School of Computer Science

CIS*2520: Data Structures

Fall 2024, Lab 3

Week of Sept. 30 to Oct. 04

---

1 Linked Lists

1. Consider the following function that takes reference to head of a Doubly Linked List as parameter. Assume that a node of doubly linked list has previous pointer as prev and next pointer as next.

```
void fun (struct node **head_ref)
{
     struct node *temp = null;
     struct node *current = *head_ref;

     while (current ≠ NULL)
     {
          temp = current->prev;
          current->prev = current->next;
          current->next = temp;
          current = current->prev;
     }
     if(temp ≠ NULL )
                  *head_ref = temp->prev;
  }
```

Assume that reference of head of following doubly linked list is passed to above function

1 ↔ 2 ↔ 3 ↔ 4 ↔ 5 ↔ 6.

What should be the modified linked list after the function call?

A. 2↔1↔4↔3↔6↔5

B. 5 ↔ 4 ↔ 3 ↔ 2 ↔ 1 ↔ 6

C. 6 ↔5 ↔4 ↔3 ↔ 2 ↔ 1

D. 6 ↔ 5 ↔ 4 ↔ 3 ↔ 1 ↔ 2

Explanation: Code reverses the doubly linked list

2. Consider the linked list 1→ 2 → 3 → 4 → 5 → 6. What is the output of the following function for a start point to the first node of this linked list?

```
void fun(struct node* start)
{
    if(start == NULL)
          return;
    printf("%d ", start->data);

    if(start->next != NULL )
          fun(start->next->next);
    printf("%d ", start->data);

}
```

A. 146641

B. 135135

C. 1235

D. 135531

Explanation: Traverse 2 nodes at a time and print element. After traversing and printing node 5, the next call will try to traverse 2 nodes, return null. From there, we come out of the recursive calls and print the

3. Write a program to create a single linked list in which nodes are 1 ⇒ 2 ⇒ 3 ⇒ 4. Then, insert node 5 in the middle of the linked list after node 3 and return the new linked list.

Explanation: Reference the lecture slides for basic flow. Ensure that prev and next pointers of all nodes are correct after insertion.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node *next;
};

// Additional function to view results – For TA’s
void printList(struct Node* head) {
   struct Node* temp = head;
   while (temp != NULL) {
       printf("%d -> ", temp->data);
       temp = temp->next;
   }
   printf("NULL\n");
}

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

head = (struct Node *)malloc(sizeof(struct Node));
second = (struct Node *)malloc(sizeof(struct Node));
third = (struct Node *)malloc(sizeof(struct Node));
fourth = (struct Node *)malloc(sizeof(struct Node));

head->data = 1
head->next = second
second->data = 2
second->next = third;
third->data = 3;
third->next = fourth;
fourth->data = 4;
fourth->next = NULL;

struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode->data = 5;
newNode->next = third->next;
third->next = newNode;
printList(head);
return 0;
}
```

4. Consider a doubly linked list with 4 nodes. The input of these nodes is as follows:

   Input data for node 1: 4

   Input data for node 2: 6

   Input data for node 3: 8

   Input data for node 4 :1

Write a program and implement the following questions on this linked list.

A. Delete a node from the last of this doubly linked list.

```
void deleteLastNode(struct Node** head) {
    struct Node* tmp = *head;

    while (tmp->next != NULL) {
        tmp = tmp->next;
    }
    tmp->prev->next = NULL;

    free(tmp); // Optional Free
```

```
        }

```

A. Find the maximum value from this doubly linked list.

```
int findMax(struct Node* head) {
    int max = head->data;
    struct Node* tmp = head->next;

    while (tmp != NULL) {
        if (tmp->data > max) {
            max = tmp->data;
        }
        tmp = tmp->next;
    }

    return max;
}
```

5. We maintain a sorted list of n integers $1,2,...,n$. Assume that we need to perform two insertions, one is $x = 0$, and the other is $y = n+1$. We need to maintain the list sorted after the insertion. So, the list after inserting x is

$$0,1,2,...,n$$

and the list after inserting *y* is

$$1,2,...,n,n+1$$

Assume that we traverse the list from the first element to the last to find out where we shall insert *x* and *y*. Please calculate the total number of operations for inserting *x* and *y* if you implement the list as:

A. Singly-linked list

Explanation:

Insert at x=0 – Will always take 1 operation, making the next pointer of the new node point to the original head of the linked list. Will be an O(1)/constant time complexity.

Reference Implementation: https://www.geeksforgeeks.org/insert-a-node-at-front-beginning-of-a-linked-list/.

Insert at y=n+1 – Need to traverse the entirety of the list(O(n)) and finally make the next pointer of last element point to the new node(O(1)). Total operations can be viewed as n+1 or O(N) time complexity.

Reference Implementation: https://www.geeksforgeeks.org/insert-node-at-the-end-of-a-linked-list/

B. Array

Explanation:

Insert at x=0 – Inserting at 0 takes constant time(O(1)). Must shift n elements to the right which takes linear time(O(n)). Total number of operations can be viewed as n+1, comes out to O(n) time complexity.

Insert at y=n+1 – Will always take only 1 operation, which just requires you to know the index of the final position. If the current array is full (size n), it also requires moving all existing elements to the new created array (size >=n+1), and the total operations can be viewed as =n+1, comes out to O(n) time complexity. If the current array is not full (size>=n+1), the total operations can be viewed as 1, comes out to O(1) time complexity.