

Data Structures in Python

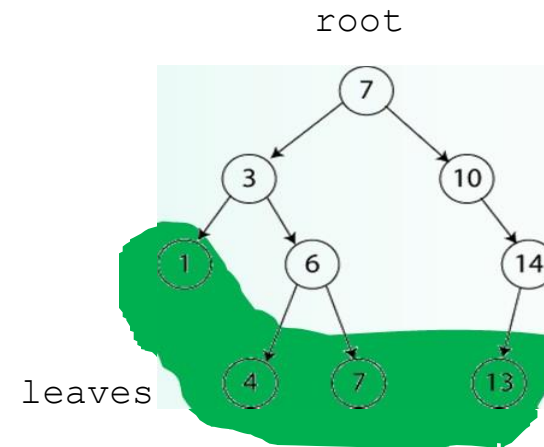
- **Tree Introduction**
- Tree Traversals
- Tree Algorithms
- Binary Search Tree
- Heap
- Heap Sort

Agenda & Readings

- Agenda
 - Tree Terminology
 - Binary Tree Properties
 - Binary Tree and Node Representation
- Reference:
 - Problem Solving with Algorithms and Data Structures
 - Chapter 6 - Tree

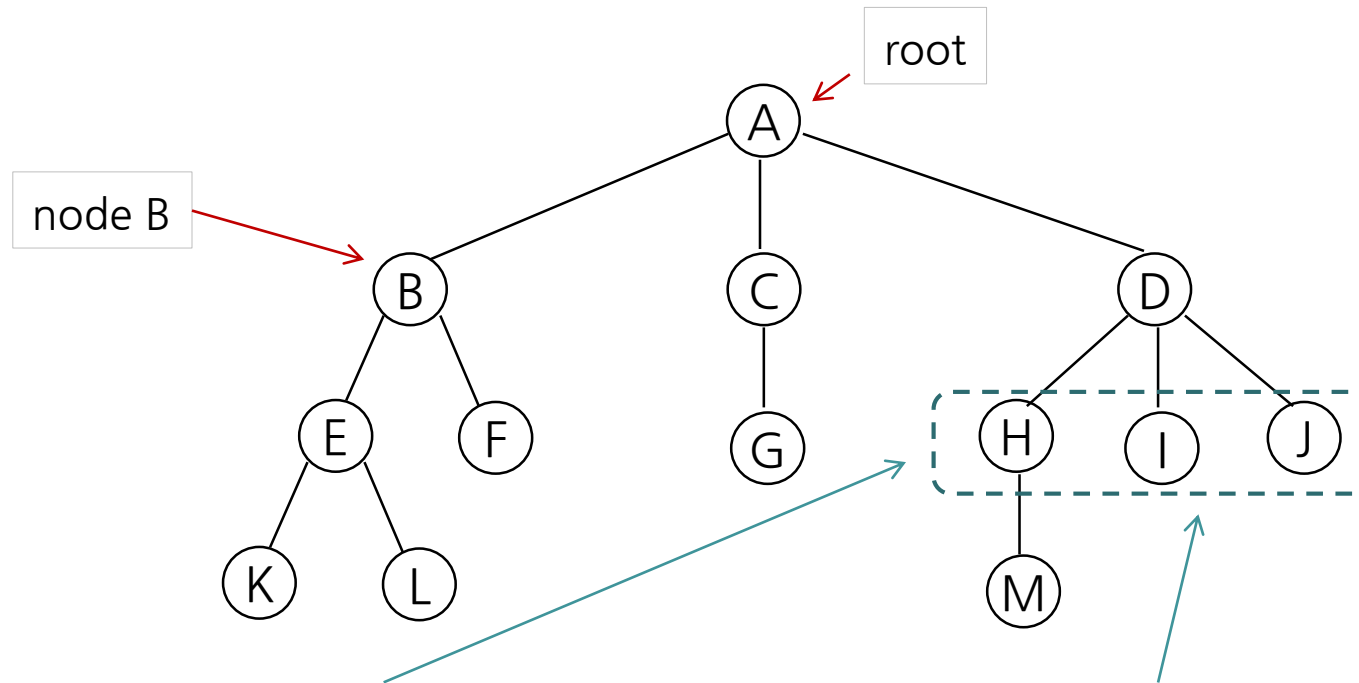
What is a Tree?

- A non-linear data structure
- An abstraction for a hierarchical structure
- It is defined as a set of points called nodes and a set of lines called edges where an edge connects two distinct nodes.



Introduction - Terminology

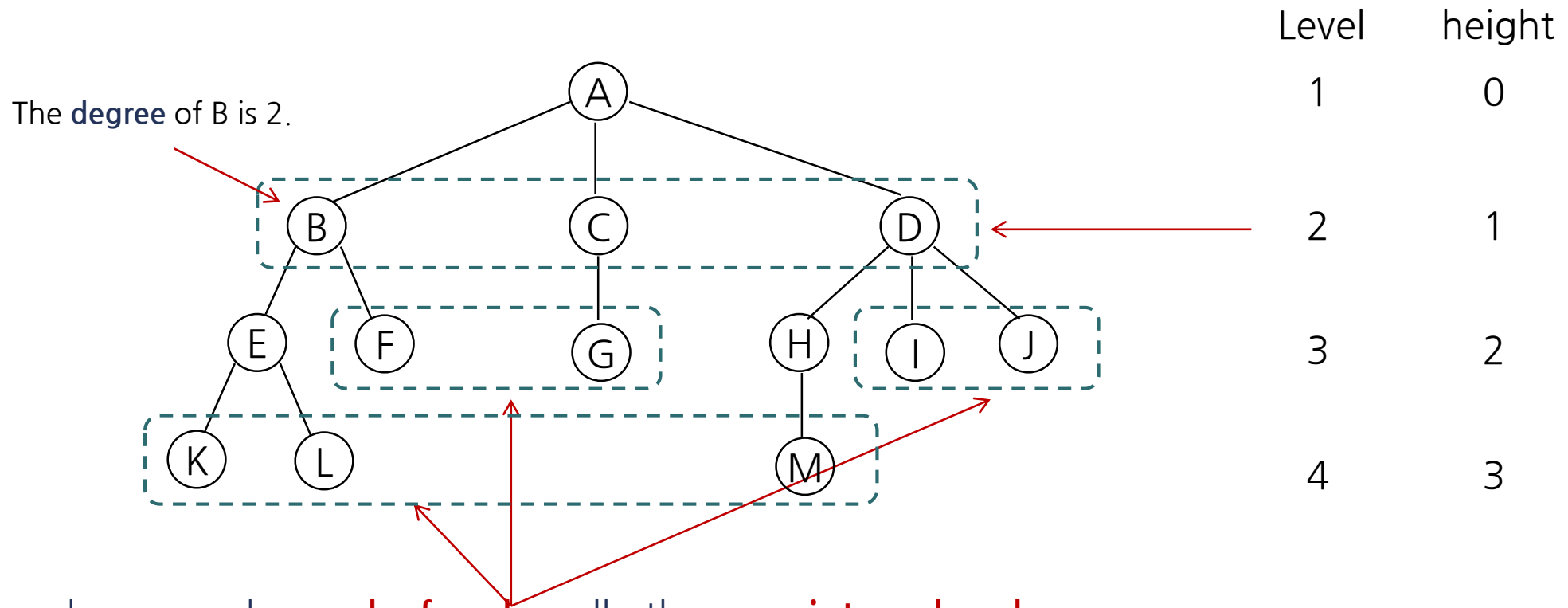
- **A tree data structure:** it is like a linked list that has a **first** node, this node is called as the **root** of the tree.
- **Example.** A **tree** with a root storing the value 'A'



- The **children** of D are H, I, and J; H, I, and J are **siblings**.
- The **parent** of D is A.

Introduction - Terminology

- **Definition.** child, parent, sibling, degree, leaf nodes, level, and internal node



- Zero degree nodes are **leaf nodes**, all others are **internal nodes**.
 - An **internal node** is any node that has at least one non-empty child.
- The **degree** of a node is the number of children.
- The **degree of a tree** is the **maximum of the degree of the nodes** in the tree.

Introduction - Representation of trees

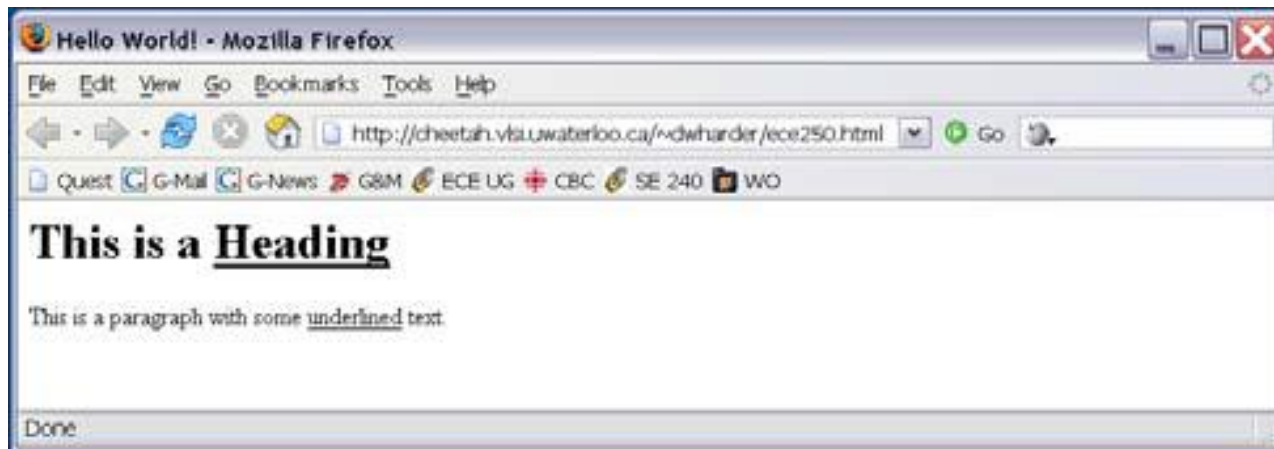
- **Exercise.** The tree representing the HTML document below:

```
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>This is a <u>Heading</u></h1>
    <p>This is a paragraph with some <u>underlined</u> text.</p>
  </body>
</html>
```

Introduction - Representation of trees

- **Exercise.** The tree representing the HTML document below:

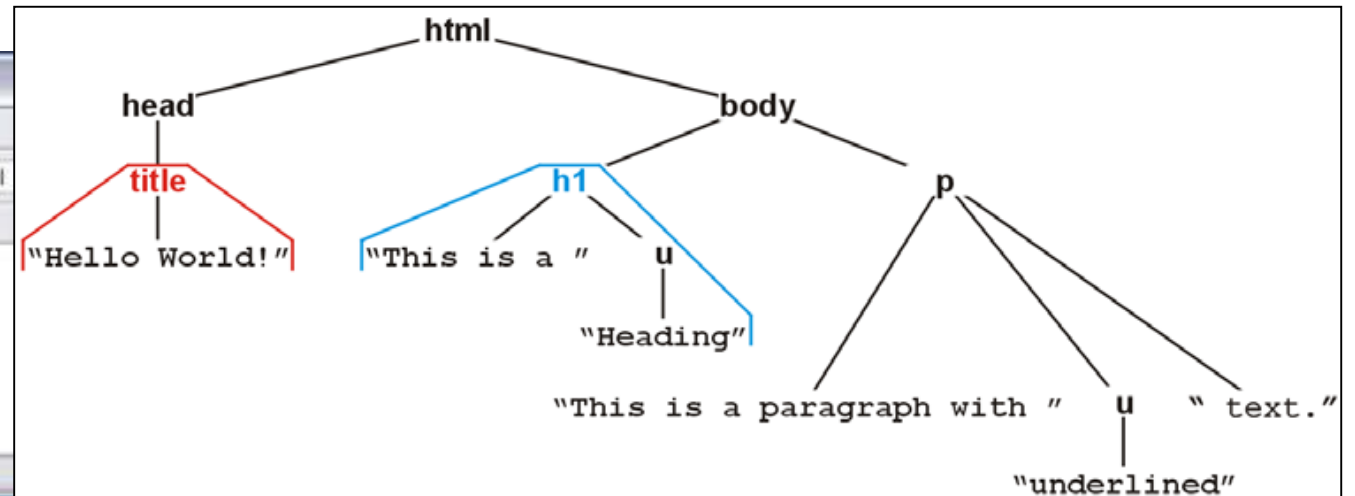
```
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>This is a <u>Heading</u></h1>
    <p>This is a paragraph with some <u>underlined</u> text.</p>
  </body>
</html>
```



Introduction - Representation of trees

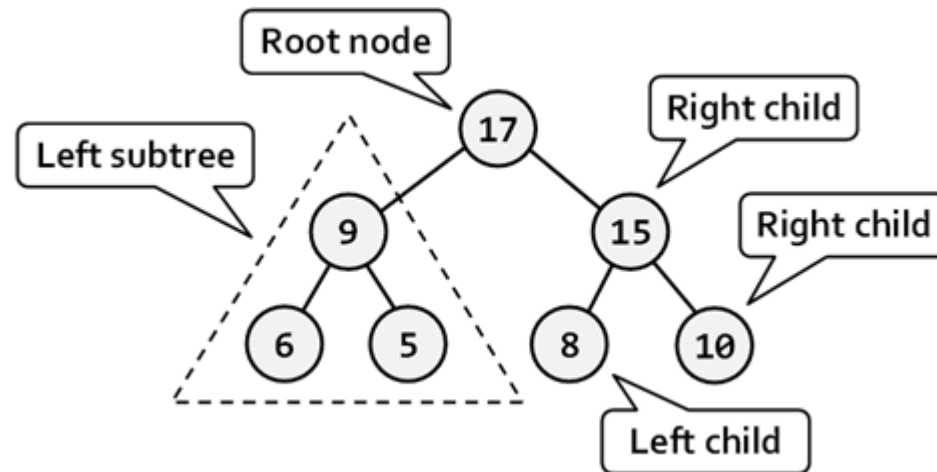
- **Exercise.** The tree representing the HTML document below:

```
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>This is a <u>Heading</u></h1>
    <p>This is a paragraph with some <u>underlined</u> text.</p>
  </body>
</html>
```



Binary trees

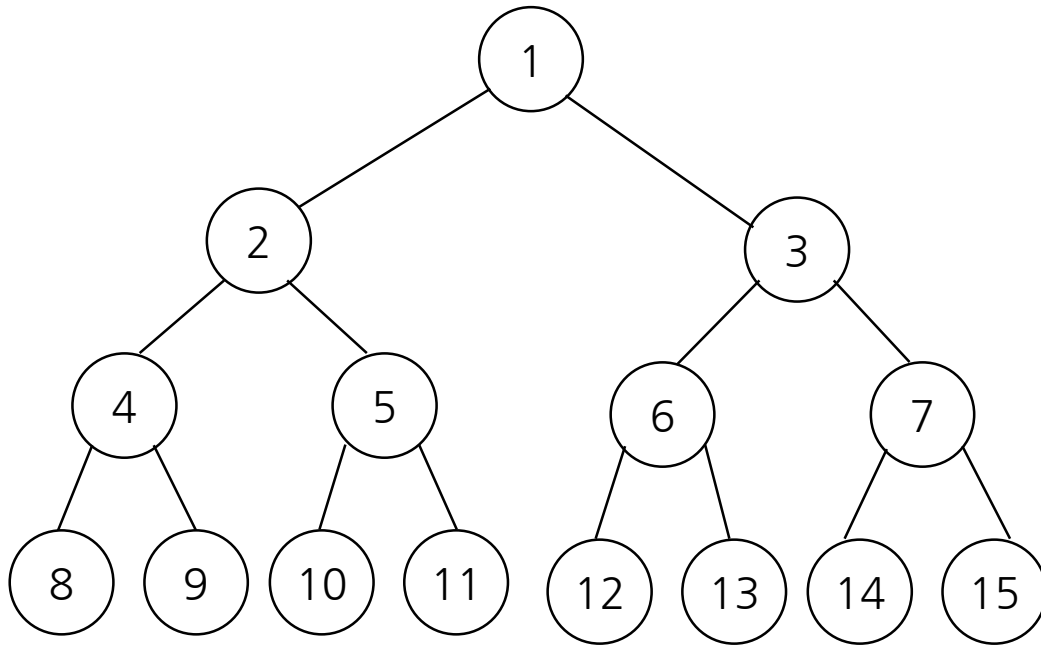
- **Definition:** A tree such that each node has *exactly* two children.
 - Notice, exactly two children - not up to two children! Because *exactly* two children means a left child **and/or** right child, no middle child.
 - Each child is either empty or another binary tree.
 - Given this constraint, we can label the two children as left and right nodes or subtrees.



Binary trees - Properties

- **Observation:**

- Q: Maximum number of nodes in binary trees in each level and all levels?
- Q: What is the max level k if there are n nodes? $k(n) = ?$

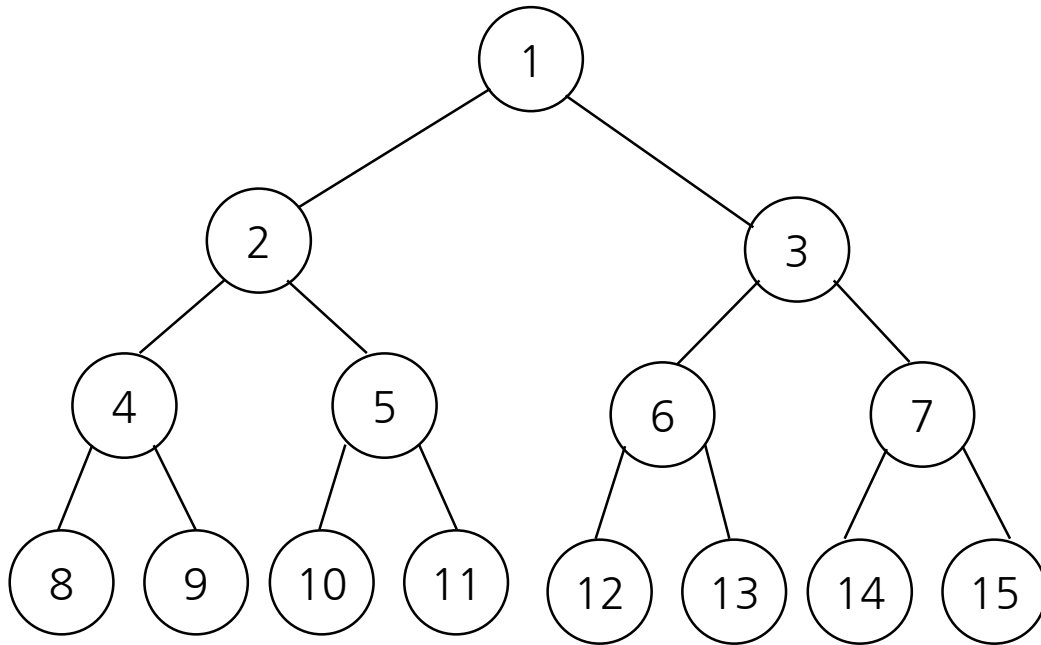


A full binary tree

Binary trees - Properties

■ Observation:

- Q: Maximum number of nodes in binary trees in each level and all levels?
- Q: What is the max level k if there are n nodes? $k(n) = ?$



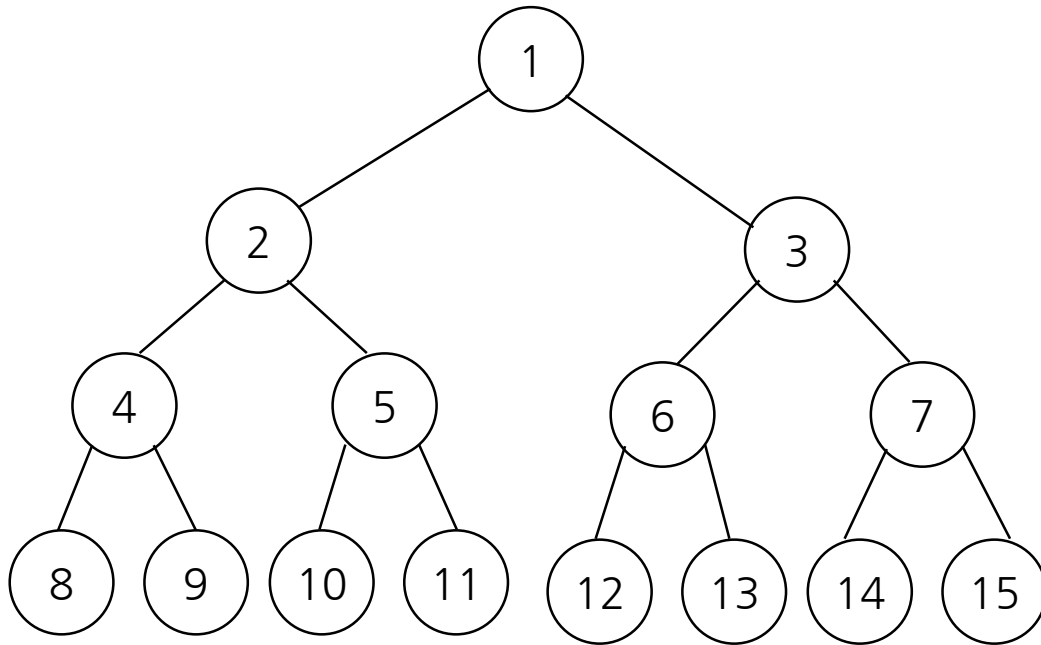
A full binary tree

Level	Node Numbers at Each Level	Total Numbers of Nodes
1	$1 = 2^0$	
2	$2 = 2^1$	
3	$4 = 2^2$	
4	$8 = 2^3$	
.	.	
11	$1024 = 2^{10}$	
.	.	
k		

Binary trees - Properties

■ Observation:

- Q: Maximum number of nodes in binary trees in each level and all levels?
- Q: What is the max level k if there are n nodes? $k(n) = ?$



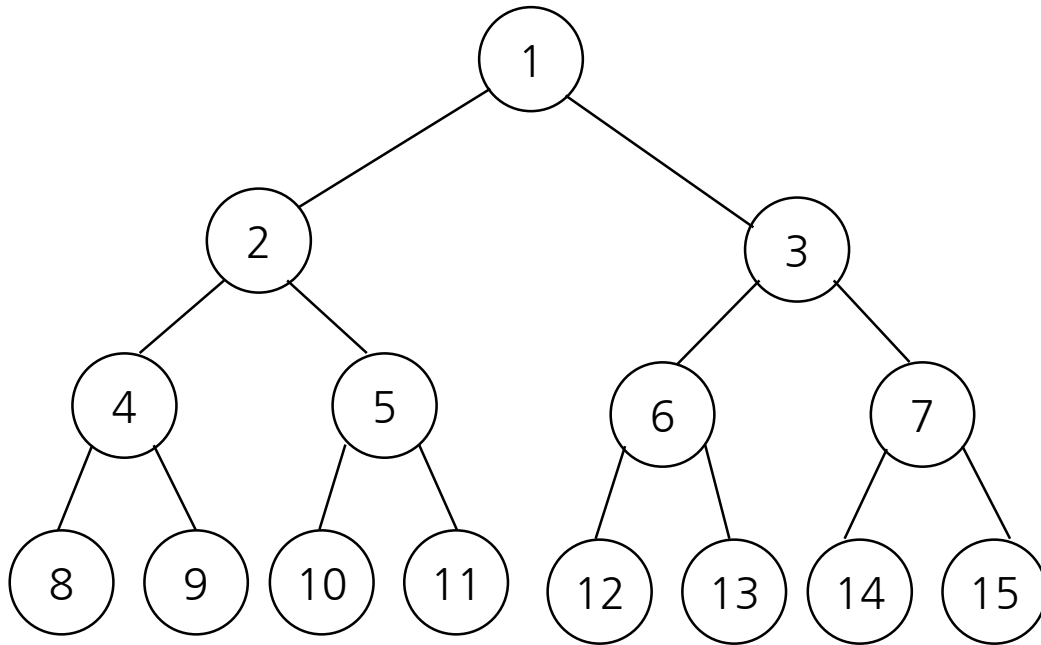
A full binary tree

Level	Node Numbers at Each Level	Total Numbers of Nodes
1	$1 = 2^0$	$1 = 2^1 - 1$
2	$2 = 2^1$	$3 = 2^2 - 1$
3	$4 = 2^2$	$7 = 2^3 - 1$
4	$8 = 2^3$	$15 = 2^4 - 1$
.	.	.
11	$1024 = 2^{10}$	$2047 = 2^{11} - 1$
.	.	.
k		

Binary trees - Properties

■ Observation:

- Q: Maximum number of nodes in binary trees in each level and all levels?
- Q: What is the max level k if there are n nodes? $k(n) = ?$

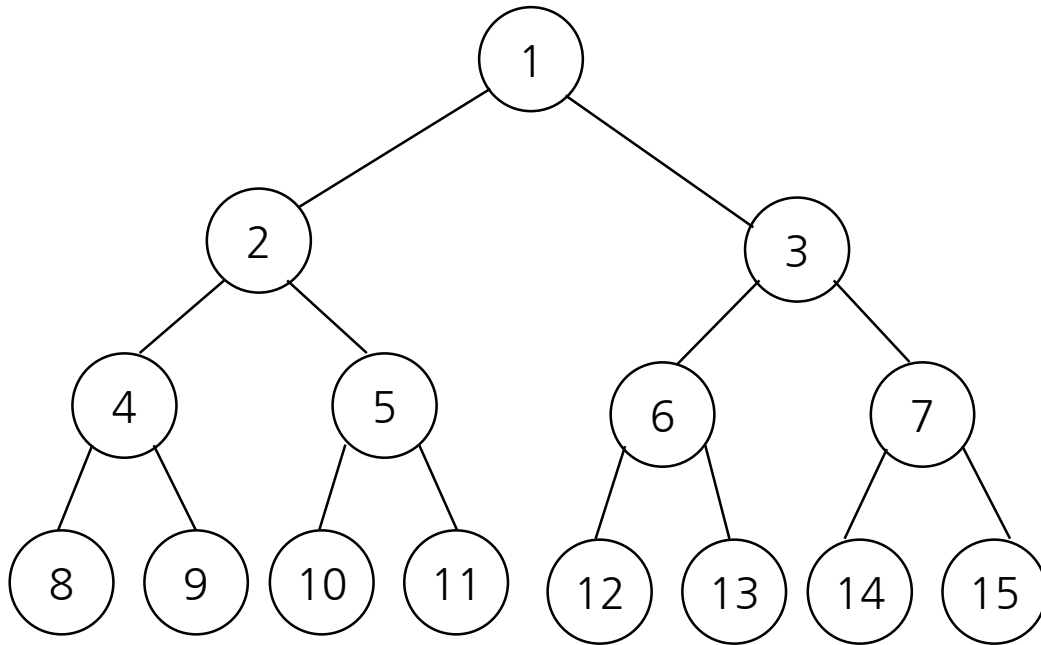


A full binary tree

Level	Node Numbers at Each Level	Total Numbers of Nodes
1	$1 = 2^0$	$1 = 2^1 - 1$
2	$2 = 2^1$	$3 = 2^2 - 1$
3	$4 = 2^2$	$7 = 2^3 - 1$
4	$8 = 2^3$	$15 = 2^4 - 1$
.	.	.
11	$1024 = 2^{10}$	$2047 = 2^{11} - 1$
.	.	.
k	2^{k-1}	$2^k - 1$
h	2^h	$2^{h+1} - 1$

Binary trees - Properties

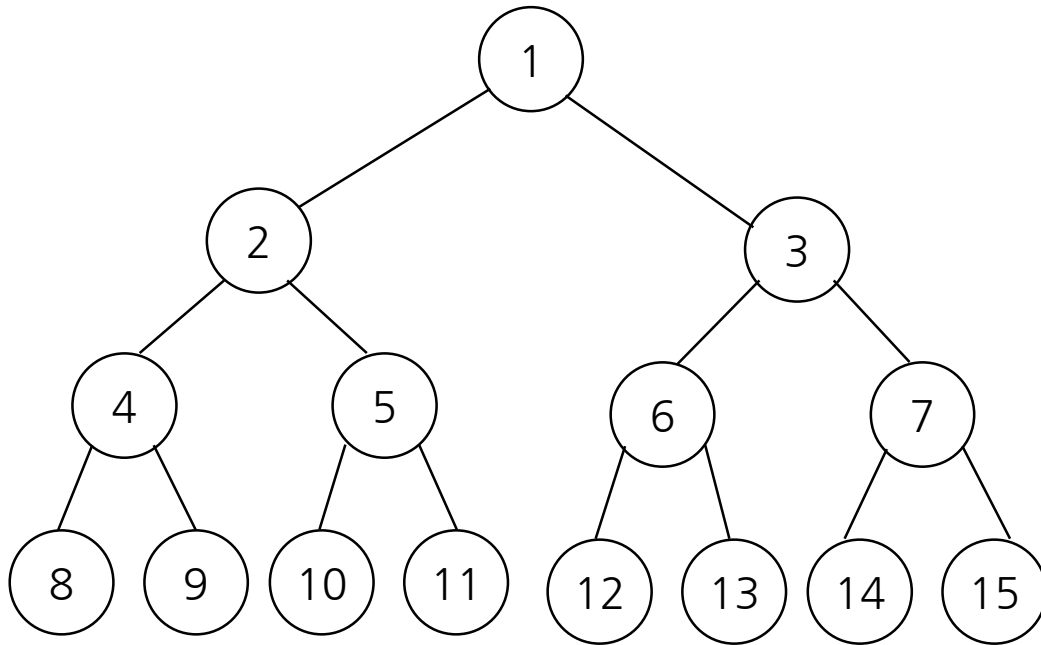
- **Definition:** *A full binary tree* of level k is a binary tree having $2^k - 1$ nodes, $k \geq 0$.
- **Definition:** A binary tree with n nodes and level k is **complete** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of level k .



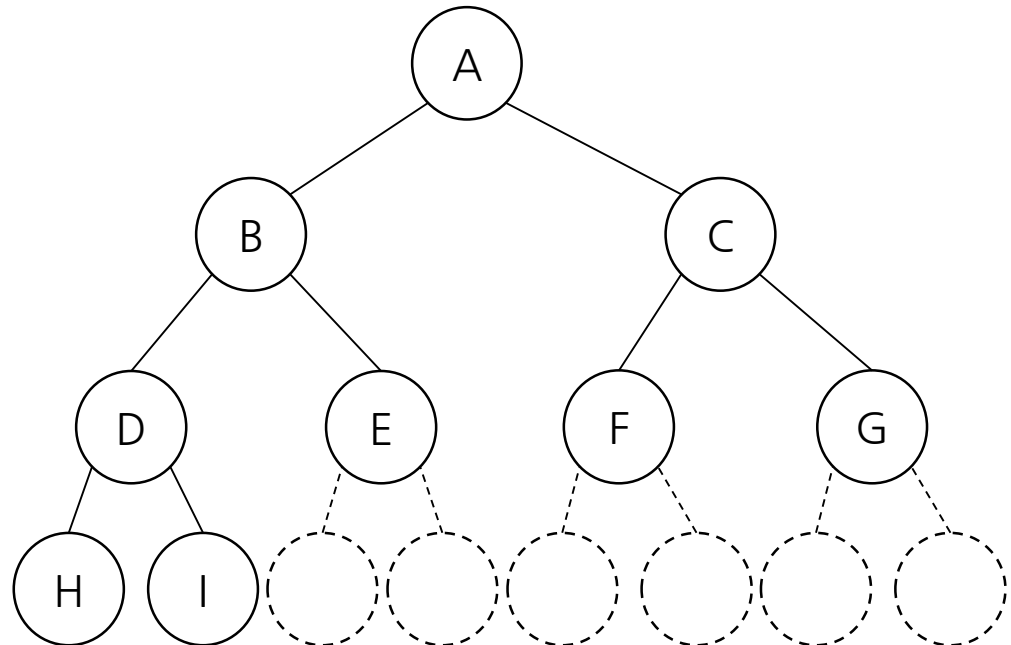
A full binary tree

Binary trees - Properties

- **Definition:** A *full binary tree* of level k is a binary tree having $2^k - 1$ nodes, $k \geq 0$.
- **Definition:** A binary tree with n nodes and level k is **complete** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of level k .



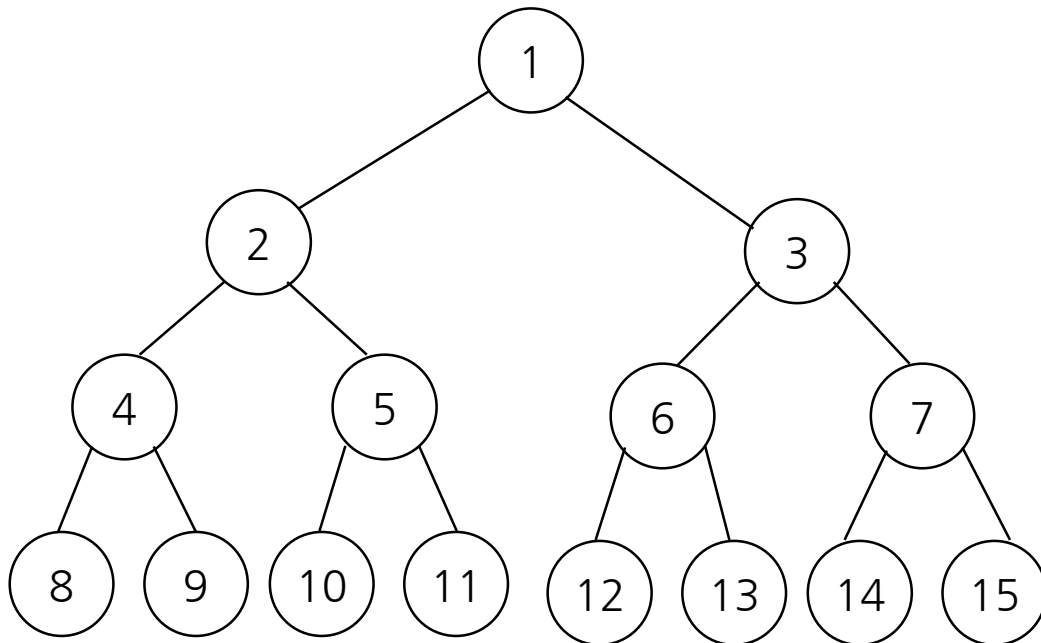
A full binary tree



A complete binary tree

Binary trees - Array representation

- **Q:** Let's suppose that you have a **complete binary tree** in an array, how can we locate node x 's parent or child?
- A **complete** binary tree with n nodes, any node index i , $1 \leq i \leq n$, we have
 - $\text{parent}(i)$ is at $\lfloor i/2 \rfloor$ If $i = 1$, i is at the root and has no parent
 - $\text{leftChild}(i)$ is at $2i$ if $2i \leq n$. If $2i > n$, then i has no left child.
 - $\text{rightChild}(i)$ is at $2i+1$ if $2i+1 \leq n$. If $2i+1 > n$, then i has no right child.



Wow! Can we use this to all binary trees?
Why not?

Problem remains:
The problem with storing an arbitrary binary tree using an array is the inefficiency *in memory usage*.

Binary trees - Properties

(1) The maximum number of **nodes on level k** of a binary tree is

$k \geq 1$

(2) The maximum number of **nodes in a binary tree of level k** is

$k \geq 1$

(3) The maximum level of a **complete binary tree** with **n** nodes is

$\lceil x \rceil$ is the smallest integer $\geq x$.

Binary trees - Properties

(1) The maximum number of **nodes on level k** of a binary tree is

$$2^{k-1}, \quad k \geq 1$$

(2) The maximum number of **nodes in a binary tree of level k** is

$$2^k - 1, \quad k \geq 1$$

(3) The maximum level of a **complete binary tree** with **n** nodes is

$$k(n) = \lceil \log_2 (n + 1) \rceil, \quad \lceil x \rceil \text{ is the smallest integer } \geq x.$$

$$n = 2^k - 1$$

$$n + 1 = 2^k$$

$$\log(n + 1) = \log 2^k$$

$$\log(n + 1) = k$$

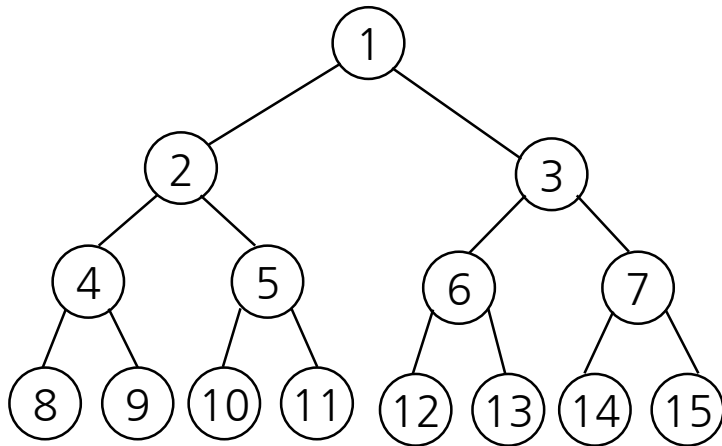
$$k(n) = \lceil \log(n + 1) \rceil$$

$$k(n) = \lceil \log(n) \rceil + 1$$

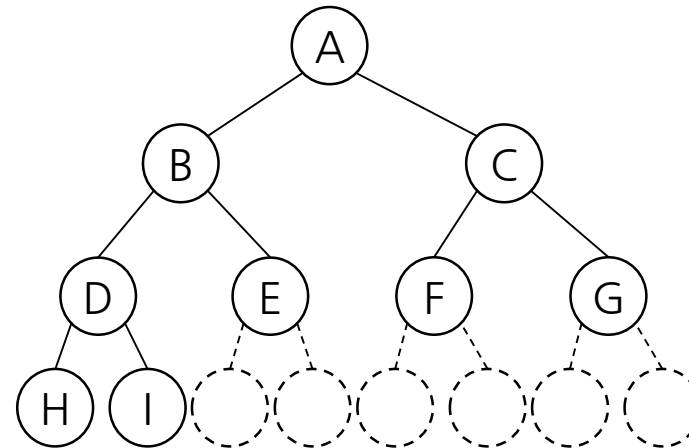
since k is an integer, and includes
the max level of complete binary tree.

Binary trees - Properties

- **Observation:** The max level of a full binary tree of n nodes is $k = \text{floor}(\log(n)) + 1$:
 - Many operations with trees have a run time that goes with the max level of some path within the tree;
 - If we have a full binary tree (or something *close* to it), we know that those operations **run in $O(\log n)$** .



A full binary tree



A complete binary tree

Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

```
root = BinaryTree('A')
print(root)
```

key	
left	right

root

'A'	
None	None

Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

```
root = BinaryTree('A')
print(root)
```

```
<__main__.BinaryTree object at
0x0000015985006A00>
```

key	
left	right

6A00
root

'A'	
None	None

Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- **Create 'B' and link with left of 'A'**

```
root = BinaryTree('A')
node = BinaryTree('B')
```

key	
left	right

6A00

root

'A'	
None	None

7A00

node

'B'	
None	None

Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- Create 'B' and link with left of 'A'.

```
root = BinaryTree('A')
node = BinaryTree('B')
root.left = node
```

key	
left	right

6A00

root

'A'	
None	None

7A00

node

'B'	
None	None

Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- Create 'B' and link with left of 'A'.

```
root = BinaryTree('A')
node = BinaryTree('B')
root.left = node
```

key	
left	right

6A00

root

'A'	
7A00	None

7A00

node

'B'	
None	None

- Simplify the code above.

Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- Create 'B' and link with left of 'A'.

```
root = BinaryTree('A')
node = BinaryTree('B')
root.left = node
```

- Simplify the code and diagram.

```
root = BinaryTree('A')
root.left = BinaryTree('B')
```

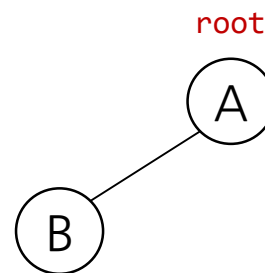
key	
left	right

6A00
root

'A'	
7A00	None

7A00
node

'B'	
None	None



Binary trees - Linked representation

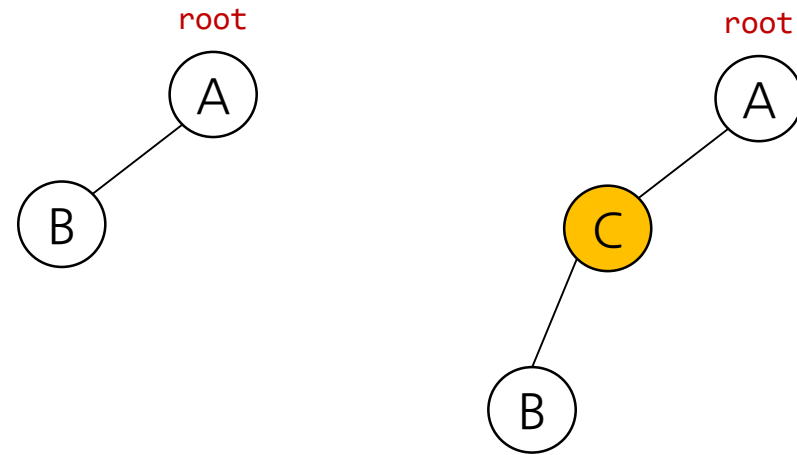
- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- Code **insertLeft()** and insert 'B' & 'C'.

```
root = BinaryTree('A')
root.insertLeft('B')
root.insertLeft('C')
```



Binary trees - Linked representation

- **Node** and **Tree** representations:

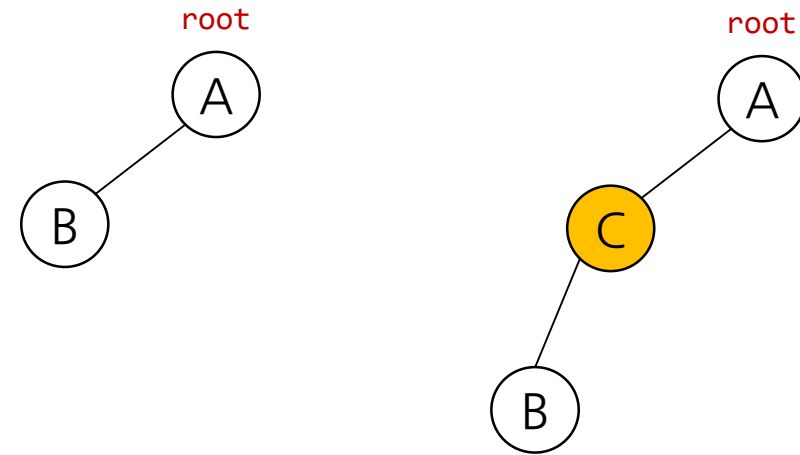
```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- Code insertLeft() and insert 'B' & 'C'.

```
root = BinaryTree('A')
root.insertLeft('B')
root.insertLeft('C')
```

```
def insertLeft(self, key):
    if self.left == None:
        self.left = BinaryTree(key)
    else:
        ...
```



Binary trees - Linked representation

- **Node** and **Tree** representations:

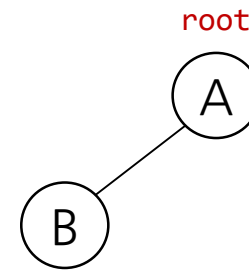
```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- Code insertLeft() and insert 'B' & 'C'.

```
root = BinaryTree('A')
root.insertLeft('B')
root.insertLeft('C')
```

```
def insertLeft(self, key):
    if self.left == None:
        self.left = BinaryTree(key)
    else:
        ...
```

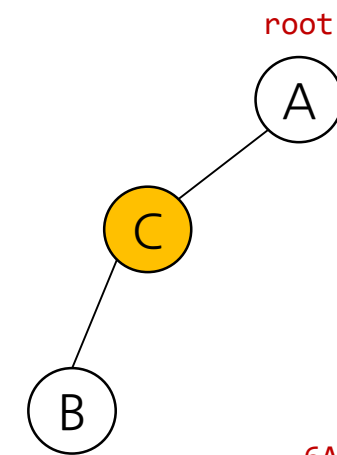


6A00
root

'A'	
7A00	None

7A00

'B'	
None	None



6A00
root

'A'	
7CFF	None

7CFF

'C'	
7A00	None

7A00

'B'	
None	None

Binary trees - Linked representation

- **Node** and **Tree** representations:

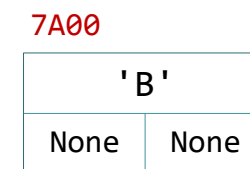
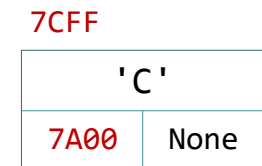
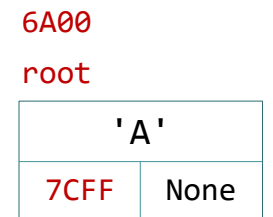
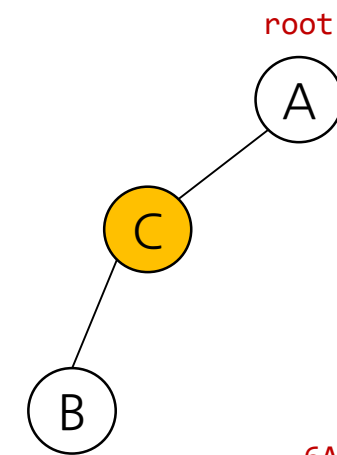
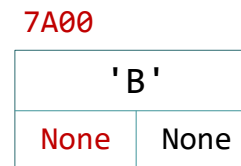
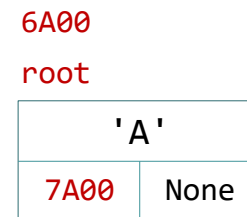
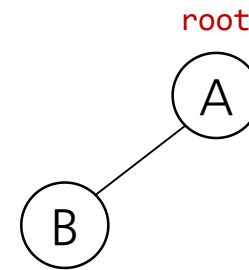
```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```

- Code insertLeft() and insert 'B' & 'C'.

```
root = BinaryTree('A')
root.insertLeft('B')
root.insertLeft('C')
```

```
def insertLeft(self, key):
    if self.left == None:
        self.left = BinaryTree(key)
    else:
        node = BinaryTree(key)
        node.left = self.left
        self.left = node
```

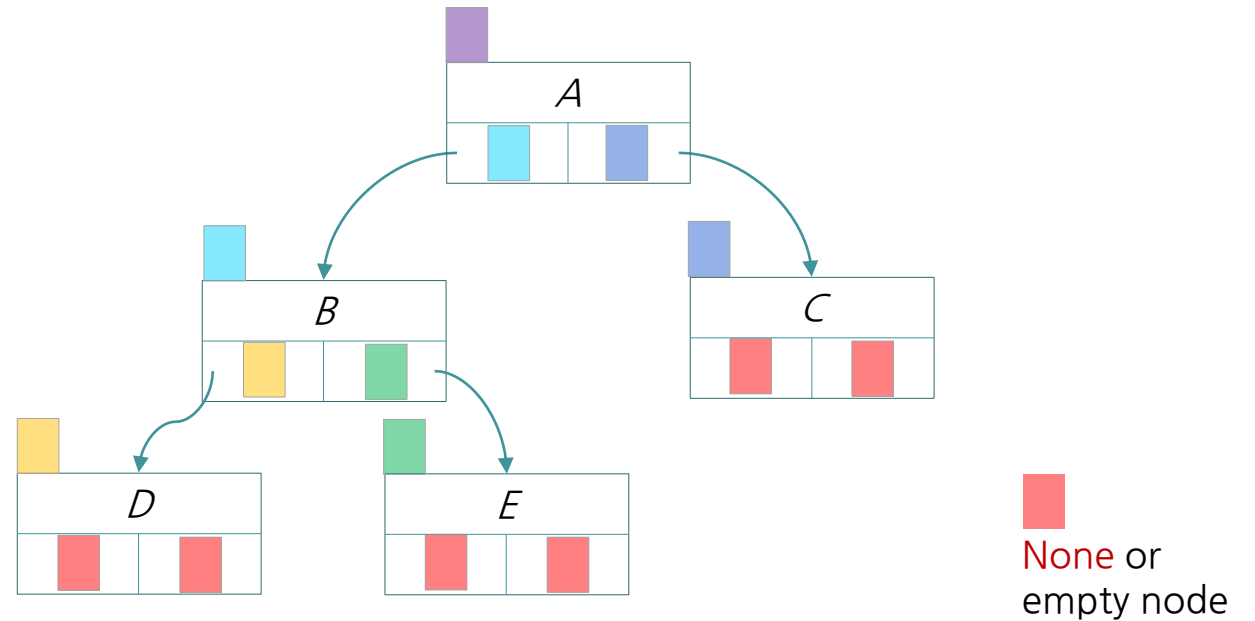


Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```



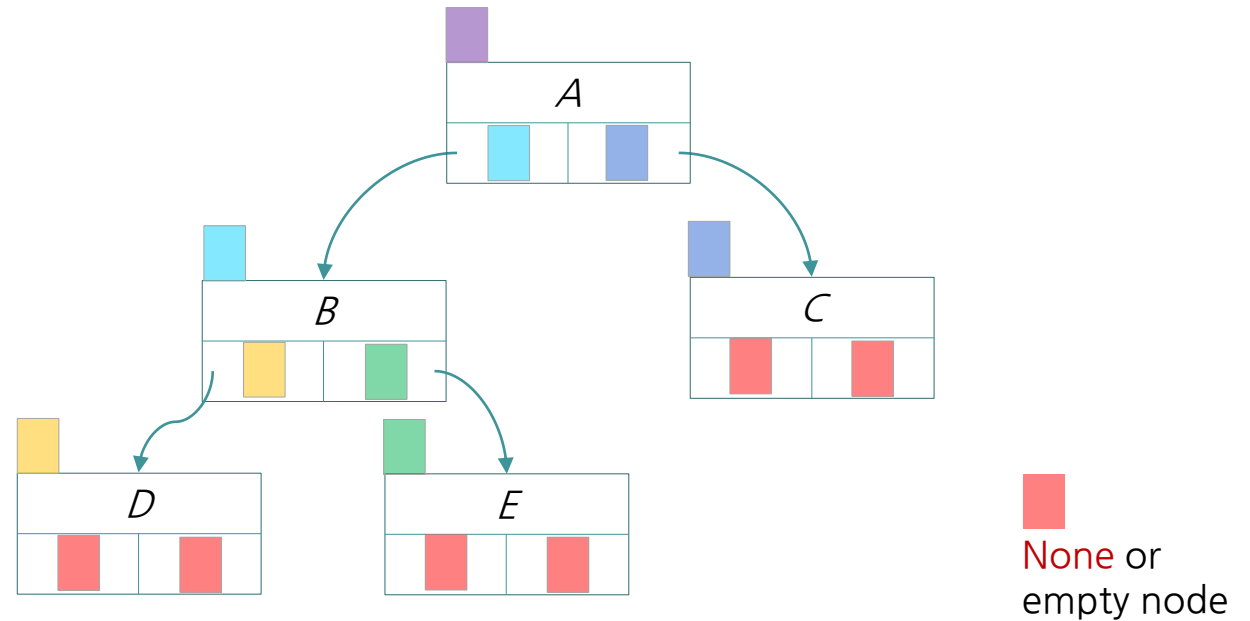
- **Q.** Is this node structure good enough?
 - Not easy to find its parent node. A parent field could be added if necessary.

Binary trees - Linked representation

- **Node** and **Tree** representations:

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
```



- Q. Is this node structure good enough?
 - Not easy to find its parent node. A parent field could be added if necessary.
- Q. It is similar to a doubly-linked list(DLL). What is different?
 - One head, but many tails. **None** points empty node conceptually.

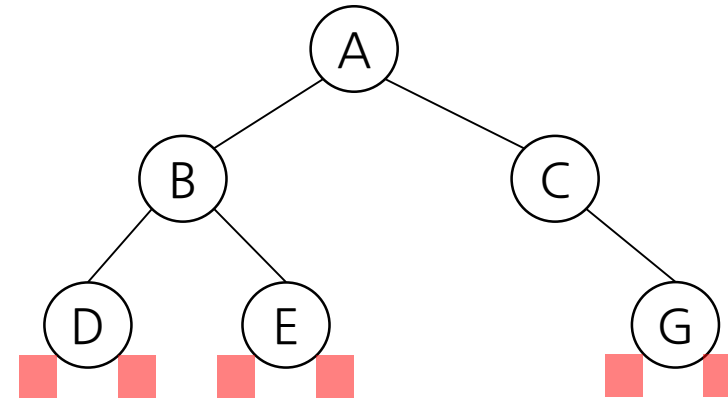
Binary trees - Exercise 1

- Build a tree shown below using insertLeft() and insertRight().

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
    def insertRight(self, key):
        ...

if __name__ == '__main__':
    root = BinaryTree('A')
    root.insertLeft('B')
    root.insertRight('C')
    # your code here
```



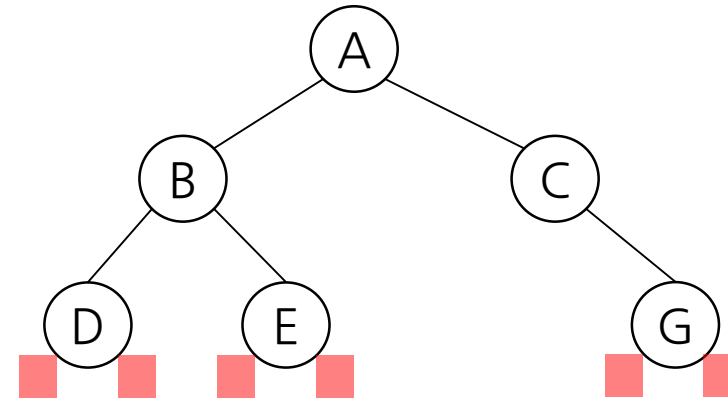
Binary trees - Exercise 2

- Extend Exercise 1 such that it exactly reproduces the output using the as shown below.

```
class BinaryTree:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

    def insertLeft(self, key):
        ...
    def insertRight(self, key):
        ...

if __name__ == '__main__':
    ...
    for node in [a, b, c, d, e, g]:
        if node.key:
            print(...)
    ...
```



```
A: (B, C)
B: (D, E)
C: (None, G)
D: (None, None)
E: (None, None)
G: (None, None)
```

Data Structures in Python

- **Tree Introduction**
- Tree Traversals
- Tree Algorithms
- Binary Search Tree
- Heap
- Heap Sort