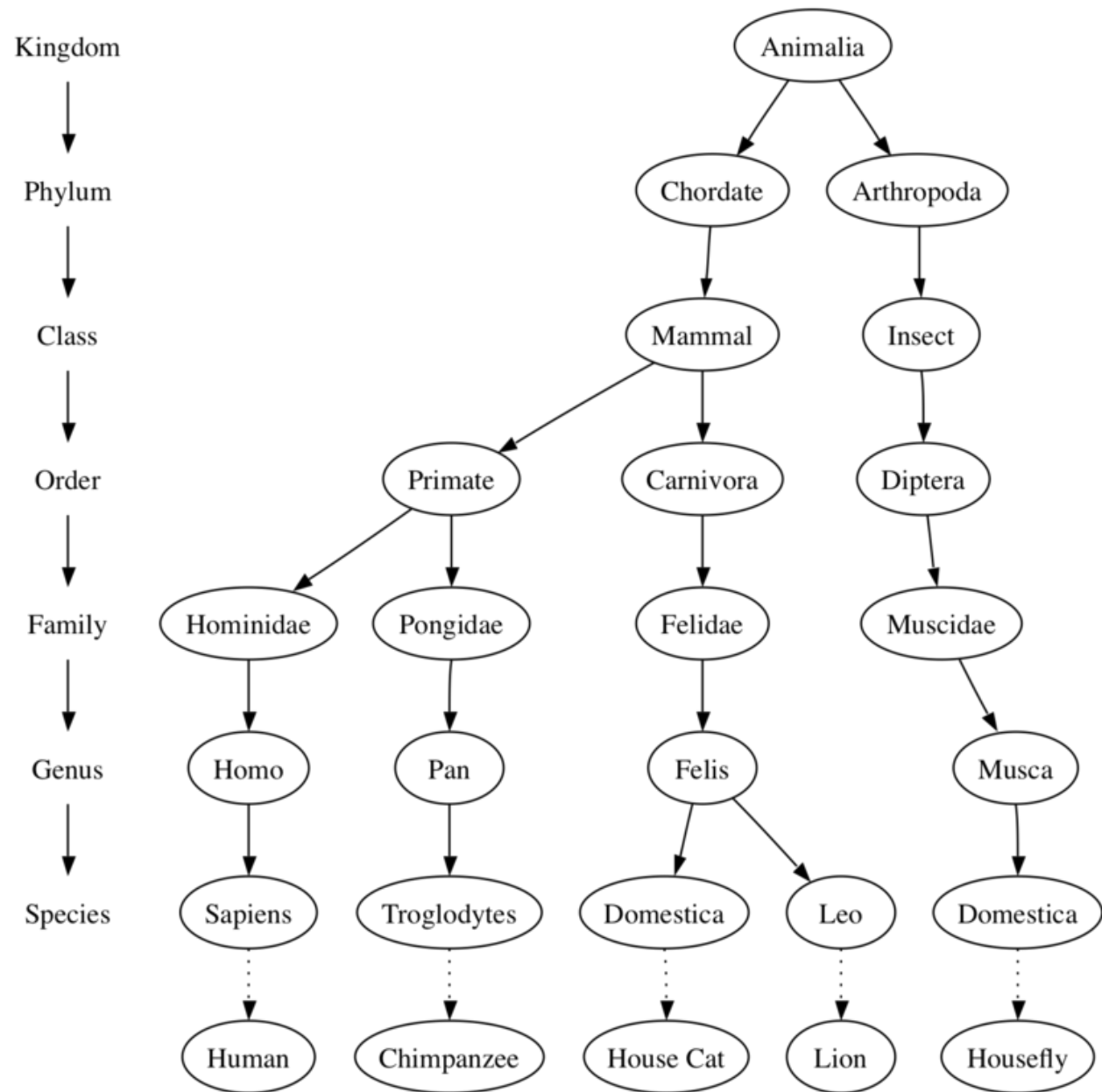
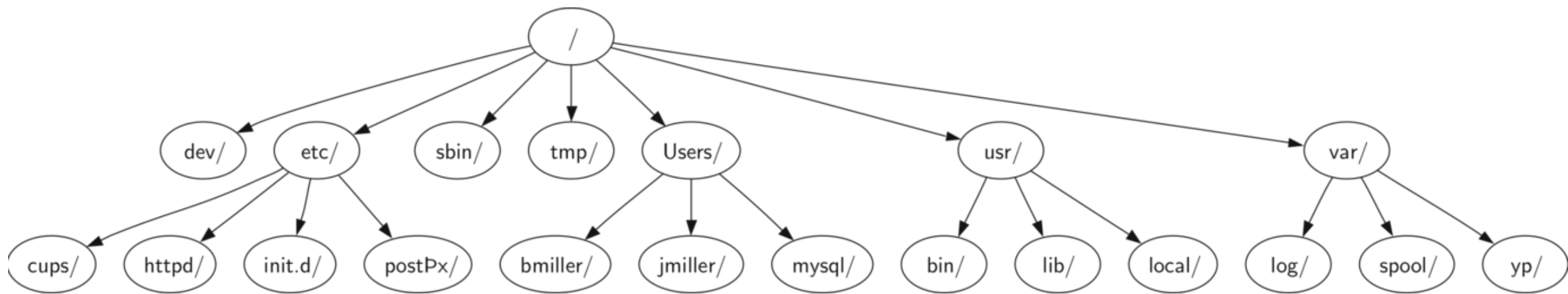


ต้นไม้ (tree)

เรียนเพื่ออะไร?

- เข้าใจหลักการของโครงสร้างข้อมูลต้นไม้
- เพื่อให้เห็นว่าต้นไม้สามารถใช้ในการสร้างโครงสร้างข้อมูลแมพได้
- เพื่อสร้างต้นไม้โดยใช้ **list**
- เพื่อสร้าง **class** ของต้นไม้
- เพื่อสร้างต้นไม้ในรูปแบบความสัมพันธ์เวียนเกิด
- เพื่อสร้างคิวแบบมีความสำคัญ (**priority queue**) ด้วยฮีพ (**heap**)





```
<html xmlns="http://www.w3.org/1999/xhtml"
```

```
  xml:lang="en" lang="en">
```

```
<head>
```

```
  <meta http-equiv="Content-Type"
```

```
    content="text/html; charset=utf-8" />
```

```
  <title>simple</title>
```

```
</head>
```

```
<body>
```

```
<h1>A simple web page</h1>
```

```
<ul>
```

```
  <li>List item one</li>
```

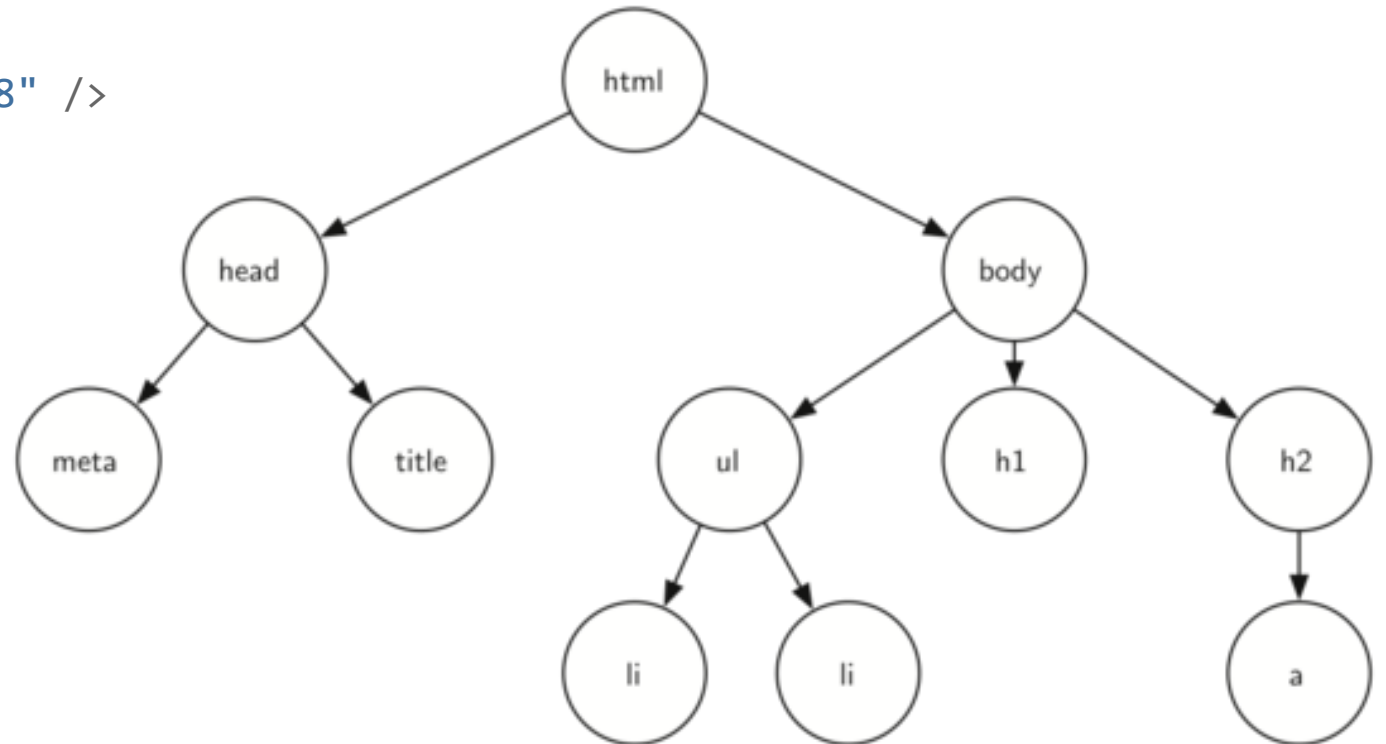
```
  <li>List item two</li>
```

```
</ul>
```

```
<h2><a href="http://www.cs.luther.edu">Luther CS </a></h2>
```

```
</body>
```

```
</html>
```

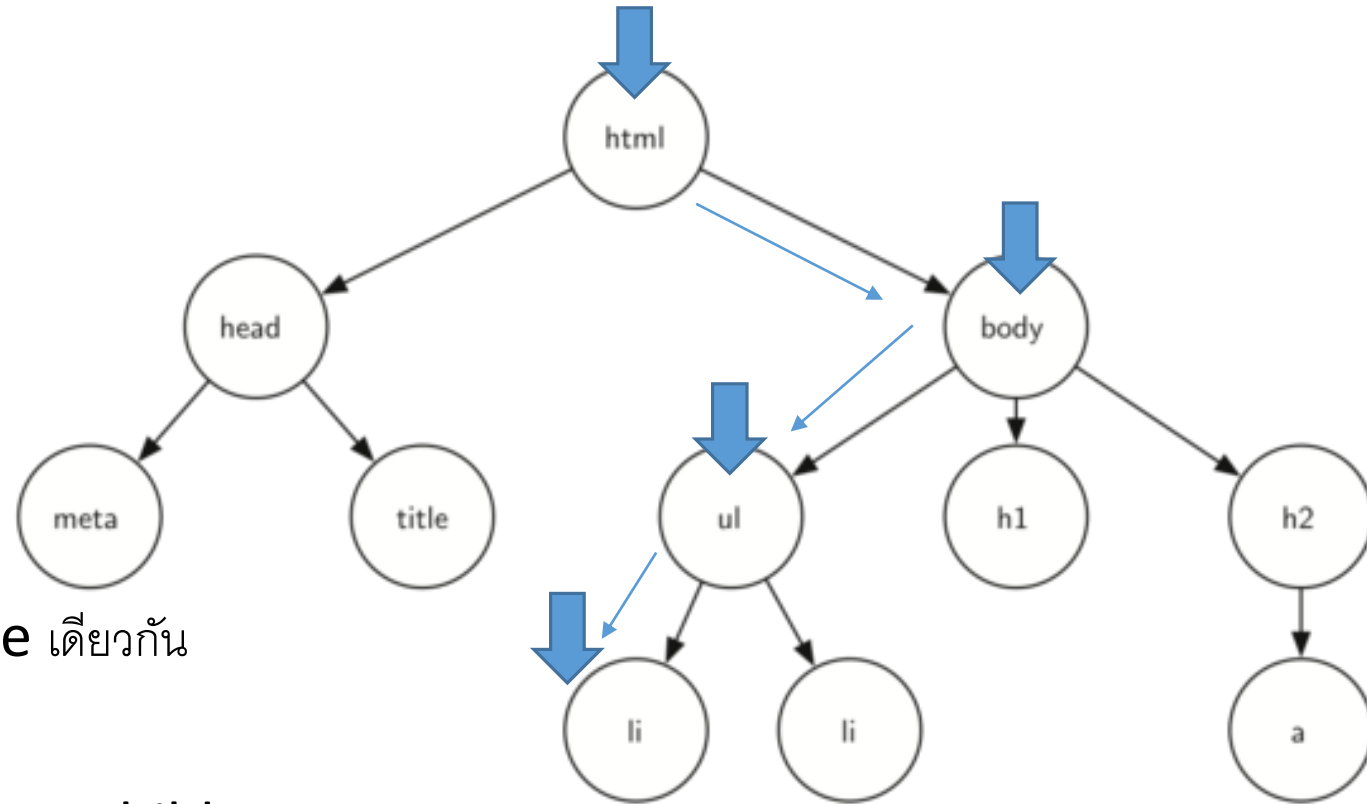


คำศัพท์

- ปม (node)
 - มีชื่อได้ เรียกว่า **key**
 - มีข้อมูลอื่นๆ เพิ่มได้ เรียกว่า **payload**
- เส้นเชื่อม (edge)
 - เชื่อมระหว่าง **2 node** เพื่อแสดงความสัมพันธ์ระหว่างกัน
 - ทุก **node** ต้องมีอย่างน้อย **1 edge** เข้าหามัน (ยกเว้น **root**)
 - **1 node** สามารถมีได้มากกว่า **1 edge** ออกจากตัวมัน
- ราก (root)
 - เป็น **node** ที่มีแต่ขาออก

คำศัพท์

- เส้นทาง (path)
 - list ของลำดับของ node ที่เชื่อมด้วย edge
- ลูก (children)
 - set ของ node ที่มี edge ขาเข้าจาก node เดียวกัน
- พ่อ (parent)
 - node ที่เป็นขาออกของ edge เพื่อไปเชื่อมต่อกับ children
- พี่น้อง (sibling)
 - node ที่มี parent เดียวกัน

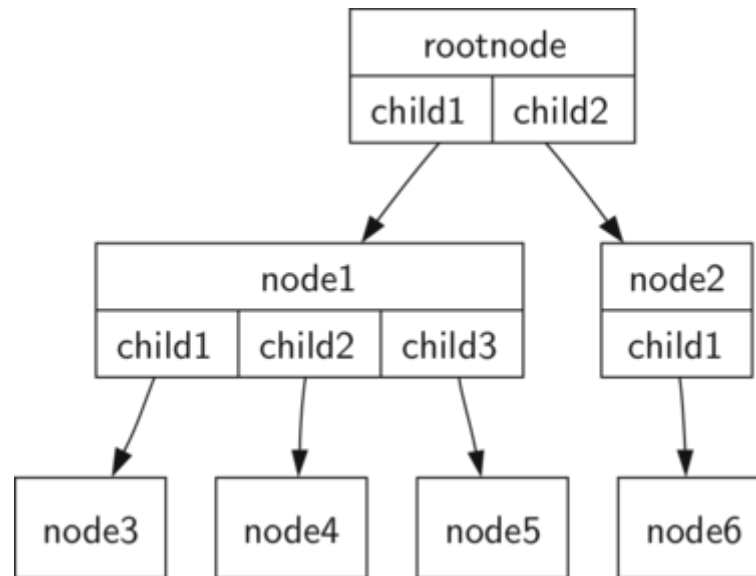


คำศัพท์

- ต้นไม้ย่อย (subtree)
 - set ของ node และ edge ประกอบด้วย parent และผู้สืบสายพันธุ์
- ปมใบไม้ (leaf node)
 - node ที่ไม่มี children
- ระดับ (level)
 - จำนวนของ edge จาก root ถึงตัวมัน
- ความสูง (height)
 - level ที่มากที่สุดของ node ในต้นไม้

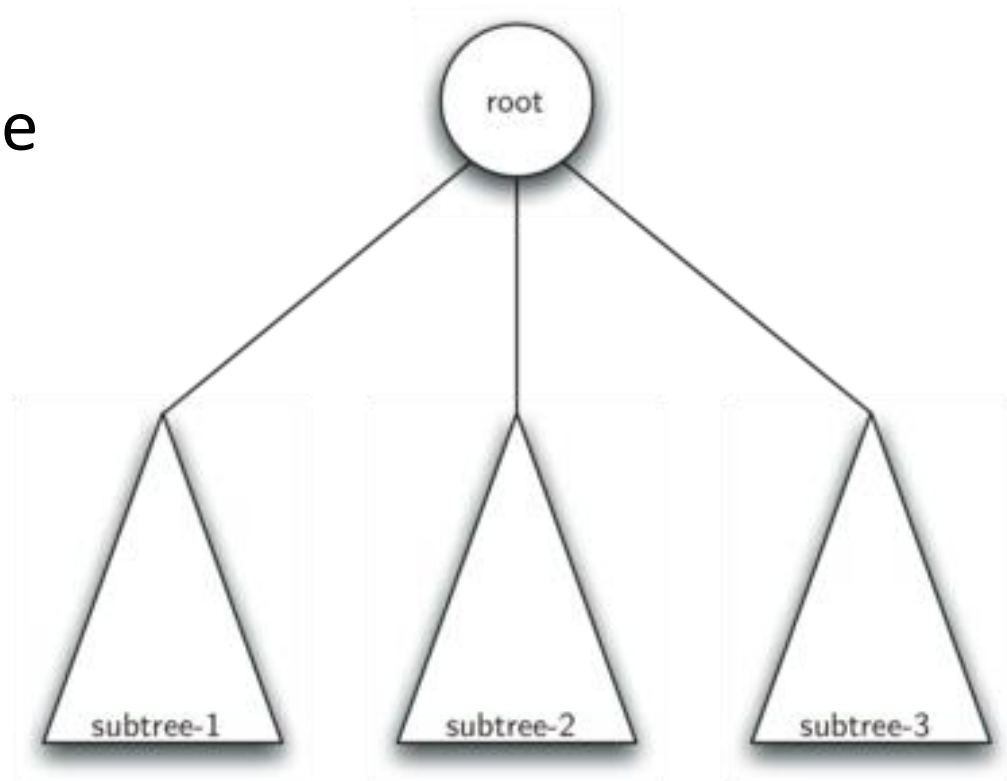
นิยาม 1

- **tree** ประกอบด้วย **set** ของ **node** และ **edge** ซึ่งเชื่อมคู่ของ **node** ซึ่ง **tree** มีคุณสมบัติดังนี้
 - ต้องมี **1 node** เป็น **root**
 - ทุก **node n** ยกเว้น **root** ต้องมี **edge** เชื่อมกับ **node p** ที่เป็น **parent** ของมัน
 - มีแค่ **path** เดียวจาก **root** ถึงแต่ละ **node**
 - ถ้าทุก **node** มีจำนวน **children** ไม่เกิน **2** จะเรียกว่า ต้นไม้ทวิภาค (**binary tree**)



นิยาม 2

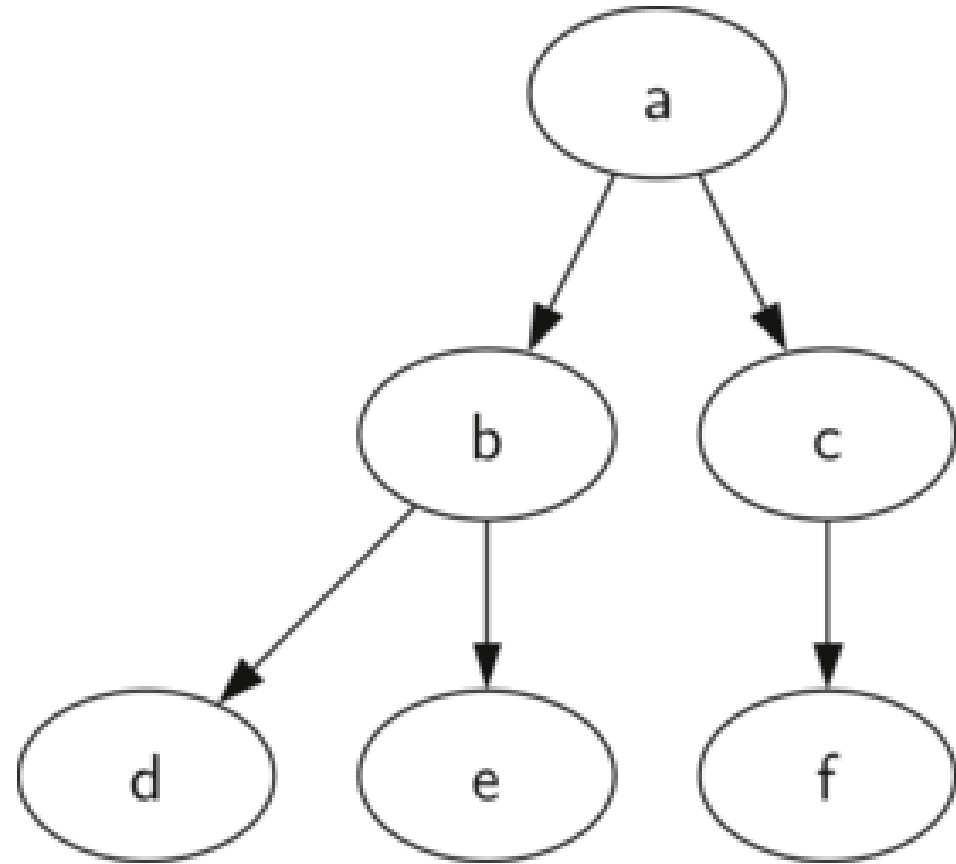
- **tree** จะว่างหรือประกอบด้วย **root** ที่มีหรือไม่มี **subtree** ก็ถือว่าเป็น **tree**
- **root** ของแต่ละ **subtree** ที่เชื่อมต่อกับ **parent** ของ **subtree** ด้วย **edge**
 - เป็นความสัมพันธ์แบบเวียนเกิด
- **tree** ในรูปมีอย่างน้อย 4 node



สร้าง tree ด้วย list ของ list

- แต่ละ node คือ list
 - เก็บค่าไว้ที่ item แรกของ list
 - item ที่ 2 เก็บ list ของ subtree ซ้าย
 - item ที่ 3 เก็บ list ของ subtree ขวา

```
myTree = ['a',  #root
          ['b',  #left subtree
            ['d', [], []],
            ['e', [], []] ],
          ['c',  #right subtree
            ['f', [], []],
            [] ]
        ]
```



การใช้งาน

- `myTree[0]` คือ ค่าที่ **root**
- `myTree[1]` คือ **subtree** ซ้าย
- `myTree[2]` คือ **subtree** ขวา
- เป็นโครงสร้างแบบความสัมพันธ์เวียนเกิด
- ไม่จำกัดว่าจะต้องเป็น **binary tree**

```
myTree = ['a', ['b', ['d',[],[]], ['e',[],[]] ], ['c', ['f',[],[]], [] ]
```

```
print(myTree)
```

```
print('left subtree = ', myTree[1])
```

```
print('root = ', myTree[0])
```

```
print('right subtree = ', myTree[2])
```

```
def BinaryTree(r):  
    return [r, [], []]
```

```
def getRootVal(root):  
    return root[0]
```

```
def setRootVal(root,newVal):  
    root[0] = newVal
```

```
def getLeftChild(root):  
    return root[1]
```

```
def getRightChild(root):  
    return root[2]
```

```
def insertLeft(root,newBranch):  
    t = root.pop(1)  
    if len(t) > 1:  
        root.insert(1,[newBranch,t,[]])  
    else:  
        root.insert(1,[newBranch, [], []])  
    return root
```

```
def insertRight(root,newBranch):  
    t = root.pop(2)  
    if len(t) > 1:  
        root.insert(2,[newBranch,[],t])  
    else:  
        root.insert(2,[newBranch,[],[]])  
    return root
```

```
r = BinaryTree(3)
insertLeft(r,4)
insertLeft(r,5)
insertRight(r,6)
insertRight(r,7)
l = getLeftChild(r)
print(l)
```

```
setRootVal(l,9)
print(r)
insertLeft(l,11)
print(r)
print(getRightChild(getRightChild(r)))
```

```
[5, [4, [], []], []]
[3, [9, [4, [], []], []], [7, [], [6, [], []]]]
[3, [9, [11, [4, [], []], []], []], [7, [], [6,
[], []]]]
[6, [], []]
```