ชื่อ – นามสกุล ชิษณุพงศ์ บุญมา รหัสนักศึกษา 64015031

Data Structures and Algorithm

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

การทดลองที่ 9 : Recursive

จุดประสงค์

1. นักศึกษาเข้าใจการทำงานของ Recursive Operation และสามารถออกแบบฟังก์ชันที่ทำงานเป็นแบบ Recursive ตามต้องการได้

ตอนที่ 1 : การออกแบบการทำงานและสร้างฟังก์ชันที่ทำงานแบบ Recursive

1. ให้นักศึกษาเขียนฟังก์ชัน

   def isPalindrome(str):

   โดย str เป็น string ที่ต้องการตรวจสอบและส่งค่าเป็น true เมื่อ str เป็น palindrome และให้ค่า false เมื่อ str ไม่เป็น palindrome

แนวคิดในการออกแบบการทำงานแบบ recursive

การทำให้ Function เรียกตัวของมันเองโดยมีการวนซ้ำไปเรื่อยๆ ซึ่ง Recursive Function จะหาจุดวนกลับมาทำงานตัวมันเอง คือให้มัน Return isPalindrome ไปเรื่อยๆ จนกว่าจะได้ Base Case ที่ต้องการจนครบข้อมูลที่อยู่ใน List

กรณีของ str ที่เป็น base case คืออะไร และ ที่ base case จะมีการทำงานอย่างไร

Base case จะทำการเช็คจำนวน str ที่ได้รับมาก่อนว่า str ใน List <= 1 มั้ย ถ้าทำจนจำนวนข้อมูลที่ได้รับมามีค่า >= 1 ให้ Return ค่า False ออกมา แล้วไปทำอีกเงื่อนไขที่ว่า str[0] != str[-1] ก็จะทำการเช็คตำแหน่งแรกและตำแหน่งสุดท้ายว่าตรงกันไหมแล้วเก็บใน str ก่อนแล้ว Return ค่า False ออกมา เพื่อกลับไปเช็ค recursive case แล้วกลับมาเช็คตัวถัดไป

กรณีของ str ที่เป็น recursive case จะมีการทำงานอย่างไร

recursive case เอาไว้ Return ตัว Function เช่นถ้าเช็คแล้วข้อมูลที่รับมาจาก base case ว่าถ้าข้อมูลที่เช็คแล้วตำแหน่งแรกกับตำแหน่งสุดท้ายเหมือนกันหรือตรงกัน ก็จะให้หาตัวถัดไปจนกว่า base case จะเทียบข้อมูลจนครบว่าตรงกันหมด แล้ว Return Output ออกไป

ฟังก์ชันที่ได้

```
def isPalindrome(str):
    if len(str) <= 1:  # base case
        return True
    elif str[0] != str[-1]:  # base case
        return False
    else:  # recursive cases
        return isPalindrome(str[1:-1])

checklist = ["abcdcba", "atoyota", "kmitl",
             "manassanan", "programming", "fundamental"]
for i in checklist:
    print(isPalindrome(i))
```

ทดสอบการทำงานของฟังก์ชัน

str	**ผลลัพธ์**
abcdcba	True
atoyota	True
kmitl	False
manassanan	False
programming	False
fundamental	False

2. ให้นักศึกษาเขียนฟังก์ชัน

def isAscending(list_of_integer):

โดย list_of_integer เป็น list ของข้อมูลตัวเลขจำนวนเต็มที่ต้องการตรวจสอบว่าเป็น list ที่เรียงลำดับจากน้อยไปมากแล้วหรือไม่ และส่งค่าเป็น true เมื่อข้อมูลใน list_of_integer เรียงลำดับจากน้อยไปมาก และให้ค่า false เมื่อ list_of_integer ไม่ได้เรียงลำดับจากน้อยไปมาก

แนวคิดในการออกแบบการทำงานแบบ recursive

การทำให้ Function เรียกตัวของมันเองโดยมีการวนซ้ำไปเรื่อยๆ ซึ่ง Recursive Function จะหาจุดวนกลับมาทำงานตัวมันเอง คือให้มัน Return isAscending ไปเรื่อยๆ จนกว่าจะได้ Base Case ที่ต้องการ จนครบข้อมูลที่อยู่ใน List

กรณีของ list ที่เป็น base case คืออะไร และ ที่ base case จะมีการทำงานอย่างไร

Base case จะทำการเช็คจำนวน str ที่ได้รับมาก่อนว่าข้อมูล str ใน List <= 1 ไหม ถ้าจนจำนวนข้อมูลที่ได้รับมามีค่า >= 1 ให้ Return ค่า False ออกมา แล้วไปทำเงื่อนไขของ recursive case

กรณีของ list ที่เป็น recursive case จะมีการทำงานอย่างไร

recursive case เอาไว้ Return ตัว Function เช่นถ้าเช็คแล้วข้อมูลที่รับมาจาก base case recursive case จะเช็คข้อมูลตำแหน่งแรกว่าน้อยกว่าตำแหน่งสุดท้ายไหม ถ้าใช่ก็จะให้หาตัวถัดไปจนกว่า base case จะเทียบข้อมูลจนครบ แล้ว Return Output ออกไป

ฟังก์ชันที่ได้

```
def isAscending(list_of_integer):
    if len(list_of_integer) < 1:  # base case
        return True
    elif list_of_integer[0] <= list_of_integer[-1]:  # recursive cases
        return isAscending(list_of_integer[1:-1])
    else:  # base case
        return False


checklist = [[1, 2, 3, 4, 5, 6, 7], [3, 4, 2, 5, 6, 1, 2], [9, 8, 7, 6, 5, 4], [
    0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5], [6, 7, 8, 9, 10, 11, 12], [6, 3, 8, 7, 9, 2, 3, 1, 5]]


for i in checklist:
    print(isAscending(i))
```

ทดสอบการทำงานของฟังก์ชัน

list_of_integer	**ผลลัพธ์**
[1,2,3,4,5,6,7]	True
[3,4,2,5,6,1,2]	False
[9,8,7,6,5,4]	False
[0,0,1,1,2,2,3,3,4,4,5,5]	True
[6,7,8,9,10,11,12]	True
[6,3,8,7,9,2,3,1,5]	False

3. ให้นักศึกษาเขียนฟังก์ชัน

def group_of_no_1(island_list,point_no):

โดย

island_list คือ list ของตัวเลข 1 และ 0 ที่เรียงตัวกัน

point_no คือ ตำแหน่งที่ต้องการตรวจสอบ

ฟังก์ชัน group_of_no_1 จะหาจำนวนตัวเลข 1 ที่ตำแหน่ง point_no ว่ามีจำนวนเลข 1 ติดกันกี่ตัว

ถ้า island_list = [0,1,1,1,1,0,0,1,1,0,1,0] และ point_no = 2 จะได้ผลลัพธ์คือ 4 เพราะมีตัวเลข 1 เรียงกัน 4 ตัวที่ตำแหน่งนั้น หรือถ้า point_no = 8 จะได้ผลลัพธ์เป็น 2

แนวคิดในการออกแบบการทำงานแบบ recursive

ทำให้โปรแกรมเช็คตำแหน่งว่า ในตำแหน่งที่ต้องการนั้นมี 0,1 อยู่กี่จำนวน โดยเอา point_no ตำแหน่งตัวที่ต้องการชี้มาเช็คกับตำแหน่งของข้อมูลใน List เมื่อเช็คตำแหน่งเจอแล้ว ก็จะไปเช็คตำแหน่งตัวที่อยู่ด้านซ้ายหรือด้านขวาของมันว่า มีตัวที่เหมือนตัวมันเองมั้ยถ้ามีก็เก็บไว้แล้ว ถ้าสมมติด้านซ้ายของตัวมันไม่มีเหมือน แต่ด้านขวาเหมือนมันก็จะไปเช็คตัวถัดไปของมันอีกว่ามีอยู่มั้ย ทำแบบนี้ไปเรื่อยจนไม่เจอแล้ว ก็จะเอาจำนวนที่ได้มาดูว่ามีกี่จำนวน

กรณีของ base case คืออะไร และ ที่ base case จะมีการทำงานอย่างไร

```
if point_no >= len(island_list) or point_no <= 0:  # base case
        return 0
```

- จะเอา point_no มาเทียบกับข้อมูลว่า >= ข้อมูลที่รับมาไหม ถ้าไม่ก็จะเป็น False หรือ ถ้า point_no <= 0 ไหมถ้าไม่ก็จะเป็น False แต่ถ้าเช็คแล้วเข้าเงื่อนไขให้ Return เป็น 0

```
if island_list[point_no] == 0 or island_list[point_no] == 2:  # base case
        return 0
```

จะเอา point_no มาเทียบกับข้อมูลว่า == ข้อมูลที่รับมาไหม ถ้าไม่ก็จะเป็น False หรือ ถ้า point_no == 2 ไหมถ้าไม่ก็จะเป็น False แต่ถ้าเช็คแล้วเข้าเงื่อนไขให้ Return เป็น 0

กรณีของ recursive case จะมีการทำงานอย่างไร

```
if island_list[point_no] == 1:  # recursive cases
        island_list[point_no] = 2
        return 1 + group_of_no_1(island_list, point_no-1) + group_of_no_1(island_list,
point_no+1)
```

ถ้า island_list[point_no] == 1 จะเป็น True จากนั้นจะเปลี่ยนตำแหน่งที่เช็คใหม่นั้นเป็น 2 และ Return ให้กลับไปทำที่ Function ใหม่เพื่อหาตัวถัดไป จนกว่า island_list[point_no] ไม่เท่ากับ 1

ฟังก์ชันที่ได้

```
def group_of_no_1(island_list, point_no):
    if point_no >= len(island_list) or point_no <= 0:  # base case
        return 0
    if island_list[point_no] == 0 or island_list[point_no] == 2:  # base case
        return 0
    if island_list[point_no] == 1:  # recursive cases
        island_list[point_no] = 2
        return 1 + group_of_no_1(island_list, point_no-1) + group_of_no_1(island_list,
point_no+1)
print(group_of_no_1([1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0], 1))
print(group_of_no_1([1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0], 5))
print(group_of_no_1([1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1], 4))
print(group_of_no_1([1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1], 10))
print(group_of_no_1([1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1], 1))
print(group_of_no_1([0, 1, 0, 1, 0, 1, 0, 1, 0, 1], 7))
```

ทดสอบการทำงานของฟังก์ชัน

island_list	point_no	ผลลัพธ์
[1,1,1,1,0,0,0,1,1,1,0,0]	1	3
[1,1,1,1,0,0,0,1,1,1,0,0]	5	0
[1,0,1,1,1,0,0,0,1,1,1,1,1,1]	4	3
[1,0,1,1,1,0,0,0,1,1,1,1,1,1]	10	6
[1,0,1,1,1,0,0,0,1,1,1,1,1,1]	1	0
[0,1,0,1,0,1,0,1,0,1]	7	1

4. ให้นักศึกษาเขียนฟังก์ชัน

def valid_parentheses(str):

โดย

str คือ string ของวงเล็บในรูปแบบต่างๆ

ฟังก์ชั่น valid_parentheses จะตรวจสอบว่าวงเล็บใน str มีการจัดลำดับได้อย่างถูกต้องหรือไม่

ถ้า str = “(()(()()))”จะได้ผลลัพธ์คือ True , ถ้า str = “(((()()(())))”จะได้ผลลัพธ์คือ False

แนวคิดในการออกแบบการทำงานแบบ recursive

จัดลำดับของวงเล็บว่า วงเล็บที่ input เข้ามานั้นเรียงลำดับถูกต้องตามหลักคณิตศาสตร์หรือไม่โดยการทำงานของ Recursive คือ นำข้อมูลวงเล็บที่รับเข้ามาใน input มาเช็คตำแหน่งก่อน จากนั้นให้เอาตำแหน่งที่เช็คออกมาแล้วไปเทียบกับเงื่อนที่ตั้งไว้ว่าแต่ละตำแหน่งที่รับมาเข้าเงื่อนไขไหน ก็จะเก็บข้อมูลไว้จนทำครบทุกตำแหน่งใน input

กรณีของ base case คืออะไร และ ที่ base case จะมีการทำงานอย่างไร

```
if i == len(str):  # base case
        return cnt == 0
```

ถ้าไม่เข้าเงื่อนไขให้เป็น False แล้วจบการทำงาน

ถ้าไม่เข้าเงื่อนไขให้เป็น return cnt == 0

```
if cnt < 0:  # base case
        return False
```

ถ้าไม่เข้าเงื่อนไขให้เป็น False แล้วจบการทำงาน

ถ้าเข้าเงื่อนไขให้เป็น return False

กรณีของ recursive case จะมีการทำงานอย่างไร

```
if str[i] == "(":  # recursive cases
        return valid_parentheses(str, i + 1, cnt + 1)
```

ถ้าไม่เข้าเงื่อนไขให้เป็น False แล้วจบการทำงาน

ถ้าเข้าเงื่อนไขให้เป็น return กลับไปที่ function

```
elif str[i] == ")":  # recursive cases
        return valid_parentheses(str, i + 1, cnt - 1)
return valid_parentheses(str, i + 1, cnt)
```

ถ้าไม่เข้าเงื่อนไขให้เป็น False แล้วจบการทำงาน

ถ้าเข้าเงื่อนไขให้เป็น return กลับไปที่ function

ฟังก์ชันที่ได้

```
def valid_parentheses(str, i=0, cnt=0):
    if i == len(str):  # base case
        return cnt == 0
    if cnt < 0:  # base case
        return False
    if str[i] == "(":  # recursive cases
        return valid_parentheses(str, i + 1, cnt + 1)
    elif str[i] == ")":  # recursive cases
        return valid_parentheses(str, i + 1, cnt - 1)
    return valid_parentheses(str, i + 1, cnt)


for ckeck in ["(()()(()))", "((()()", "())()()(", "((((()))(()))))", "()()(((())))", "()"]:
    print("str =", ckeck, "=", valid_parentheses(ckeck))
```

ทดสอบการทำงานของฟังก์ชัน

str	**ผลลัพธ์**
(()()(()()))	True
(((()()	False
())()()(	False
(((()))(((())))	True
()()((((()))))	True
()	True

ชื่อ – นามสกุล ชิษณุพงศ์ บุญมา รหัสนักศึกษา 64015031