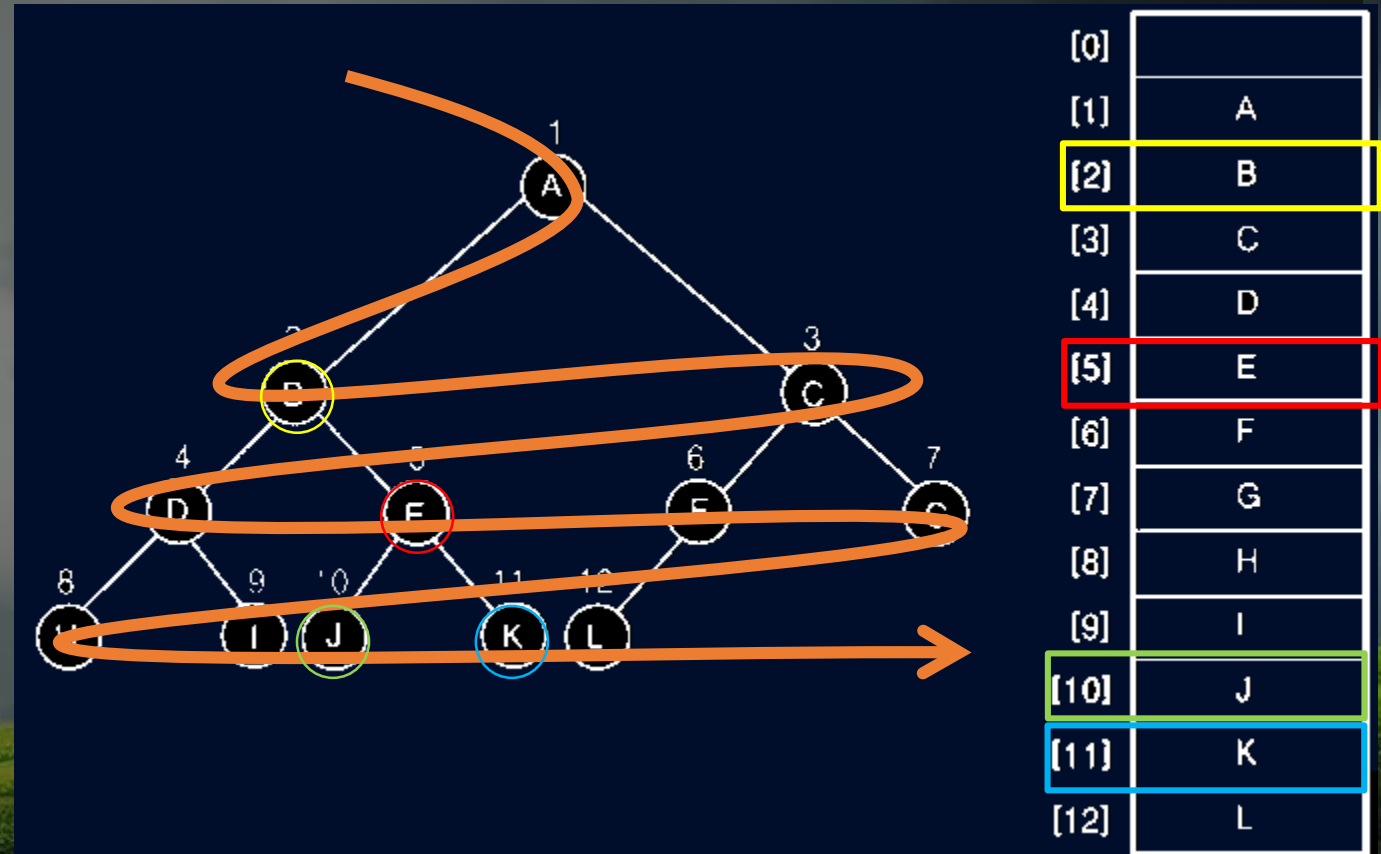
- Child → left child
- Sibling → right child



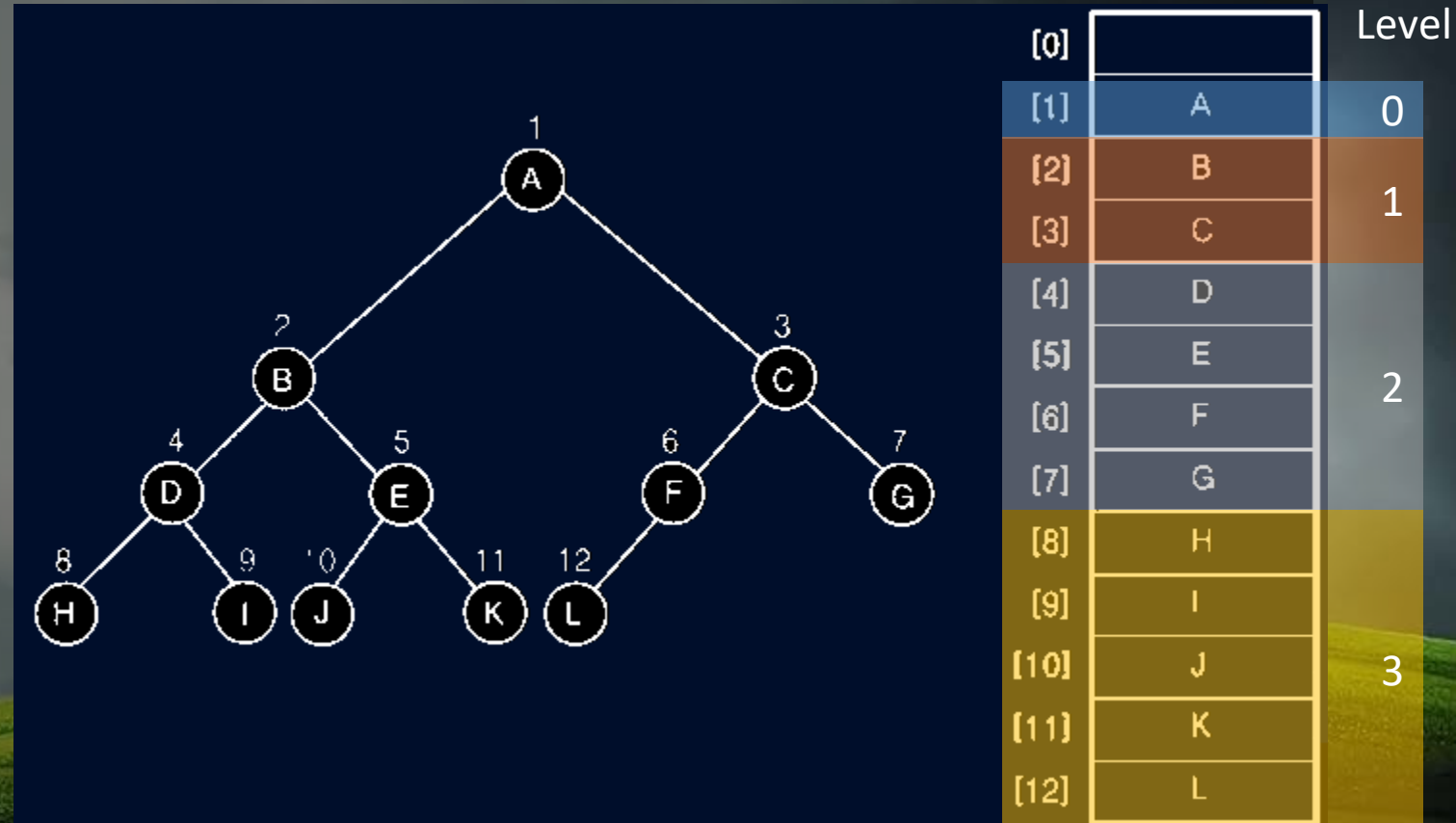
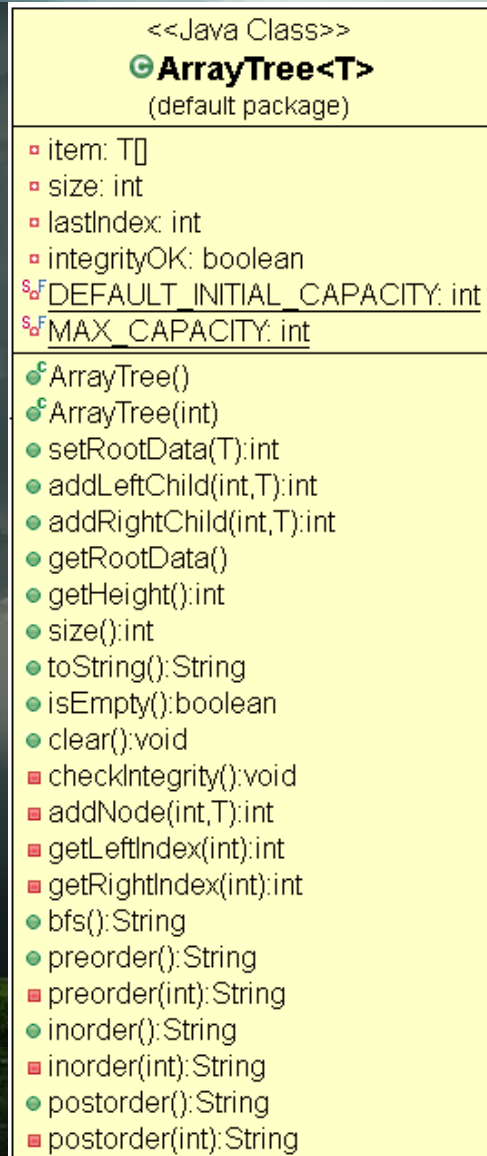
Complete Binary Tree

- Index of node i
 - Parent : $\lfloor i/2 \rfloor$
 - Left child : $\text{index} * 2$
 - Right child : $\text{index} * 2 + 1$

<<Java Interface>> TreeInterface<T> (default package)	<<Java Class>> ArrayTree<T> (default package)
<ul style="list-style-type: none">• getRootData()• getHeight():int• size():int• isEmpty():boolean• clear():void• bfs():String• preorder():String• inorder():String• postorder():String	<ul style="list-style-type: none">▪ item: T[]▪ size: int▪ lastIndex: int▪ integrityOK: boolean▪ DEFAULT_INITIAL_CAPACITY: int▪ MAX_CAPACITY: int• ArrayTree()• ArrayTree(int)• setRootData(T):int• addLeftChild(int,T):int• addRightChild(int,T):int• getRootData()• getHeight():int• size():int• toString():String• isEmpty():boolean• clear():void▪ checkIntegrity():void▪ addNode(int,T):int▪ getLeftIndex(int):int▪ getRightIndex(int):int• bfs():String• preorder():String▪ preorder(int):String• inorder():String▪ inorder(int):String• postorder():String▪ postorder(int):String

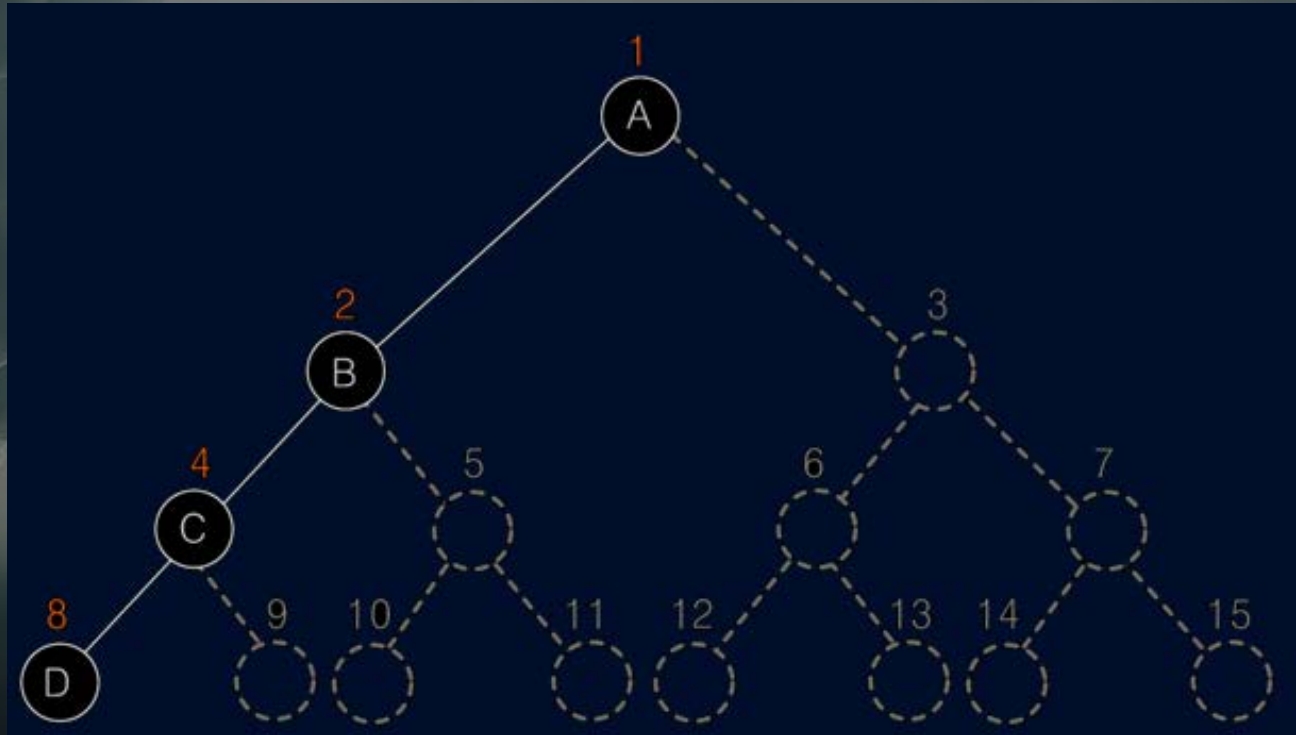


UML model of a Tree



Disadvantage of ArrayTree

- a skewed tree wastes the memory



[0]	
[1]	A
[2]	B
[3]	
[4]	C
[5]	
[6]	
[7]	
[8]	D
[9]	
[10]	
[11]	
[12]	
[13]	
[14]	
[15]	

Test Program

```

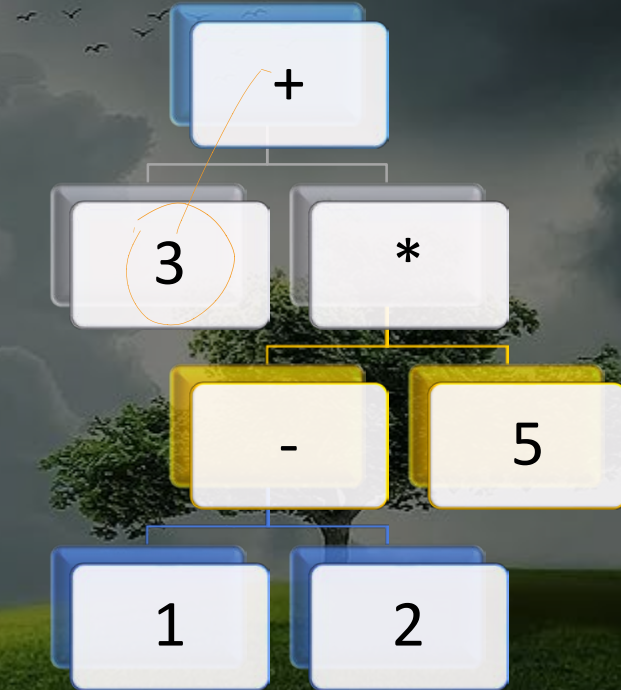
1 public class ArrayTest {
2     public static void main(String[] args) {
3         ArrayTree<Character> exTree = new ArrayTree<>(15);
4         System.out.println("Is the tree empty? " + exTree.isEmpty());
5         int index = exTree.setRootData('+');
6         exTree.addLeftChild(index, '3');
7         index = exTree.addRightChild(index, '*');
8         exTree.addRightChild(index, '5');
9         index = exTree.addLeftChild(index, '-');
10        exTree.addLeftChild(index, '1');
11        exTree.addRightChild(index, '2');
12        System.out.println("After adding, is the tree empty? " + exTree.isEmpty());
13        System.out.println("The current height is " + exTree.getHeight());
14        System.out.println(exTree.getRootData() + "[" + exTree.size() + "]: " + exTree.toString());
15        System.out.println("BFS: " + exTree.bfs());
16        System.out.println("pre-order traversal: " + exTree.preorder());
17        System.out.println("in-order traversal: " + exTree.inorder());
18        System.out.println("post-order traversal: " + exTree.postorder());
19        exTree.clear();
20        System.out.println("After clearing, is the tree empty? " + exTree.isEmpty());
21        if(!exTree.isEmpty()) System.out.println("The current height is " + exTree.getHeight());
22        System.out.println(exTree.getRootData() + "[" + exTree.size() + "]: " + exTree.toString());
23    }
24 }

```

Problems Javadoc Declaration Console

<terminated> ArrayTest [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (2020. 10. 20. 오후 1:00:51 - 오후 1:00:52)

Is the tree empty? true
 After adding, is the tree empty? false
 The current height is 3
 +[7]: [null, +, 3, *, null, null, -, 5, null, null, null, null, 1, 2, null]
 BFS: + 3 * - 5 1 2
 pre-order traversal: + 3 * - 1 2 5 → root 부터 다 나열함
 in-order traversal: 3 + 1 - 2 * 5 → 왼쪽 → 오른쪽
 post-order traversal: 3 1 2 - 5 * + → 밑부터
 After clearing, is the tree empty? true
 null[0]: [null, null, null, null, null, null, null, null, null, null, null, null, null, null, null]



[0]	
[1]	+
[2]	3
[3]	*
[4]	
[5]	
[6]	-
[7]	5
[8]	
[9]	
[10]	
[11]	
[12]	1
[13]	2
[14]	

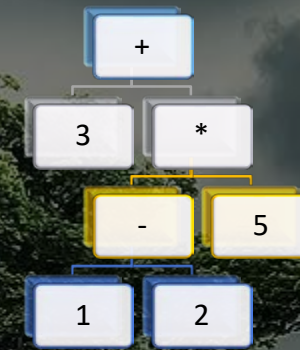
Add Root

Client program

```
public class ArrayTest {  
    public static void main(String[] args) {  
        ArrayTree<Character> exTree = new ArrayTree<>(15);  
        System.out.println("Is the tree empty? " + exTree.isEmpty());  
        int index = exTree.setRootData('+');  
        exTree.addLeftChild(index, '3');  
        index = exTree.addRightChild(index, '*');  
        exTree.addRightChild(index, '5');  
        index = exTree.addLeftChild(index, '-');  
        exTree.addLeftChild(index, '1');  
        exTree.addRightChild(index, '2');  
    }  
}
```

In ArrayTree<T>

```
public int setRootData(T data) {  
    return addNode(1, data);  
}  
  
public int addLeftChild(int parentIndex, T data) {  
    return addNode(parentIndex*2, data);  
}  
  
public int addRightChild(int parentIndex, T data) {  
    return addNode(parentIndex*2+1, data);  
}
```



[0]	
[1]	+
[2]	3
[3]	*
[4]	
[5]	
[6]	-
[7]	5
[8]	
[9]	
[10]	
[11]	
[12]	1
[13]	2
[14]	

Add Children

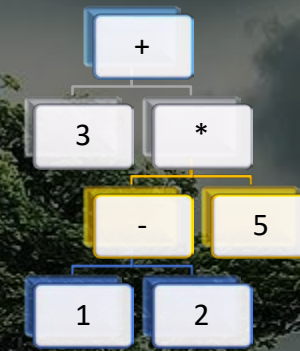
Be careful not to create a child node without a parent node

Client program

```
exTree.addLeftChild(index, '3');  
index = exTree.addRightChild(index, '*');  
exTree.addRightChild(index, '5');  
index = exTree.addLeftChild(index, '-');  
exTree.addLeftChild(index, '1');  
exTree.addRightChild(index, '2');
```

In ArrayTree<T>

```
private int addNode(int index, T data) {  
    checkIntegrity();  
    if(index < item.length) {  
        if(item[index]==null)size++;  
        item[index] = data;  
        if(index>lastIndex) lastIndex = index;  
    } else {  
        index=-1;  
        System.err.println(data+" cannot be added to "+(index*2)+" because of the array size");  
    }  
    return index;  
}
```



[0]	
[1]	+
[2]	3
[3]	*
[4]	
[5]	
[6]	-
[7]	5
[8]	
[9]	
[10]	
[11]	
[12]	1
[13]	2
[14]	

Get information

height, the number of nodes, root data

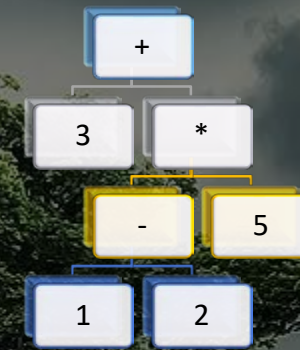
Client program

```
System.out.println("After adding, is the tree empty? "+ exTree.isEmpty());  
System.out.println("The current height is "+exTree.getHeight());  
System.out.println(exTree.getRootData()+"["+exTree.size()+"]": "+exTree.toString());
```

In ArrayTree<T>

```
public T getRootData() {  
    checkIntegrity();  
    T result = item[1];  
    return result;  
}  
public int getHeight() {  
    return (int)(Math.Log(lastIndex)/Math.Log(2.));  
}  
public int size() {  
    return size;  
}  
public String toString() {  
    checkIntegrity();  
    return Arrays.toString(item);  
}  
public boolean isEmpty() {  
    return size == 0;  
}
```

$\log_2(\text{lastIndex})$



[0]	
[1]	+
[2]	3
[3]	*
[4]	
[5]	
[6]	-
[7]	5
[8]	
[9]	
[10]	
[11]	
[12]	1
[13]	2
[14]	

traversal

bfs: using array

Pre order traversal

```
public String preorder() {  
    return preorder(1);  
}  
private String preorder(int i) {  
    String s = "";  
    int left = getLeftIndex(i);  
    int right = getRightIndex(i);  
  
    s += item[i] + " ";  
    if(left>0) s += preorder(left);  
    if(right>0) s += preorder(right);  
    return s;  
}
```

in order traversal

```
public String inorder() {  
    return inorder(1);  
}  
private String inorder(int i) {  
    String s = "";  
    int left = getLeftIndex(i);  
    int right = getRightIndex(i);  
    if(left>0) s += inorder(left);  
    s += item[i] + " ";  
    if(right>0) s += inorder(right);  
    return s;  
}
```

post order traversal

```
public String postorder() {  
    return postorder(1);  
}  
private String postorder(int i) {  
    String s = "";  
    int left = getLeftIndex(i);  
    int right = getRightIndex(i);  
    if(left>0) s += postorder(left);  
    if(right>0) s += postorder(right);  
    s += item[i] + " ";  
    return s;  
}
```





Linked **Tree**

Design Issue

Tree is defined

as a composition of nodes

LinkedTree<T>

- root : TreeNode<T>



LinkedNode<T>

- data : T
- left : TreeNode<T>
- right: TreeNode<T>

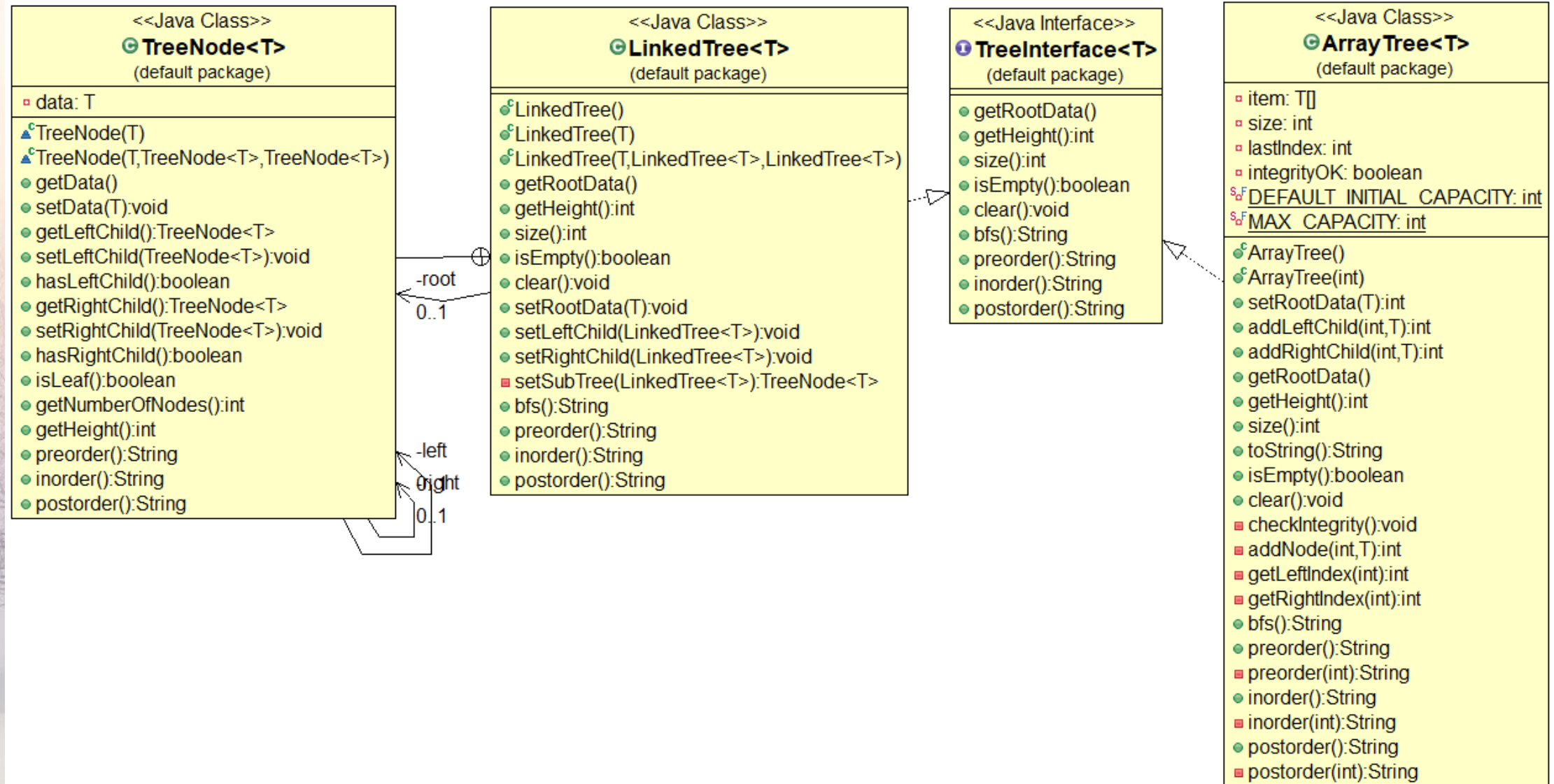
the definition of a rooted tree

- Empty tree is a tree.
- If S is a set of trees
any trees of S do not share a node.
 $T = (r, S)$ is a tree
 r is a root
a tree in S is a sub-tree of T

LinkedTree<T>

- data: T
- left: LinkedTree<T>
- right: LinkedTree<T>

UML class diagram

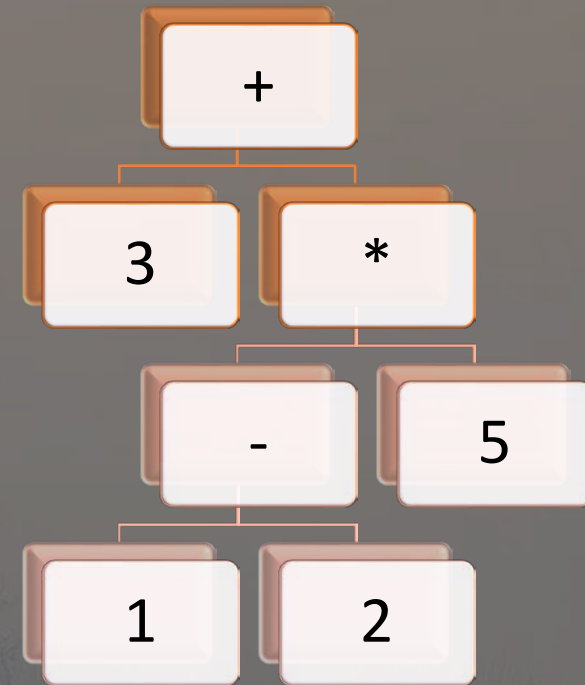


Test Program

```
1 public class LinkedTest {
2     public static void main(String[] args) {
3         LinkedTree<Character> exTree = new LinkedTree<>();
4         System.out.println("Is the tree empty? "+ exTree.isEmpty());
5         LinkedTree<Character> minus
6             = new LinkedTree<>('-', new LinkedTree<Character>('1'), new LinkedTree<Character>('2'));
7         LinkedTree<Character> five = new LinkedTree<Character>('5');
8         LinkedTree<Character> times = new LinkedTree<Character>('*', minus, five);
9         exTree.setRootData('+');
10        exTree.setLeftChild(new LinkedTree<Character>('3'));
11        exTree.setRightChild(times);
12        System.out.println("After adding, is the tree empty? "+ exTree.isEmpty());
13        System.out.println("The current height is "+exTree.getHeight());
14        System.out.println(exTree.getRootData()+"["+exTree.size()+"]": "+exTree.toString());
15        System.out.println("BFS: "+exTree.bfs());
16        System.out.println("pre-order traversal: "+exTree.preorder());
17        System.out.println("in-order traversal: "+exTree.inorder());
18        System.out.println("post-order traversal: "+exTree.postorder());
19        exTree.clear();
20        System.out.println("After clearing, is the tree empty? "+ exTree.isEmpty());
21        if(!exTree.isEmpty()) System.out.println("The current height is "+exTree.getHeight());
22        System.out.println(exTree.getRootData()+"["+exTree.size()+"]": "+exTree.toString());
23    }
24 }
```

Problems Javadoc Declaration Console
<terminated> LinkedTest [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (2020. 10. 20. 오후 9:03:14 - 오후 9:03:14)

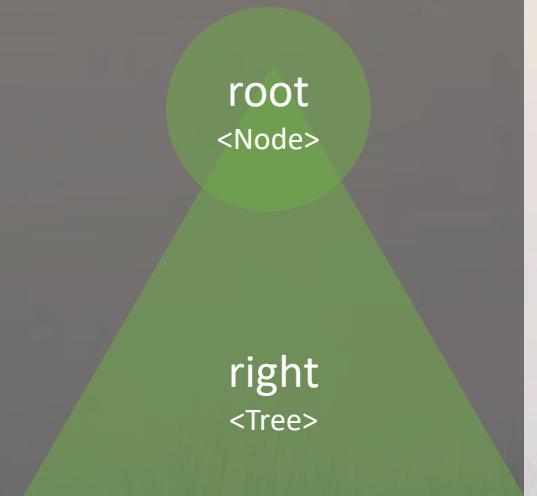
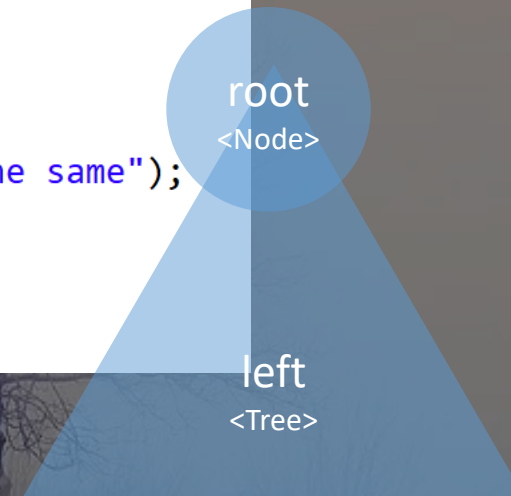
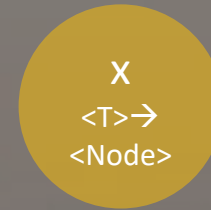
```
Is the tree empty? true
After adding, is the tree empty? false
The current height is 3
+[7]: LinkedTree@1f32e575
BFS: + 3 * - 5 1 2
pre-order traversal: + 3 * - 1 2 5
in-order traversal: 3 + 1 - 2 * 5
post-order traversal: 3 1 2 - 5 * +
After clearing, is the tree empty? true
null[0]: LinkedTree@1f32e575
```



Add Node

Three kind of constructors

```
public LinkedTree() {  
    root = null;  
}  
public LinkedTree(T x) {  
    setRootData(x);  
}  
public LinkedTree(T x, LinkedTree<T> left, LinkedTree<T> right) {  
    setRootData(x);  
    root.left = setSubTree(left);  
    TreeNode<T> temp = setSubTree(right);  
    if(temp != null && temp == root.left) {  
        System.err.println("Both children cannot be the same");  
        temp = null;  
    }  
    root.right = temp;  
}
```



Get Information

TreeNode calculates the information recursively.

In class LinkedTree

```
public T getRootData() {
    if(root==null) return null;
    return root.getData();
}
public int getHeight() {
    if(root == null) {
        System.err.println("You try to know the height of an empty tree");
        return -1;
    }
    return root.getHeight();
}
public int size() {
    if(root == null) return 0;

    return root.getNumberOfNodes();
}

public boolean isEmpty() {
    return root==null;
}
```

In inner class TreeNode

```
public int getNumberOfNodes() {
    if(isLeaf()) return 1;

    int num = 1;
    if(hasLeftChild()) num += left.getNumberOfNodes();
    if(hasRightChild()) num += right.getNumberOfNodes();
    return num;
}
public int getHeight() {
    if(isLeaf()) return 0;
    int leftHeight=0, rightHeight=0;
    if(hasLeftChild()) leftHeight = left.getHeight();
    if(hasRightChild()) rightHeight = right.getHeight();

    return Math.max(leftHeight, rightHeight)+1;
}
```


BFS Traversal

It uses queue for TreeNode.

```
public String bfs() {  
    Queue    String s = "";  
    if(!isEmpty()) {  
        q.offer(root);  
        for(TreeNode<T> node = q.poll() ; node!=null ; node = q.poll()) {  
            s+=node.data + " ";  
            if(node.hasLeftChild()) q.offer(node.left);  
            if(node.hasRightChild()) q.offer(node.right);  
        }  
    }  
    return s;  
}
```

DFS Traversal

In class LinkedTree

```
public String preorder() {  
    return root.preorder();  
}  
public String inorder() {  
    return root.inorder();  
}  
public String postorder() {  
    return root.postorder();  
}
```

In inner class TreeNode

```
public String preorder() {  
    String s = "";  
    s+=data + " ";  
    if(hasLeftChild())s+=left.preorder();  
    if(hasRightChild())s+=right.preorder();  
    return s;  
}  
public String inorder() {  
    String s = "";  
    if(hasLeftChild())s+=left.inorder();  
    s+=data + " ";  
    if(hasRightChild())s+=right.inorder();  
    return s;  
}  
public String postorder() {  
    String s = "";  
    if(hasLeftChild())s+=left.postorder();  
    if(hasRightChild())s+=right.postorder();  
    s+=data + " ";  
    return s;  
}
```


Thank you

