파이썬으로 배우는
데이터 구조

한동대학교
전산전자공학부
김영섭 교수

학습 목표

트리(Tree) 구조와 용어에 대해 학습하고

간단하게 구현할 수 있다

Data Structures in Python
Chapter 7 - 1

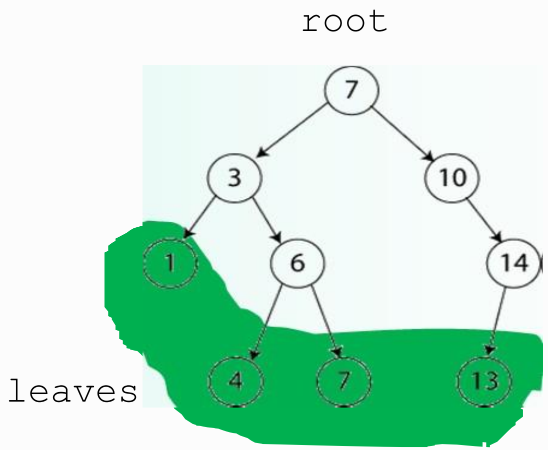
- **Tree Introduction**
- Tree Traversals
- Tree Algorithms

Agenda & Readings

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

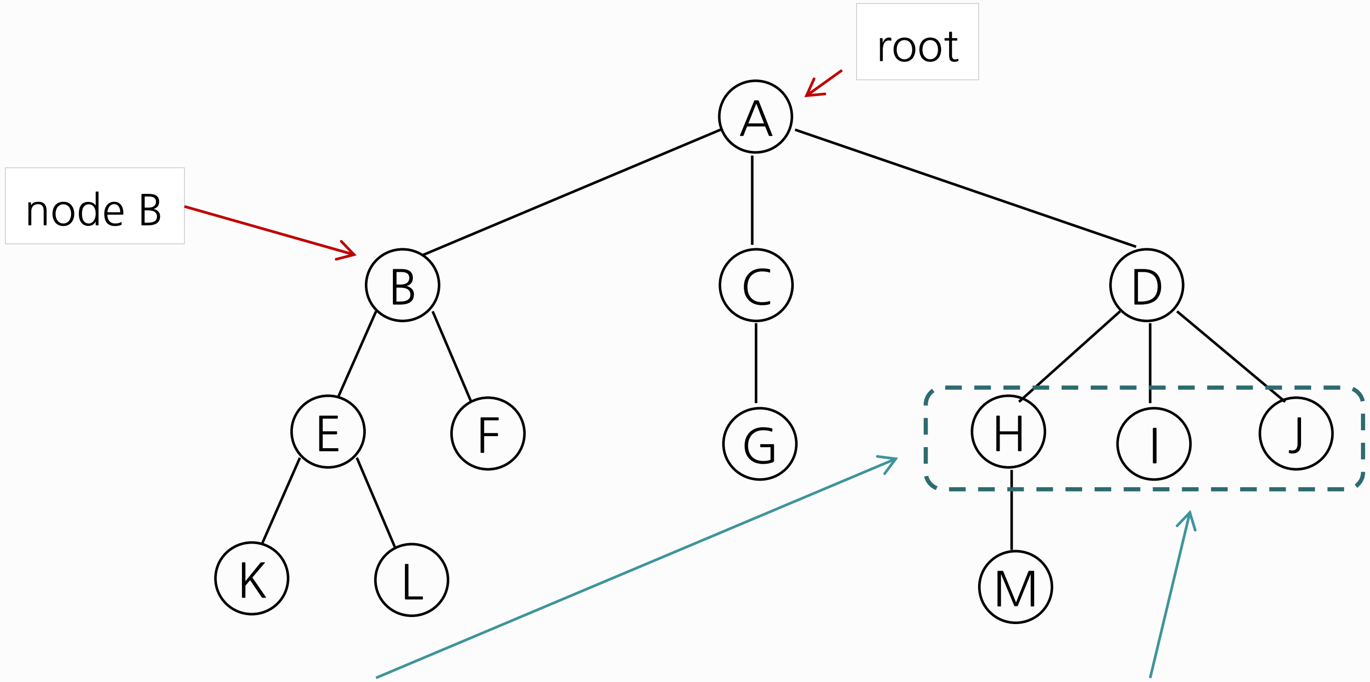
What is a Tree?

- 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[2] connects two distinct nodes.



Introduction - Terminology

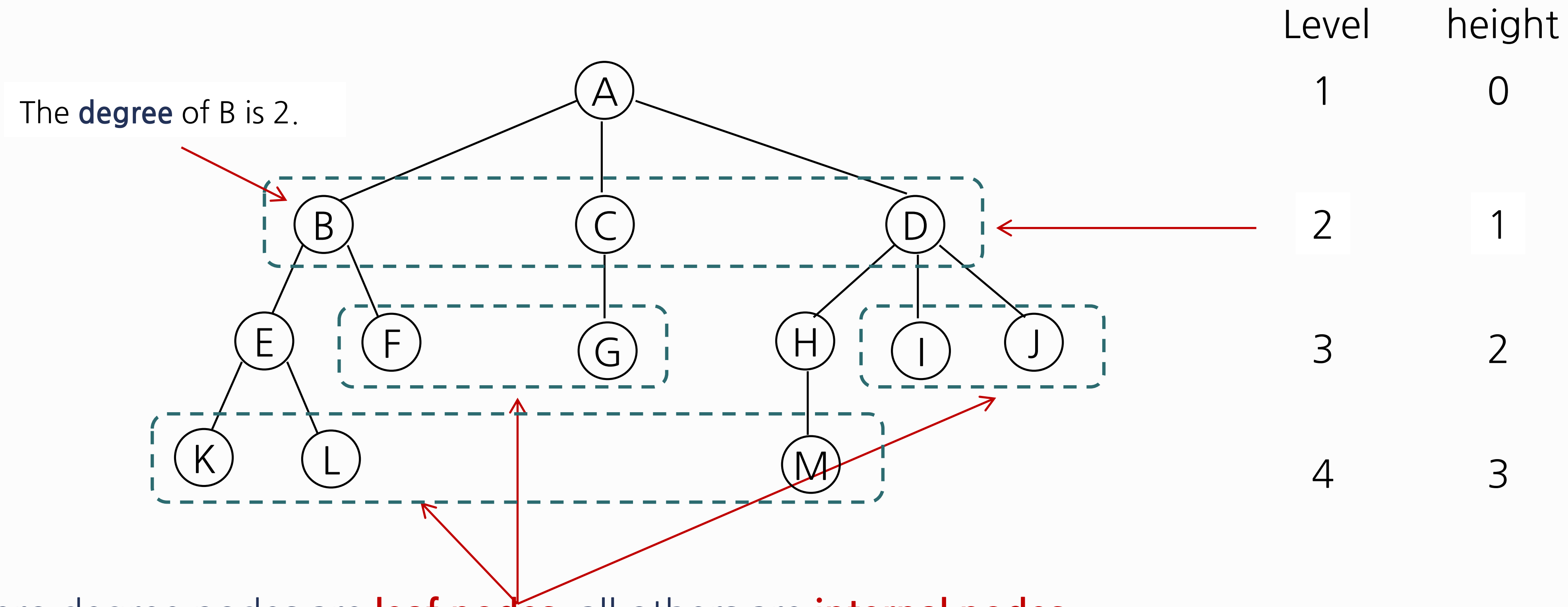
- **A tree data structure:** it is like a linked list that has a **first** node, this node is called as the **root** of the tree.
- **Example.** A **tree** with a root storing the value 'A'



- 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

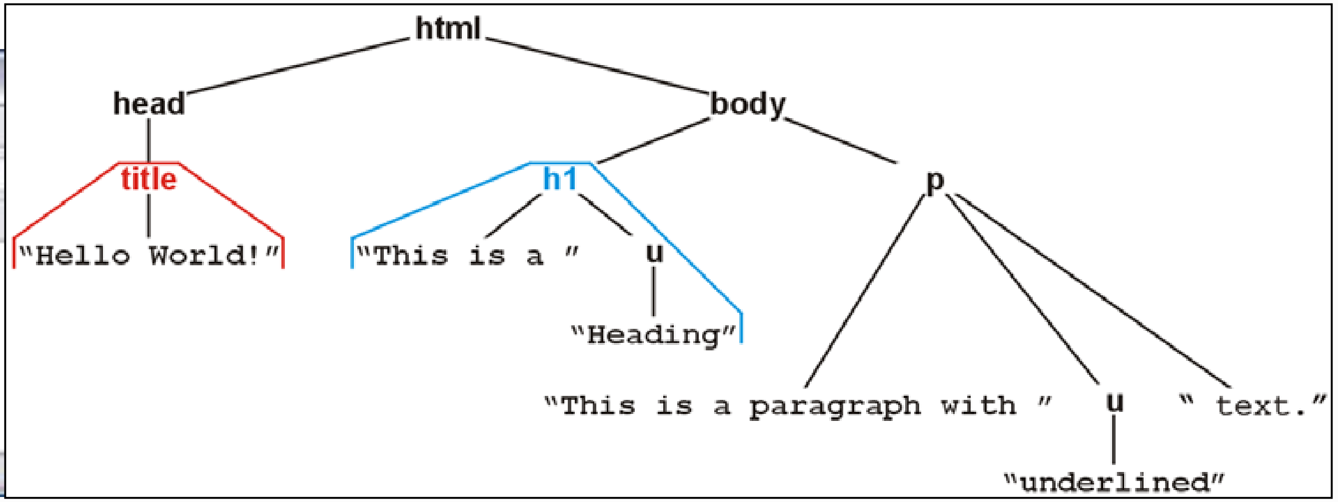
- **Exercise.** The tree representing the HTML document below:

```html
<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

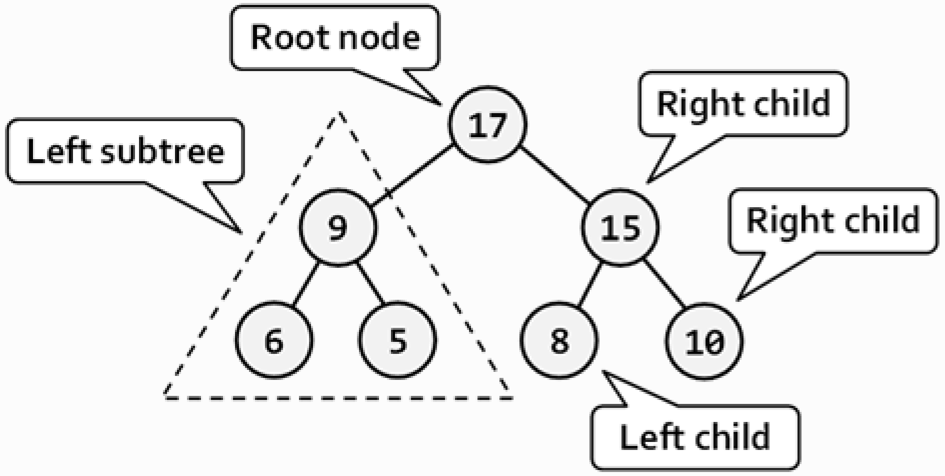
- **Exercise.** The tree representing the HTML document below:

```
<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>
```



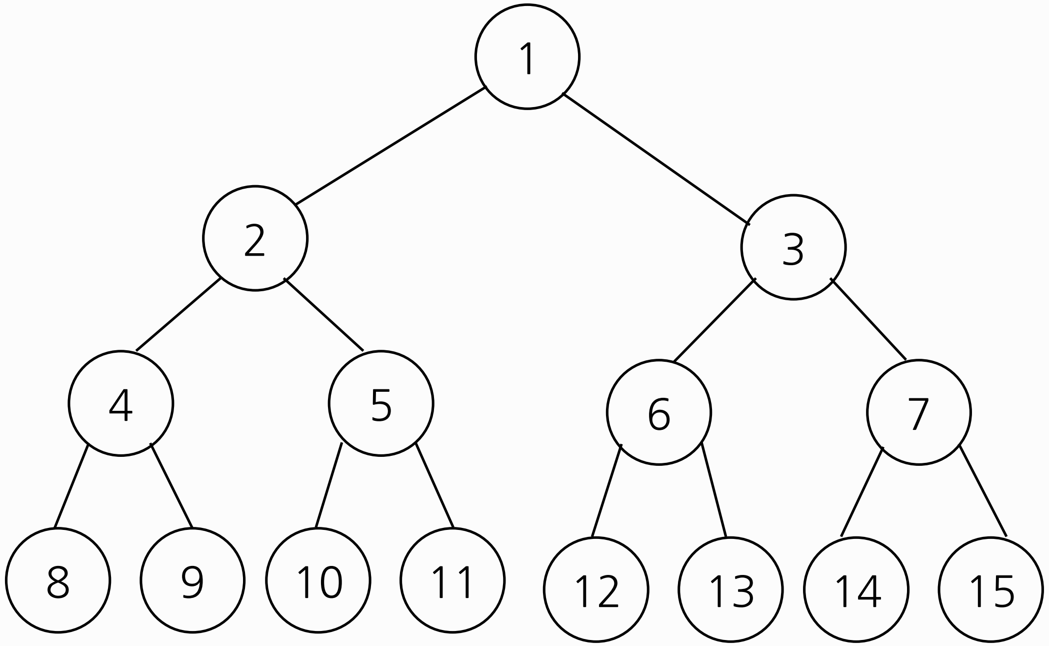
Binary trees

- **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

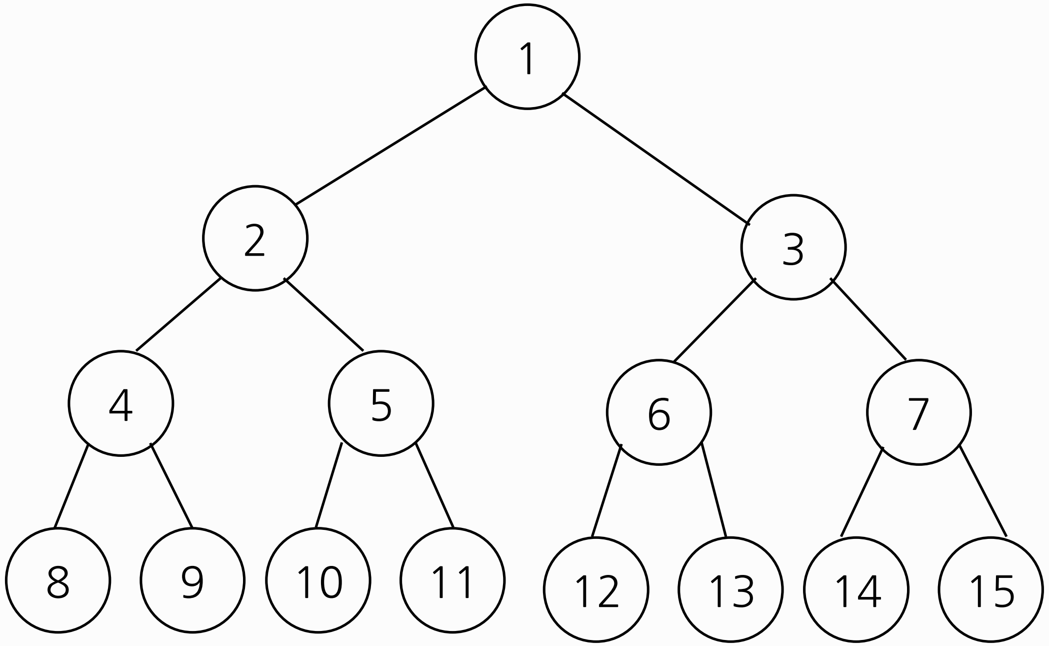
- **Observation:**
  - **Q:** Maximum number of nodes in binary trees in each level and all levels?
  - **Q:** What is the max level k if there are n nodes? k(n) = ?



*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) = ?

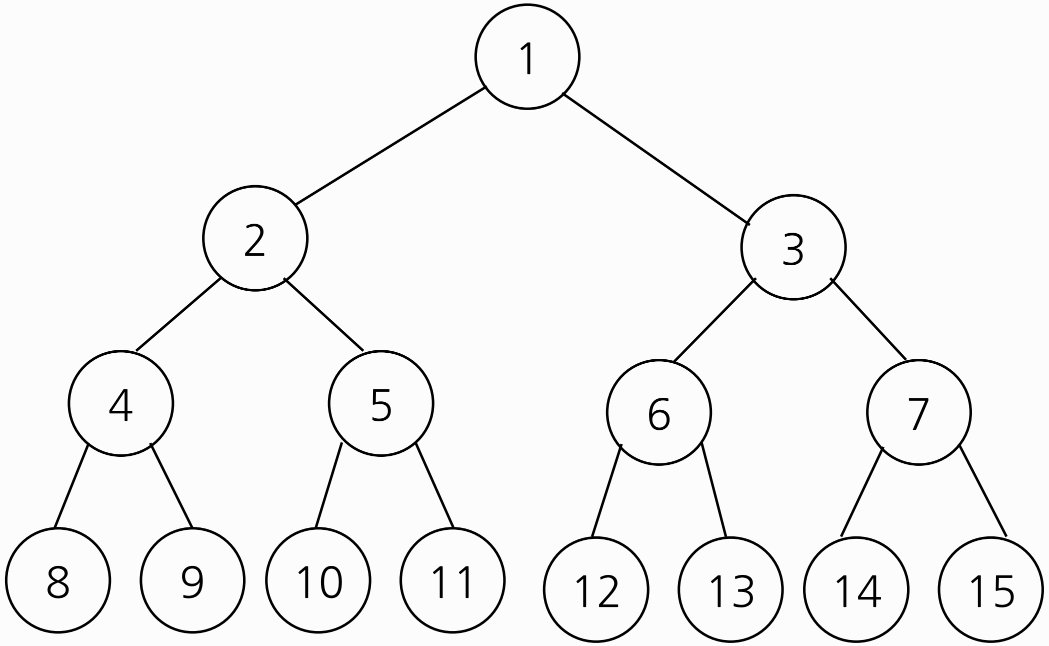


*A **full** binary tree*

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) = ?

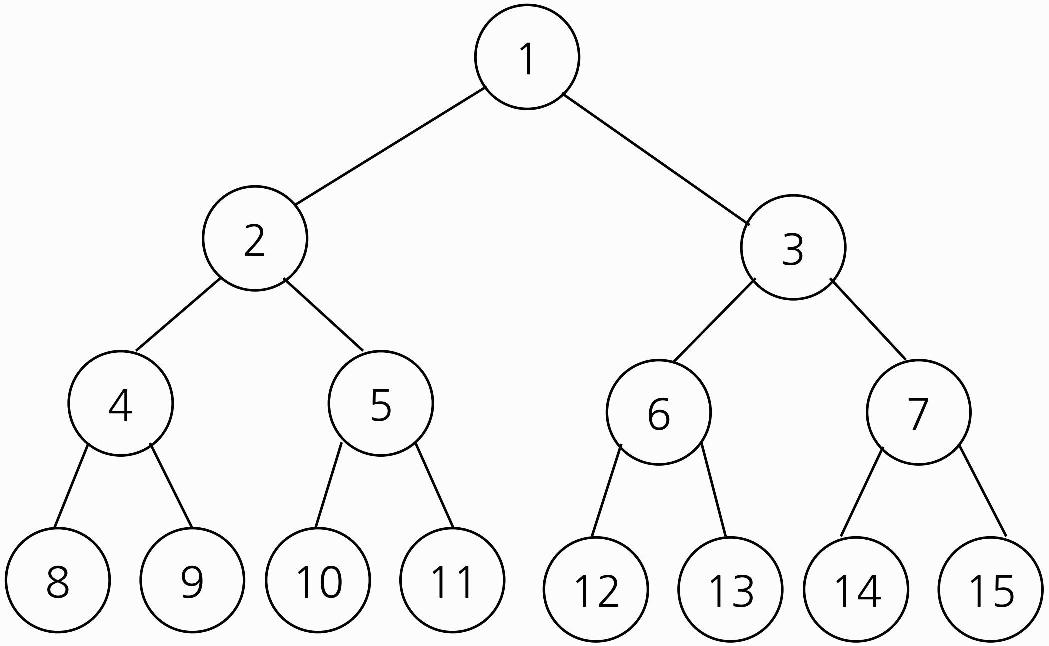


*A **full** binary tree*

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) = ?

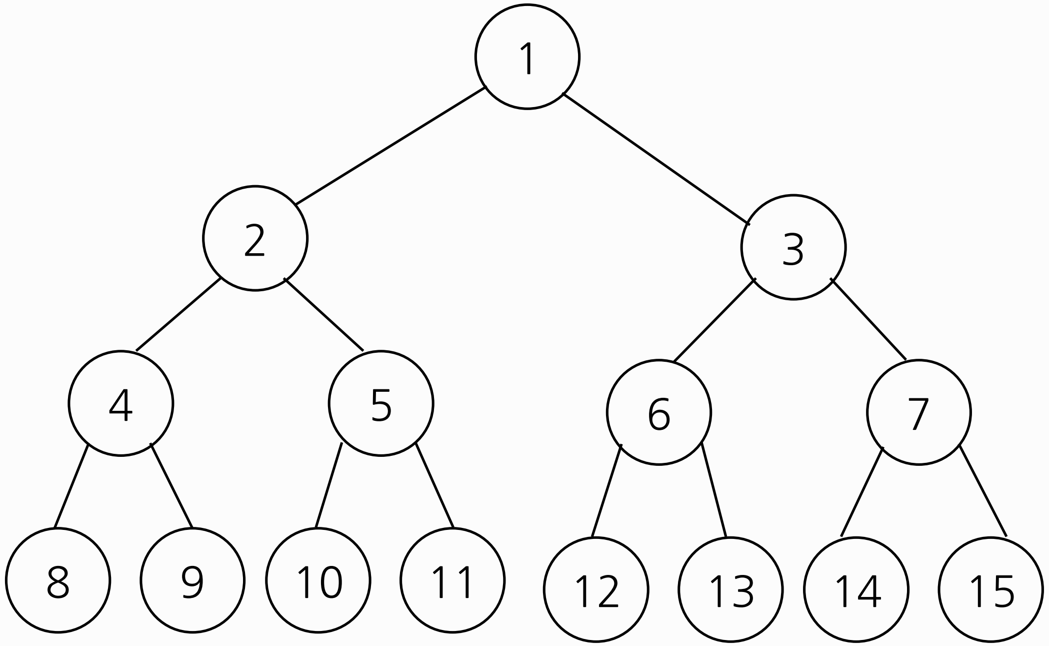


*A **full** binary tree*

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	$\mathbf{2^{k-1}}$	$\mathbf{2^k - 1}$
h	2^h	$\mathbf{2^{h+1} - 1}$

Binary trees - Properties

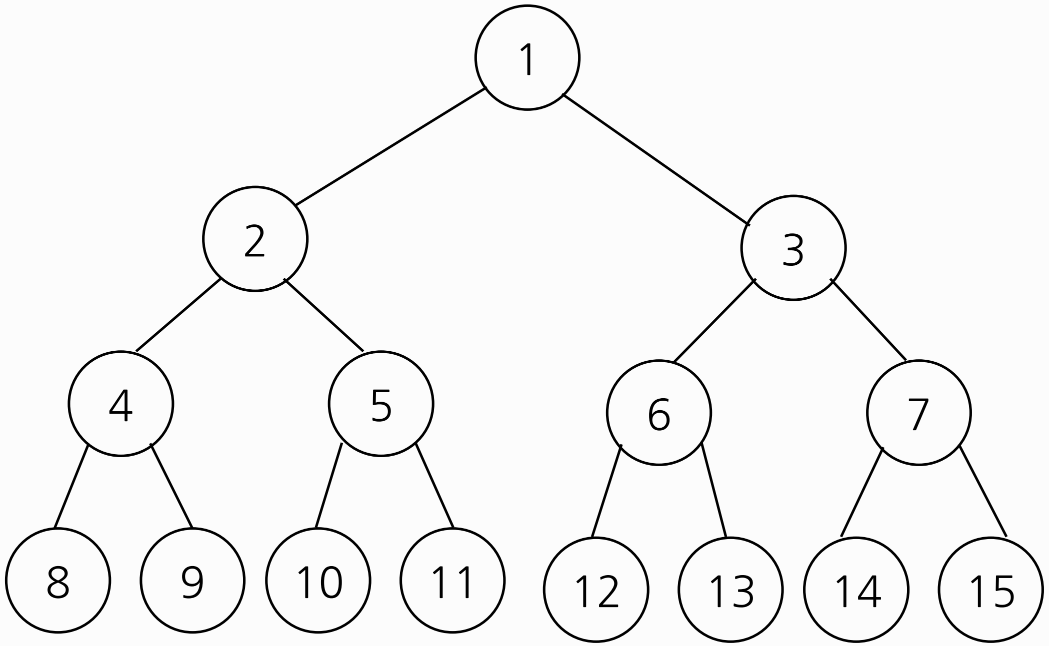
- **Definition:** *A **full** binary tree* of level k is a binary tree having **2^k - 1 nodes**, $k \geq 0$.
- **Definition**: A binary tree with n nodes and level k is **complete** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of level k .



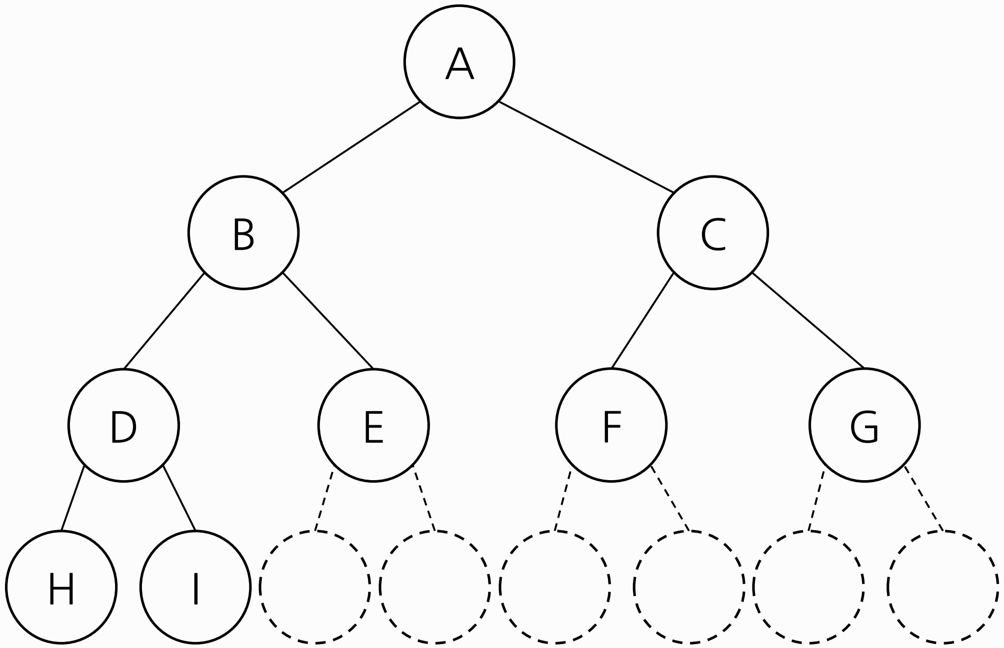
*A **full** binary tree*

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 .



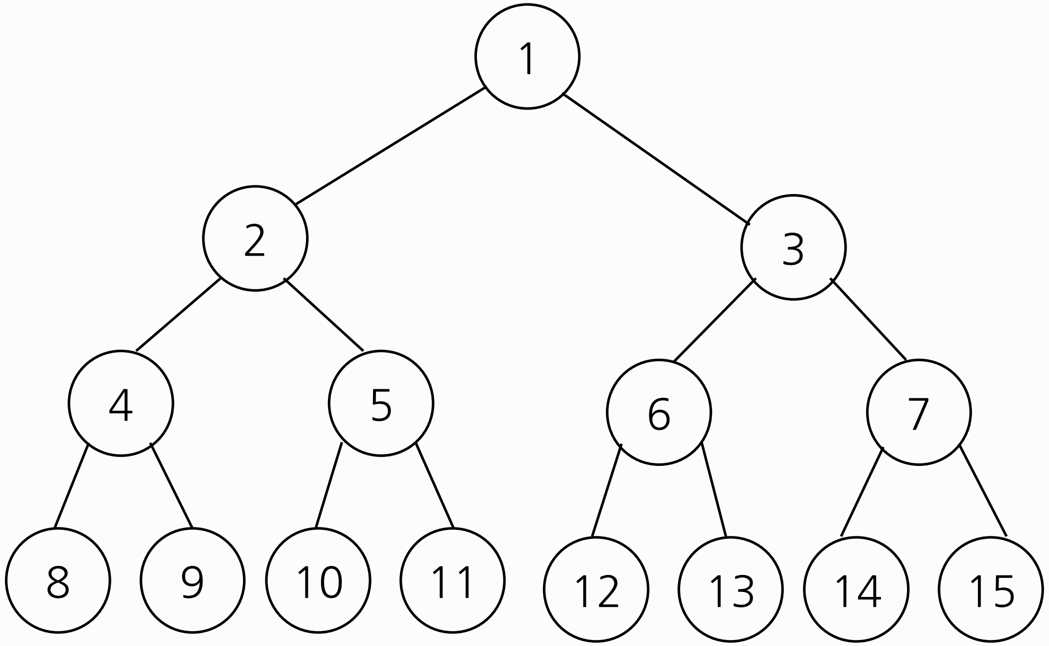
*A **full** binary tree*



*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
  - `parent(i) is at` $\lfloor i/2 \rfloor$ `If i = 1, i is at the root and has no parent`
  - `leftChild(i) is at 2i  if 2i <= n. If 2i > n, then i has no left child.`
  - `rightChild(i) is at 2i+1 if 2i+1 <= 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 level k binary tree of** 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

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

(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

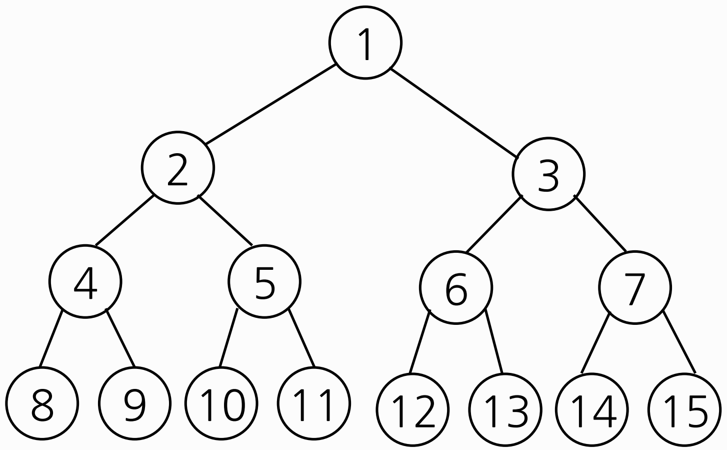
$k(n) = \lceil log_2\,(n+1) \rceil$, $\lceil x \rceil$ is the smallest integer $\geq x$.

$$\begin{aligned} n &= 2^{k} - 1 \\ n + 1 &= 2^{k} \\ \log(n+1) &= log 2^{k} \\ \log(n+1) &= k \\ k(n) &= \lceil \log(n+1) \rceil \\ k(n) &= \lfloor \log(n) \rfloor + 1 \end{aligned}$$

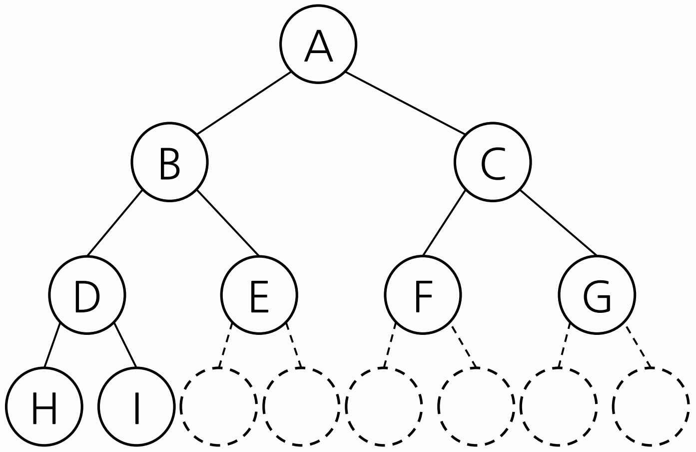
since k is an integer, and includes
the max level of complete binary tree.

Binary trees - Properties

- **Observation:** The max level of a full binary tree of *n* nodes is k = 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 $\boldsymbol{O}(\log \boldsymbol{n})$.



*A **full** binary tree*

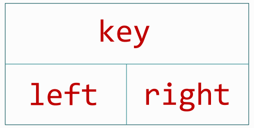


*A **complete** binary tree*

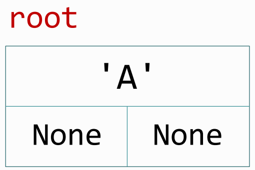
Binary trees - Linked representation

- Node and Tree representations:

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

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




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



Binary trees - Linked representation

- Node and Tree representations:

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

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

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

```
<__main__.Node object at
0x000015985006A00>
```

key	
left	right

6A00

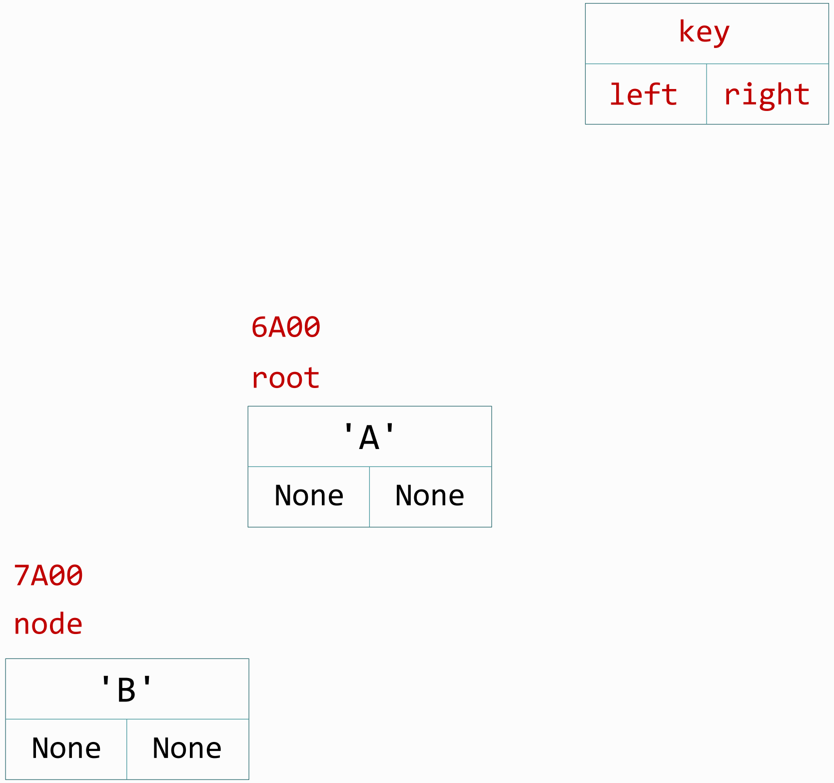
root

'A'	
None	None

Binary trees - Linked representation

- Node and Tree representations:

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

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



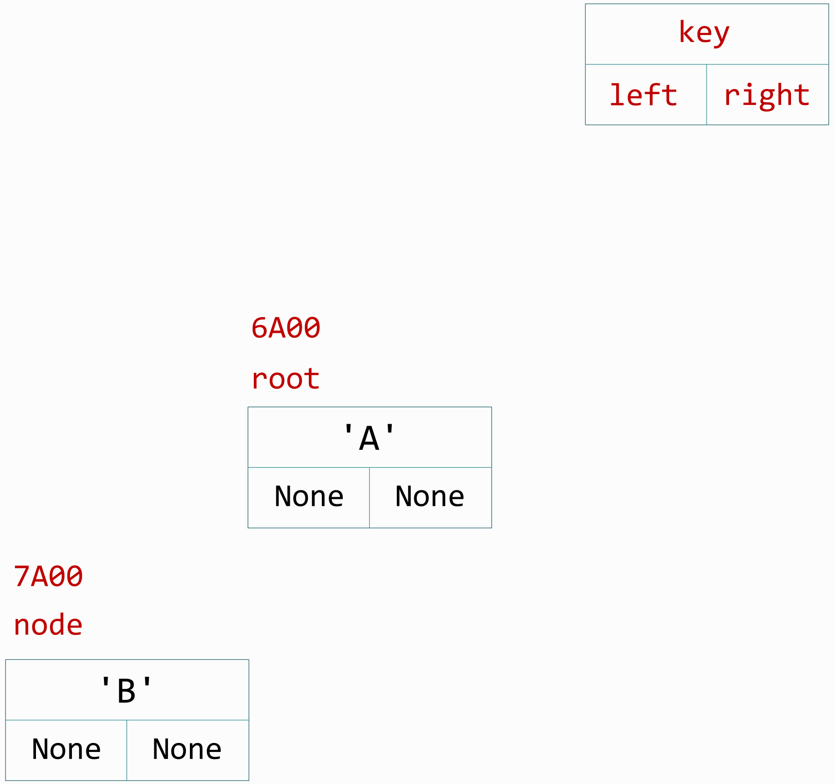
- Create 'B' and link with left of 'A'

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

Binary trees - Linked representation

- Node and Tree representations:

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

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

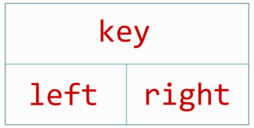


- Create 'B' and link with left of 'A'.

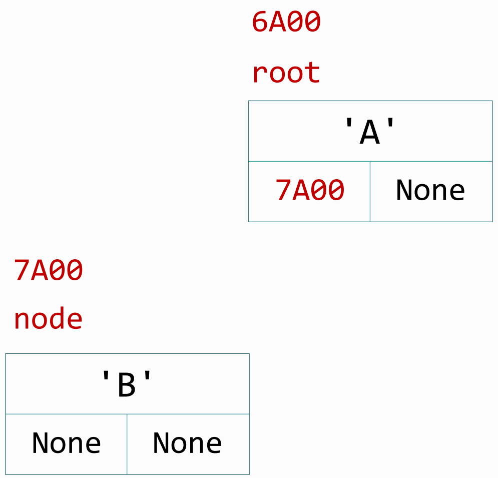
```
root = Node('A')
node = Node('B')
root.left = node
```

Binary trees - Linked representation

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




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

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



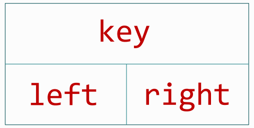
- Create 'B' and link with left of 'A'.

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

- Simplify the code and diagram.

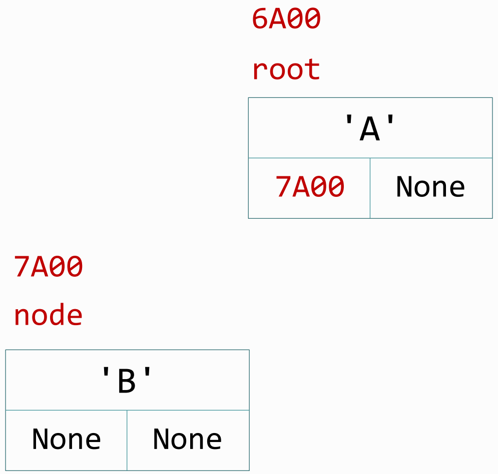
Binary trees - Linked representation

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



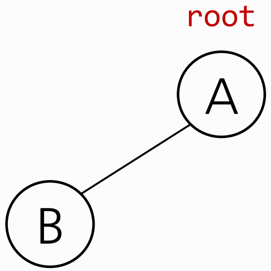

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

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

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




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

- Simplify the code and diagram.

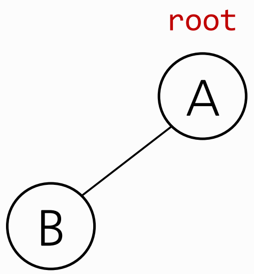


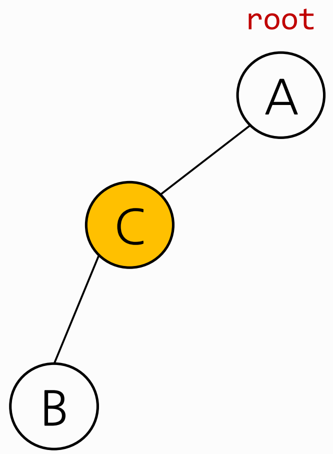

```
root = Node('A')
root.left = Node('B')
```

Binary trees - Linked representation

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

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

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





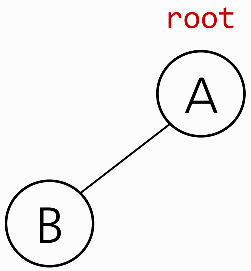
- Code **insertLeft()** and insert 'B' & 'C'.

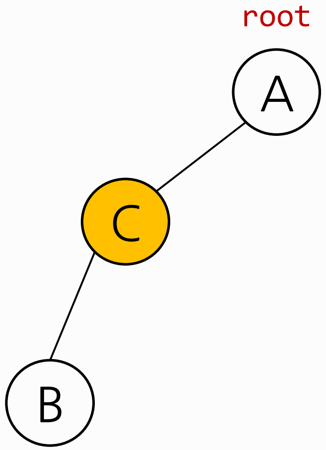
```
root = Node('A')
root.insertLeft('B')
root.insertLeft('C')
```

Binary trees - Linked representation

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

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

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





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

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

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

Binary trees - Linked representation

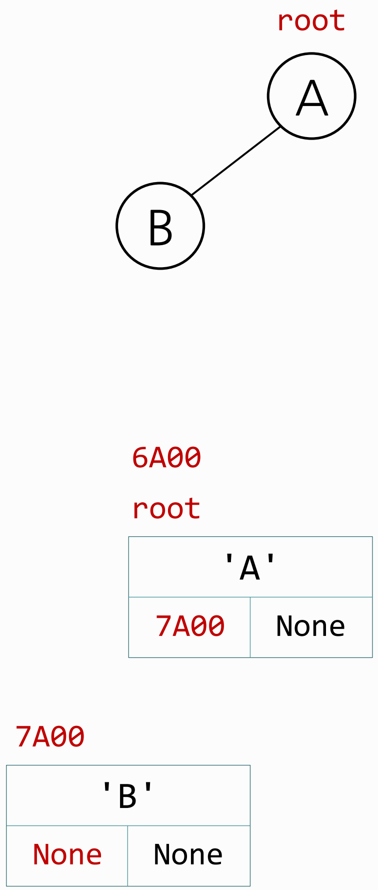
- Node and Tree representations:

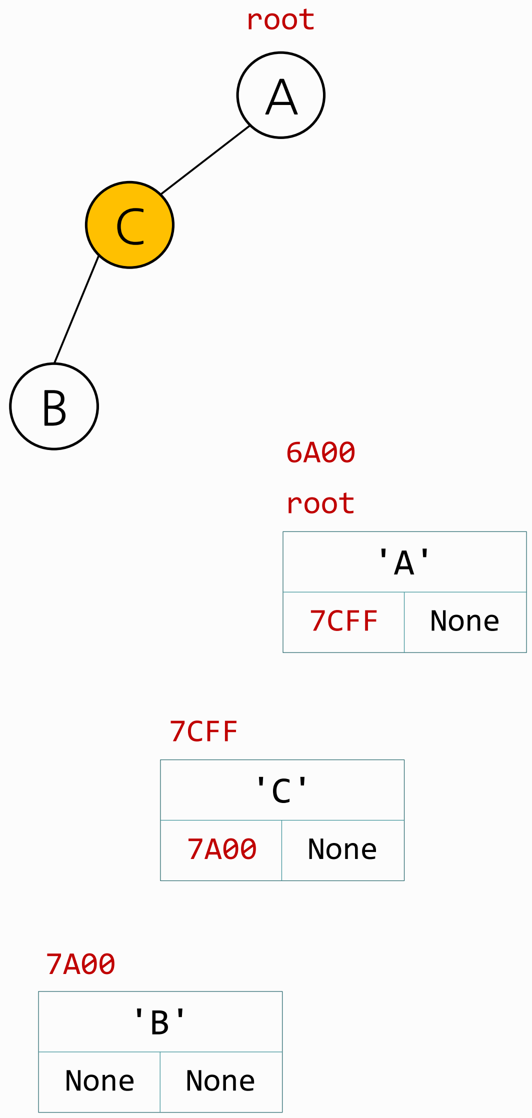
```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

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

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

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

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





Binary trees - Linked representation

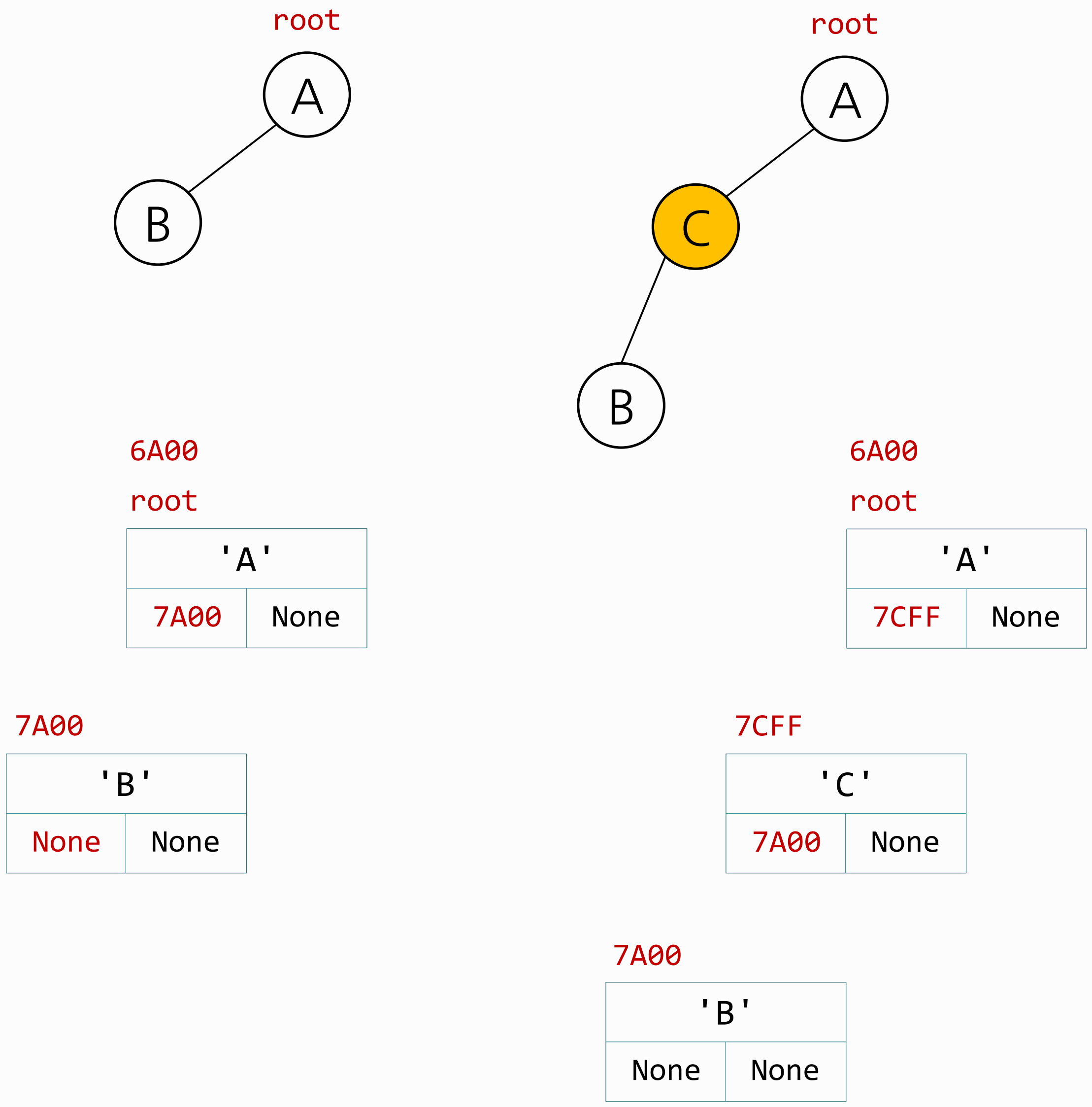
- Node and Tree representations:

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

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

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

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

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



Binary trees - Linked representation

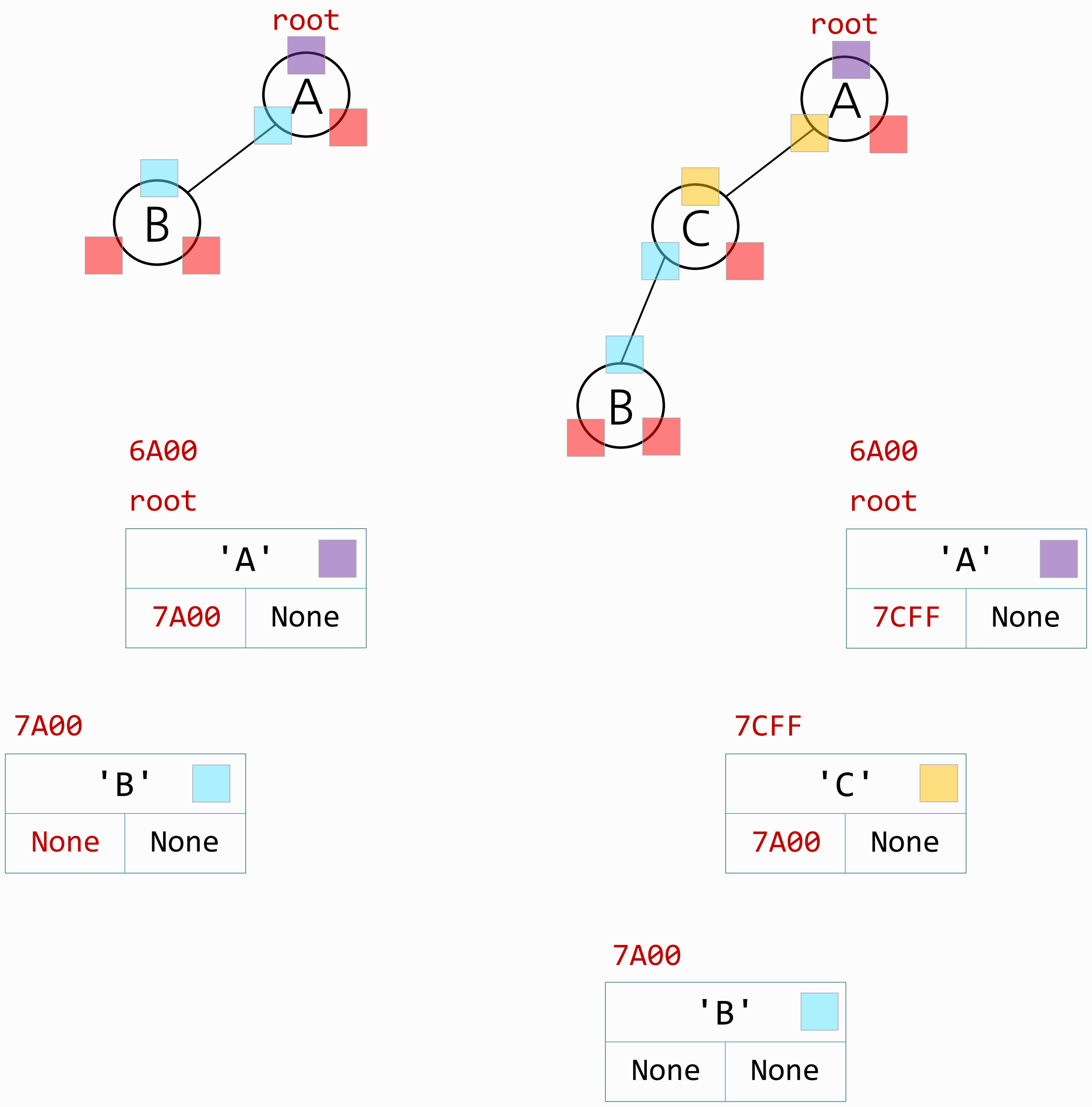
- Node and Tree representations:

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

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

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

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

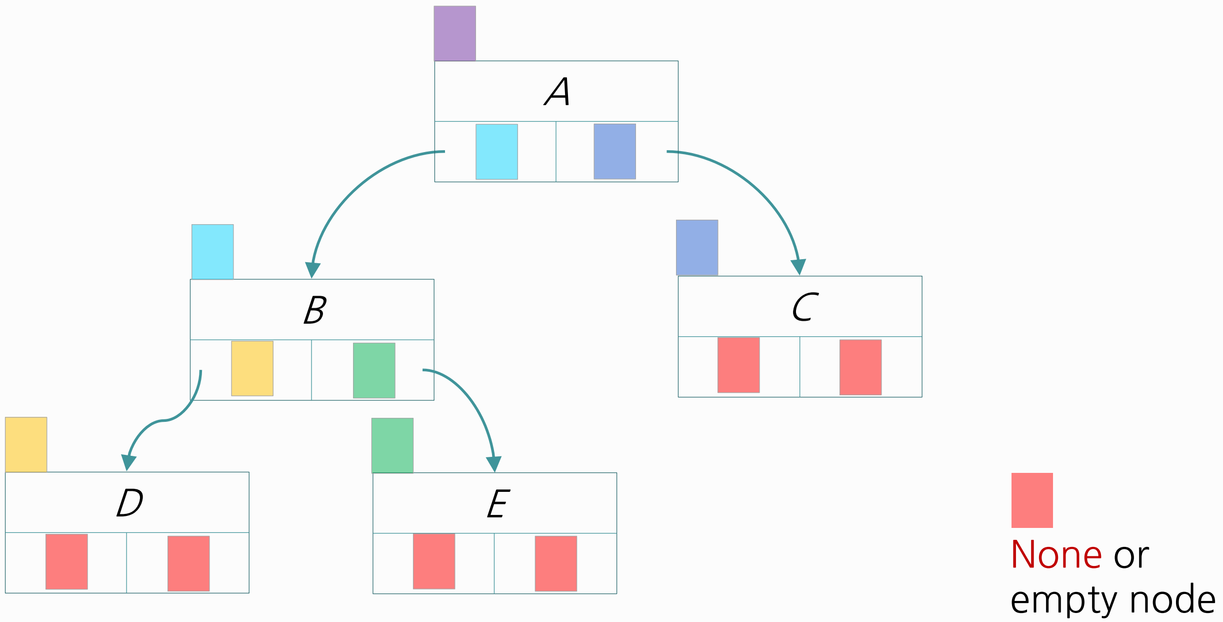
```
def insertLeft(self, key):
    if self.left == None:
        self.left = Node(key)
    else:
        node = Node(key)
        node.left = self.left
        self.left = node
```



Binary trees - Linked representation

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

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

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

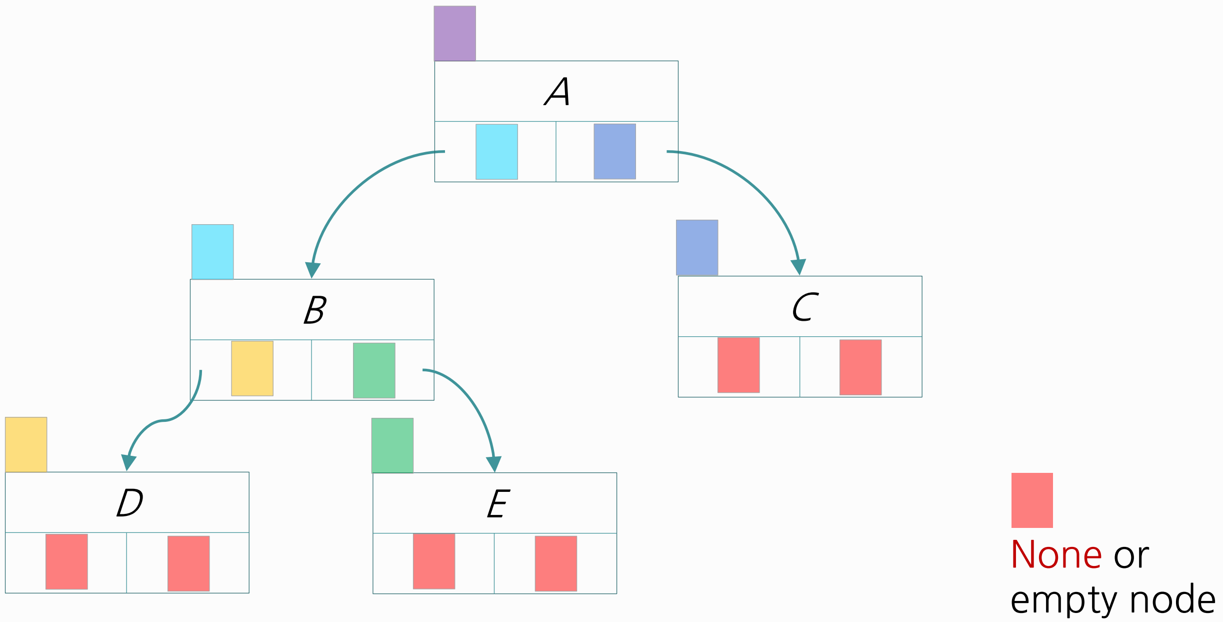


- **Q.** Is this node structure good enough?
  - Not easy to find its parent node. A parent field could be added if necessary.

Binary trees - Linked representation

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

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

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

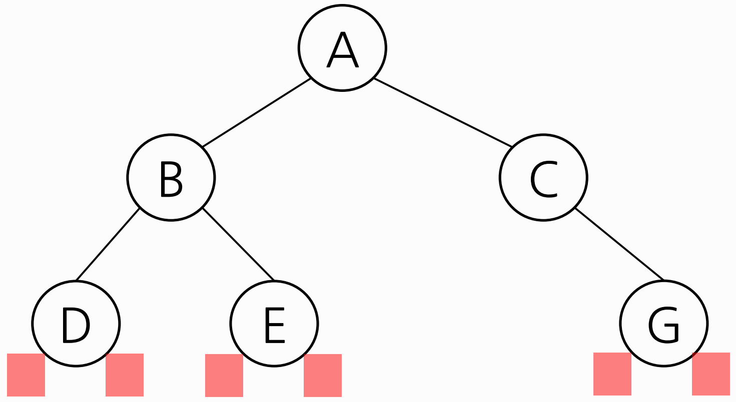


- **Q.** Is this node structure good enough?
  - Not easy to find its parent node. A parent field could be added if necessary.
- **Q.** It is similar to a doubly-linked list(DLL). What is different?
  - One head, but many tails. None points empty node conceptually.

Binary trees - Exercise 1

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

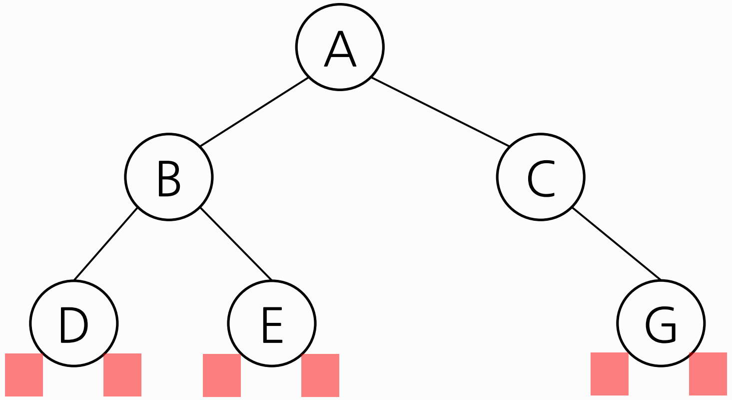
```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

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

if __name__ == '__main__':
    root = Node('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 Node:
    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)
```

학습 정리

1) 트리(Tree)는 비선형 자료 구조이다

2) 완전이진트리(Complete binary tree)에 대한 시간복잡도는 O($\log n$)으로 빠른 알고리즘이다


파이썬으로 배우는
데이터 구조
수고했습니다
곧 다음 시간에
다시 뵙겠습니다