โครงสร้างซอร์ฟแวร์แก้ไขปัญหา hilbert hotel

จัดทำโดย
กลุ่ม Hyper Hotel (4D)

ธีรวัฒน์ ลาภพานิช	67010448
เนติธร สุธรรมวราพร	67010497
ปรเมศวร์ โรจน์หล่อสกุล	67010534
ธีภพ มหาสุข	67011464

รายงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาในรายวิชา
01076109 OBJECT ORIENTED DATA STRUCTURES และ 01076110 OBJECT ORIENTED DATA STRUCTURES PROJECT
สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ ปีการศึกษา 2568

บทนำและทฤษฎีโรงแรม

รายงานนี้ใช้เพื่ออธิบายแนวคิดทฤษฎี hibert hotel และพัฒนาซอฟแวร์เพื่อแก้ไขปัญหาแนวคิดนี้

Hilbert's Hotel เป็นแนวคิดของ David Hilbert เพื่อใช้อธิบายลักษณะของจำนวนที่มีค่าเป็นอนันต์ (infinity) โดยยกตัวอย่างโรงแรมที่มีจำนวนห้องเป็นอนันต์ และทุกห้องมีคนพักทุกห้องจนเต็ม หากมีแขกใหม่เข้ามา ก็สามารถย้ายแขกเดิมไปห้องอื่นได้เนื่องจากมีห้องเป็นอนันต์ เจ้าของโรงแรมต้องหาวิธีที่ในการจัดแขกให้แขกเดิม และแขกใหม่ทุกคนมีห้องพักโดยที่หมายเลขห้องไม่ซ้ำกัน

กรณีมีแขกใหม่จำนวน n คน จะย้ายแขกเดิมไปที่ ห้องเดิม + n เพื่อให้มีห้องว่าง n ห้องสำหรับแขกใหม่

กรณีมาเป็นคันรถจำนวน m คัน และในแต่ละคันมี n คน เจ้าของโรงแรมต้องสามารถจัดย้ายหมายเลขห้องแขกเดิมไปที่ห้องใหม่ เพื่อให้แขกที่เข้าใหม่มีห้องพัก

กรณีมีแขกจำนวนอนันต์เข้ามาพัก ก็สามารถย้ายแขกเดิมไปที่ห้องที่มีหมายเลขเป็น 2 เท่าของห้องเดิมได้ เพื่อทำให้ห้องเลขที่คี่ว่างและสามารถรองรับแขกจำนวนอนันต์ได้

การจัดการแขกในโรงแรม

ในโครงงานนี้จะให้ AVL Tree เพื่อการจัดเก็บข้อมูลที่มีประสิทธิภาพและทำงานได้เร็วในการ Search หาข้อมูลจำนวนมาก

ในส่วนของการจัดการแขกจะมีแขก 3 แบบดังนี้

1.แขกที่พักอยู่ในโรงแรม

2.แขกที่ Check-in ปกติ (เดิน,คนในรถ,รถบนเรือ,เรือ)

3.แขกที่ Check-in แบบ Manual

Data structure ที่ใช้

AVL Tree

โครงสร้างข้อมูลแบบ AVL Tree เป็นโครงสร้างข้อมูลที่มีการปรับสมดุลอัตโนมัติ (Self-Balancing) หลังการเพิ่มหรือลบข้อมูลทุกครั้ง โดยจะทำให้ความสูงของต้นไม้แตกต่างกันระหว่างด้านซ้ายและขวาไม่เกิน 1 ทำให้การค้นหา เพิ่ม และลบข้อมูลมีประสิทธิภาพอยู่ในระดับ O(log n) เสมอ

โครงสร้างของ AVL Tree

- Root : โหนดรากของต้นไม้ เป็นจุดเริ่มต้นของทุกการดำเนินการ ค้นหา แทรก ลบ
- getHeight() : ความสูงของโหนด ใช้คำนวณสมดุลของต้นไม้
- getBalance() : ค่าผลต่างระหว่างความสูงของ subtree ซ้ายและขวา
  ถ้าค่า Balance Factor อยู่นอกช่วง -1 ถึง 1 จะต้องมีการหมุนเพื่อปรับสมดุล

หลักการทำงานของ AVL Tree

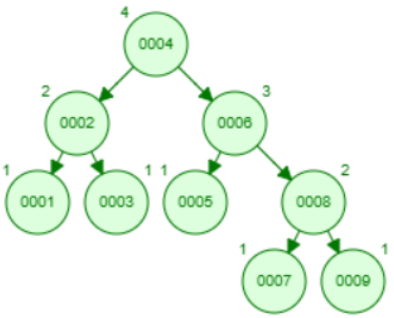
เมื่อมีการเพิ่มหรือลบโหนด จะมีการคำนวณค่า Balance Factor ย้อนกลับขึ้นไปจากโหนดที่เปลี่ยนแปลงจนถึงราก และทำการหมุนหากพบว่าไม่สมดุล โดยมีรูปแบบการหมุน 4 แบบ

1. LL Rotation
2. RR Rotation
3. LR Rotation
4. RL Rotation

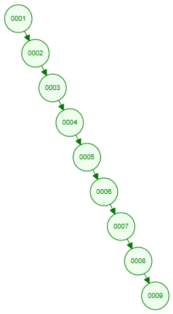
เหตุผลที่เลือกใช้ AVL Tree

ตัว AVL Tree จะทำ balance โหนดหากไม่สมดุล ทำให้ความลึกของ Leaf Node มีค่าใกล้เคียงกันเสมอ ทำให้ความเร็วในการค้นหาโหนดดีกว่า Tree แบบปกติที่ไม่มีการ balance ดังตัวอย่างได้ทำการเพิ่มเลข 1 - 9 เข้าไปใน AVL และ binary tree

ภาพ AVL tree



ภาพ binary tree



การออกแบบโปรแกรม

หน้าต่างเริ่มต้น

หลังจากที่เปิดโปรแกรมมา ผู้ใช้จะเจอกับตัวเลือก 7 ตัวเลือก ได้แก่

1 : Check-in guests by channel

2 : Manually check-in guests

3 : Manually check-out guests

4 : Search room

5 : Export to CSV file

6 : Print Available Room

7 : Quit App

โดยให้ผู้ใช้ใส่ตัวเลือกเป็นตัวเลข 1 - 7 หากกรอกอย่างอื่นที่ไม่ใช่ตัวเลข จะแสดงคำว่า Invalid Input

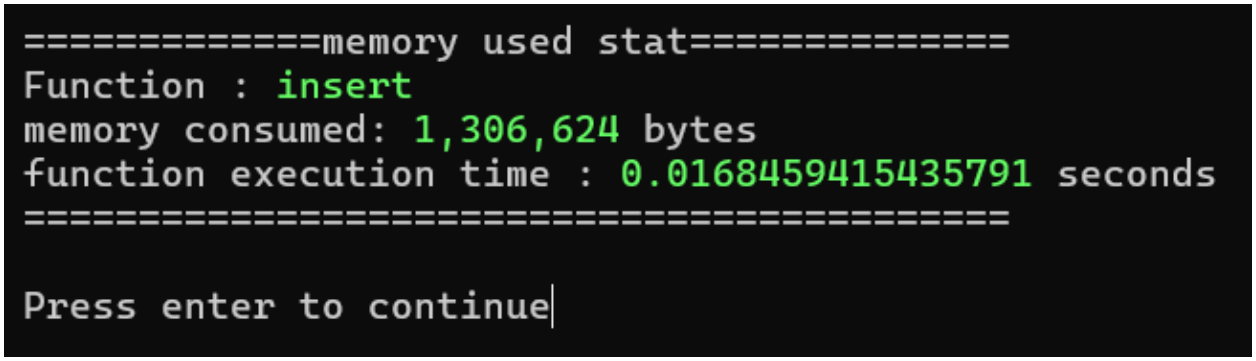
ผู้ใช้สามารถกด enter เพื่อกลับไปหน้าหลัก

โดยตัวเลือกที่แสดงผลในแต่ละตัวเลือก จะอธิบายในส่วนถัดๆไป

1.รับแขกเข้ามาจากแต่ละช่องทาง (Check-in guests by channel)

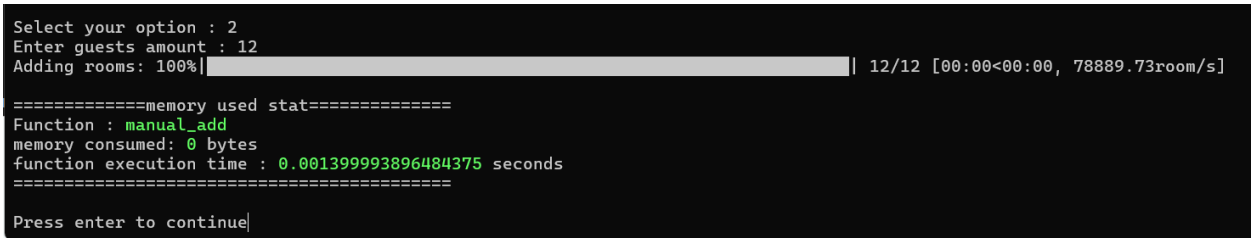
เมื่อเลือกเมนูนี้ผู้ใช้จะต้องกรอกแขกซึ่งเป็นทัวร์ขนาดใหญ่ ที่เข้ามาในแต่ละช่องทาง โดยจะให้มีเรือ x ลำ ในแต่ละลำมีรถ y คัน และในแต่ละคันมีแขก z คน และอาจมีแขกที่เกิน และแยกมาอีก เมื่อกรอกครบจะแสดงหลอดความคืบหน้า (progress bar) แสดงการความคืบหน้าย้ายแขกเข้าโรงแรม (AVL Tree)

เมื่อหลอดความคืบหน้า (progress bar) เต็มแล้วจะแสดงผลลัพธ์การทำงานของ function ทั้ง memory ที่ใช้ และเวลาที่ใช้ในการทำงาน โดยจำนวนคนที่เข้ามาจะแปรผันตรงกับเวลาที่ใช้ในการทำงานของฟังก์ชันนี้



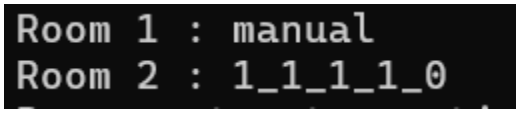
2. เพิ่มหมายเลขห้องแบบ manual (Manually check-in guests)

เมื่อเลือกเมนูนี้ ผู้ใช้จะต้องกรอกจำนวนหมายเลขที่ต้องการจะเพิ่ม จากนั้นตัวโปรแกรมจะเพิ่มห้องใหม่เข้าไปในโรงแรม (AVL Tree) พร้อมทั้งแสดงผลลัพธ์การทำงานของ function ทั้ง memory ที่ใช้ และเวลาที่ใช้ในการทำงาน โดยค่าของเลขที่รับเข้ามาจะแปรผันตรงกับเวลาที่ใช้ในการทำงานของฟังก์ชันนี้



ข้อมูลของแขก

โปรแกรมที่พวกเราออกแบบจะจำลองการแก้ปัญหา โดยแขกที่ได้รับมาจะมีอยู่ 2 แบบ คือ แขกที่เข้ามาในแต่ละช่องทาง ดังตัวอย่าง



ในห้องแรกเกิดจากการเพิ่มคนโดยใช้ตัวเลือกที่ 2 เพิ่มหมายเลขห้องแบบ manual (Manually check-in guests)

ในห้องที่ 2 เกิดจากการเพิ่มคนโดยใช้ตัวเลือกที่หนึ่งรับแขกเข้ามาจากแต่ละช่องทาง (Check-in guests by channel) โดยตัวเลขจะถูกคั่นด้วย _ ซึ่งแต่ละตัวมีความหมายดังนี้

เลขตัวที่ 1 : ลำดับของกลุ่มที่แขกกลุ่มนี้เข้าพัก

เลขตัวที่ 2 : ช่องทาง (ลำดับของคนบนรถ)

เลขตัวที่ 3 : ช่องทาง (ลำดับของรถบนเรือ)

เลขตัวที่ 4 : ช่องทาง (ลำดับของเรือ)

เลขตัวที่ 5 : ช่องทาง (แขกที่เดินทางมาเอง)

3. ลบหมายเลขห้องแบบ manual (Manually check-out guests)

เมื่อเลือกเมนูนี้ ผู้ใช้จะต้องกรอกหมายเลขห้องที่ต้องการจะลบ จากนั้นตัวโปรแกรมจะลบห้องในโรงแรม (AVL Tree) หากไม่เจอห้องนั้น จะแสดงผล Room not found พร้อมทั้งแสดงผลลัพธ์การทำงานของ function ทั้ง memory ที่ใช้ และเวลาที่ใช้ในการทำงาน

```
Select your option : 3
Enter room number : 1
Remove 1

=============memory used stat==============
Function : manual_remove
memory consumed: 0 bytes
function execution time : 0.00044536590576171875 seconds
===========================================

Press enter to continue
```

4.ค้นหาห้อง (Search room)

เมื่อเลือกเมนูนี้ ผู้ใช้จะต้องกรอกหมายเลขห้องของแขกที่ต้องการจะค้นหา จากนั้นตัวโปรแกรมจะแสดงผลแขกพร้อช่องทางที่แขกคนนั้นเข้ามา (AVL Tree) หากไม่เจอห้องนั้น จะแสดงผล Room not found พร้อมทั้งแสดงผลลัพธ์การทำงานของ function ทั้ง memory ที่ใช้ และเวลาที่ใช้ในการทำงาน

```
Select your option : 4
Enter room number : 2

Room 2 : 2_1_2_2_0

=============memory used stat==============
Function : search
memory consumed: 0 bytes
function execution time : 0.00061440467834472 66 seconds
===========================================

Press enter to continue
```

5. บันทึกข้อมูลเป็นไฟล์ (Export to CSV file)

เมื่อเลือกเมนูนี้ โปรแกรมจะไปดึงข้อมูลของห้องที่มีแขกทั้งหมดในโรงแรม (AVL Tree) มาแปลงเป็น List แล้วเขียนแต่ละห้องเป็นบรรทัดในไฟล์ hotel.csv

```
Select your option : 5

Export Success

============memory used stat==============
Function : export_to_file
memory consumed: 8,192 bytes
function execution time : 0.00194787979125976 seconds
==========================================

Press enter to continue
```

ตัวอย่างข้อมูลในไฟล์

hotel.csv

```
Room 1 : 2_2_2_2_0
Room 2 : 2_1_2_2_0
Room 3 : 2_2_1_2_0
Room 4 : 2_1_1_2_0
Room 5 : 2_2_2_1_0
Room 6 : 2_1_2_1_0
Room 7 : 2_2_1_1_0
Room 8 : 2_1_1_1_0
Room 9 : 2_0_0_0_2
Room 10 : 2_0_0_0_1
Room 11 : manual
Room 12 : manual
Room 13 : manual
Room 14 : manual
Room 15 : manual
Room 16 : 1_2_2_2_0
Room 17 : 1_1_2_2_0
Room 18 : 1_2_1_2_0
Room 19 : 1_1_1_2_0
Room 20 : 1_2_2_1_0
Room 21 : 1_1_2_1_0
Room 22 : 1_2_1_1_0
Room 23 : 1_1_1_1_0
Room 24 : 1_0_0_0_2
Room 25 : 1_0_0_0_1
```

6.แสดงแขกทั้งหมดในโรงแรม

เมื่อเลือกเมนูนี้ โปรแกรมจะไปดึงข้อมูลของห้องที่มีแขกทั้งหมดในโรงแรม (AVL Tree) มาแปลงเป็น List แล้วแสดงผลบน terminal

```
Select your option : 6
Room 1 : 2_2_2_2_0
Room 2 : 2_1_2_2_0
Room 3 : 2_2_1_2_0
Room 4 : 2_1_1_2_0
Room 5 : 2_2_2_1_0
Room 6 : 2_1_2_1_0
Room 7 : 2_2_1_1_0
Room 8 : 2_1_1_1_0
Room 9 : 2_0_0_0_2
Room 10 : 2_0_0_0_1
Room 11 : manual
Room 12 : manual
Room 13 : manual
Room 14 : manual
Room 15 : manual
Room 16 : 1_2_2_2_0
Room 17 : 1_1_2_2_0
Room 18 : 1_2_1_2_0
Room 19 : 1_1_1_2_0
Room 20 : 1_2_2_1_0
Room 21 : 1_1_2_1_0
Room 22 : 1_2_1_1_0
Room 23 : 1_1_1_1_0
Room 24 : 1_0_0_0_2
Room 25 : 1_0_0_0_1
Press enter to continue
```

7. Quit App

เมื่อเลือกเมนูนี้ ผู้ใช้จะสามารถออกจากโปรแกรมนี้ได้ เมื่อกด enter ตัวโปรแกรมจะปิดทันที ผู้ใช้สามารถสามารถออกจากโปรแกรมได้อีกวิธีด้วยการกดคีย์ลัดปุ่ม Ctrl+C

การวิเคราะห์ Big O ของแต่ละฟังก์ชัน

1.การเพิ่มหมายเลขห้องแบบ manual

การเพิ่มห้องในโรงแรมนั้นจะเรียกใช้ผ่านฟังก์ชัน manual_add() ตามรูปด้านล่าง

```
#For requirement 4
@track
def manual_add(self, count):
    with tqdm(total=count, desc="Adding rooms", unit="room") as pbar:
        if self.last_room > 0:
            self.tree.update(count)
        for i in range(1, count + 1):
            self.tree.add(Room(f"manual", i))
            pbar.update(1)
    return
```

Function manual_add() มี parameter รับค่า count เป็นจำนวนห้องที่ต้องการจะเพิ่ม ในส่วนของบรรทัด with tqdm() และ pbar เป็นคำสั่งไว้แสดง progress bar ความคืบหน้าของการเพิ่มห้องจากทั้งหมด ซึ่งไม่เกี่ยวข้องกับการทำงานของโปรแกรม จะขอไม่อธิบายคำสั่งนี้พร้อมกับไม่นำไปวิเคราะห์ร่วมกับค่า Big O ในโปรแกรม และในส่วนของ function update() ห้องจะถูกนำไปขยายความในหัวข้อ 3.การจัดเรียงลำดับหมายเลขห้อง ซึ่ง Big O ของ function นี้จะมีค่าเป็น **O(n)** โดยที่ n คือจำนวน node ทั้งหมดใน tree

ในส่วนของการ add node ใน tree จะมี for loop ทำงานทั้งหมดตาม count หรือ ตามจำนวนห้อง จะได้เป็นค่า **O(count) * O(add)** โดยที่ O(add) จะสามารถหาค่าได้ตามรูปด้านล่าง

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

def _add(self, root, data):
    if root is None:
        return AVLNode(data)
    if int(data) < int(root.data):
        root.left = self._add(root.left, data)
    else:
        root.right = self._add(root.right, data)
    root = self.rebalance(root)
    return root
```

Function add() จะรับ parameter ในชื่อ data ซึ่งเป็น class Room และส่งต่อไปให้ function _add() ที่รับ parameter เป็น root ซึ่งเป็น node และ data ที่เป็น Room โดยเริ่มจากการเช็คว่าที่ root ว่างหรือไม่ ถ้าใช่จะทำการส่งและสร้าง node ใหม่ แล้วต่อมาจะทำ search หา node ที่ data นั้นควรอยู่ด้วยการท่อง tree ไปตาม node ต่างๆโดยใช้วิธีเปรียบเทียบหมายเลขห้องมากกว่าหรือน้อยกว่า ซึ่งจะได้ค่า Big O จากการค้นหานี้เป็น **O(log n)**

ในขั้นตอนสุดท้าย tree จะทำการ rebalance ตัวเองผ่าน function rebalance() หลังจากการ add node เสร็จสิ้น โดยปกติการหมุน tree จะมีค่าเป็น **O(1)** แต่เนื่องจากการ rebalance จะถูกเรียกเป็น stack เริ่มจากล่างสุด (ที่ไม่ใช่ node ที่พึ่ง add) ไปจนถึงบนสุด (root / top node) ซึ่งจากการเรียกใช้นี้จะได้ค่า Big O จากการ rebalance เป็น **O(log n)**

จากทั้งหมดใน function นี้จะสามารถวิเคราะห์ค่า Big O ของ function manual_add() เป็น O(count) * (O(log n) + O(log n)) + O(n) = **O(count*log n)**

2.การลบหมายเลขห้องแบบ manual

การลบห้องในโรงแรมนั้นจะเรียกใช้ผ่านฟังก์ชัน manual_remove() ตามรูปด้านล่าง

```python
#For requirement 5
@track
def manual_remove(self, room_number):
    result = self.tree.search(room_number)
    if not result:
        print("Room not found")
        return
    self.tree.remove(int(room_number))
    print(f"Remove Room {room_number}")
    return
```

Function manual_remove() มี parameter รับค่า room_number เป็นหมายเลขห้องที่ต้องการจะ remove ห้องออก ซึ่งในส่วนของ function search() ห้องจะถูกนำไปขยายความในหัวข้อ 4.การค้นหาหมายเลขห้อง ซึ่ง Big O ของ function นี้จะมีค่าเป็น **O(log n)** โดยที่ n คือจำนวน node ทั้งหมดใน tree

หลังจากการค้นหาห้องใน Hotel สำเร็จจะทำการเช็คว่าเจอห้องที่หาไม่ ถ้าไม่เจอก็ print ข้อความไม่เจอ พร้อมกับ return เพื่อจบการทำงานของโปรแกรม ในกรณีที่เจอจะทำการเรียกใช้ function remove() โดยการส่ง parameter เป็นเลขห้องไป แล้ว print แสดงผลว่าทำการ remove room เรียบร้อยแล้ว พร้อมกับ return เพื่อจบการทำงานของ function

```python
def remove(self, data):
    self.root = self._remove(self.root, data)

def _remove(self, node, data):
    if node is None:
        return None
    if data < node.data:
        node.left = self._remove(node.left, data)
    elif data > node.data:
        node.right = self._remove(node.right, data)
    else:
        if node.left is None:
            return node.right
        elif node.right is None:
            return node.left
        temp = self.get_successer(node.right)
        node.data = temp.data
        node.right = self._remove(node.right, temp.data)
    return self.rebalance(node)
```

Function remove() จะเรียกใช้ function _remove() ที่รับ parameter เป็น node และ data เป็นหมายเลขห้องที่ต้องการจะลบ ในส่วนแรกจะทำการค้นหา node ใน tree ที่ต้องการจะลบโดยใช้การเปรียบเทียบหมายเลขห้องเพื่อที่จะท่อง tree ไปตาม node ต่างๆจนเจอ node ตัวที่ต้องการจะลบ ซึ่งสามารถคำนวณค่า Big O ออกมาได้เป็น **O(log n)**

ส่วนถัดมาจะทำการเช็คว่า node ตัวนั้น มีลูกไหม ถ้าไม่มีลูกหรือมีลูกหนึ่ง จะให้ node ลูกขึ้นมาแทนตัวมัน หรือ remove เลย (กลายเป็น None) ในกรณีที่ไม่มีลูก แต่ถ้ามีลูกถึงสองฝั่งจะให้ทำการหา successer คือ หา leaf node ที่มีค่าน้อยที่สุดของลูกฝั่งที่มากกว่า node นั้น เพื่อนำ data ของ leaf node นั้นมาแทนที่ node ที่เราต้องการจะลบ แล้วทำการเปลี่ยนไปลบ successer หรือ leaf node นั้นทิ้ง ซึ่งสามารถคำนวณ Big O การหา successer ได้เป็น **O(log n)** ในสุดท้ายจะทำการ rebalance tree เพื่อให้ tree ยังคง balance ซึ่งมีค่า Big O เป็น **O(log n)**

จากทั้งหมดใน function นี้จะสามารถวิเคราะห์ค่า Big O ของ function manual_remove() เป็น O(log n) + O(log n) + O(log n) + O(log n) = **O(log n)**

3.การจัดเรียงลำดับหมายเลขห้อง

การจัดเรียงเลขห้องในโรงแรมนั้นจะเรียกใช้ผ่านฟังก์ชัน update() ตามรูปด้านล่าง

```
def update(self, value):
    self._update(self.root, value)

def _update(self, focus, value):
    if focus is None:
        return None
    if focus.left is not None:
        self._update(focus.left, value)
    focus.data.number += value
    if focus.right is not None:
        self._update(focus.right, value)
```

Function update() มี parameter เป็น value ซึ่งเป็น int หมายถึงค่าเลขห้องที่จะอัปเดตเพิ่มขึ้นไปในแต่ละ Room โดยจะอัปเดตทุกห้องใน Hotel การทำงานใน function จะส่งต่อไปให้ functioin _update() ที่รับ parameter เป็น focus ซึ่งเป็น node และ value ที่เป็น int เริ่มจากการเช็คว่าที่ focus ว่างหรือไม่ ถ้าว่างให้ return เพื่อจบการทำงาน ในส่วนถัดไปจะทำการเช็คว่า node นั้นมีลูกทางไหนบ้าง ถ้ามีลูกจะเรียกใช้ฟังก์ชันนี้กับลูกเช่นเดียวกับ node แล้วทำการเปลี่ยนค่าหมายเลขห้องของ room ที่ node นั้นเก็บ ซึ่งจะได้ค่า Big O จากการอัปเดตค่านี้เป็น **O(n)**

4.การค้นหาหมายเลขห้อง

การค้นหาห้องในโรงแรมด้วยหมายเลขห้องนั้นจะเรียกใช้ผ่านฟังก์ชัน search() ตามรูปด้านล่าง

```
#For requirement 7
@track
def search(self, room_number):
    result = self.tree.search(room_number)
    if result:
        op = f"{result}"
    else:
        op = "Room Not found"
    print(f'\n{op}')
```

```
def search(self, data):
    return self._search(self.root, data)

def _search(self, node, data):
    if node == None:
        return None

    if node.data == data:
        return node
    if node.data > data:
        return self._search(node.left, data)
    else:
        return self._search(node.right, data)
```

Function search() จะรับ parameter data เป็นเลขห้องแล้วเรียกใช้ _search() ซึ่งจะทำการ Search ใน AVL Tree แบบ Recursive และเนื่องจากการเก็บข้อมูลใน AVL Tree มีการ rebalance node ทำให้ Big O มีค่าเป็น **O(log n)**

ภาคผนวก

Source code : https://github.com/PluemDontKnowToCode/MaanMaiRood

Hibert Hotel: https://en.wikipedia.org/wiki/Hilbert%27s_paradox_of_the_Grand_Hotel
https://www.youtube.com/watch?v=HLTjDXT9SqQ