



Hilbert's Hotel

นายยกพล	นิลบาร์นัต	67010751
นายสิรภาพ	ลั้มไกรสรณ์	67010943
นายศิวักร	สุขชมทอง	67010889
นายพิทวัส	ฉิน	67011509
นายอิทธิวัชร	ทินประภา	67011583

รายงานประกอบวิชา 01076110 Object Oriented Data Structures Project

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคการศึกษาที่ 1 ปีการศึกษา 2568

คำนำ

รายงานฉบับนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของรายวิชา Object Oriented Data Structures Project โดยมีจุดประสงค์เพื่อศึกษาและทำความเข้าใจทฤษฎี Hilbert's Hotel ของ David Hilbert ซึ่งเป็นแนวคิดเชิงปรัชญาและคณิตศาสตร์เกี่ยวกับ “ความไม่สิ้นสุด” การศึกษาแนวคิดช่วยเสริมสร้างความเข้าใจด้านทฤษฎี เพื่อพัฒนาทักษะการคิดเชิงนามธรรมและการออกแบบเชิงโครงสร้างเชิงลึก

ผู้จัดทำหวังว่ารายงานฉบับนี้จะเป็นประโยชน์แก่ผู้อ่าน ในการสร้างความรู้ ความเข้าใจ เกี่ยวกับแนวคิดของทฤษฎี Hilbert's Hotel ของ David Hilbert ตลอดจนเห็นแนวทางในการเชื่อมโยงทฤษฎีดังกล่าวกับการประยุกต์ใช้ในด้านวิศวกรรมคอมพิวเตอร์

คณะผู้จัดทำ

สารบัญ

	หน้า
คำนำ	ก
สารบัญ	ข
สารบัญ(ต่อ)	ค
สารบัญรูปภาพ	ง
บทที่ 1 บทนำ	1
1.1 ความเป็นมาและความสำคัญของทฤษฎี Hilbert's Hotel	1
1.2 วัตถุประสงค์ของรายงาน	2
1.3 ขอบเขตของการศึกษา	2
บทที่ 2 โครงสร้างโปรแกรม	3
2.1 โครงสร้างโปรแกรม	3
2.1.1 Class Guest Travel	3
2.1.2 Class Hotel	4
2.1.3 โปรแกรมหลัก	5
2.2 Source Code	6
บทที่ 3 การอธิบายการทำงานของแต่ละฟังก์ชันและการวิเคราะห์ประสิทธิภาพ	13
3.1 การวิเคราะห์ประสิทธิภาพของฟังก์ชัน	13
3.1.1 การเพิ่มหมายเลขห้องแบบแมนนวล	13
3.1.2 การลบหมายเลขห้องแบบแมนนวล	13
3.1.3 การจัดเรียงลำดับหมายเลขห้อง	14
3.1.4 การค้นหาหมายเลขห้อง	14
3.2 การจำลองการทำงานของโปรแกรม	15
3.2.1 ลำดับการทำงานหลักของโปรแกรม	15
3.2.2 จำลองการทำงานของโปรแกรม	16
บทที่ 4 สรุปผลและข้อเสนอแนะ	21
4.1 สรุปผล	21
4.2 ข้อเสนอแนะ	21

สารบัญ(ต่อ)

บรรณานุกรม

ภาคผนวก

หน้า

จ

ฉ

สารบัญรูปภาพ

	หน้า
รูปที่ 1 ภาพ source code ของคลาส GuestTravel	6
รูปที่ 2 ภาพ source code ของคลาส Hotel(1)	7
รูปที่ 3 ภาพ source code ของคลาส Hotel(2)	8
รูปที่ 4 ภาพ source code ของคลาส Hotel(3)	8
รูปที่ 5 ภาพ source code ของคลาส Hotel(4)	9
รูปที่ 6 ภาพ source code ของคลาส Hotel(5)	9
รูปที่ 7 ภาพ source code ของคลาส Hotel(6)	10
รูปที่ 8 ภาพ source code ของคลาส Hotel(7)	10
รูปที่ 9 ภาพ source code ของคลาส Hotel(8)	11
รูปที่ 10 ภาพ source code ของคลาส main(1)	11
รูปที่ 11 ภาพ source code ของคลาส main(2)	12
รูปที่ 12 ภาพ source code ของคลาส main(3)	12
รูปที่ 13 ภาพของฟังก์ชัน manual_add_guest	13
รูปที่ 14 ภาพของฟังก์ชัน remove_room	14
รูปที่ 15 ภาพของฟังก์ชัน search_room	15
รูปที่ 16 การกรอกค่า “2 3 4 5” เพื่อสร้างกลุ่มแขกแรกจำนวน 120 ห้อง	16
รูปที่ 17 ระบบตรวจสอบว่าห้องยังว่างอยู่และเพิ่มข้อมูลสำเร็จ	17
รูปที่ 18 ระบบแสดงรายชื่อแขกอัตโนมัติและแขกที่เพิ่มด้วยมือ	17
รูปที่ 19 ระบบแสดงข้อมูลของหมายเลขห้องที่ผู้ใช้กรอก	18
รูปที่ 20 ระบบแสดงลบข้อมูลของหมายเลขห้องที่ผู้ใช้กรอก	18
รูปที่ 21 ระบบแสดงเวลาการทำงานรวมของระบบ	19
รูปที่ 22 ระบบแสดงหน่วยข้อมูลที่ใช้ทั้งหมด	19
รูปที่ 23 แสดงการบันทึกข้อมูลลงไฟล์ hotel_rooms	20
รูปที่ 24 แสดงข้อมูลที่อยู่ในไฟล์ที่บันทึก	20

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของทฤษฎี Hilbert's Hotel

ทฤษฎี Hilbert's Hotel เป็นแนวคิดที่ตั้งขึ้นโดยนักคณิตศาสตร์ชาวเยอรมัน David Hilbert ในปี ค.ศ. 1924 เพื่อลองชี้ให้เห็นลักษณะของ อนันต์ และวิธีการที่เราสามารถทำงานกับคณิตศาสตร์ที่เกี่ยวข้องกับจำนวนที่ไม่สามารถนับได้ (infinite sets) หรือ “อนันต์” ที่ไม่สามารถมองเห็นและเข้าใจในเชิงลึกได้ในโลกแห่งความจริง

เรื่องราวของ Hilbert's Hotel ถูกใช้เป็นคำอธิบายที่สมมุติขึ้นเพื่อเข้าใจคุณสมบัติของเซตอนันต์ เช่น เซตของจำนวนเต็ม ซึ่งมีคุณสมบัติที่แตกต่างจากเซตจำนวนจำกัด (finite sets) ตัวอย่างที่ Hilbert ยกมาเกี่ยวข้องกับโรงแรมที่มีห้องพักจำนวนอนันต์ โดยแต่ละห้องมีหมายเลขกำหนด (1, 2, 3, ...) และเมื่อมีแขกใหม่มาถึงในขณะที่โรงแรมเต็มแล้ว ก็สามารถจัดที่พักรับแขกได้โดยไม่ต้องเพิ่มห้องใหม่ โดยการย้ายแขกคนเดิมจากห้องหนึ่งไปห้องอื่น เช่น การย้ายแขกจากห้อง 1 ไปห้อง 2, ห้อง 2 ไปห้อง 3, และต่อไป ซึ่งจะทำให้ห้อง 1 ว่างสำหรับแขกใหม่ได้

ความสำคัญของทฤษฎีนี้ไม่ได้จำกัดแค่การเป็นตัวอย่างที่แสดงให้เห็นถึงลักษณะของอนันต์ แต่ยังชี้ให้เห็นถึงความแตกต่างที่สำคัญระหว่างเซตที่มีขนาดจำกัดและเซตที่มีขนาดอนันต์ เช่น ความสามารถในการจัดระเบียบหรือย้ายตำแหน่งของสมาชิกในเซตเหล่านั้น ซึ่งเป็นประเด็นสำคัญในหลายๆ ด้านของคณิตศาสตร์ เช่น การศึกษาทฤษฎีเซต (Set Theory) และการพัฒนาความเข้าใจในวิธีการทำงานกับอนันต์

ในเชิงคณิตศาสตร์, ทฤษฎีนี้ยังมีบทบาทในการศึกษาของฟังก์ชันที่มีค่าอนันต์, แนวคิดของจำนวนจริง และพื้นที่ในเชิงลึกที่ช่วยเสริมการวิเคราะห์ในสาขาต่างๆ ทั้งในเชิงทฤษฎีและประยุกต์ เช่น ฟิสิกส์, วิศวกรรมศาสตร์, และคณิตศาสตร์ประยุกต์ที่เกี่ยวข้องกับเซตอนันต์และอนันต์ในเชิงคณิตศาสตร์

การเข้าใจและการใช้ Hilbert's Hotel จึงไม่เพียงแต่เป็นการศึกษาคณิตศาสตร์ในเชิงลึก แต่ยังเป็นวิธีการที่ช่วยสร้างภาพการคิดเชิงคณิตศาสตร์ที่ซับซ้อนและเข้าใจยากให้เป็นเรื่องที่เป็นรูปธรรมและเข้าใจได้ง่ายยิ่งขึ้น

1.2 วัตถุประสงค์ของรายงาน

1. เพื่อศึกษาความเป็นมาและแนวคิดของทฤษฎี Hilbert's Hotel โดยเฉพาะการใช้โรงแรมสมมุติในการอธิบายลักษณะของเซตอนันต์ และคุณสมบัติของจำนวนอนันต์ในเชิงคณิตศาสตร์
2. เพื่อทำความเข้าใจเกี่ยวกับลักษณะและคุณสมบัติของเซตอนันต์และการจัดการกับเซตที่มีจำนวนอนันต์ ผ่านตัวอย่างของการจัดห้องในโรงแรมที่มีห้องพักอนันต์
3. เพื่อเสริมสร้างความเข้าใจในการจัดการกับอนันต์และช่วยให้สามารถมองเห็นแนวคิดที่เกี่ยวข้องกับอนันต์ในรูปแบบที่เข้าใจง่ายและสามารถนำไปใช้ในการศึกษาและการประยุกต์ในอนาคตได้

1.3 ขอบเขตของการศึกษา

การศึกษาในรายงานนี้จะมุ่งเน้นไปที่การทำความเข้าใจและอธิบายทฤษฎี Hilbert's Hotel ซึ่งเป็นแนวคิดทางคณิตศาสตร์ที่เกี่ยวข้องกับลักษณะและคุณสมบัติของเซตอนันต์ โดยจะศึกษาความเป็นมาและหลักการทำงานของทฤษฎีนี้ ผ่านการใช้ตัวอย่างของโรงแรมที่มีห้องพักอนันต์ที่เต็มไปด้วยแขก และสามารถรองรับแขกใหม่ได้

บทที่ 2

โครงสร้างโปรแกรม

2.1 โครงสร้างโปรแกรม

โปรแกรม Hotel Management System ได้รับการออกแบบภายใต้แนวคิด เชิงวัตถุ (Object Oriented Programming) เพื่อจำลองการจัดการระบบโรงแรมอย่างเป็นระบบและมีลำดับขั้นตอน โดยแบ่งออกเป็น สามส่วนหลัก ได้แก่ Class Hotel ตัวจัดการหลักของระบบ Class GuestTravel ตัวแทนของกลุ่มแขกและโครงสร้างการเข้าพัก และโปรแกรมหลัก ส่วนควบคุมการทำงานและการโต้ตอบกับผู้ใช้

2.1.1 Class GuestTravel

คลาสนี้ใช้สำหรับเก็บข้อมูลและจำลอง กลุ่มแขก (Guest Group) ที่เข้าพักในโรงแรม โดยแต่ละกลุ่มมีการจัดรูปแบบการเดินทางในลักษณะสัมฤทธิ์ ได้แก่ ท่าเรือ (Quay) เรือ (Boat) รถบัส (Bus) และที่นั่ง (Seat)

2.1.1.1 โครงสร้างตัวแปร (Attributes)

ตัวแปร	คำอธิบาย
last_guest_index	ค่าดัชนีของแขกล่าสุดในกลุ่มก่อนหน้านี้
Travel	รายการจำนวนหน่วยในแต่ละมิติของการเดินทาง
shift	ค่าการเลื่อนตำแหน่งของห้อง
guest_count	จำนวนแขกทั้งหมดในกลุ่ม

2.1.1.2 ตัวแปรระดับคลาส (Class-Level Variables)

ตัวแปร	คำอธิบาย
deleted_room	เซตเก็บหมายเลขห้องที่ถูกลบ
manually_added_guest	พจนานุกรมเก็บข้อมูลแขกที่เพิ่มด้วยตนเอง

2.1.1.3 เมธอด (Methods)

1. `room_index(guest_id)` ใช้คำนวณหมายเลขห้องจริงของแขก โดยบวกค่า `shift` เข้าไปใน `guest_id`
2. `Shift(n)` ใช้สำหรับเลื่อนตำแหน่งห้องทั้งหมดในกลุ่ม เมื่อมีการเพิ่มกลุ่มใหม่เข้ามา
3. `print_index()` ทำหน้าที่แสดงรายชื่อแขกทั้งหมดที่ยังไม่ถูกลบออกจากระบบ พร้อมหมายเลขห้อง

2.1.2 Class Hotel

คลาส Hotel ทำหน้าที่เป็น ตัวควบคุมหลักของระบบ มีหน้าที่จัดการข้อมูลกลุ่มแขกทั้งหมด (ผ่านการเก็บอ็อบเจกต์ของ `GuestTravel` ในลิสต์) รวมถึงการลบ เพิ่ม และค้นหาห้อง ตลอดจนการวัดประสิทธิภาพและการบันทึกข้อมูลออกไฟล์

2.1.2.1 โครงสร้างตัวแปร (Attributes)

ตัวแปร	คำอธิบาย
<code>guestHotel</code>	รายการของกลุ่มแขกทั้งหมด (list ของอ็อบเจกต์ <code>GuestTravel</code>)
<code>tempShift</code>	เก็บค่าการเลื่อนห้องแต่ละครั้งที่เกิดขึ้น
<code>travel</code>	ดัชนีการเดินทางของกลุ่มที่ค้นหา
<code>total_runtime</code>	เวลาทำงานรวมของทุกฟังก์ชัน

2.1.2.2 เมธอด (Methods)

1. การจัดการแขก
 - a) `add_guest(guest: list[int])` เพิ่มกลุ่มแขกใหม่เข้าระบบ โดยคำนวณจำนวนแขกทั้งหมดจากผลคูณของแต่ละมิติ และอัปเดตค่าการเลื่อนตำแหน่งห้อง (`shift`)
 - b) `manual_add_guest(room_num)` เพิ่มแขกด้วยตนเองลงในหมายเลขห้องที่กำหนด พร้อมตรวจสอบว่าห้องนั้นไม่ถูกจองอยู่ก่อนแล้ว

c) `remove_room(room_id)` ลบข้อมูลห้องออกจากระบบ โดยเก็บรหัสไว้ในเซต `deleted_room`

2. การค้นหาและแสดงผล

a) `print_sorted_room()` แสดงรายชื่อห้องและแขกทั้งหมดในลำดับที่จัดเรียง พร้อมระบุประเภทแขก (อัตโนมัติหรือเพิ่มด้วยมือ)

b) `search_room(room_num)` ค้นหาหมายเลขห้องในทุกกลุ่มแขก และคืนข้อมูลเชิงโครงสร้าง เช่น กลุ่มที่อยู่, รหัสแขก, และเส้นทางการเดินทาง

c) `find_path()` ใช้ค้นหาพิกัดของแขกในโครงสร้างสี่มิติ

3. การประมวลผลและบันทึก

a) `code_runtime(func, *args, **kwargs)` ใช้วัดเวลาการทำงานของฟังก์ชันย่อย

b) `memory_used()` แสดงปริมาณหน่วยความจำที่ใช้โดยใช้โมดูล `tracemalloc`

c) `save_to_file(filename)` เขียนข้อมูลแขกทั้งหมดลงในไฟล์ `.csv` โดยจัดรูปแบบข้อมูลให้ชัดเจนด้วยการจัดตำแหน่งคอลัมน์ และถ้าไม่มีชื่อไฟล์ จะตั้งชื่อไฟล์ Default เป็น `hotel_rooms` ให้อัตโนมัติ

2.1.2 โปรแกรมหลัก

ส่วนของ `main` เป็นจุดเริ่มต้นของโปรแกรม ทำหน้าที่เป็น ส่วนติดต่อผู้ใช้แบบข้อความ โดยให้ผู้ใช้เลือกคำสั่งจากเมนู เช่น เพิ่มกลุ่มแขก เพิ่มห้องด้วยตนเอง แสดงรายชื่อห้อง ค้นหาห้อง ลบห้อง แสดงเวลาการทำงานรวม แสดงหน่วยความจำที่ใช้ บันทึกข้อมูลลงไฟล์

2.2 Source Code

2.2.1 Class GuestTravel

```

from typing import List
from functools import reduce

class GuestTravel:
    deleted_guest = set()
    deleted_room = set()
    manually_added_guest = {}
    def __init__(self, first_guest_index: int, travel : List[int]):
        self.last_guest_index = first_guest_index
        self.travel = travel
        self.shift = 0
        self.guest_count = reduce(lambda x, y: x * y, travel) + first_guest_index

    def Shift(self, n):
        self.shift += n

    def room_index(self, guest_id):
        return guest_id + self.shift

    def last_guest(self):
        return self.last_guest_index

    def print_index(self):
        guest_count = 0
        if len(self.travel) == 4:
            for m in range(self.travel[0]):
                offset_a = m * self.travel[1] * self.travel[2] * self.travel[3]
                for l in range(self.travel[1]):
                    offset_b = offset_a + l * self.travel[2] * self.travel[3]
                    for k in range(self.travel[2]):
                        offset_c = offset_b + k * self.travel[3]
                        for j in range(self.travel[3]):
                            offset_d = offset_c + j
                            for i in range(1):
                                guest_id = i + offset_d
                                room_index = self.room_index(guest_id)
                                if guest_id not in GuestTravel.deleted_guest and room_index not in self.deleted_room:
                                    final_guest_id = guest_id + self.last_guest_index
                                    guest_count += 1
                                    print(f" Guest #{final_guest_id:<10} → Room {room_index:>6}")
        return guest_count

```

รูปที่ 1 ภาพ source code ของ คลาส GuestTravel

2.2.2 Class Hotel

```
import tracemalloc
from guestHotel import GuestTravel
import math
from functools import reduce
import time

class Hotel:
    def __init__(self, guestHotel:list =[]):
        self.guestHotel = []
        self.tempShift = []
        self.travel = 0
        self.total_runtime = 0

    def remove_guest(self, guest_id: int):
        if guest_id in GuestTravel.deleted_guest:
            print("❌ Guest already deleted!")
            return
        GuestTravel.deleted_guest.add(guest_id)
        print(f"✅ Successfully removed guest from room {guest_id}")

    def remove_room(self, room_id: int):
        if room_id in GuestTravel.deleted_room:
            print("❌ Room already deleted!")
            return
        if room_id in GuestTravel.manually_added_guest.values():
            for k, v in GuestTravel.manually_added_guest.items():
                if v == room_id:
                    del GuestTravel.manually_added_guest[k]
                    break
        GuestTravel.deleted_room.add(room_id)
        print(f"✅ Successfully removed room {room_id}")

    def shift_TheManuallyAdded(self, shift_value):
        for key, value in GuestTravel.manually_added_guest.items():
            GuestTravel.manually_added_guest[key] = value + shift_value

    def shift_All(self, shift_value:int):
        self.tempShift.append(shift_value)
        self.shift_TheManuallyAdded(shift_value)
        for guest in self.guestHotel:
            guest.Shift(shift_value)
```

รูปที่ 2 ภาพ source code ของ คลาส Hotel (1)

```

def add_guest(self, guest:list[int]):
    if(len(self.guestHotel) == 0):
        new_guest = GuestTravel(0, guest)
        self.shift_TheManuallyAdded(reduce(lambda x, y: x * y, guest))
        self.guestHotel.append(new_guest)
    else:
        self.shift_All(reduce(lambda x, y: x * y, guest))
        new_guest = GuestTravel(self.guestHotel[-1].guest_count, guest)
        self.guestHotel.append(new_guest)

    total_guests = reduce(lambda x, y: x * y, guest)
    print(f"✅ Added {total_guests} guests successfully!")
    print(f"    Travel configuration: {guest}")

def manual_add_guest(self, room_num):
    if((len(self.guestHotel) > 0 and self.guestHotel[-1].guest_count >= room_num) or room_num in GuestTravel.manually_added_guest.values()):
        print(f"❌ Room is already occupied!")
    else:
        guest_id = "M" + str(len(GuestTravel.manually_added_guest))
        GuestTravel.manually_added_guest[guest_id] = room_num
        print(f"✅ Successfully added manual guest {guest_id} to room {room_num}")

```

รูปที่ 3 ภาพ source code ของ คลาส Hotel (2)

```

def print_sorted_room(self):
    print("\n" + "=" * 60)
    print("🏨 HOTEL ROOM STATUS REPORT".center(60))
    print("=" * 60)

    total_rooms = 0

    for idx, group in enumerate(reversed(self.guestHotel)):
        group_num = len(self.guestHotel) - idx
        print(f"\n┌{'-' * 58}┐")
        print(f"| 📋 Guest Group #{group_num:<40}|")
        print(f"| Configuration: {str(group.travel):<38}|")
        print(f"└{'-' * 58}┘")

        guest_count = group.print_index()
        total_rooms += guest_count

    if GuestTravel.manually_added_guest:
        print(f"\n┌{'-' * 58}┐")
        print(f"| 👤 Manually Added Guests{' ' * 31}|")
        print(f"└{'-' * 58}┘")
        for guest_id, room_num in sorted(GuestTravel.manually_added_guest.items(), key=lambda x: x[1]):
            print(f"    Guest {guest_id:<12} → Room {room_num:>6}")
        total_rooms += len(GuestTravel.manually_added_guest)

    print("\n" + "=" * 60)
    print(f"🏠 Total Occupied Rooms: {total_rooms}")
    print("=" * 60 + "\n")

```

รูปที่ 4 ภาพ source code ของ คลาส Hotel (3)

```

def find_path(self, room_num, travel):
    room_idx = room_num
    seat = (room_idx % travel[3]) + 1
    room_idx //= travel[3]
    bus = (room_idx % travel[2]) + 1
    room_idx //= travel[2]
    boat = (room_idx % travel[1]) + 1
    room_idx //= travel[1]
    quay = (room_idx % travel[0]) + 1
    return quay, boat, bus, seat

def find_insert(self, target: int) -> int:
    if not self.guestHotel:
        return 0
    lo, hi = 0, len(self.guestHotel) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        mid_val = self.guestHotel[mid].shift
        seed_range = self.guestHotel[mid].guest_count - self.guestHotel[mid].last_guest_index
        if mid_val <= target < seed_range + mid_val:
            return mid
        if mid_val > target:
            lo = mid + 1
        elif mid_val < target:
            hi = mid - 1
    return mid

```

รูปที่ 5 ภาพ source code ของ คลาส Hotel (4)

```

def search_room(self, room_num):
    print(f"\n🔍 Searching for room {room_num}...")
    print("-" * 50)
    if room_num in GuestTravel.deleted_room:
        print(f"\n❌ Room Deleted!")
        print(f"    Room Number : {room_num}")
        print(f"    Type : Manually Deleted")
        print("-" * 50 + "\n")
        return
    if room_num in GuestTravel.manually_added_guest.values():
        keys = [key for key, val in GuestTravel.manually_added_guest.items() if val == room_num]
        print(f"\n✅ Room Found!")
        print(f"    Guest ID : {keys[0]}")
        print(f"    Room Number : {room_num}")
        print(f"    Type : Manual")
        print("-" * 50 + "\n")
        return
    if room_num > self.guestHotel[-1].guest_count-1:
        print(f"\n❌ Room {room_num} not found in hotel!")
        print("-" * 50 + "\n")
        return
    idx = self.find_insert(room_num)
    self.travel = idx
    detail = self.find_path(room_num - self.guestHotel[idx].shift, self.guestHotel[idx].travel)
    guest_id = (room_num - self.guestHotel[idx].shift) + self.guestHotel[idx].last_guest_index
    print(f"\n✅ Room Found!")
    print(f"    Guest ID : {guest_id}")
    print(f"    Room Number : {room_num}")
    print(f"    Group Index : {idx+1}")
    print(f"    Travel Path : Quay {detail[0]}, Boat {detail[1]}, Bus {detail[2]}, Seat {detail[3]}")
    print("-" * 50 + "\n")

```

รูปที่ 6 ภาพ source code ของ คลาส Hotel (5)

```

def save_to_file(self, filename: str):
    try:
        all_data = []
        for idx, group in enumerate(self.guestHotel):
            if len(group.travel) == 4:
                for m in range(group.travel[0]):
                    offset_a = m * group.travel[1] * group.travel[2] * group.travel[3]
                    for l in range(group.travel[1]):
                        offset_b = offset_a + l * group.travel[2] * group.travel[3]
                        for k in range(group.travel[2]):
                            offset_c = offset_b + k * group.travel[3]
                            for j in range(group.travel[3]):
                                offset_d = offset_c + j
                                guest_id = offset_d
                                room_index = group.room_index(guest_id)
                                if guest_id not in GuestTravel.deleted_guest and room_index not in GuestTravel.deleted_room:
                                    final_guest_id = guest_id + group.last_guest_index
                                    travel_path = f"({m+1},{l+1},{k+1},{j+1})"
                                    all_data.append({
                                        'guest_id': str(final_guest_id),
                                        'room_num': str(room_index),
                                        'group': str(idx + 1),
                                        'travel_path': travel_path,
                                        'type': 'Auto'
                                    })

        for guest_id, room_num in sorted(GuestTravel.manually_added_guest.items(), key=lambda x: x[1]):
            all_data.append({
                'guest_id': guest_id,
                'room_num': str(room_num),
                'group': '-',
                'travel_path': '-',
                'type': 'Manual'
            })
    
```

รูปที่ 7 ภาพ source code ของ คลาส Hotel (6)

```

def code_runtime(self, func, *args, **kwargs):
    start = time.time()
    result = func(*args, **kwargs)
    end = time.time()
    runtime = end - start
    self.total_runtime += runtime
    print(f"🕒 Runtime: {runtime:.6f} seconds")
    return result

def print_total_runtime(self):
    print("\n" + "=" * 50)
    print(f"🕒 Total Runtime: {self.total_runtime:.6f} seconds")
    print("=" * 50 + "\n")

def memory_used(self):
    current, peak = tracemalloc.get_traced_memory()
    print("\n" + "=" * 50)
    print("📊 Memory Usage Report")
    print("=" * 50)
    print(f"Peak Memory : {peak / 1024:.2f} KB")
    print(f"Current Memory: {current / 1024:.2f} KB")
    print("=" * 50 + "\n")
    
```

รูปที่ 8 ภาพ source code ของ คลาส Hotel (7)

```

max_guest_id = max(len(row['guest_id']) for row in all_data) if all_data else 7
max_room_num = max(len(row['room_num']) for row in all_data) if all_data else 10
max_group = max(len(row['group']) for row in all_data) if all_data else 5
max_travel = max(len(row['travel_path']) for row in all_data) if all_data else 11
max_type = 6

max_guest_id = max(max_guest_id, 7)
max_room_num = max(max_room_num, 10)
max_group = max(max_group, 5)
max_travel = max(max_travel, 11)

with open(filename + '.csv', 'w', encoding='utf-8') as f:
    header = f'{"GuestID":^{max_guest_id}} | {"RoomNumber":^{max_room_num}} | {"Group":^{max_group}} | {"TravelPath":^{max_travel}} | {"Type":^{max_type}}\n'
    separator = f'{"-"*max_guest_id}+{"-"*max_room_num}+{"-"*max_group}+{"-"*max_travel}+{"-"*max_type}\n'
    f.write(header)
    f.write(separator)

    for row in all_data:
        line = f'{"row['guest_id']>{max_guest_id}} | {"row['room_num']>{max_room_num}} | {"row['group']>{max_group}} | {"row['travel_path']>{max_travel}} | {"row['type']>{max_type}}\n'
        f.write(line)

print(f"✔ Hotel data successfully saved to '{filename}'")
print(f"    Total records: {len(all_data)}")
except IOError as e:
    print(f"✖ Error saving file: {e}")

```

รูปที่ 9 ภาพ source code ของ คลาส Hotel (8)

2.2.3 โปรแกรมหลัก

```

from Hotel import Hotel
import tracemalloc

def print_menu():
    print("\n" + "=" * 60)
    print("🏨 HOTEL MANAGEMENT SYSTEM".center(60))
    print("=" * 60)
    print(" 1. ➕ Add guest group")
    print(" 2. 👤 Add room and people manually")
    print(" 3. 📋 Print sorted room list")
    print(" 4. 🔍 Search room")
    print(" 5. 🗑 Remove room")
    print(" 6. ⚙ Print all code runtime")
    print(" 7. 💾 Print memory usage")
    print(" 8. 💾 Save to file")
    print(" x. 🚪 Exit")
    print("=" * 60)

def main():
    control = True
    hotel = Hotel()
    tracemalloc.start()

    print("\n" + "=" * 60)
    print("🏨 Welcome to Hotel Management System".center(60))
    print("=" * 60)
    print("\n📝 Please enter initial guest configuration")
    print("    (Format: 4 numbers separated by spaces)")
    print("    Example: 2 3 4 5")
    print("-" * 60)

    INPUT_INITIAL = True
    while INPUT_INITIAL:
        try:
            initial_guest = list(map(int, input("➡ Initial guest: ").split()))
            if len(initial_guest) != 4:
                print("✖ Please enter exactly 4 numbers.")
                continue

            if any(i <= 0 for i in initial_guest):
                print("✖ Error Input! All numbers must be positive.")
                continue

            INPUT_INITIAL = False
        except ValueError:

```

รูปที่ 10 ภาพ source code ของ คลาส main (1)


```

        print("❌ Invalid input! Please enter only numbers separated by spaces.")
    hotel.add_guest(initial_guest)

while control:
    print_menu()
    opt = input("\n> Select option: ").strip()
    print()

    if opt == '1':
        print("📄 Enter guest configuration (4 numbers)")
        print("    Example: 2 3 4 5")
        try:
            inp_ppl = list(map(int, input("> Guest configuration: ").split()))
            if len(inp_ppl) != 4:
                print("❌ Please enter exactly 4 numbers.")
                continue
            if any(i <= 0 for i in inp_ppl):
                print("❌ Error Input! All numbers must be positive.")
                continue
            hotel.code_runtime(hotel.add_guest, inp_ppl)
        except ValueError:
            print("❌ Invalid input! Please enter only numbers separated by spaces.")

    elif opt == '2':
        try:
            inp_mul = int(input("> Enter room number: "))
            if inp_mul <= 0:
                print("❌ Room number must be a positive integer.")
                continue
            hotel.code_runtime(hotel.manual_add_guest, inp_mul)
        except ValueError:
            print("❌ Invalid input! Please enter a valid number.")

    elif opt == '3':
        hotel.code_runtime(hotel.print_sorted_room)

    elif opt == '4':
        search_room = int(input("> Enter room number to search: "))
        hotel.code_runtime(hotel.search_room, search_room)

    elif opt == '5':
        guest_number = int(input("> Enter room number to remove: "))

```

รูปที่ 11 ภาพ source code ของ คลาส main (2)

```

        hotel.code_runtime(hotel.remove_room, guest_number)

    elif opt == '6':
        hotel.print_total_runtime()

    elif opt == '7':
        hotel.memory_used()

    elif opt == '8':
        filename = input("> Enter filename (default: hotel_rooms): ").strip()
        if not filename:
            filename = "hotel_rooms"
        hotel.code_runtime(hotel.save_to_file, filename)

    elif opt == 'x' or opt == 'X':
        print("\n" + "=" * 60)
        print("🌟 Thank you for using Hotel Management System!".center(60))
        print("=" * 60 + "\n")
        control = False

    else:
        print("❌ Invalid selection! Please try again.")

tracemalloc.stop()

if __name__ == "__main__":
    main()

```

รูปที่ 12 ภาพ source code ของ คลาส main (3)

บทที่ 3

การอธิบายการทำงานของแต่ละฟังก์ชันและการวิเคราะห์ประสิทธิภาพ

3.1 การวิเคราะห์ประสิทธิภาพของฟังก์ชัน

3.1.1 การเพิ่มหมายเลขห้องแบบแมนนวล

การเพิ่มหมายเลขห้องแบบแมนนวล มีความซับซ้อนเชิงเวลาที่สูงขึ้นอยู่กับจำนวนครั้งที่มีการเพิ่ม guest แบบแมนนวล ก่อนหน้า กล่าวคือ หากกำหนดให้ขนาดของ dictionary ที่เก็บข้อมูล guest ที่ถูกเพิ่มแบบ manual มีชื่อว่า manually_added_guest และมีขนาดเท่ากับ m การดำเนินการเพิ่มหมายเลขห้องในแต่ละครั้งจะมีความซับซ้อนเป็นฟังก์ชันของ m ส่วนการตรวจสอบเงื่อนไขอื่น ๆ เช่น การตรวจสอบว่าเลขห้องมีค่าน้อยกว่าจำนวนแขกทั้งหมดหรือไม่ ถือเป็นการดำเนินการคงที่ ดังนั้น ความซับซ้อนเชิงเวลารวมของกระบวนการนี้คือ $O(m)$

```
def manual_add_guest(self, room_num):
    if ((len(self.guestHotel) > 0 and self.guestHotel[-1].guest_count >= room_num) or room_num in GuestTravel.manually_added_guest.values()):
        print("❌ Room is already occupied!")
    else:
        guest_id = "M" + str(len(GuestTravel.manually_added_guest))
        GuestTravel.manually_added_guest[guest_id] = room_num
        print(f"✅ Successfully added manual guest {guest_id} to room {room_num}")
```

รูปที่ 13 ภาพของฟังก์ชัน manual_add_guest

3.1.2 การลบหมายเลขห้องแบบแมนนวล

การลบหมายเลขห้องแบบแมนนวล จำเป็นต้องตรวจสอบก่อนว่าเลขห้องปัจจุบันถูกลบไปแล้วหรือไม่ โดยทำการตรวจสอบว่าเลขห้องดังกล่าวอยู่ภายในเซต deleted_room หรือไม่ การตรวจสอบนี้มีความซับซ้อนที่สูงขึ้นกับขนาดของเซต deleted_room ซึ่งให้ขนาดเท่ากับ d นอกจากนี้ยังต้องตรวจสอบเพิ่มเติมว่าเลขห้องดังกล่าวถูกเพิ่มแบบ manual มาก่อนหรือไม่ หากใช่ จำเป็นต้องทำการวนซ้ำ (for loop) เพื่อลบเลขห้องนั้นออกจาก dictionary manually_added_guest ซึ่งมีขนาดเท่ากับ m ในส่วนของการเพิ่มหมายเลขห้องเข้าไปในเซตถือเป็นการดำเนินการแบบคงที่ โดยมีความซับซ้อน $O(1)$ ดังนั้น ความซับซ้อนเชิงเวลารวมของฟังก์ชันนี้คือ $O(m+d)$

```
def remove_room(self, room_id: int):
    if room_id in GuestTravel.deleted_room:
        print("❌ Room already deleted!")
        return
    if room_id in GuestTravel.manually_added_guest.values():
        for k, v in GuestTravel.manually_added_guest.items():
            if v == room_id:
                del GuestTravel.manually_added_guest[k]
                break
    GuestTravel.deleted_room.add(room_id)
    print(f"✅ Successfully removed room {room_id}")
```

รูปที่ 14 ภาพของฟังก์ชัน remove_room

3.1.3 การจัดเรียงลำดับหมายเลขห้อง

การจัดเรียงลำดับหมายเลขห้องไม่จำเป็นต้องมีการดำเนินการจัดเรียงเพิ่มเติม เนื่องจากข้อมูลที่เก็บอยู่ใน guestHotel ถูกจัดเรียงตามลำดับอยู่แล้ว การดำเนินการเพียงอย่างเดียวคือการวนซ้ำ (loop) ผ่านรายการ guestHotel จากหลังไปหน้า เพื่อเข้าถึงหรือแสดงผลตามลำดับที่ต้องการ ดังนั้นการจัดเรียงหมายเลขห้องในขั้นตอนนี้อาจถือเป็นการดำเนินการแบบคงที่ ซึ่งมีความซับซ้อนเชิงเวลา คือ $O(1)$

3.1.4 การค้นหาหมายเลขห้อง

การค้นหาหมายเลขห้อง เริ่มต้นด้วยการตรวจสอบว่าเลขห้องดังกล่าวถูกลบไปแล้วหรือไม่ โดยการตรวจสอบในเซตของ delete_room ซึ่งมีขนาดเท่ากับ d มีความซับซ้อนเชิงเวลาเท่ากับ $O(d)$ จากนั้นจะค้นต่อว่า ห้องดังกล่าวถูกเพิ่มแบบแมนนวล หรือไม่ โดยการตรวจสอบนี้ขึ้นอยู่กับขนาดของ dictionary manually_added_guest ซึ่งมีขนาดเท่ากับ m มีความซับซ้อนเชิงเวลาเท่ากับ $O(m)$ จากนั้นจะทำการค้นหาว่าเลขห้องนั้นอยู่ในช่วงของ guestHotel ไດ โดยขึ้นอยู่กับขนาดของ guestHotel ซึ่งมีขนาดเท่ากับ g และใช้วิธีการค้นหาแบบ Binary Search ทำให้ส่วนนี้มีความซับซ้อนเชิงเวลาเท่ากับ $O(\log(g))$ ต่อมาจะต้องค้นหาเส้นทางการเดินทางของแขก โดยฟังก์ชัน find_path(self, room_num, travel) ซึ่งจะรับค่ามาคำนวณนั้นคือ room_num จะเป็นเลขของที่นั่งจริงๆของมัน คือเลขห้องลบออกด้วยค่าที่ shift ไป และ travel คือ มิติการเดินทางรอบที่ตรงกับเลขห้องที่ต้องการ (quay, ship, bus, seat) หลังจากนั้นจะเอา room_num และ มิติการเดินทางทั้งสี่มาคำนวณ ความซับซ้อนเชิงเวลาจึงเป็น $O(1)$ ดังนั้น ความซับซ้อนเชิงเวลารวมของกระบวนการค้นหาหมายเลขห้องคือ $O(d+m+\log(g))$

```

def search_room(self, room_num):
    print(f"\n🔍 Searching for room {room_num}...")
    print("-" * 50)
    if room_num in GuestTravel.deleted_room:
        print(f"\n❌ Room Deleted!")
        print(f"Room Number : {room_num}")
        print(f"Type : Manually Deleted")
        print("-" * 50 + "\n")
        return
    if room_num in GuestTravel.manually_added_guest.values():
        keys = [key for key, val in GuestTravel.manually_added_guest.items() if val == room_num]
        print(f"\n✅ Room Found!")
        print(f"Guest ID : {keys[0]}")
        print(f"Room Number : {room_num}")
        print(f"Type : Manual")
        print("-" * 50 + "\n")
        return
    if room_num > self.guestHotel[-1].guest_count-1:
        print(f"\n❌ Room {room_num} not found in hotel!")
        print("-" * 50 + "\n")
        return

    idx = self.find_insert(room_num)
    self.travel = idx
    detail = self.find_path(room_num - self.guestHotel[idx].shift, self.guestHotel[idx].travel)
    guest_id = (room_num - self.guestHotel[idx].shift) + self.guestHotel[idx].last_guest_index

    print(f"\n✅ Room Found!")
    print(f"Guest ID : {guest_id}")
    print(f"Room Number : {room_num}")
    print(f"Group Index : {idx+1}")
    print(f"Travel Path : Quay {detail[0]}, Boat {detail[1]}, Bus {detail[2]}, Seat {detail[3]}")
    print("-" * 50 + "\n")

```

รูปที่ 15 ภาพของฟังก์ชัน search_room

3.2 การจำลองการทำงานของโปรแกรม

โปรแกรม Hotel Management System ได้รับการออกแบบเพื่อจำลองการทำงานของระบบจัดการห้องพักในโรงแรม โดยผู้ใช้งานสามารถเพิ่มกลุ่มแขก (Guest Group) เข้าพัก, เพิ่มแขกด้วยตนเอง, ลบห้อง, ค้นหาหมายเลขห้อง, และบันทึกข้อมูลลงไฟล์ได้

ระบบทำงานแบบโต้ตอบผ่านเมนู (Interactive Menu System) ซึ่งผู้ใช้งานจะเลือกคำสั่งจากตัวเลือกที่ปรากฏบนหน้าจอ และโปรแกรมจะดำเนินการตามคำสั่งนั้นโดยอัตโนมัติ

3.2.1 ลำดับการทำงานหลักของโปรแกรม

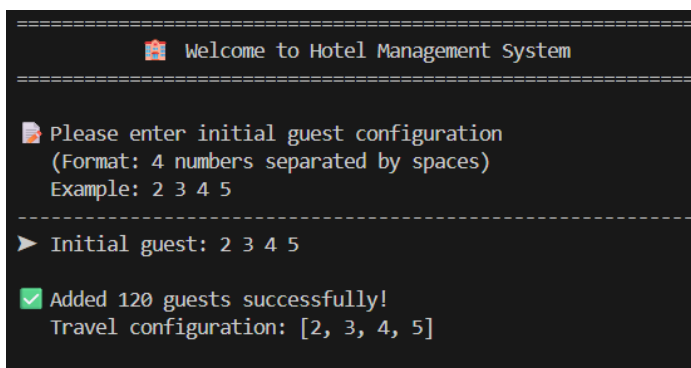
ขั้นตอนการทำงานหลักของโปรแกรมีดังนี้

1. เริ่มต้นโปรแกรม (Initialization) โปรแกรมแสดงข้อความต้อนรับ และให้ผู้ใช้งานกรอกค่าการตั้งต้นของกลุ่มแขกแรกในรูปแบบ 4 มิติ (เช่น 2 3 4 5)
2. สร้างกลุ่มแขกเริ่มต้น (Create Initial Guest Group) ระบบคำนวณจำนวนแขกทั้งหมดจากผลคูณของแต่ละมิติ ($2 \times 3 \times 4 \times 5 = 120$) และกำหนดหมายเลขห้องอัตโนมัติ

3. เข้าสู่เมนูหลัก (Main Menu) ผู้ใช้สามารถเลือกคำสั่งได้ เช่น เพิ่มแขก, ลบห้อง, ค้นหาห้อง, ดูรายชื่อห้องทั้งหมด, หรือบันทึกข้อมูลลงไฟล์
4. ประมวลผลตามคำสั่ง (Execution) เมื่อผู้ใช้เลือกคำสั่ง โปรแกรมจะเรียกเมธอดที่เกี่ยวข้องจากคลาส Hotel เพื่อดำเนินการ
5. รายงานผลลัพธ์ (Output) ผลลัพธ์ เช่น รายชื่อแขก, หมายเลขห้อง, เส้นทางการเดินทาง (Travel Path), และเวลาทำงานของระบบ จะถูกแสดงทางหน้าจอ
6. สิ้นสุดการทำงาน (Termination) เมื่อผู้ใช้เลือก “Exit” โปรแกรมจะสรุปเวลาการทำงานทั้งหมดและปิดการทำงานโดยปลอดภัย

3.2.2 จำลองการทำงานของโปรแกรม

1. ผู้ใช้เริ่มต้นด้วยการกรอกค่า “2 3 4 5” เพื่อสร้างกลุ่มแขกแรกจำนวน 120 ห้อง



```
=====
Welcome to Hotel Management System
=====

Please enter initial guest configuration
  (Format: 4 numbers separated by spaces)
  Example: 2 3 4 5
-----
> Initial guest: 2 3 4 5

✓ Added 120 guests successfully!
  Travel configuration: [2, 3, 4, 5]
```

รูปที่ 16 การกรอกค่า “2 3 4 5” เพื่อสร้างกลุ่มแขกแรกจำนวน 120 ห้อง

2. ต่อมาเลือกเมนูหมายเลข 2 เพื่อเพิ่มแขกด้วยตนเองในห้องหมายเลข 125

```
=====
HOTEL MANAGEMENT SYSTEM
=====
1. + Add guest group
2. Add room and people manually
3. Print sorted room list
4. Search room
5. Remove room
6. Print all code runtime
7. Print memory usage
8. Save to file
x. Exit
=====

> Select option: 2

> Enter room number: 125
[✓] Successfully added manual guest M0 to room 125
🕒 Runtime: 0.000247 seconds
```

รูปที่ 17 ระบบตรวจสอบว่าห้องยังว่างอยู่และเพิ่มข้อมูลสำเร็จ

3. จากนั้นเลือกเมนู 3 เพื่อพิมพ์รายชื่อห้องทั้งหมด

```
=====
HOTEL ROOM STATUS REPORT
=====

Guest Group #1
Configuration: [2, 3, 4, 5]

Guest #0    → Room    0
Guest #1    → Room    1
Guest #2    → Room    2
Guest #3    → Room    3
Guest #4    → Room    4
Guest #5    → Room    5
Guest #6    → Room    6
Guest #7    → Room    7
Guest #8    → Room    8
Guest #9    → Room    9
Guest #10   → Room   10
Guest #11   → Room   11
Guest #12   → Room   12
Guest #13   → Room   13
Guest #14   → Room   14
Guest #15   → Room   15
Guest #16   → Room   16
Guest #17   → Room   17
Guest #18   → Room   18
```

รูปที่ 18 ระบบแสดงรายชื่อแขกอัตโนมัติและแขกที่เพิ่มด้วยมือ

5. เลือกเมนู 4 เพื่อค้นหาห้อง โดยการใส่หมายเลขห้อง

```

> Select option: 4
> Enter room number to search: 12
🔍 Searching for room 12...
-----
✅ Room Found!
Guest ID      : 12
Room Number   : 12
Group Index   : 1
Travel Path   : Quay 1, Boat 1, Bus 3, Seat 3
-----
🕒 Runtime: 0.001097 seconds

```

รูปที่ 19 ระบบแสดงข้อมูลของหมายเลขห้องที่ผู้ใช้กรอก

6. เลือกเมนู 5 เพื่อลบห้องที่ต้องการโดยการใส่หมายเลขห้อง

```

=====
🏨 HOTEL MANAGEMENT SYSTEM
=====
1. ➕ Add guest group
2. 👤 Add room and people manually
3. 📄 Print sorted room list
4. 🔍 Search room
5. 🗑 Remove room
6. 🕒 Print all code runtime
7. 📊 Print memory usage
8. 💾 Save to file
x. 🚪 Exit
=====
> Select option: 5
> Enter room number to remove: 12
✅ Successfully removed room 12
🕒 Runtime: 0.000198 seconds

```

รูปที่ 20 ระบบแสดงลบข้อมูลของหมายเลขห้องที่ผู้ใช้กรอก

7. เลือกเมนู 6 เพื่อแสดงการทำงานเวลารวมของระบบทั้งหมด

```
=====
HOTEL MANAGEMENT SYSTEM
=====
1. + Add guest group
2. 👤 Add room and people manually
3. 📄 Print sorted room list
4. 🔍 Search room
5. 🗑 Remove room
6. ⌚ Print all code runtime
7. 📊 Print memory usage
8. 💾 Save to file
x. 🚪 Exit
=====
> Select option: 6
=====
⌚ Total Runtime: 0.044367 seconds
=====
```

รูปที่ 21 ระบบแสดงเวลาการทำงานรวมของระบบ

8. เลือกเมนู 7 เพื่อแสดงหน่วยข้อมูลที่ใช้ทั้งหมด

```
=====
HOTEL MANAGEMENT SYSTEM
=====
1. + Add guest group
2. 👤 Add room and people manually
3. 📄 Print sorted room list
4. 🔍 Search room
5. 🗑 Remove room
6. ⌚ Print all code runtime
7. 📊 Print memory usage
8. 💾 Save to file
x. 🚪 Exit
=====
> Select option: 7
=====
📊 Memory Usage Report
=====
Peak Memory   : 71.02 KB
Current Memory: 15.09 KB
=====
```

รูปที่ 22 ระบบแสดงหน่วยข้อมูลที่ใช้ทั้งหมด

9. สุดท้ายผู้ใช้เลือกเมนู 8 เพื่อบันทึกข้อมูลลงไฟล์ “hotel_rooms.csv”

```
=====
HOTEL MANAGEMENT SYSTEM
=====
1. + Add guest group
2. 👤 Add room and people manually
3. 📄 Print sorted room list
4. 🔍 Search room
5. 🗑️ Remove room
6. ⚙️ Print all code runtime
7. 📊 Print memory usage
8. 💾 Save to file
x. 🚪 Exit
=====

> Select option: 8

> Enter filename (default: hotel_rooms): hotel_rooms

✅ Hotel data successfully saved to 'hotel_rooms'
   Total records: 121
   ⚙️ Runtime: 0.023683 seconds
```

รูปที่ 23 แสดงการบันทึกข้อมูลลงไฟล์ hotel_rooms

GuestID	RoomNumber	Group	TravelPath	Type
0	0	1	(1,1,1,1)	Auto
1	1	1	(1,1,1,2)	Auto
2	2	1	(1,1,1,3)	Auto
3	3	1	(1,1,1,4)	Auto
4	4	1	(1,1,1,5)	Auto
5	5	1	(1,1,2,1)	Auto
6	6	1	(1,1,2,2)	Auto
7	7	1	(1,1,2,3)	Auto
8	8	1	(1,1,2,4)	Auto
9	9	1	(1,1,2,5)	Auto
10	10	1	(1,1,3,1)	Auto
11	11	1	(1,1,3,2)	Auto
12	12	1	(1,1,3,3)	Auto
13	13	1	(1,1,3,4)	Auto

รูปที่ 24 แสดงข้อมูลที่อยู่ในไฟล์ที่บันทึก

บทที่ 4

สรุปผลและข้อเสนอแนะ

4.1 สรุปผล

จากการพัฒนาโปรแกรมสามารถสรุปผลได้ว่า ระบบที่ออกแบบขึ้นสามารถจำลองการทำงานของโรงแรมในลักษณะการจัดการห้องพักได้อย่างมีประสิทธิภาพ โดยมีการใช้แนวคิด การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) เป็นหลักในการออกแบบ เพื่อให้การจัดการข้อมูลมีความเป็นระบบและสามารถขยายผลได้ในอนาคต

ผลการทดสอบจำลองการทำงานพบว่า ระบบสามารถเพิ่มกลุ่มแขกหลายชุดได้โดยไม่เกิดข้อผิดพลาดในการคำนวณหมายเลขห้อง การลบหรือค้นหาข้อมูลทำได้ถูกต้องครบถ้วน และมีการจัดการทรัพยากรของหน่วยความจำได้อย่างเหมาะสม

4.2 ข้อเสนอแนะ

แม้ระบบจะสามารถทำงานได้อย่างสมบูรณ์ในระดับต้นแบบแต่ยังมีแนวทางในการพัฒนาเพิ่มเติมเพื่อเพิ่มประสิทธิภาพและความสมบูรณ์ของระบบ ดังนี้

1. เพิ่มส่วนติดต่อผู้ใช้แบบกราฟิก เพื่อให้ผู้ใช้งานทั่วไปสามารถสั่งงานได้สะดวกยิ่งขึ้น โดยไม่ต้องใช้คำสั่งผ่าน Terminal หรือ Command Line
2. ประสิทธิภาพการทำงาน ปรับปรุงอัลกอริทึมการค้นหาและการจัดเก็บข้อมูลให้ทำงานได้รวดเร็วยิ่งขึ้น เมื่อมีจำนวนแขกและห้องเพิ่มขึ้นในระดับใหญ่
3. การบันทึกและโหลดข้อมูลอัตโนมัติ เพื่อให้ผู้ใช้งานต่อเนื่องได้แม้ปิดโปรแกรม โดยไม่ต้องบันทึกหรือโหลดไฟล์ด้วยตนเอง

บรรณานุกรม

แม่ทเล้าให้ฟัง | MLHF. (2568, 1 ตุลาคม). ปริศนา Infinite Hotel Paradox | Hilbert's Infinity Hotel. YouTube. <https://www.youtube.com/watch?v=HLTjDXT9SqQ>

ภาคผนวก

ลิ้งค์ github ที่มี source code ของโปรเจกต์นี้

-https://github.com/Nonnnchun/OOD_project.git