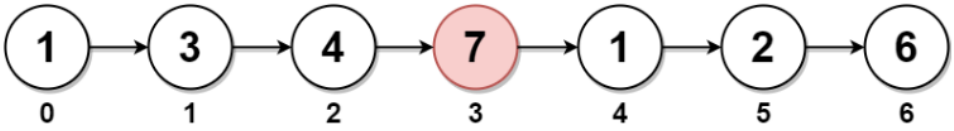
PD-W17-ANS

Q1:Delete the middle Node of a Linked List

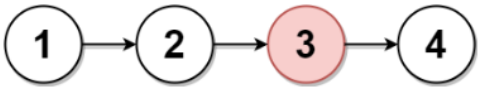
You are given the head of a linked list. Delete the middle node, and return the head of the modified linked list.
(給定一條鏈結串列，並刪除中間的節點)

Example 1:




```
Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list. The indices of the nodes
are written below.
Since n = 7, node 3 with value 7 is the middle node, which is marked in red.
We return the new list after removing this node.
```

Example 2:




```
Input: head = [1,2,3,4]
Output: [1,2,4]
Explanation:
The above figure represents the given linked list.
For n = 4, node 2 with value 3 is the middle node, which is marked in red.
```

code:

PD1-W17-ANS

```c
#include<stdio.h>
#include<stdlib.h>

struct LinkedList{
    int data;
    struct LinkedList *next;
};
struct LinkedList* Insert(struct LinkedList* head,int newData){
    //TODO
    struct LinkedList* newNode = (struct LinkedList*)malloc(sizeof(struct LinkedList));
    struct LinkedList* last = head;
    newNode->data = newData;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        return head;
    }

    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
    return head;
    //
}
void printLinkedList(struct LinkedList* head) {
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
void deleteMiddle(struct LinkedList* head){
    if(!head->next) return;

    struct LinkedList *fast = head->next;
    struct LinkedList *slow = head;

    while(fast && fast->next){
        fast = fast->next->next;
        if(!fast) break;
        slow = slow->next;
    }
    struct LinkedList *q = slow->next;
    slow->next = slow->next->next;
    free(q);
    //return head;
}

int main(){
    struct LinkedList *head=NULL;
    int i, total, num;
    scanf("%d", &total);
    for(i = 0; i < total; i++){
        scanf("%d", &num);
        head = Insert(head,num);
```

```
        }
    printLinkedList(head);
    //TO DO
        deleteMiddle(head);
    printLinkedList(head);
}
```

Q2:Doubly Linked List

雙向鏈結串列

每個節點會變成一個資料欄與兩個指標欄，讓資料可以從頭或尾巴開始找，優點是可以讓被破壞或遺失的節點被找回來，但在追加或刪除資料時，必須更動比單向鏈結串列更多的指標次數。

code:

```c
#include <stdio.h>
#include <stdlib.h>

// 定義雙向鏈表結構
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// 在雙向鏈表中插入新節點
void insertNode(struct Node** head, int newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    newNode->prev = NULL;
    newNode->next = *head;

    if (*head != NULL) {
        (*head)->prev = newNode;
    }

    *head = newNode;
}

// 在雙向鏈表中刪除給定值的節點
void deleteNode(struct Node** head, int delData) {
    struct Node* current = *head;

    while (current != NULL && current->data != delData) {
        current = current->next;
    }

    if (current == NULL) {
        printf("Node with data %d not found\n", delData);
        return;
    }

    if (current->prev != NULL) {
        current->prev->next = current->next;
    } else {
        *head = current->next;
    }

    if (current->next != NULL) {
        current->next->prev = current->prev;
    }

    free(current);
}

// 顯示雙向鏈表
void printList(struct Node* head) {
    while (head != NULL) {
        printf("%d <-> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
```

```
}

// 釋放雙向鏈表內存
void freeList(struct Node* head) {
    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    struct Node* head = NULL;

    insertNode(&head, 1);
    insertNode(&head, 2);
    insertNode(&head, 3);

    printf("Original Double Linked List: ");
    printList(head);

    deleteNode(&head, 2);

    printf("Double Linked List after deleting node with data 2: ");
    printList(head);

    // Free memory
    freeList(head);

    return 0;
}

```