**รายงานโครงงาน**

**Hilsbert’s Hotel**

**เสนอ**

**ผศ. เกียรติณรงค์ ทองประเสริฐ**

**โดย**

**66010882 สุวิจักขณ์ หยกพิทักษ์โชค**

**66010873 สุภเวช เสพศิริสุข**

**66011463 รัตนพล มาสูตร**

**65010801 ภวัต ประกายทิพย์**

**65010983 วัศพล สัมฤทธิ์**

**โครงงานนี้เป็นส่วนหนึ่งของการศึกษา  
วิชา OBJECT ORIENTED DATA STRUCTURES**

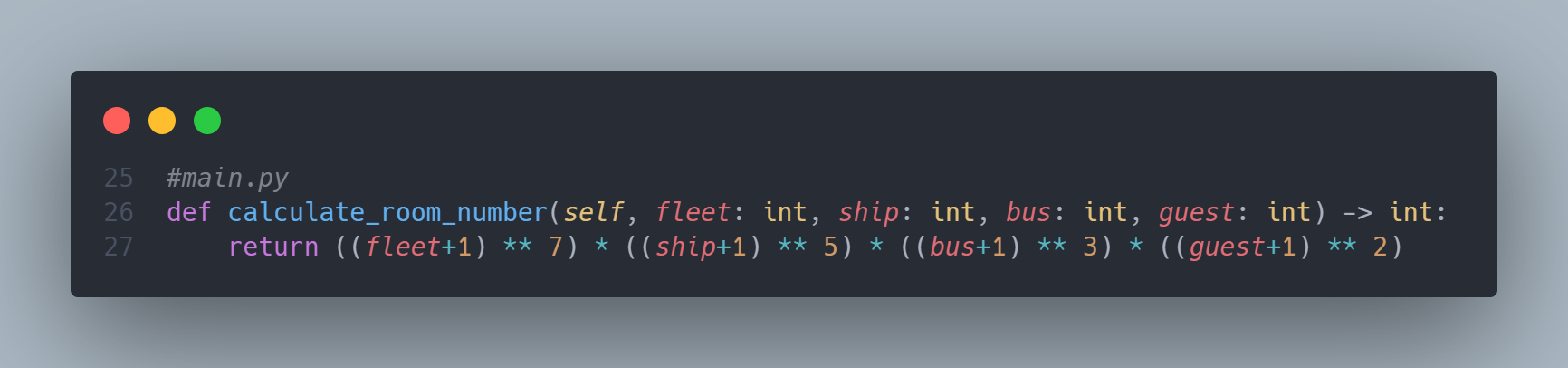
**ปีการศึกษา 2567**

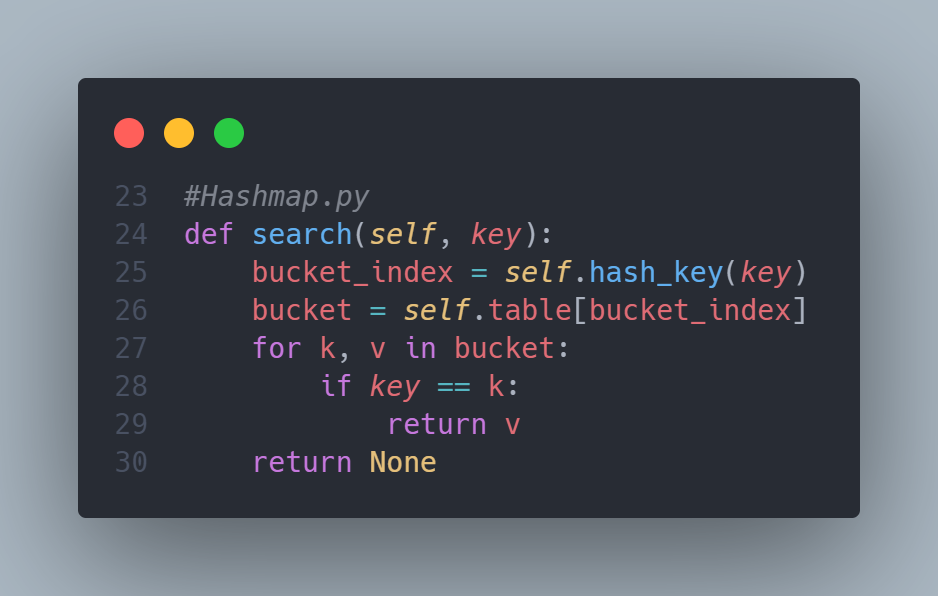
**Function exec\_time**

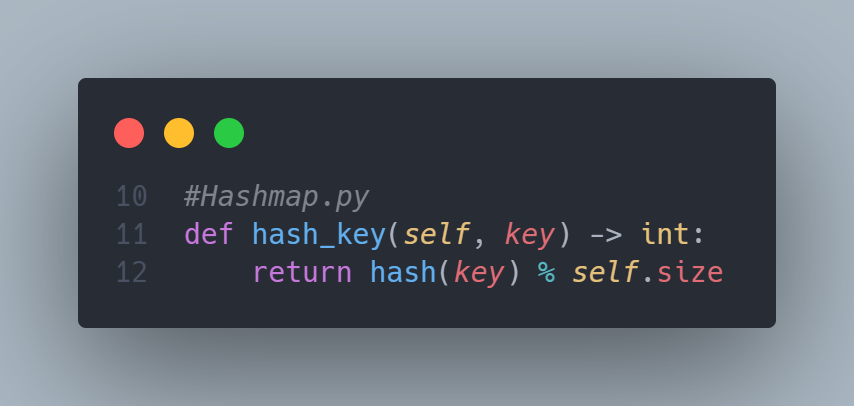
exec\_time เป็น decorator function ที่ใช้ในการนับเวลาที่ใช้ในแต่ละ function function นี้จะรับค่าเป็น function และ return ค่าเป็น function ที่เพิ่มการนับเวลาเริ่มก่อนเรียก function ที่ pass เข้ามา และหยุดเวลาเมื่อ function นั้นเสร็จสิ้นการทำงานและจำแสดงผลเวลาเป็นเลขทศนิยมสี่จุดทาง terminal

**สรุป Big O = O(1)**

**Function add\_room**

Method ในการเพิ่มห้องจากการนำข้อมูลช่องทางในการมาของแขกในการคำนวณเป็นเลขห้องโดยการเรียกใช้

1. room\_number = self.calculate\_room\_number(fleet, ship, bus, guest) O(1)
2. self.hash\_table.search(room\_number)
   1. bucket\_index = self.hash\_key(key) O(1)

นำ key ไปคำนวนหา index ใน hash table โดยการนำ key modulus ด้วย ขนาดของ hash table

* 1. bucket = self.table[bucket\_index] O(1)
  2. for k, v in bucket: O(n)

วนลูปหา key ใน list ของ index นั้น ถ้าเจอจะ return ข้อมูลของคนที่อยู่ในห้องออกมา

* 1. return None กรณีที่ไม่เจอห้อง

1. self.hash\_table.insert(room\_number, (fleet, ship, bus, guest))

A computer code with colorful text

Description automatically generated with medium confidence

* 1. bucket\_index = self.hash\_key(key) O(1)

นำ key ไปคำนวนหา index ใน hash table โดยการนำ key mod ด้วย ขนาดของ hash table

* 1. bucket = self.table[bucket\_index] O(1)
  2. bucket.append((key, value)) O(1)

เพิ่มห้องและข้อมูลคนอยู่ต่อท้าย bucket index นั้นๆ

1. ****กรณีพบว่าห้องที่จะเพิ่มมีอยู่แล้วเพิ่มทับไม่ได้ จะใช้ quadratic probing ในการหาเลขห้องที่  
   สามารถเพิ่มได้
   1. while self.hash\_table.search(room\_number) is not None ขนาดของ bucket \* O(n)
   2. self.hash\_table.insert(room\_number, (fleet, ship, bus, guest)) O(n)
2. การเพิ่มเลขห้องไปเก็บใน avlTree
   1. การแทรกข้อมูล: O(logn)

การแทรกข้อมูลใน AVL Tree เป็นการแทรกในลักษณะของ Binary Search Tree (BST)

* 1. การอัปเดตความสูงและตรวจสอบความสมดุล: O(logn)

หลังจากแทรกข้อมูลสำเร็จแล้ว ทุกโหนดจากตำแหน่งที่แทรกกลับขึ้นไปยังรากของต้นไม้จะต้องมีการ

อัปเดตค่าความสูง (height) ซึ่งทำได้ในเวลาO(1) สำหรับแต่ละโหนด และเนื่องจากโหนดทั้งหมดที่ต้องอัปเดตอยู่ในเส้นทางจากโหนดที่แทรกถึงราก ความซับซ้อนของขั้นตอนนี้จึงเป็น

* 1. การหมุน: O(logn)

ในกรณีแย่ที่สุด การแทรกข้อมูลอาจทำให้เกิดการหมุนมากที่สุดหนึ่งครั้งต่อระดับของต้นไม้ นั่นคือไม่เกิน

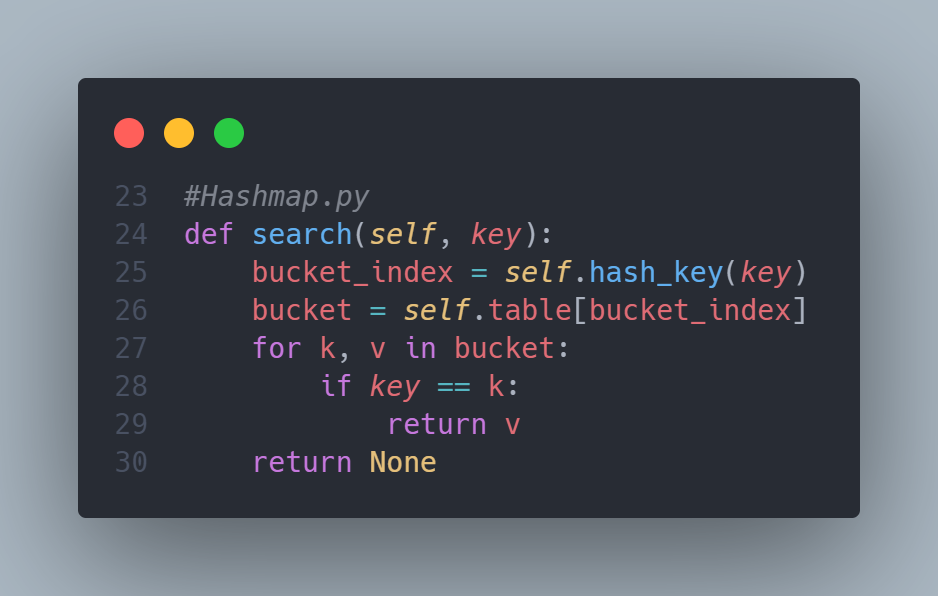
**สรุปการเพิ่มเลขห้องของ avlTree มี complexity เป็น O(logn)**

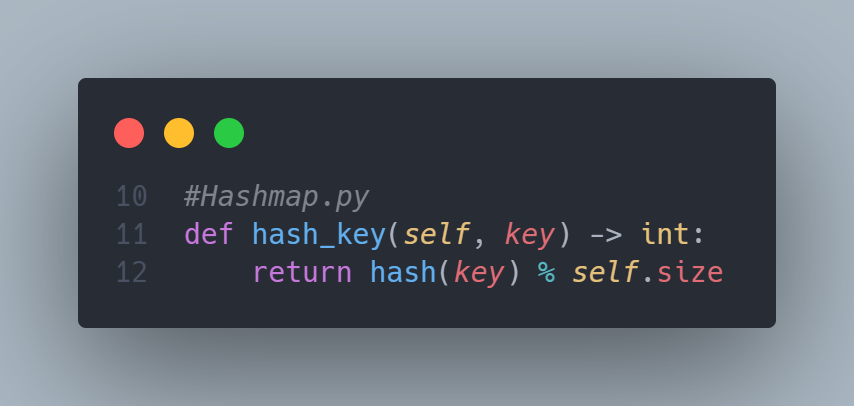
โดยรวมการเพิ่มห้องด้วย methode add\_room ของ hotel ที่มีการใช้ hash แบบ open hashing ร่วมกับ avlTree มี complexity มากสุดที่ O(n)

**Function remove\_room**

**A computer screen shot of text

Description automatically generated**

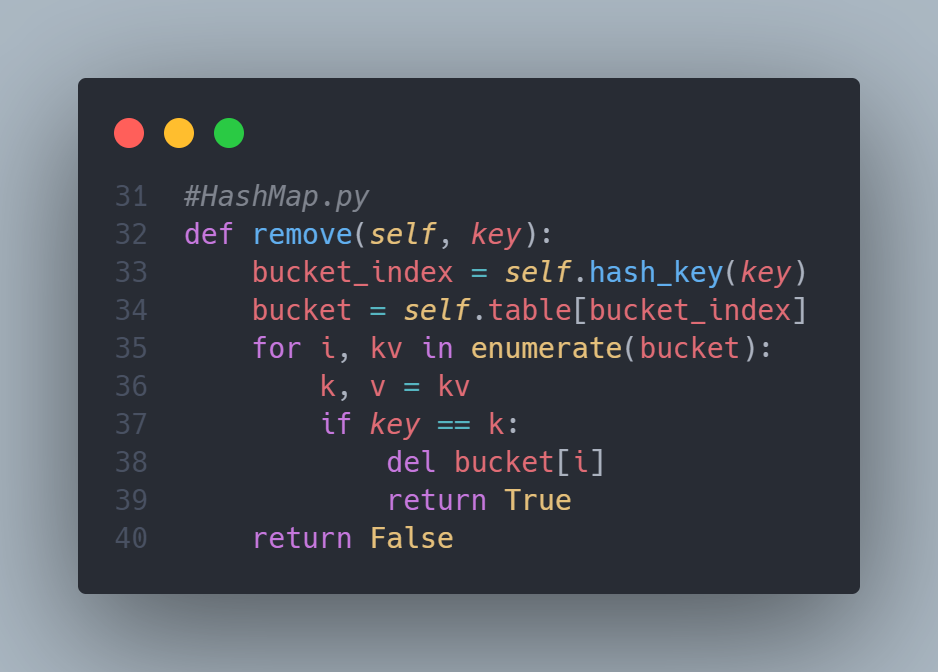
1. if self.hash\_table.search(room\_number):
   1. bucket\_index = self.hash\_key(key) O(1)

นำ key ไปคำนวนหา index ใน hash table โดยการนำ key modulus ด้วย ขนาดของ hash table

* 1. bucket = self.table[bucket\_index] O(1)
  2. for k, v in bucket: O(n)

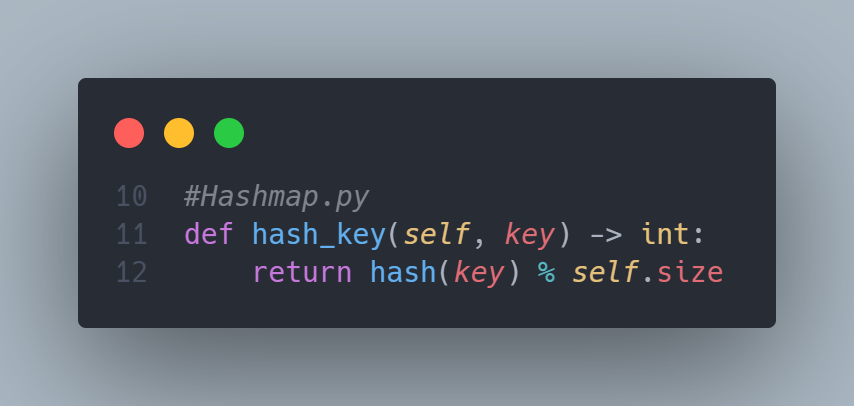
วนลูปหา key ใน list ของ index นั้น ถ้าเจอจะ return ข้อมูลของคนที่อยู่ในห้องออกมา

* 1. return None กรณีที่ไม่เจอห้อง

1. self.hash\_table.remove(room\_number)
   1. bucket\_index = self.hash\_key(key) O(1)

คำนวนหา index ใน hash table โดยการนำ key mod ด้วย ขนาดของ hash table

* + 1. Bucket\_index = self.hash\_key(key) O(1)

นำ key ไปคำนวนหา index ใน hash table โดยการนำ key mod ด้วย ขนาดของ hash table

* 1. bucket = self.table[bucket\_index] O(1)
  2. for i, kv in enumerate(bucket): O(n)

วนลูปหา key ใน list ของ index นั้น ถ้าเจอจะ delete bucket ในตำแหน่งนั้น แล้ว return True

* 1. return False กรณีไม่เจอห้อง O(1)

1. A screenshot of a computer screen

   Description automatically generatedself.root = self.avl\_tree.remove(self.root, room\_number)

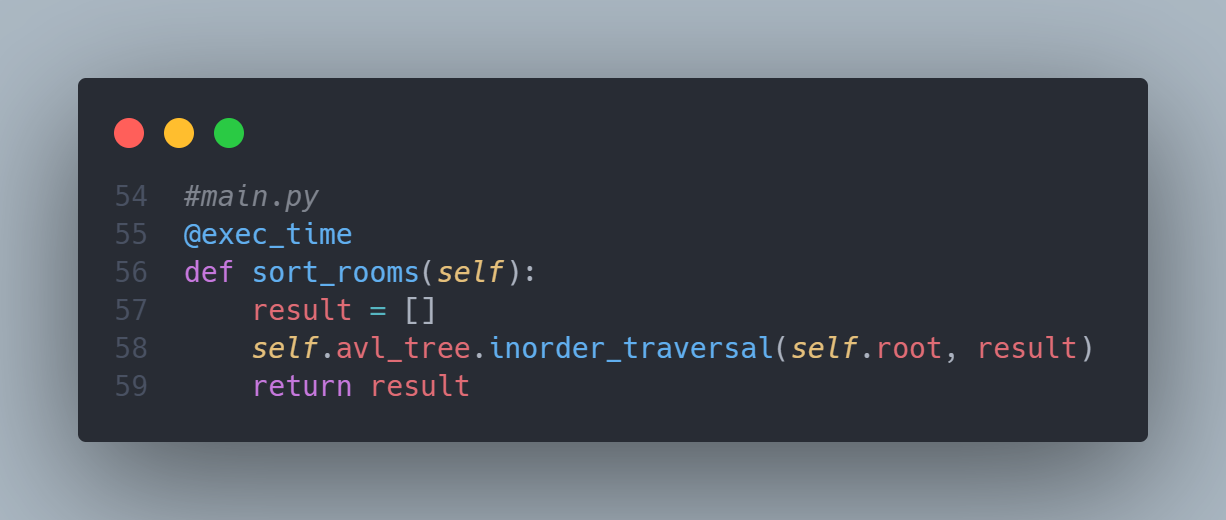
3.1 def remove O(log(n))

การ remove Node ใน AVL tree หาก key มีค่าน้อยกว่า root Node จะทำการ traversal ไปทางซ้ายกลับกันหากมากกว่า ก็จะไปทางขวาเมื่อเจอ root Node ที่มีค่าเท่ากับ key ก็ทำการลบ Node ตัวนั้น และหากมี Node ที่จะทำการลบมี child Node ก็จะนำตัวที่มีค่าน้อยที่สุดของ Node ฝั่งขวามาแทนที่ Node ที่ลบไป จากนั้นปรับ balance ของ AVL tree

3.2 def get\_min\_value Node O(log(n))

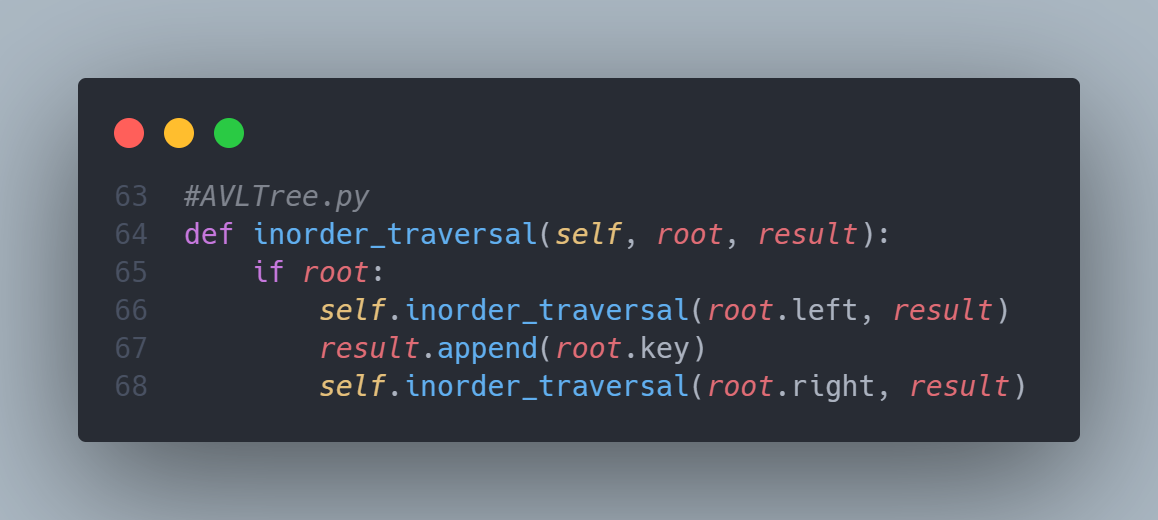
การหาค่าที่น้อยที่สุดใน AVL tree คือการ traversal ไปจนถึงค่า left Node ตัวสุดท้ายจากนั้น return ค่าของ Node ออกมา

**สรุป Big O = O(n)**

**Function sort\_room**

1. result = [] O(1)

คือการสร้างlistเปล่าขึ้นมาเพื่อเก็บข้อมูล

1. self.avl\_tree.inorder\_traversal(self.root, result)
   1. if root: O(1)

ถ้า root ไม่ใช่ None ให้เข้าเงื่อนไข

* 1. self.inorder\_traversal(root.left, result) O(n)

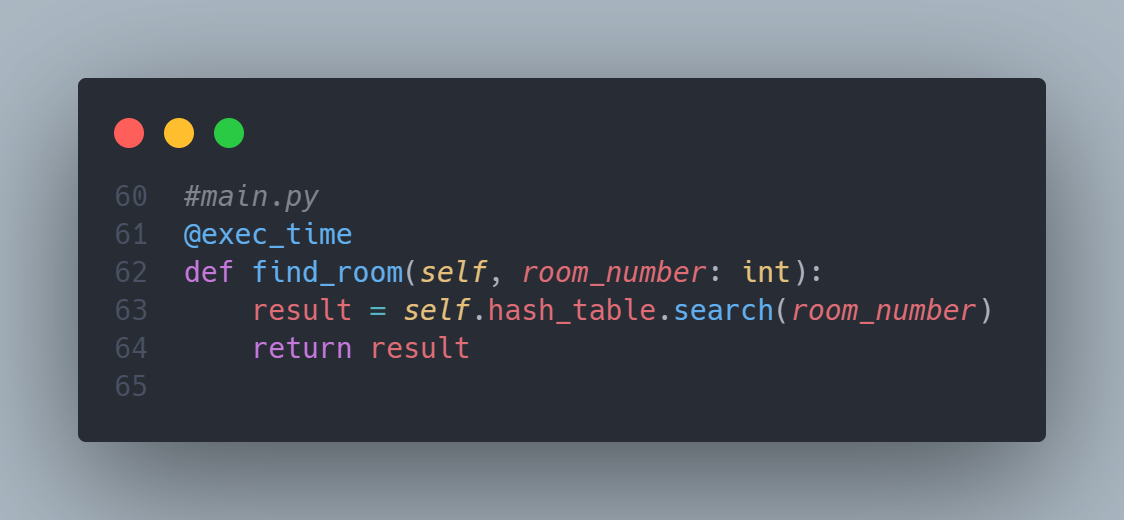
เป็นการเลื่อนค่า root ไปทางซ้ายเพื่อที่จะเรียงห้อง

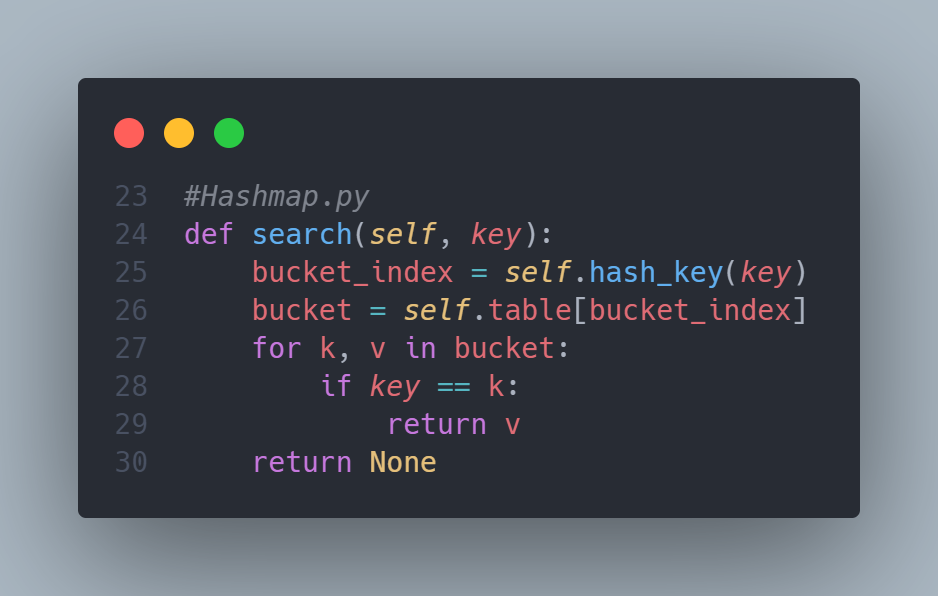
* 1. result.append(root.key) O(1)
  2. self.inorder\_traversal(root.right, result) O(n)

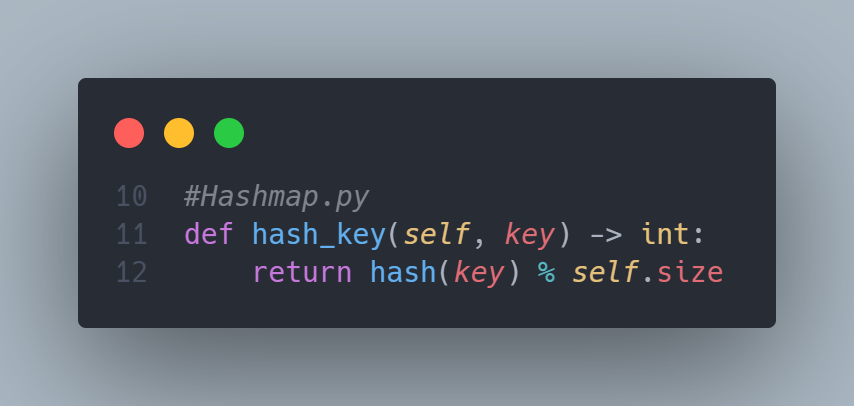
1. return result O(1)

เป็นการคืนค่า result หลังจากผ่านการ inorder\_traversalแล้ว

**สรุป Big O = O(n)**

**Function find\_room การค้นหาหมายเลขห้อง**

1. result = self.hash\_table.search(room\_number)
   1. bucket\_index = self.hash\_key(key) O(1)

นำ key ไปคำนวนหา index ใน hash table โดยการนำ key modulus ด้วย ขนาดของ hash table

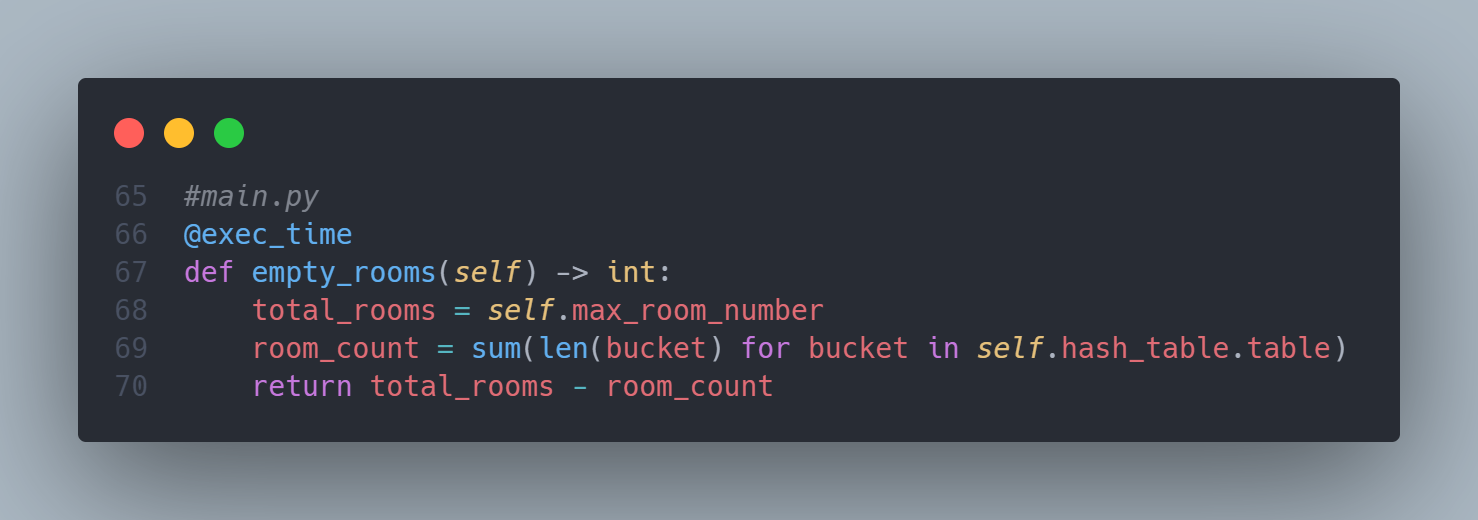
* 1. bucket = self.table[bucket\_index] O(1)
  2. for k, v in bucket: O(n)

วนลูปหา key ใน list ของ index นั้น ถ้าเจอจะ return ข้อมูลของคนที่อยู่ในห้องออกมา

* 1. return None กรณีที่ไม่เจอห้อง O(1)

1. return result O(1)

**สรุป Big O = O(n)**

**Function empty\_rooms แสดงจำนวนห้องที่ว่าง**

1. total\_rooms = self.max\_room\_number O(1)

ห้องที่หมายเลขมากที่สุดในโรงแรม

1. sum(len(bucket) for bucket in self.hast\_table.table O(n)
2. return total\_rooms – room\_count O(1)

**สรุป Big O = O(n)**