**School of Computer Science**

**CIS\*2520: Data Structures**

**Fall 2024, Lab 3**

**Week of Sept. 30 to Oct. 04**

# 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?

1. 2⇿1⇿4⇿3⇿6⇿5
2. 5 ⇿ 4 ⇿ 3 ⇿ 2 ⇿ 1 ⇿6
3. 6 ⇿5 ⇿4 ⇿3 ⇿ 2 ⇿ 1 v
4. 6 ⇿ 5 ⇿ 4 ⇿ 3 ⇿ 1 ⇿ 2
5. 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);

}

* 1. 146641
  2. 135135
  3. 1235
  4. 135531 -> stack 재귀함수 다시 역방향으로 돌아감

1. 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.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

return newNode;

};

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 = createNode(1);

head->next = createNode(2);

head->next->next = createNode(3);

head->next->next->next = createNode(4);

struct Node\* temp = head;

while (temp->data != 3) {

temp = temp->next;

}

struct Node\* five = createNode(5);

struct Node\* temp2 = temp->next;

temp->next = five;

temp->next->next = temp2;

printList(head);

return 0;

}

1. 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.

* 1. Delete a node from the last of this doubly linked list.
  2. Find the maximum value from this doubly linked list.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

struct Node\* previous;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

newNode->previous = NULL;

return newNode;

};

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 = createNode(4);

head->next = createNode(6);

head->next->previous = head;

head->next->next = createNode(8);

head->next->next->previous = head->next;

head->next->next->next = createNode(1);

head->next->next->next->previous = head->next->next;

struct Node\* temp = head;

int max = head->data;

while (temp->next != NULL) {

if (max < temp->data) {

max = temp->data;

}

temp = temp->next;

}

printList(head);

//Deleting

temp->previous->next = NULL;

free(temp);

printList(head);

//Maximum value

printf("%d", max);

return 0;

}

1. 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:

* 1. Singly-linked list
  2. Array

### Problem Breakdown:

We need to insert two values, x=0*x*=0 and y=n+1*y*=*n*+1, into a sorted list of n*n* integers. After each insertion, the list must remain sorted. We are asked to calculate the number of operations needed to find the insertion points and insert these values if the list is implemented as:

1. A singly linked list.
2. An array.

### Key Assumptions:

* **Traversal cost**: The number of operations for each insertion corresponds to the number of elements we must traverse in the list to find the correct insertion position.
* **Insertion cost**: The cost of inserting into a singly linked list and an array differs based on how the structure behaves when elements are added.

### A. Singly-Linked List

In a singly-linked list, we can only traverse the list in one direction, from the head (first element) to the tail (last element). The time complexity for inserting an element depends on how far the element is from the head.

1. **Inserting x=0*x*=0**:
   * 00 must be inserted at the beginning of the list.
   * Traversal cost: We don't need to traverse the list to insert 00 because it goes at the beginning. This is an **O(1)** operation.
2. **Inserting y=n+1*y*=*n*+1**:
   * y=n+1*y*=*n*+1 must be inserted at the end of the list.
   * Traversal cost: To find the end of the list, we must traverse through all n*n* elements.
   * Traversal takes **O(n)** operations.
3. **Total operations for singly-linked list**:
   * **Inserting x=0*x*=0**: O(1)*O*(1)
   * **Inserting y=n+1*y*=*n*+1**: O(n)*O*(*n*)

Total operations for singly-linked list=O(1+n)=O(n)Total operations for singly-linked list=*O*(1+*n*)=*O*(*n*)

### B. Array

In an array, elements are stored in contiguous memory locations. Inserting an element requires shifting elements to make space for the new element if it's not added at the end. The time complexity for inserting an element depends on the position of the insertion.

1. **Inserting x=0*x*=0**:
   * 00 must be inserted at the beginning of the array. This requires shifting all existing n*n* elements one position to the right.
   * Insertion cost: **O(n)** operations (to shift all elements).
2. **Inserting y=n+1*y*=*n*+1**:
   * y=n+1*y*=*n*+1 must be inserted at the end of the array.
   * Insertion cost: Inserting at the end of the array is an **O(1)** operation because no elements need to be shifted.
3. **Total operations for array**:
   * **Inserting x=0*x*=0**: O(n)*O*(*n*)
   * **Inserting y=n+1*y*=*n*+1**: O(1)*O*(1)

Total operations for array=O(n+1)=O(n)Total operations for array=*O*(*n*+1)=*O*(*n*)