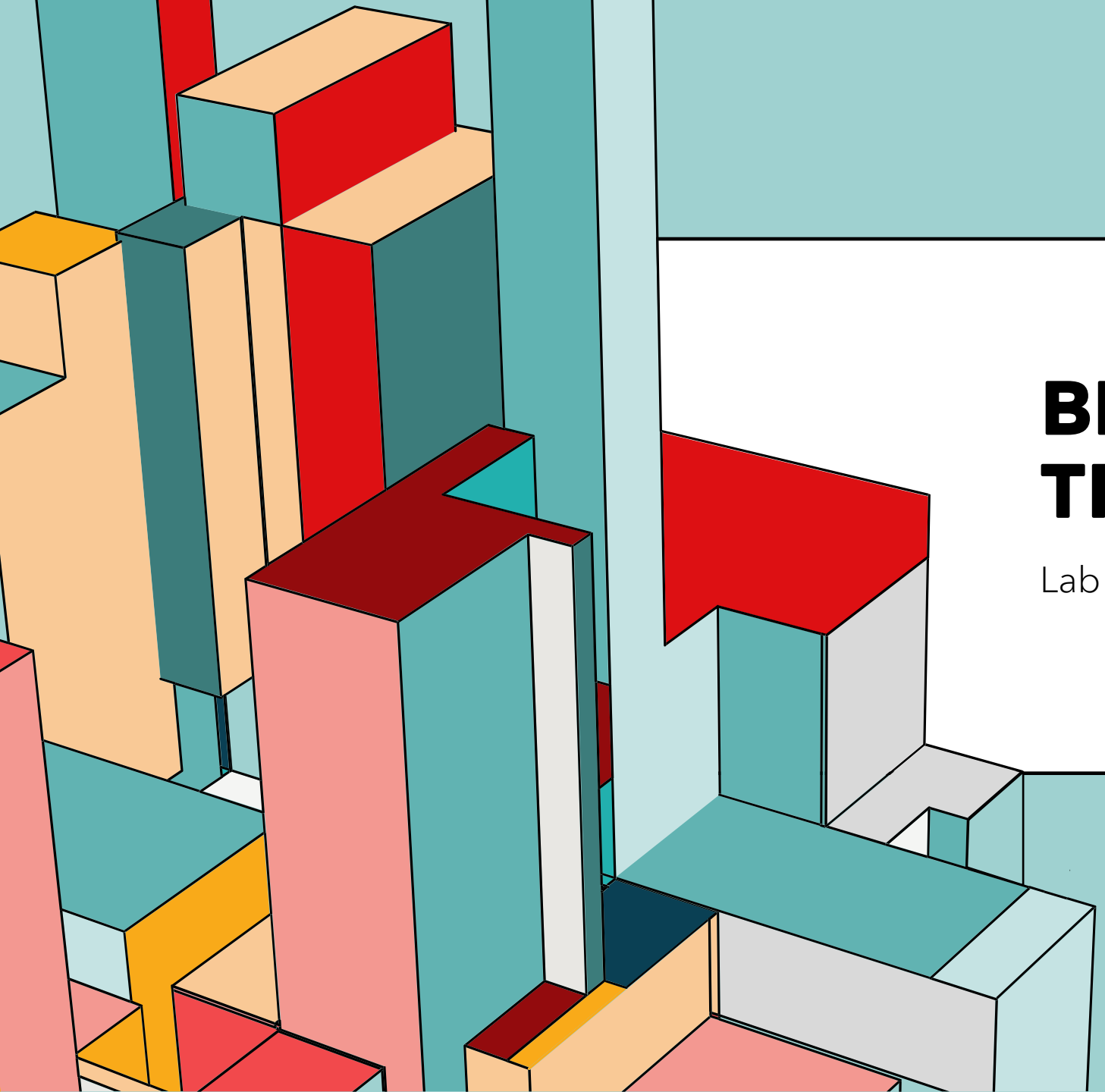


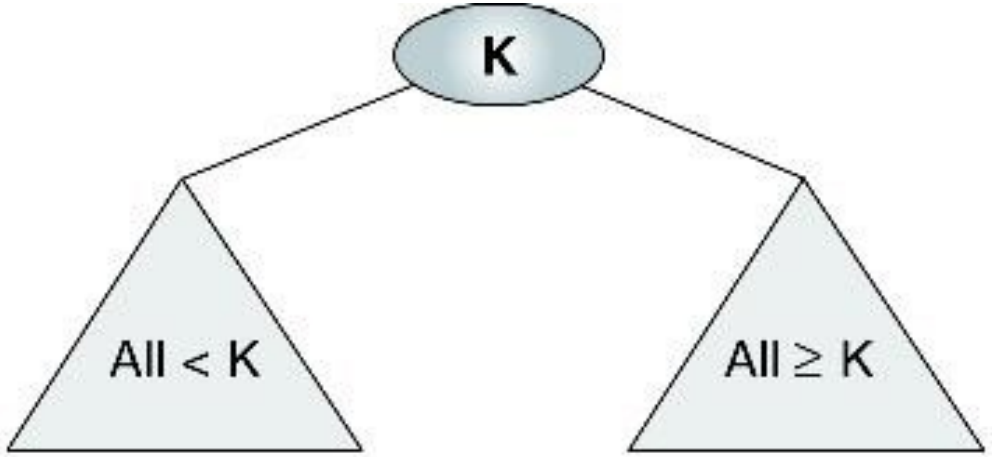
BINARY SEARCH TREE (BST)

Lab Document (Lab 5-6)

BINARY SEARCH TREE

เป็น Binary Tree ประเภทหนึ่ง โดยกำหนดให้

- ทุกโหนดใน left subtree ต้องมีค่าน้อยกว่า root
- ทุกโหนดใน Right Subtree ต้องมีค่ามากกว่าหรือเท่ากับ root
- ในแต่ละ Subtree ต้องคงคุณสมบัติของ Binary Search Tree



LAB 5-6

- จงเขียนโปรแกรมเพื่อสร้าง Binary Search Tree และสามารถทำ Operation ต่างๆ ได้ ดังนี้
  - is_empty() : คืนค่าบูลีน เพื่อแสดงว่า Binary search tree นั้นว่างหรือไม่
    - คืนค่า True ถ้า binary search tree ว่าง ; ถ้าไม่ใช่ ให้คืนค่า False
  - insert(data) : ทำการเพิ่มข้อมูล data เข้าไปใน Binary search tree
  - delete(data) : ทำการลบข้อมูล data ออกจาก Binary search tree
    - คืนค่าข้อมูลที่ถูกลบ ; กรณีที่ไม่สามารถลบข้อมูลได้ ให้คืนค่า None
  - traverse() : ท่องเข้าไปใน Binary search tree และแสดงข้อมูลตามลำดับการ traverse แบบต่างๆ
    - Preorder, Inorder, Postorder
    - มีการเรียกฟังก์ชัน preorder(root), inorder(root) และ postorder(root) ตามลำดับ
  - findMin() : คืนค่าข้อมูลที่มีค่าน้อยที่สุดใน Binary search tree
  - findMax() : คืนค่าข้อมูลที่มีค่ามากที่สุดใน Binary search tree

ตัวอย่างการทดสอบโปรแกรม

myBST = BST()

myBST.insert(14)

myBST.insert(23)

myBST.insert(7)

myBST.insert(10)

myBST.insert(33)

myBST.traverse() // โปรแกรมแสดงข้อมูลการ traverse แบบต่างๆ

- Preorder: -> 14 -> 7 -> 10 -> 23 -> 33
- Inorder: -> 7 -> 10 -> 14 -> 23 -> 33
- Postorder: -> 10 -> 7 -> 33 -> 23 -> 14

ตัวอย่างการทดสอบโปรแกรม (ต่อ)

myBST.delete(14)

myBST.traverse() // โปรแกรมแสดงข้อมูลการ traverse แบบต่างๆ

- Preorder: -> 10 -> 7 -> 23 -> 33
- Inorder: -> 7 -> 10 -> 23 -> 33
- Postorder: -> 7 -> 33 -> 23 -> 10

print("Min: ", myBST.findMin()) // โปรแกรมแสดงข้อความ Min: 7

print("Max: ", myBST.findMax()) // โปรแกรมแสดงข้อความ Max: 33

ตัวอย่างโครงสร้างโปรแกรม

```
Class BSTNode :
    def __init__(self, data) :
        self.data = data
        self.left = None
        self.right = None
```

```
Class BST :
    def __init__(self) :
        self.root = None
    def is_empty(self) :
        // return True if BST is empty
    def insert(self, data) :
        // insert data into BST
    def delete(self, data) :
        // delete data from BST
    def preorder(self, root) :
        // preorder traversal
    def inorder(self, root) :
        // inorder traversal
    def postorder(self, root) :
        // postorder traversal
    def traverse(self) :
        // call preorder(), inorder(), postorder()
    def findMin(self) :
        // return minimum value
    def findMax(self) :
        // return maximum value
```

PREORDER TRAVERSAL

```
def preorder(self, root):
    if (root != None):
        print("->", root.data, end=" ")
        self.preorder(root.left)
        self.preorder(root.right)
```

INORDER TRAVERSAL

```
def inorder(self, root):
    if (root != None):
        self.inorder(root.left)
        print("->", root.data, end=" ")
        self.inorder(root.right)
```

POSTORDER TRAVERSAL

```
def postorder(self, root):
    if (root != None):
        self.postorder(root.left)
        self.postorder(root.right)
        print("->", root.data, end=" ")
```

INSERT

```
def insert(self, data):
    pNew = BSTNode(data)
    // ถ้าเป็น Empty tree แล้ว กำหนด pNew เป็น root node
    // ถ้าไม่ใช่ ก็วนลูปเริ่มจาก root วิ่งไปตามกิ่งต้นไม้ (อาจใช้ pointer prev, start ช่วย)
            data < root data : วิ่งไปกิ่งซ้าย
            ถ้าไม่ใช่ : วิ่งไปกิ่งขวา
    // วิ่งไปเรื่อยๆ จนไปต่อไม่ได้
    // ทำการเชื่อม pNew เข้ากับต้นไม้
```

DELETE

def delete(self, data):

// ถ้าเป็น Empty tree แล้ว ไม่สามารถลบ data ได้

// ถ้าไม่ใช่ ก็วนลูปเริ่มจาก root วิ่งไปตามกิ่งต้นไม้ (อาจใช้ pointer prev, start ช่วย)

data < root data : วิ่งไปกิ่งซ้าย

ถ้าไม่ใช่ : วิ่งไปกิ่งขวา

// วิ่งไปเรื่อยๆ จน start ชี้ที่โหนดที่มีข้อมูลเท่ากับ data

// ทำการลบโหนดที่ start ชี้อยู่ ขึ้นอยู่กับกรณีต่างๆ

- กรณีโหนดที่ลบ เป็น leaf node
- กรณีโหนดที่ลบ มีลูกฝั่งซ้ายหรือขวา อย่างใดอย่างหนึ่งเท่านั้น
- กรณีโหนดที่ลบ มีลูกทั้ง 2 ฝั่ง