การบ้าน Data Structure for Optimal Solution เรื่อง Linked-List

ขอให้นิสิตอธิบายกระบวนการอย่างเป็นขั้นเป็นตอน (จะเป็นการอธิบายด้วยภาพ หรือจะเป็นการอธิบายด้วย pseudo code ก็ได้) ต่อการกระทำตามโจทย์ที่กำหนดให้

1. ถ้าคุณมี singly linked list ที่มีโหนด A -> B -> C -> D -> E (โดย A เป็น head) จงอธิบายขั้นตอนในการแทรกโหนดใหม่ X ระหว่าง C และ D

ขั้นตอนในการแทรก Node X ระหว่าง C และ D

1. ค้นหา Node C:
   - เริ่มต้นที่ head และเดินลิสต์จนกระทั่งถึง node C
2. สร้าง Node ใหม่ X:
   - สร้าง node ใหม่ X และตั้งค่า X->next เป็น node D
3. อัปเดตการเชื่อมโยง:
   - ตั้งค่า C->next เป็น node X เพื่อเชื่อมโยง node C กับ node X

ผลลัพธ์: A -> B -> C -> X -> D -> E

Pseudo Code:

```
function insertAfter(head, targetValue, newValue):
    current = head

    while current != NULL and current->data != targetValue:
        current = current->next

    if current != NULL:
        X = createNewNode(newValue)
        X->next = current->next
        current->next = X
    return head

function createNewNode(value):
    node = allocateMemoryForNode()
    node->data = value
    node->next = NULL
    return node
```

2. ต่อจากข้อ 1 ให้ลบ node สุดท้าย (node E) ออกจาก list นี้

ขั้นตอนการลบ Node สุดท้าย (Node E)

1. ตรวจสอบลิสต์:
   - ถ้า head เป็น NULL, ลิสต์ว่าง ไม่ต้องทำอะไร
   - ถ้า head->next เป็น NULL, ลิสต์มี node เดียว, ลบ node ตั้งค่า head เป็น NULL

2. เริ่มต้นที่หัวของลิสต์:
   - ตั้งค่า previous เป็น NULL และ current เป็น head
   - ใช้ลูป while เพื่อหาตำแหน่งของ node ก่อนหน้านี้สุดท้าย
   - ลูปจะหยุดเมื่อ current->next เป็น NULL

3. อัปเดตและลบ:
   - ตั้งค่า previous->next เป็น NULL
   - ใช้ free(current) เพื่อลบ node สุดท้าย

ผลลัพธ์: A -> B -> C -> X -> D

Pseudo Code:

```
function deleteLastNode(head):
    if head is NULL:
        return NULL
    if head->next is NULL:
        free(head)
        return NULL

    previous = NULL
    current = head

    while current->next != NULL:
        previous = current
        current = current->next

    previous->next = NULL
    free(current)
    return head
```

3. ต่อจากข้อ 2 ให้เพิ่ม node Y เข้าไปเป็น node แรกของ list นี้

ขั้นตอนการเพิ่ม Node ใหม่ (Node Y) เป็น Node แรก

1. สร้าง Node ใหม่ (Y):
   - สร้าง node ใหม่ Y และตั้งค่า Y->next เป็น head
2. อัปเดตหัวของลิสต์:
   - ตั้งค่า head ให้ชี้ไปที่ Y

ผลลัพธ์: Y -> A -> B -> C -> X -> D

Pseudo Code:

```
function addNodeAtHead(head, newValue):
  Y = createNewNode(newValue)
  Y->next = head
  head = Y
  return head

function createNewNode(value):
  node = allocateMemoryForNode()
  node->data = value
  node->next = NULL
  return node
```

4. ต่อจากข้อ 4 ให้อธิบายกระบวนการค้นหาข้อมูล Z ใน list นี้

ขั้นตอนการค้นหาข้อมูล Z

1. เริ่มต้นที่โหนด head:
   - ตั้งค่า current ให้ชี้ไปที่โหนด head
2. วนลูปผ่านลิสต์:
   - ใช้ลูป while เพื่อเดินผ่านลิสต์จนกว่า current จะเป็น NULL หรือจนกว่าจะพบข้อมูลที่ต้องการ
3. ตรวจสอบข้อมูลในโหนดปัจจุบัน:
   - ตรวจสอบว่า current->data ตรงกับ Z หรือไม่
   - ถ้าตรง, แสดงข้อความ "Data found" และหยุดการค้นหา
4. ไปที่โหนดถัดไป:
   - ถ้าไม่ตรง, ตั้งค่า current เป็น current->next เพื่อไปยังโหนดถัดไปในลิสต์
5. ถ้าลูปสิ้นสุด:
   - หากลูปสิ้นสุดลงและยังไม่พบข้อมูล Z, แสดงข้อความ "Data not found”

Pseudo Code:

```
function findNode(head, Z):

    current = head


    while current != NULL:

        if current->data == Z:

            print "Data found"

            return


        current = current->next


    print "Data not found”
```

5. จงอธิบายแนวคิดของ circular linked list และยกตัวอย่างการใช้งานที่เหมาะสม

Circular Linked List:

- Circular Linked List เป็นโครงสร้างข้อมูล linked list ที่โหนดสุดท้าย (tail) จะชี้ pointer next กลับไปยังโหนดแรก (head) ทำให้ไม่มีโหนดใดที่ next เป็น NULL.
- Circular Linked List ไม่มีจุดเริ่มต้นหรือจุดสิ้นสุดที่ชัดเจน เราสามารถเดินทางผ่าน list ไปได้เรื่อย ๆ โดยจะกลับมาที่จุดเริ่มต้นอัตโนมัติเมื่อถึงโหนดสุดท้าย

ตัวอย่างการใช้งานที่เหมาะสม:

- **การวนซ้ำข้อมูลที่ต้องการการจัดการในรูปแบบวงกลม**: เช่นการทำงานในระบบ scheduling ของ CPU หรือการเล่นเกมที่มีผู้เล่นหมุนเวียนกันไปเรื่อย ๆ
- **การจัดเก็บข้อมูลของ buffer แบบหมุน**: เช่นการใช้งานใน Circular Buffer สำหรับเก็บข้อมูลที่ต้องการการจัดการแบบ FIFO และลบข้อมูลเก่าเมื่อ buffer เต็มโดยการหมุนกลับไปที่จุดเริ่มต้น