

Lab 8 : Dynamic Binary Search Tree

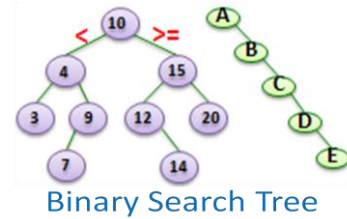
วัตถุประสงค์ ฝึกหัดเขียน Dynamic Binary Tree และ Recursion

ทฤษฎี

Binary Tree: node ใดๆ มีลูกได้ไม่เกิน 2 ตัว (0, 1 หรือ 2) ($bi=2$)

Binary Search Tree: node ใด ๆ มี ลูกซ้ายซ้าย < มัน และ ลูกซ้ายขวา \geq มัน

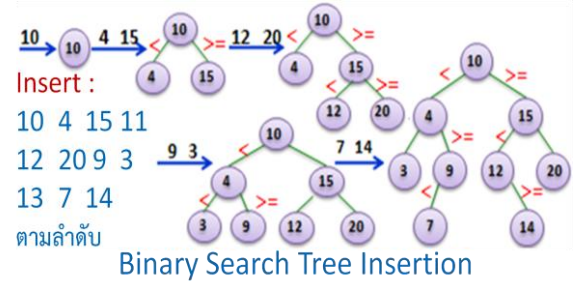
Dynamic Implementation: สร้าง link โดยใช้ pointer (python : เก็บตำแหน่งลูกข้างซ้ายและขวา)



Binary Search Tree

Binary Search Tree Operations :

1. **Insertion** : ใส่เทียบค่า data ที่ต้องการ insert ไปเรื่อยๆ จาก root ลงไปทางซ้ายหากค่า data น้อยกว่า มิฉะนั้นไปทางขวา จนถึง leaf แล้ว insert เป็น leaf ใหม่



Binary Search Tree Insertion

2. **Trasverses** : ท่องไปใน tree เพื่อ visit node ละ 1 ครั้ง แบ่งตามลำดับการ visit
 - a. **inOrder(root)** : inOrder(left subtree), visit root, inOrder(right subtree)
 - b. **preOrder(root)** : visit root, preOrder(left subtree), preOrder(right subtree)
 - c. **postOrder(root)** : postOrder(left subtree), postOrder(right subtree), visit root

Algorithm : postOrder(root)
if (root is not empty)
 postOrder(left subtree)
 postOrder(right subtree)
 visit root

3. **Deletion** : แบ่งเป็น 3 กรณี เมื่อ n

- a. เป็น leaf : delete ได้เลย
- b. มีลูกข้างเดียว : delete โดยต่อพ่อเข้ากับลูก
- c. มีลูก 2 ข้าง :

หา isp = inorder successor รูปม่วง

หรือ inorder predecessor รูปเขียว

copy ค่าของ isp ไปแทน n

แล้ว delete isp ซึ่งมีลูกข้างเดียว

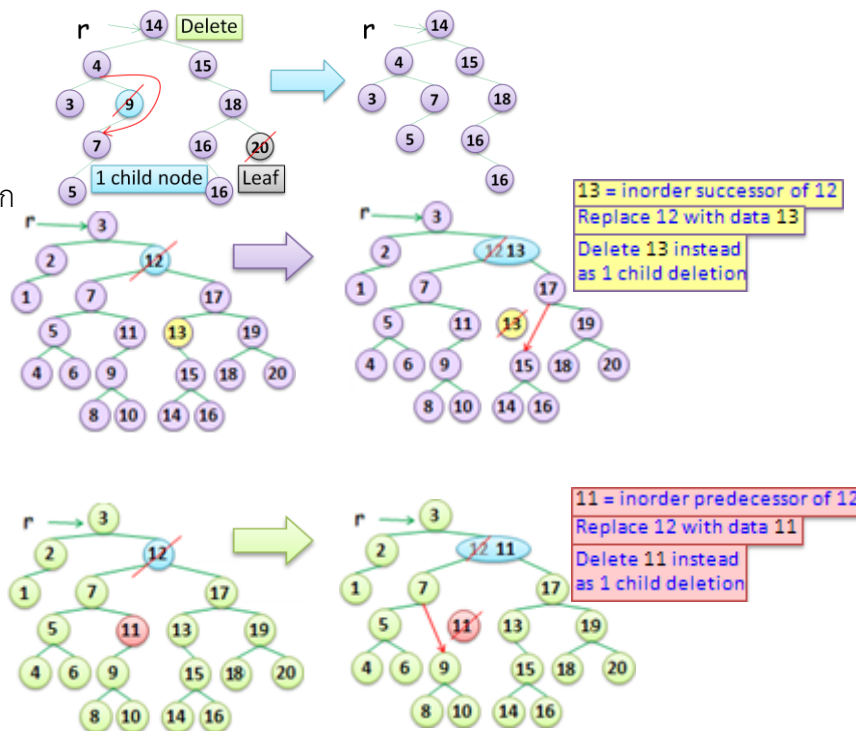
หาก inorder คือ ...10 11 12 13 14 ...

predecessor คือตัวก่อนมัน 1 ตัว

successor คือตัวหลังมัน 1 ตัว


predecessor และ successor ของ 12 คือ 11 และ 13 ตามลำดับ

การ delete ที่มีลูก 2 ข้างอีกวิธีหนึ่งคือ **lazy deletion** แต่จะทำให้ได้ height ยาวขึ้น (ศึกษาในทฤษฎี)



การทดลอง ทำ Dynamic Binary Search Tree (BST) หากนักศึกษาทำได้เอง ไม่ต้องดู code เผลย

- สร้าง class node ของ BST ประกอบด้วยข้อมูล 3 ส่วน คือ data left subtree และ right subtree โดยให้ค่า default ของ left subtree และ right subtree เป็น None เช่น รูป tree ข้างล่างเกิดจาก เรียก

t = node(14) 

class node สามารถสร้างได้หลายอย่าง เช่น มีเฉพาะฟังก์ชัน __init__() หรือจะมี **accessors** และ **mutators** เพื่อใช้ access data ของ class node ตามหลักของ OOP ก็ได้ แต่เพื่อให้ง่ายต่อการเข้าใจ จะใช้เฉพาะฟังก์ชัน __init__() หาก นศ. เขียน accessors และ mutators เมื่อจะ access data ของ class node จะควรเรียกฟังก์ชันเหล่านี้

```
class node:
    def __init__(self, data, left = None, right = None):
        self.data = data
        self.left = None if left is None else left
        self.right = None if right is None else right

    def __str__(self):
        return str(self.data)

    def getData(self):          # accessor
        return self.data

    def getLeft(self):          # accessor
        return self.left

    def getRight(self):         # accessor
        return self.right

    def setData(self, data):    # mutator
        self.data = data

    def setLeft(self, left):    # mutator
        self.left = left

    def setRight(self, right):  # mutator
        self.right = right
```

ตัวอย่าง code ข้างล่าง access data ของ node โดยไม่ใช้ และ ใช้ accessor ตามลำดับ

ไม่ใช้ accessor

```
if data < root.data:
    root.left = node(data)
```

ใช้ accessor

```
if data < root.getData():
    root.setLeft(node(data))
```

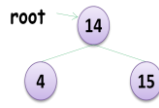
2. สร้าง class BST สำหรับ Binary Search Tree ซึ่งภายในมี data ตัวเดียวคือ root โดยให้ค่า default เป็น None

`t = BST()` ทำให้ root ของ t ซึ่ที่ None **root** → None

3. เพิ่มฟังก์ชันต่างๆ ของ class BST ตามลำดับ พร้อมทดสอบความถูกต้อง

- 3.1. โดยขั้นแรกเพิ่ม iterative function `def addI(self, data)` เพื่อเพิ่ม node ที่มีข้อมูล data เข้าไปใน BST ตามทฤษฎี

ที่เรียนมา ดังนั้น เมื่อเรียก `t.addI(14)` จากรูปในข้อที่แล้ว จะได้รูป  และต่อมาเมื่อ `t.addI(4)`



และ `t.addI(15)` ตามลำดับ จะได้รูป

- 3.2. ทดสอบฟังก์ชันของท่านด้วยการเขียนและเรียกฟังก์ชัน `def inOrder(self)` เพื่อ พิมพ์ inorder traverse ของ tree โดยในฟังก์ชันนี้ให้เรียก ฟังก์ชันอีกฟังก์ชันหนึ่งซึ่งเป็น recursive function pass root เข้าไป

inorder ของ tree จะได้ข้อมูลเรียงจากน้อยไปหามาก ascending order

- 3.3. เพิ่ม iterative function `def add(self, data)` เพื่อ insert data เข้าไปใน tree ดูตัวอย่างการสร้าง recursive function จากข้อที่แล้ว
- 3.4. เขียนฟังก์ชัน `printSideway` เพื่อพิมพ์ tree แบบหันข้าง 90 องศา ตาม algorithm :

```

Algorithm printSideway (tree, level)
  if (tree is not empty)
    printSideway (rightSubtree, level + 1)
    print space 3*level times, print item at root & endOfLine
    printSideway (leftSubtree, level + 1)
  
```

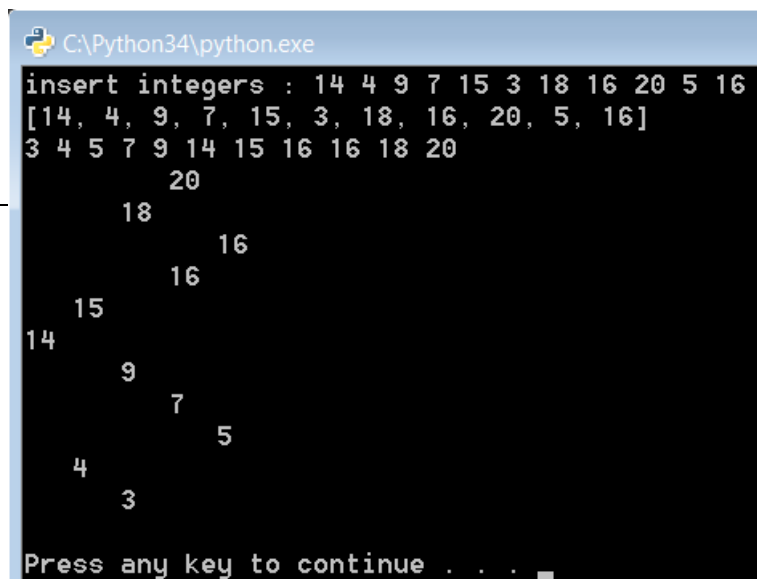
ดังนั้น output ของ code ข้างล่างเป็นดังนี้

```

l = [int(e) for e in input("insert integers : ").split()]
print(l)

t = BST()
for ele in l:
    t.addI(ele)

t.inOrder()
t.printSideway()
  
```



```

C:\Python34\python.exe
insert integers : 14 4 9 7 15 3 18 16 20 5 16
[14, 4, 9, 7, 15, 3, 18, 16, 20, 5, 16]
3 4 5 7 9 14 15 16 16 18 20
    20
  18
    16
  16
15
14
  9
    7
    5
  4
    3

Press any key to continue . . .
  
```

```

class BST:
    def __init__(self, root = None):
        self.root = None if root is None else root

    def addI(self, data):
        if self.root is None:
            self.root = node(data)
        else:
            fp = None          #father of p
            p = self.root      #start comparing from root
            while p:          # while p is not None
                fp = p
                p = p.left if data < p.data else p.right # if data < p.data
                                                         #   p = p.left
                                                         # else:
                                                         #   p = p.right

            if data < fp.data:
                fp.left = node(data)
            else:
                fp.right = node(data)

    def add(self, data):
        self.root = BST._add(self.root, data)

    def _add(root, data):    # recursive _add
        if root is None:
            return node(data)
        else:
            if data < root.data:
                root.left = BST._add(root.left, data)
            else:
                root.right = BST._add(root.right, data)
        return root

    def inOrder(self):
        BST._inOrder(self.root)
        print()

    def _inOrder(root):
        if root:          # if root is not None
            BST._inOrder(root.left)
            print(root.data, end = ' ')
            BST._inOrder(root.right)

    def printSideway(self):
        BST._printSideway(self.root, 0)
        print()

    def _printSideway(root, level):
        if root :          # if root is not None
            BST._printSideway(root.right, level+1)
            print(' '*level, root.data, sep = ' ')
            BST._printSideway(root.left, level+1)

```

- 3.5. เขียนฟังก์ชัน `def preOrder(self)` และ `def postOrder(self)` เพื่อ พิมพ์ traverse ของ tree ในแต่ละแบบ
- 3.6. เขียนฟังก์ชัน `def search(self, data)` เพื่อหาว่ามี data อยู่ใน tree หรือไม่ return pointer ที่ชี้ไปที่ node นั้น
(python : return node นั้น) หากมี หรือ return None หากไม่มี
- 3.7. เขียนฟังก์ชัน `def path(self, data)` พิมพ์ path จาก root ไปยัง node ที่มีข้อมูล data
- 3.8. เขียนฟังก์ชัน `def del(self, data)` เพื่อ delete node ที่มี data ออกจาก BST

หมายเหตุ: ในการสอบ เนื่องจากมีเวลาจำกัด อาจกำหนดให้นักศึกษาไม่ต้องสร้าง class BST มีแต่ class node เท่านั้น ซึ่งทำให้การเขียน code นั้นง่ายและสั้นขึ้นมาก เช่น ข้างล่างเป็นตัวอย่าง function insert() และ inOrder() เมื่อกำหนดเพียง

class node

```
class node:

    def __init__(self, data, left = None, right = None):
        self.data = data
        self.left = None if left is None else left
        self.right = None if right is None else right

#-----No class BST
def insert(r, data):
    if not r:
        return node(data)
    else:
        if data < r.data:
            r.left = insert(r.left, data)
        else:
            r.right = insert(r.right, data)
    return r

def inOrder(r):
    if r:
        inOrder(r.left)
        print(r.data, end = ' ')
        inOrder(r.right)

l = [14,4,9,7,15,3,18,16,20,5,16]
r = None

for ele in l:
    r = insert(r, ele)
inOrder(r)
print()
```