

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

On

DATA STRUCTURES (23CS3PCDST)

Submitted by

DARSHAN P N (1BM24CS090)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019 August-

December 2025

**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by **DARSHAN P N (1BM25CS090)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 20252026. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

ANUSHA V
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index

Sl. No.	Experiment Title	Page No.
1	Push, Pop, Display using Stack	4
2	Infix to postfix Conversion	6
3	Queues	11
4	Insertion in Singly Linked List	14
5	Deletion in Singly Linked List	18
6	Sort, Reverse And Concatenate	22
7	Doubly Linked List	28
8	Binary Search Tree	32
9	Traverse a Graph Using BFS method	36
10	Linear Probing Hash Table Implementation	39
11	Leetcode Problem 1	41
12	Leetcode Problem 2	44
13	Leetcode Problem 3	47
14	Leetcode Problem 4	49
15	Leetcode Problem 5	51
16	Leetcode Problem 6	54
7	Leetcode Problem 7	57

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:

a) Push

b) Pop

c) Display

The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
void push(int st[],int *top)
{
    int item;
    if(*top==STACK_SIZE-1)
        printf("Stack overflow\n");
    else
    {
        printf("\nEnter an item :");
        scanf("%d",&item); (*top)+
        +;
        st[*top]=item;
    }
}
void pop(int st[],int *top)
{ if(*top== -1) printf("Stack underflow\n");
  else
  {
      printf("\n%d item was
      deleted",st[( *top)--]); }
}
void display(int st[],int *top)
{
    int i;
    if(*top== -1) printf("Stack is
    empty\n");
    for(i=0;i<=*top;i++)
        printf("%d\t",st[i]);
} void
main()
{
    int st[10],top=-1, c,val_del;
    while(1)
    { printf("\n1. Push\n2. Pop\n3. Display\n");
      printf("\nEnter your choice :");
      scanf("%d",&c);
```

```

switch(c)
{ case 1: push(st,&top);
  break;
  case 2: pop(st,&top);
  break;
  case 3: display(st,&top);
  break;
  default: printf("\nInvalid choice!!!");
  exit(0);
}
}
}

```

Output:

```

main.c
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #define MAX 5
5
6 //GAGAN NAIK
7 int s[MAX], top = -1;
8
9 void push();
10 void pop();
11 void display();
12
13 void main()
14 {
15     int ch;
16     char str[10];
17     for (;;)
18     {
19         printf("\n IMPLEMENTATION OF STACK OF INTEGERS USING ARRAY \n");
20         printf("\n 1 : PUSH");
21         printf("\n 2 : POP");
22         printf("\n 3 : DISPLAY");
23         printf("\n 4 : EXIT");
24         printf("\n Enter your Choice : ");
25         scanf("%d", &ch);
26
27         switch (ch)
28         {
29             case 1:
30                 push();
31                 break;
32
33             case 2:
34                 pop();
35                 break;
36
37             case 3:
38                 display();
39                 break;
40

```

Output

```

IMPLEMENTATION OF STACK OF INTEGERS USING ARRAY
1 : PUSH
2 : POP
3 : DISPLAY
4 : EXIT
Enter your Choice : 1

Enter the Element : 5

IMPLEMENTATION OF STACK OF INTEGERS USING ARRAY
1 : PUSH
2 : POP
3 : DISPLAY
4 : EXIT
Enter your Choice : 2

The Deleted Element is 5

IMPLEMENTATION OF STACK OF INTEGERS USING ARRAY
1 : PUSH
2 : POP
3 : DISPLAY
4 : EXIT
Enter your Choice : 3

The Stack is Empty

IMPLEMENTATION OF STACK OF INTEGERS USING ARRAY
1 : PUSH
2 : POP
3 : DISPLAY
4 : EXIT
Enter your Choice : 4

*** Code Execution Successful ***

```

Lab Program 2

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).

```
#include <stdio.h>
#include <ctype.h>
#define MAX 100
char stack[MAX];
int top = -1;

void push(char x) {
    stack[++top] = x;
}

char pop() {
    if (top == -1)
        return -1;
    else
        return stack[top--];
}

int priority(char x) {
    if (x == '+' || x == '-')
        return 1;
    if (x == '*' || x == '/')
        return 2;
    return 0;
}

int main() {
    char infix[MAX], postfix[MAX];
    int i = 0, k = 0;
    char ch;

    printf("Enter a valid parenthesized infix expression: ");
    scanf("%s", infix);

    while ((ch = infix[i++]) != '\0') {
```

```

        if (isalnum(ch)) {
            postfix[k++] = ch;
        }

        else if (ch == '(') {
            push(ch);
        }

        else if (ch == ')') {
            while (stack[top] != '(') {
                postfix[k++] = pop();
            }
            pop();
        }

        else {
            while (top != -1 && priority(stack[top]) >= priority(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }

    while (top != -1) {
        postfix[k++] = pop();
    }

    postfix[k] = '\0';

    printf("Postfix expression: %s\n", postfix);

    return 0;
}

```

Output:

```

main.c
1 #include <stdio.h>
2 #include <ctype.h>
3
4 //GAGAN NATH//
5
6 int stack[40];
7 int top = -1;
8
9 void push(int x)
10 {
11     stack[++top] = x;
12 }
13
14 int pop()
15 {
16     return stack[top--];
17 }
18
19 int main()
20 {
21     char exp[20];
22     char *ptr;
23     int n1, n2, n3, num;
24
25     Enter the expression: 245*+
26     The result of expression 245*+ = 18
27
28     === Code Execution Successful ===

```

Lab Program 2

a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>

#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

void insert() {
    int x;
    if (rear == MAX - 1) {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1)
        front = 0;
    printf("Enter element: ");
    scanf("%d", &x);
    queue[++rear] = x;
}

void delete() {
    if (front == -1 || front > rear) {
        printf("Queue Empty\n");
        return;
    }
    printf("Deleted element: %d\n", queue[front++]);
}

void display() {
    int i;
    if (front == -1 || front > rear) {
        printf("Queue Empty\n");
        return;
    }
}
```



```

    }
    for (i = front; i <= rear; i++)
        printf("%d ", queue[i]);
    printf("\n");
}

int main() {
    int choice;
    while (1) {
        printf("1.Insert 2.Delete 3.Display 4.Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1: insert(); break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4: return 0;
            default: printf("Invalid choice\n");
        }
    }
}

```

The screenshot shows a C program titled 'main.c' in an IDE. The code implements a queue using an array. The output window shows the program's execution, including menu prompts and user input.

```

main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 //GAGAN NAIK//
4 #define MAX 5
5
6 int queue[MAX];
7 int front = -1, rear = -1;
8
9 void insert()
10 {
11     int item;
12
13     if (rear == MAX - 1)
14     {
15         printf("\n QUEUE OVERFLOW! Cannot insert element.\n");
16         return;
17     }
18
19     printf("\n Enter the element to insert: ");
20     scanf("%d", &item);
21
22     if (front == -1)
23         front = 0;
24
25     queue[rear] = item;
26     rear++;
27 }
28
29 void delete()
30 {
31     if (front == -1)
32     {
33         printf("\n Queue is empty.\n");
34         return;
35     }
36     printf("\n Deleted element is: ");
37     printf("%d\n", queue[front]);
38     front++;
39 }
40
41 void display()
42 {
43     if (front == -1 || rear == -1)
44     {
45         printf("\n Queue is empty.\n");
46         return;
47     }
48     for (i = front; i <= rear; i++)
49         printf("%d ", queue[i]);
50     printf("\n");
51 }
52
53 int main()
54 {
55     int choice;
56     while (1)
57     {
58         printf("1.Insert 2.Delete 3.Display 4.Exit\n");
59         scanf("%d", &choice);
60         switch (choice)
61         {
62             case 1: insert(); break;
63             case 2: delete(); break;
64             case 3: display(); break;
65             case 4: return 0;
66             default: printf("Invalid choice\n");
67         }
68     }
69 }

```

Output

```

--- QUEUE OPERATIONS USING ARRAY ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1

Enter the element to insert: 10

Element 10 inserted successfully.

--- QUEUE OPERATIONS USING ARRAY ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted element is 10

--- QUEUE OPERATIONS USING ARRAY ---
1. Insert
2. Delete

```

Output

Clear

```
3. Display
4. Exit
Enter your choice: 2

Deleted element is 10

--- QUEUE OPERATIONS USING ARRAY ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

Queue is Empty.

--- QUEUE OPERATIONS USING ARRAY ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4

=== Code Execution Successful ===
```

Lab Program 3

b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>

#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

void insert() {
    int x;
    if ((front == 0 && rear == MAX - 1) || (front == rear + 1)) {
        printf("Queue Overflow\n");
        return;
    }
    printf("Enter element: ");
    scanf("%d", &x);
    if (front == -1) {
        front = 0;
        rear = 0;
    } else if (rear == MAX - 1) {
        rear = 0;
    } else {
        rear++;
    }
    queue[rear] = x;
}

void delete() {
    if (front == -1) {
        printf("Queue Empty\n");
        return;
    }
    printf("Deleted element: %d\n", queue[front]);
    if (front == rear) {
        front = -1;
        rear = -1;
    } else if (front == MAX - 1) {
```

```

        front = 0;
    } else {
        front++;
    }
}

void display() {
    int i;
    if (front == -1) {
        printf("Queue Empty\n");
        return;
    }
    i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear)
            break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

int main() {
    int choice;
    while (1) {
        printf("1.Insert 2.Delete 3.Display 4.Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1: insert(); break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4: return 0;
            default: printf("Invalid choice\n");
        }
    }
}

```

Output

```
main.c  Run  Clear
1 #include <stdio.h>
2 #include <stdlib.h>
3 //GAGAN NAIK//
4
5 #define MAX 5
6
7 int cq[MAX];
8 int front = -1, rear = -1;
9
10 void insert()
11 {
12     int item;
13
14     if ((front == 0 && rear == MAX - 1) || (front == rear + 1))
15     {
16         printf("\n CIRCULAR QUEUE OVERFLOW! Cannot insert element.\n");
17         return;
18     }
19
20     printf("\n Enter the element to insert: ");
21     scanf("%d", &item);
22
23     printf("\n Element %d inserted successfully.\n", item);
24 }
```

--- CIRCULAR QUEUE OPERATIONS ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to insert: 10
Element 10 inserted successfully.
--- CIRCULAR QUEUE OPERATIONS ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted element is 10
--- CIRCULAR QUEUE OPERATIONS ---
1. Insert
2. Delete

Output Clear

3. Display
4. Exit
Enter your choice: 2
Deleted element is 10
--- CIRCULAR QUEUE OPERATIONS ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Circular Queue is Empty.
--- CIRCULAR QUEUE OPERATIONS ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
=== Code Execution Successful ===

Lab Program 4

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

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

struct node {
    int data;
    struct node *next;
};

struct node *head = NULL;

void createList(int n) {
    struct node *newNode, *temp;
    int data;

    for (int i = 0; i < n; i++) {
        newNode = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            temp = head;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
}

void insertAtFirst(int data) {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
}

void insertAtEnd(int data) {
```

```

struct node *newNode = (struct node *)malloc(sizeof(struct node));
struct node *temp = head;

newNode->data = data;
newNode->next = NULL;

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

while (temp->next != NULL)
    temp = temp->next;

temp->next = newNode;
}

void insertAtPosition(int data, int pos) {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    struct node *temp = head;

    newNode->data = data;

    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return;
    }

    for (int i = 1; i < pos - 1; i++) {
        if (temp == NULL)
            return;
        temp = temp->next;
    }

    newNode->next = temp->next;
    temp->next = newNode;
}

void displayList() {
    struct node *temp = head;

    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("%d -> ", temp->data);

```

```

        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int n, data, pos;

    printf("Enter number of nodes: ");
    scanf("%d", &n);

    printf("Enter node values:\n");
    createList(n);

    printf("Linked List:\n");
    displayList();

    printf("Enter value to insert at first: ");
    scanf("%d", &data);
    insertAtFirst(data);
    displayList();

    printf("Enter value to insert at end: ");
    scanf("%d", &data);
    insertAtEnd(data);
    displayList();

    printf("Enter value and position to insert: ");
    scanf("%d %d", &data, &pos);
    insertAtPosition(data, pos);
    displayList();

    return 0;
}

```

```

main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // GAGAN NAIK
5 struct node
6 {
7     int data;
8     struct node *next;
9 };
10 struct node *head = NULL;
11
12 void create()
13 {
14     int n, i, item;
15     struct node *temp, *newNode;
16
17     printf("\n Enter number of nodes: ");
18     scanf("%d", &n);
19     for (i = 0; i < n; i++)
20     {
21         newNode = (struct node *)malloc(sizeof(struct node));
22         printf(" Enter data for node %d: ", i + 1);
23         scanf("%d", &item);
24

```

```

Output
--- SINGLY LINKED LIST OPERATIONS ---
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Any Position
5. Display
6. Exit
Enter your choice: 1

Enter number of nodes: 2
Enter data for node 1: 1
Enter data for node 2: 2

--- SINGLY LINKED LIST OPERATIONS ---
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Any Position
5. Display
6. Exit
Enter your choice: 2

Enter element to insert at beginning: 3

```


Output

Clear

```
Enter element to insert at beginning: 3

--- SINGLY LINKED LIST OPERATIONS ---
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Any Position
5. Display
6. Exit
Enter your choice: 3

Enter element to insert at end: 4

--- SINGLY LINKED LIST OPERATIONS ---
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Any Position
5. Display
6. Exit
Enter your choice: 2

Enter element to insert at beginning: 5
```

Output

Clear

```
--- SINGLY LINKED LIST OPERATIONS ---
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Any Position
5. Display
6. Exit
Enter your choice: 4

Enter position: 2
Enter element: 7

--- SINGLY LINKED LIST OPERATIONS ---
1. Create Linked List
2. Insert at Beginning
3. Insert at End
4. Insert at Any Position
5. Display
6. Exit
Enter your choice: 6

=== Code Execution Successful ===
```

Lab Program 5

WAP to Implement Singly Linked List with following operations

- a) Create a linked list.
- b) Deletion of first element, specified element and last element in the list.
- c) Display the contents of the linked list.

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

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

```
struct node {  
    int data;  
    struct node *next;  
};
```

```
struct node *head = NULL;
```

```
void delete_first() {  
    struct node *temp;  
    if (head == NULL) {  
        printf("List is empty\n");  
        return;  
    }  
    temp = head;  
    head = head->next;  
    free(temp);  
}
```

```
void delete_end() {  
    struct node *temp, *prev;  
    if (head == NULL) {  
        printf("List is empty\n");  
        return;  
    }  
    if (head->next == NULL) {  
        free(head);  
        head = NULL;  
        return;  
    }  
    temp = head;  
    while (temp->next != NULL) {  
        prev = temp;  
        temp = temp->next;
```

```

    }
    prev->next = NULL;
    free(temp);
}

void delete_element() {
    int x;
    struct node *temp, *previf (head
== NULL) { printf("List is
empty\n"); return;
    }
    scanf("%d", &x);
    if (head->data == x) {
        delete_first();
        return;
    }
    temp = head;
    while (temp != NULL && temp->data != x) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Element not found\n");
        return;
    }
    prev->next = temp->next;
    free(temp);
}

void display() {
    struct node *temp = head;
    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice;
    while (1) {
        printf("1.Create 2.DeleteFirst 3.DeleteElement 4.DeleteEnd 5.Display 6.Exit\n");

```

```

        scanf("%d", &choice);
        switch (choice) {
case 1: create(); break;
case 2: delete_first(); break;
case 3: delete_element(); break;
case 4: delete_end(); break;
case 5: display(); break;
case 6: return 0;
        default: printf("Invalid choice\n");
        }
}
}

```

```

main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 //GAGAN NAIK
4 struct node
5 {
6     int data;
7     struct node *next;
8 };
9
10 struct node *head = NULL;
11
12 void create()
13 {
14     int n, i, item;
15     struct node *temp, *newNode;
16
17     printf("\n Enter number of nodes: ");
18     scanf("%d", &n);
19
20     for (i = 0; i < n; i++)
21     {
22         newNode = (struct node *)malloc(sizeof(struct node));
23         printf(" Enter data for node %d: ", i + 1);
24         scanf("%d", &item);

```

Output

```

--- SINGLY LINKED LIST DELETION OPERATIONS ---
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display
6. Exit
Enter your choice: 1

Enter number of nodes: 2
Enter data for node 1: 1
Enter data for node 2: 2

--- SINGLY LINKED LIST DELETION OPERATIONS ---
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display
6. Exit
Enter your choice: 2

```

Output

Clear

```
Deleted element is 1

--- SINGLY LINKED LIST DELETION OPERATIONS ---
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display
6. Exit
Enter your choice: 3

Deleted element is 2

--- SINGLY LINKED LIST DELETION OPERATIONS ---
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display
6. Exit
Enter your choice: 5

Linked List is Empty.
```

Output

Clear

```
Deleted element is 2

--- SINGLY LINKED LIST DELETION OPERATIONS ---
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display
6. Exit
Enter your choice: 5

Linked List is Empty.

--- SINGLY LINKED LIST DELETION OPERATIONS ---
1. Create Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display
6. Exit
Enter your choice: 6

=== Code Execution Successful ===
```

Lab Program 6

a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

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

```
struct node {
    int data;
    struct node *next;
};
```

```
struct node *head = NULL;
```

```
void create(int n) {
    struct node *newNode, *temp;
    int data;
    for (int i = 0; i < n; i++) {
        newNode = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;
            temp = head;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
}
```

```
void sortList() {
    struct node *i, *j;
    int temp;
    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}
```

```

    }
}
}

void display() {
    struct node *temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter values:\n");
    create(n);
    sortList();
    display();
    return 0;
}

```

```

main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 //GAGAN NAIK
4 struct node
5 {
6     int data;
7     struct node *next;
8 };
9
10 struct node *head = NULL;
11
12 void create()
13 {
14     int n, i, item;
15     struct node *temp, *newNode;
16
17     printf("Enter number of nodes: ");
18     scanf("%d", &n);
19
20     for (i = 0; i < n; i++)
21     {
22         newNode = (struct node *)malloc(sizeof(struct node));
23         printf("Enter data: ");
24         scanf("%d", &item);

```

Output

```

Enter number of nodes: 5
Enter data: 8
Enter data: 3
Enter data: 1
Enter data: 6
Enter data: 5
Sorted Linked List:
1 -> 3 -> 5 -> 6 -> 8 -> NULL

=== Code Execution Successful ===

```

```

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

struct node {
    int data;
    struct node *next;
};

struct node *head = NULL;

void create(int n) {
    struct node *newNode, *temp;
    int data;
    for (int i = 0; i < n; i++) {
        newNode = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;
            temp = head;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
}

void reverse() {
    struct node *prev = NULL, *curr = head, *next = NULL;
    while (curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}

void display() {
    struct node *temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

```



```

int main() {
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter values:\n");
    create(n);
    reverse();
    display();
    return 0;
}

```

The screenshot shows a C code editor with a file named 'main.c'. The code implements a linked list with functions for creating, reversing, and displaying nodes. The output window shows the program's execution with user input for 5 nodes and values 1, 5, 2, 8, 6. The reversed list is printed as 6 -> 8 -> 2 -> 5 -> 1 -> NULL.

```

main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 //GADAN HAIN
4 struct node
5 {
6     int data;
7     struct node *next;
8 };
9
10 struct node *head = NULL;
11
12 void create()
13 {
14     int n, i, item;
15     struct node *temp, *newNode;
16
17     printf("Enter number of nodes: ");
18     scanf("%d", &n);
19
20     for (i = 0; i < n; i++)
21     {
22         newNode = (struct node *)malloc(sizeof(struct node));
23         printf("Enter data: ");
24         scanf("%d", &item);

```

Output

```

Enter number of nodes: 5
Enter data: 1
Enter data: 5
Enter data: 2
Enter data: 8
Enter data: 6
Reversed Linked List:
6 -> 8 -> 2 -> 5 -> 1 -> NULL

=== Code Execution Successful ===

```

```

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

struct node {
    int data;
    struct node *next;
};

struct node *head1 = NULL, *head2 = NULL;

struct node* create(int n) {
    struct node *newNode, *temp = NULL, *head = NULL;
    int data;
    for (int i = 0; i < n; i++) {
        newNode = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;
            temp = head;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
    return head;
}

void concatenate() {
    struct node *temp = head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = head2;
}

void display(struct node *head) {
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

```

```

int main() {
    int n1, n2;
    printf("Enter nodes for List 1: ");
    scanf("%d", &n1);
    head1 = create(n1);

    printf("Enter nodes for List 2: ");
    scanf("%d", &n2);
    head2 = create(n2);

    concatenate();
    display(head1);

    return 0;
}

```

```

main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 //GAGAN NAIK
4 struct node
5 {
6     int data;
7     struct node *next;
8 };
9
10 struct node *head1 = NULL;
11 struct node *head2 = NULL;
12
13 struct node* create()
14 {
15     int n, i, item;
16     struct node *head = NULL, *temp, *newNode;
17
18     printf("Enter number of nodes: ");
19     scanf("%d", &n);
20
21     for (i = 0; i < n; i++)
22     {
23         newNode = (struct node *)malloc(sizeof(struct node));
24         printf("Enter data: ");

```

Output

```

Create First Linked List
Enter number of nodes: 5
Enter data: 1
Enter data: 5
Enter data: 2
Enter data: 5
Enter data: 8
Create Second Linked List
Enter number of nodes: 3
Enter data: 6
Enter data: 7
Enter data: 9
Concatenated Linked List:
1 -> 5 -> 2 -> 5 -> 8 -> 6 -> 7 -> 9 -> NULL

*** Code Execution Successful ***

```

Lab Program 7

WAP to Implement doubly link list with primitive operations

- a) Create a doubly linked list.
- b) Insert a new node to the left of the node.
- c) Delete the node based on a specific value
- d) Display the contents of the list

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

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

```
struct node {  
    int data;  
    struct node *prev;  
    struct node *next;  
};
```

```
struct node *head = NULL;
```

```
void create() {  
    int n, x;  
    struct node *temp, *newnode;  
    scanf("%d", &n);  
    while (n--) {  
        newnode = malloc(sizeof(struct node));  
        scanf("%d", &x);  
        newnode->data = x;  
        newnode->prev = newnode->next = NULL;  
        if (head == NULL) {  
            head = newnode;  
        } else {  
            temp = head;  
            while (temp->next != NULL)  
                temp = temp->next;  
            temp->next = newnode;  
            newnode->prev = temp;  
        }  
    }  
}
```

```
void insert_left() {  
    int key, x;
```

```

struct node *temp, *newnode;
scanf("%d", &key);
scanf("%d", &x);
temp = head;
while (temp != NULL && temp->data != key)
    temp = temp->next;
if (temp == NULL) {
    printf("Element not found\n");
    return;
}
newnode = malloc(sizeof(struct node));
newnode->data = x;
newnode->prev = temp->prev;
newnode->next = temp;
if (temp->prev != NULL)
    temp->prev->next = newnode;
else
    head = newnode;
temp->prev = newnode;
}

```

```

void delete_value() {
    int x;
    struct node *temp;
    scanf("%d", &x);
    temp = head;
    while (temp != NULL && temp->data != x)
        temp = temp->next;
    if (temp == NULL) {
        printf("Element not found\n");
        return;
    }
    if (temp->prev != NULL)
        temp->prev->next = temp->next;
    else
        head = temp->next;
    if (temp->next != NULL)
        temp->next->prev = temp->prev;
    free(temp);
}

```

```

void display() {
    struct node *temp = head;

```

```

    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice;
    while (1) {
        printf("1.Create 2.InsertLeft 3.DeleteValue 4.Display 5.Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1: create(); break;
            case 2: insert_left(); break;
            case 3: delete_value(); break;
            case 4: display(); break;
            case 5: return 0;
            default: printf("Invalid choice\n");
        }
    }
}

```

main.c	Output
<pre> 1 #include <stdio.h> 2 #include <stdlib.h> 3 //GAGAN NAIK 4 struct node 5 { 6 int data; 7 struct node *prev; 8 struct node *next; 9 }; 10 11 struct node *head = NULL; 12 13 void create() 14 { 15 int n, i, item; 16 struct node *newNode, *temp; 17 18 printf("Enter number of nodes: "); 19 scanf("%d", &n); 20 21 for (i = 0; i < n; i++) 22 { 23 newNode = (struct node *)malloc(sizeof(struct node)); 24 printf("Enter data: "); </pre>	<pre> 1.Create 2.Insert Left 3.Delete by Value 4.Display 5.Exit Enter choice: 1 Enter number of nodes: 3 Enter data: 1 Enter data: 2 Enter data: 5 1.Create 2.Insert Left 3.Delete by Value 4.Display 5.Exit Enter choice: 2 Enter node value to insert left of: 6 Enter new element: 7 Element not found 1.Create 2.Insert Left </pre>

Output

Clear

```
1.Create
2.Insert Left
3.Delete by Value
4.Display
5.Exit
Enter choice: 3
Enter value to delete: 5
Node deleted
```

```
1.Create
2.Insert Left
3.Delete by Value
4.Display
5.Exit
Enter choice: 4
```

```
1 <-> 2 <-> NULL
```

```
1.Create
2.Insert Left
3.Delete by Value
4.Display
5.Exit
Enter choice: 5
```

Lab Program 8

Write a program

- To construct a binary search tree.
- To traverse the tree using all the methods i.e., in-order, preorder and post order
- To display the elements in the tree.

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

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

```
struct node {  
    int data;  
    struct node *left;  
    struct node *right;  
};
```

```
struct node *root = NULL;
```

```
struct node* insert(struct node *root, int x) {  
    if (root == NULL) {  
        root = malloc(sizeof(struct node));  
        root->data = x;  
        root->left = root->right = NULL;  
        return root;  
    }  
    if (x < root->data)  
        root->left = insert(root->left, x);  
    else if (x > root->data)  
        root->right = insert(root->right, x);  
    return root;  
}
```

```
void inorder(struct node *root) {  
    if (root != NULL) {  
        inorder(root->left);  
        printf("%d ", root->data);  
        inorder(root->right);  
    }  
}
```

```
void preorder(struct node *root) {
```

```

    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

int main() {
    int n, x, i, choice;
    while (1) {
        printf("1.Insert 2.Inorder 3.Preorder 4.Postorder 5.Exit\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf("%d", &n);
                for (i = 0; i < n; i++) {
                    scanf("%d", &x);
                    root = insert(root, x);
                }
                break;
            case 2:
                inorder(root);
                printf("\n");
                break;
            case 3:
                preorder(root);
                printf("\n");
                break;
            case 4:
                postorder(root);
                printf("\n");
                break;
            case 5:
                return 0;
        }
    }
}

```

```

    }
}

```

main.c

Share

Run

Output

Clear

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 //GAGAN NAIK
4 struct node
5 {
6     int data;
7     struct node *left;
8     struct node *right;
9 };
10
11 struct node *root = NULL;
12
13 struct node* insert(struct node *root, int item)
14 {
15     if (root == NULL)
16     {
17         struct node *newNode = (struct node *)malloc(sizeof(struct node));
18         newNode->data = item;
19         newNode->left = NULL;
20         newNode->right = NULL;
21         return newNode;
22     }
23
24     if (item < root->data)

```

1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 1
Enter element: 2

1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 1
Enter element: 6

1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 1
Enter element: 2

Output

Clear

```

1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 2
Inorder Traversal: 2 6

1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 3
Preorder Traversal: 2 6

1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 4
Postorder Traversal: 6 2

```

Output

Clear

```
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 3
Preorder Traversal: 2 6
```

```
1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 4
Postorder Traversal: 6 2
```

```
1.Insert into BST
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Exit
Enter your choice: 5
```

```
=== Code Execution Successful ===
```

Lab Program 9

- a) Write a program to traverse a graph using BFS method.
- b) Write a program to check whether given graph is connected or not using DFS method.

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

#define MAX 20

int queue[MAX], front = -1, rear = -1;

void enqueue(int x) {
    if (rear == MAX - 1) return;
    if (front == -1) front = 0;
    queue[++rear] = x;
}

int dequeue() {
    if (front == -1) return -1;
    int x = queue[front];
    if (front == rear) front = rear = -1;
    else front++;
    return x;
}

void bfs(int n, int adj[n][n], int start) {
    int visited[n];
    for (int i = 0; i < n; i++) visited[i] = 0;
    enqueue(start);
    visited[start] = 1;
    printf("BFS: ");
    while (front != -1) {
        int u = dequeue();
        printf("%d ", u);
        for (int v = 0; v < n; v++) {
            if (adj[u][v] && !visited[v]) {
                enqueue(v);
                visited[v] = 1;
            }
        }
    }
}
```

```

    }
}
printf("\n");
}

void dfs_util(int n, int adj[n][n], int visited[n], int u) {
    visited[u] = 1;
    for (int v = 0; v < n; v++) {
        if (adj[u][v] && !visited[v])
            dfs_util(n, adj, visited, v);
    }
}

int is_connected(int n, int adj[n][n]) {
    int visited[n];
    for (int i = 0; i < n; i++) visited[i] = 0;
    dfs_util(n, adj, visited, 0);
    for (int i = 0; i < n; i++)
        if (!visited[i]) return 0;
    return 1;
}

int main() {
    int n, i, j;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    int adj[n][n];
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
    bfs(n, adj, 0);
    if (is_connected(n, adj))
        printf("Graph is connected\n");
    else
        printf("Graph is not connected\n");
    return 0;
}

```

main.c

Share

Run

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //GAGAN NAIK
5
6 #define MAX 20
7 int queue[MAX], front = -1, rear = -1; void enqueue(int x) {
8 if (rear == MAX - 1) return; if (front == -1) front = 0; queue[++rear] = x;
9 }
10
11 int dequeue() {
12 if (front == -1) return -1; int x = queue[front];
13 if (front == rear) front = rear = -1; else front++;
14 return x;
15 }
16
17 void bfs(int n, int adj[n][n], int start) { int visited[n];
18 for (int i = 0; i < n; i++) visited[i] = 0; enqueue(start);
19 visited[start] = 1; printf("BFS: "); while (front != -1) {
20 int u = dequeue(); printf("%d ", u);
21 for (int v = 0; v < n; v++) {
22 if (adj[u][v] && !visited[v]) { enqueue(v);
23 visited[v] = 1;
24 }
```

Enter number of vertices: 3
Enter adjacency matrix:
2
1
2
3
4
5
6
7
8
BFS: 0 1 2
Graph is connected

=== Code Execution Successful ===

Lab Program 10

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers.

Design and develop a Program in C that uses Hash function $H: K \rightarrow L$ as $H(K) = K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>
#define M 100
```

```
typedef struct {
    int key;
    int used;
} Record;
```

```
int hash(int k) {
    return k % M;
}
```

```
void insert(Record ht[], int k) {
    int i = hash(k);
    int start = i;
    while (ht[i].used) {
        i = (i + 1) % M;
        if (i == start)
            return;
    }
    ht[i].key = k;
    ht[i].used = 1;
}
```

```
int search(Record ht[], int k) {
    int i = hash(k);
    int start = i;
    while (ht[i].used) {
        if (ht[i].key == k)
            return i;
        i = (i + 1) % M;
        if (i == start)
            break;
    }
    return -1;
}
```

```

int main() {
    Record ht[M];
    int n, k, i, pos, choice;

    for (i = 0; i < M; i++)
        ht[i].used = 0;

    printf("Enter number of employee records: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Enter 4-digit key: ");
        scanf("%d", &k);
        insert(ht, k);
    }

    do {
        printf("Enter key to search: ");
        scanf("%d", &k);
        pos = search(ht, k);
        if (pos == -1)
            printf("Key not found\n");
        else
            printf("Key found at address %02d\n", pos);

        printf("Search another key? (1-Yes / 0-No): ");
        scanf("%d", &choice);
    } while (choice == 1);

    return 0;
}

```

The screenshot shows a C code editor with a file named 'main.c'. The code implements a hash table with a separate chaining method. It includes a 'Record' struct with 'key' and 'used' fields. The 'insert' function uses a hash function to find a slot in the array and inserts a new record if the slot is empty. The 'search' function finds a slot and traverses the linked list of records to find the desired key. The main function prompts the user for the number of records, inserts three records with keys 123, 111, and 222, and then searches for keys 111, 123, 222, and 112. The output window shows the program's execution, confirming that keys 111, 123, and 222 were found at addresses 11, 23, and 22 respectively, while key 112 was not found. The code execution was successful.

```

main.c
1 #include <stdio.h>
2 #define M 100
3
4 //GAGAN NAIK
5
6 typedef struct {
7     int key;
8     int used;
9 } Record;
10
11 int hash(int k) {
12     return k % M;
13 }
14
15 void insert(Record ht[], int k) {
16     int i = hash(k);
17     int start = i;
18     while (ht[i].used) {
19         i = (i + 1) % M;
20         if (i == start)
21             return;
22     }
23     ht[i].key = k;
24     ht[i].used = 1;
25 }
26
27 int search(Record ht[], int k) {
28     int i = hash(k);
29     while (ht[i].used) {
30         if (ht[i].key == k)
31             return i;
32         i = (i + 1) % M;
33     }
34     return -1;
35 }
36
37 int main() {
38     Record ht[M];
39     int n, k, i, pos, choice;
40
41     for (i = 0; i < M; i++)
42         ht[i].used = 0;
43
44     printf("Enter number of employee records: ");
45     scanf("%d", &n);
46
47     for (i = 0; i < n; i++) {
48         printf("Enter 4-digit key: ");
49         scanf("%d", &k);
50         insert(ht, k);
51     }
52
53     do {
54         printf("Enter key to search: ");
55         scanf("%d", &k);
56         pos = search(ht, k);
57         if (pos == -1)
58             printf("Key not found\n");
59         else
60             printf("Key found at address %02d\n", pos);
61
62         printf("Search another key? (1-Yes / 0-No): ");
63         scanf("%d", &choice);
64     } while (choice == 1);
65
66     return 0;
67 }

```

Output

```

Enter number of employee records: 3
Enter 4-digit key: 123
Enter 4-digit key: 111
Enter 4-digit key: 222
Enter key to search: 111
Key found at address 11
Search another key? (1-Yes / 0-No): 1
Enter key to search: 123
Key found at address 23
Search another key? (1-Yes / 0-No): 1
Enter key to search: 222
Key found at address 22
Search another key? (1-Yes / 0-No): 1
Enter key to search: 112
Key not found
Search another key? (1-Yes / 0-No): 0

=== Code Execution Successful ===

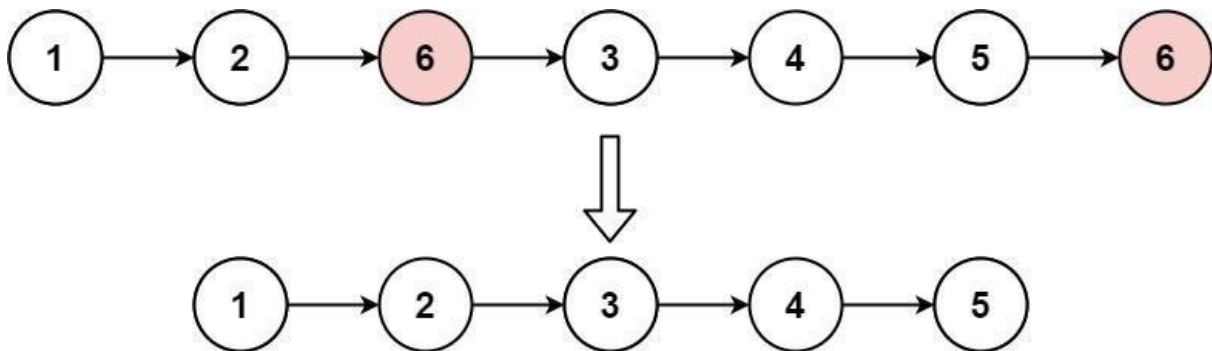
```


Leetcode Problem 1

Remove Linked List Elements

Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head*.

Example 1:



Input: `head = [1,2,6,3,4,5,6]`, `val = 6`

Output: `[1,2,3,4,5]`

Example 2:

Input: `head = []`, `val = 1`

Output: `[]`

Example 3:

Input: `head = [7,7,7,7]`, `val = 7`

Output: `[]`

Constraints:

- The number of nodes in the list is in the range `[0, 104]`.
- `1 <= Node.val <= 50`
- `0 <= val <= 50`

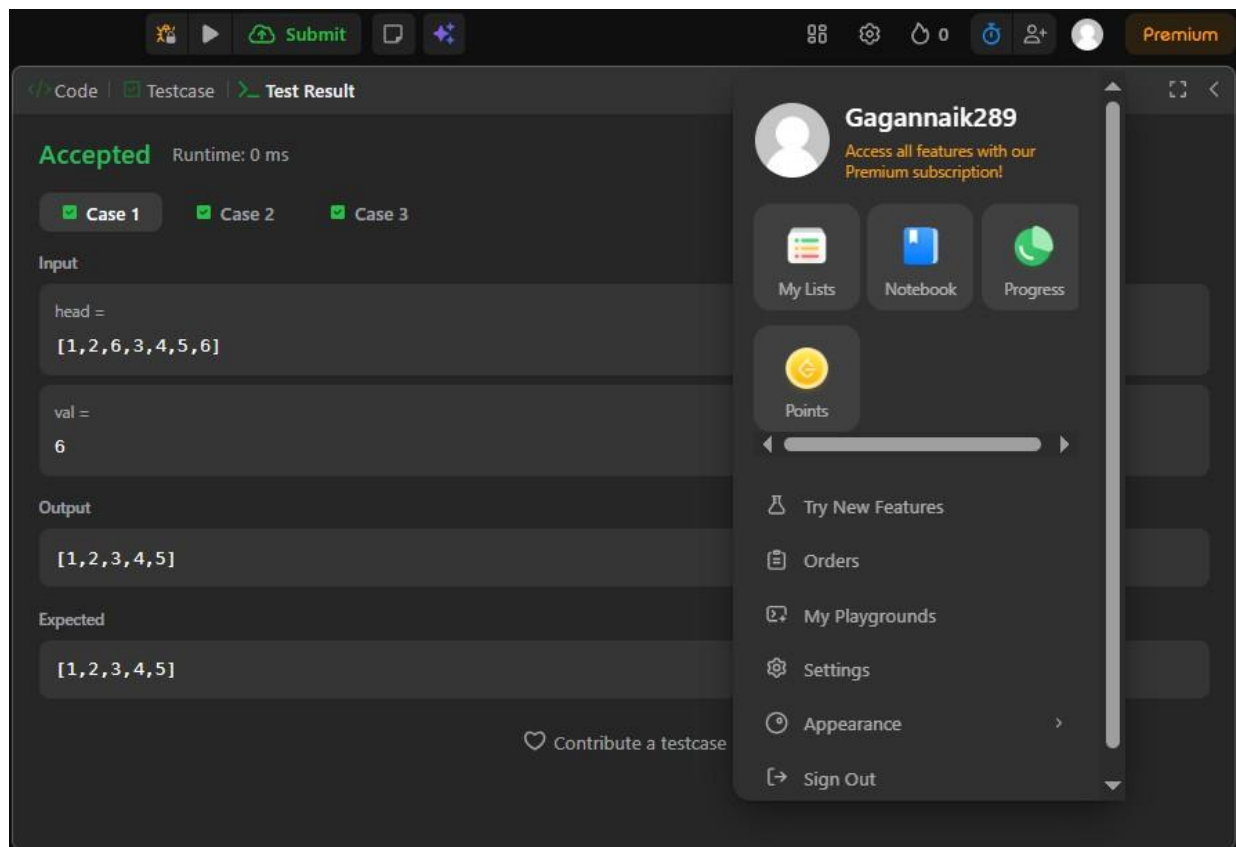
```
struct ListNode* removeElements(struct ListNode* head, int val) {  
    struct ListNode dummy;  
    dummy.next = head;  
    struct ListNode* current = &dummy;  
  
    while (current->next != NULL) {
```

```

if (current->next->val == val) {
    struct ListNode* temp = current->next;
    current->next = current->next->next;
    free(temp);
} else {
    current = current->next;
}
}

return dummy.next;
}

```



Submit

Code

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

head =
[]

val =
1

Output

[]

Expected

[]

Contribute a testcase

Gagannaik289

Access all features with our Premium subscription!

My Lists

Notebook

Progress

Points

Try New Features

Orders

My Playgrounds

Settings

Appearance

Sign Out

Submit

Code

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

head =
[7,7,7,7]

val =
7

Output

[]

Expected

[]

Contribute a testcase

Gagannaik289

Access all features with our Premium subscription!

My Lists

Notebook

Progress

Points

Try New Features

Orders

My Playgrounds

Settings

Appearance

Sign Out

Leetcode Problem 2

Sort List

Given the `head` of a linked list, return *the list after sorting it in ascending order*.

Example 1:

Input: `head = [4,2,1,3]`

Output: `[1,2,3,4]`

Example 2:

Input: `head = [-1,5,3,4,0]`

Output: `[-1,0,3,4,5]`

Example 3:

Input: `head = []`

Output: `[]`

Constraints:

- The number of nodes in the list is in the range `[0, 5 * 104]`.
- `-105 <= Node.val <= 105`

```
struct ListNode* merge(struct ListNode* l1, struct ListNode* l2) {
    struct ListNode dummy;
    struct ListNode* tail = &dummy;
    dummy.next = NULL;

    while (l1 && l2) {
        if (l1->val < l2->val) {
            tail->next = l1;
            l1 = l1->next;
        } else {
            tail->next = l2;
            l2 = l2->next;
        }
        tail = tail->next;
    }

    tail->next = l1 ? l1 : l2;
```

```

    return dummy.next;
}

struct ListNode* getMid(struct ListNode* head) {
    struct ListNode* slow = head;
    struct ListNode* fast = head->next;

    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
    }

    return slow;
}

struct ListNode* sortList(struct ListNode* head) {
    if (!head || !head->next)
        return head;

    struct ListNode* mid = getMid(head);
    struct ListNode* right = mid->next;
    mid->next = NULL;

    struct ListNode* leftSorted = sortList(head);
    struct ListNode* rightSorted = sortList(right);

    return merge(leftSorted, rightSorted);
}

```

Submit

Code

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

head =
[4, 2, 1, 3]

Output

[1, 2, 3, 4]

Expected

[1, 2, 3, 4]

Contribute a testcase

Gagannaik289

Access all features with our Premium subscription!

My Lists

Notebook

Progress

Points

Try New Features

Orders

My Playgrounds

Settings

Appearance

Sign Out

Submit

Code

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

head =
[-1, 5, 3, 4, 0]

Output

[-1, 0, 3, 4, 5]

Expected

[-1, 0, 3, 4, 5]

Contribute a testcase

Gagannaik289

Access all features with our Premium subscription!

My Lists

Notebook

Progress

Points

Try New Features

Orders

My Playgrounds

Settings

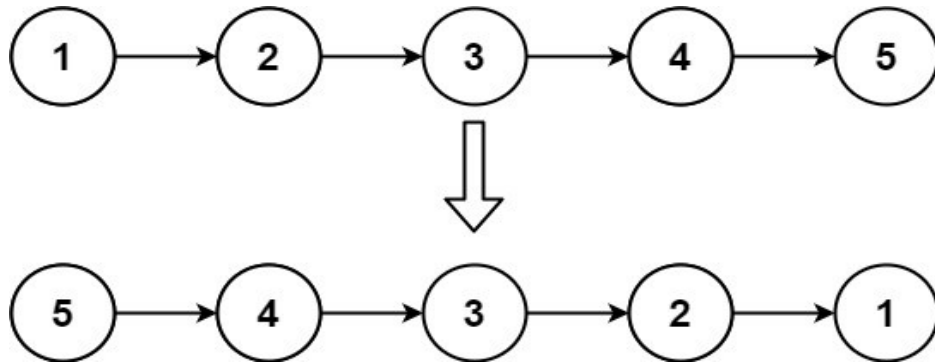
Appearance

Sign Out

Leetcode Problem 3

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

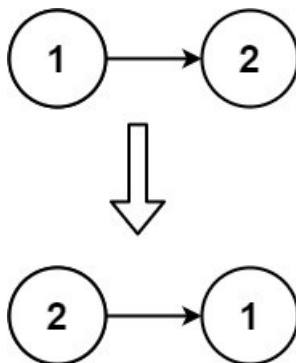
Example 1:



Input: `head = [1,2,3,4,5]`

Output: `[5,4,3,2,1]`

Example 2:



Input: `head = [1,2]`

Output: `[2,1]`

Example 3:

Input: `head = []`

Output: `[]`

Constraints:

- The number of nodes in the list is the range `[0, 5000]`.
- `-5000 <= Node.val <= 5000`

```

struct ListNode* reverseList(struct ListNode* head) {
    struct ListNode* prev = NULL;
    struct ListNode* curr = head;
    struct ListNode* next = NULL;

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

    return prev;
}

```

The screenshot displays a coding platform interface with a dark theme. At the top, there are navigation icons and a 'Premium' badge. The main area shows the 'Test Result' tab, indicating the code was 'Accepted' with a runtime of 0 ms. Below this, three test cases (Case 1, Case 2, Case 3) are marked as passed. The input section shows 'head = [1,2,3,4,5]' and the output section shows '[5,4,3,2,1]', which matches the expected output '[5,4,3,2,1]'. A 'Contribute a testcase' link is visible at the bottom of the test result section. On the right side, a user profile dropdown menu is open for 'Gagannaik289', showing options like 'My Lists', 'Notebook', 'Progress', 'Points', 'Try New Features', 'Orders', 'My Playgrounds', 'Settings', 'Appearance', and 'Sign Out'.

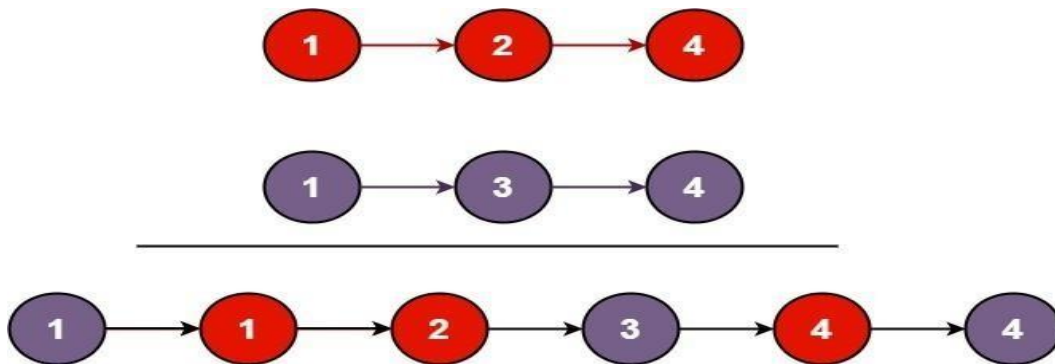
Leetcode Program 4

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Example 1:



Input: `list1 = [1,2,4]`, `list2 = [1,3,4]`

Output: `[1,1,2,3,4,4]`

Example 2:

Input: `list1 = []`, `list2 = []`

Output: `[]`

Example 3:

Input: `list1 = []`, `list2 = [0]`

Output: `[0]`

Constraints:

- The number of nodes in both lists is in the range `[0, 50]`.
- `-100 <= Node.val <= 100`
- Both `list1` and `list2` are sorted in **non-decreasing** order.

```
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode* list2) {  
    struct ListNode dummy;
```

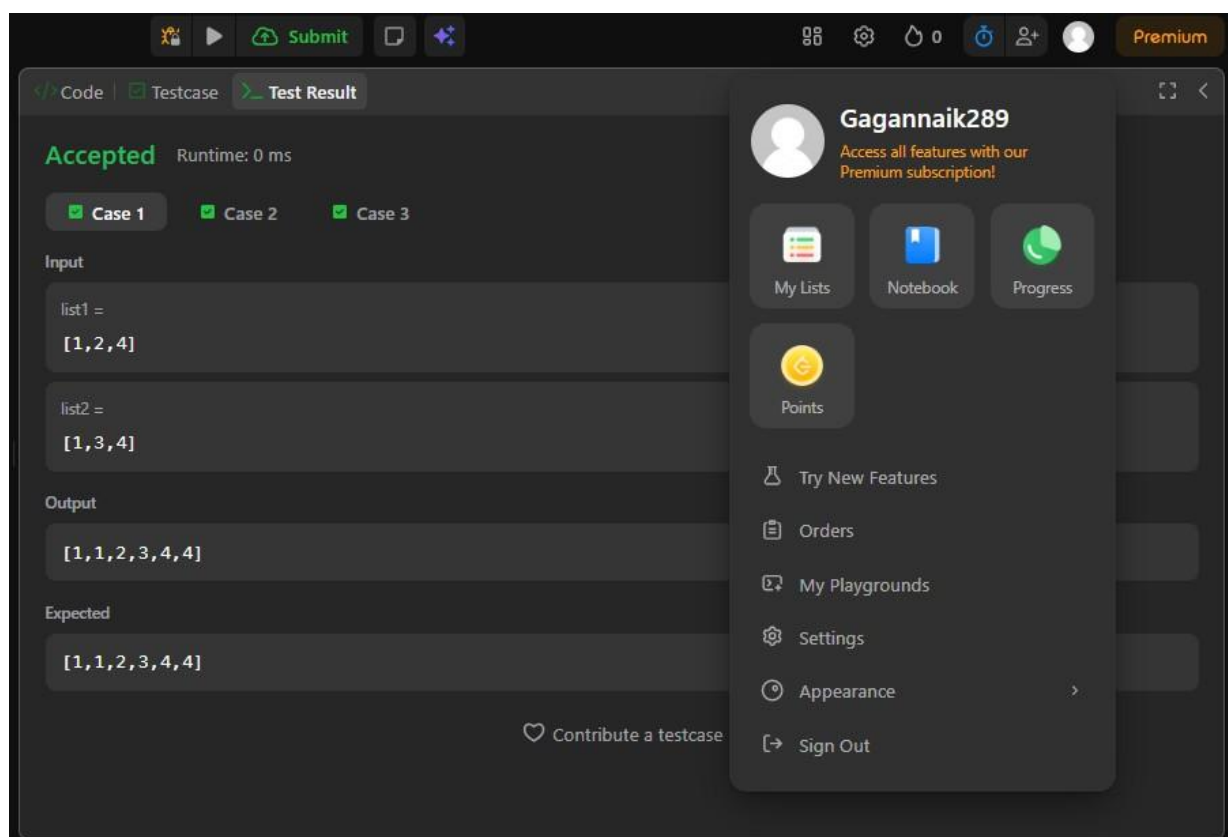
```

struct ListNode* tail = &dummy;
dummy.next = NULL;

while (list1 && list2) {
    if (list1->val <= list2->val) {
        tail->next = list1;
        list1 = list1->next;
    } else {
        tail->next = list2;
        list2 = list2->next;
    }
    tail = tail->next;
}

tail->next = list1 ? list1 : list2;
return dummy.next;
}

```



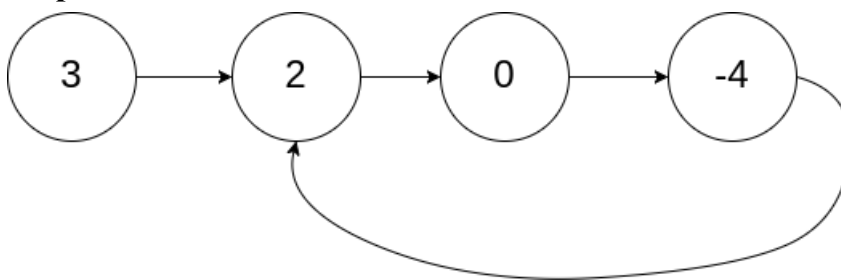
Leetcode Program 5

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:

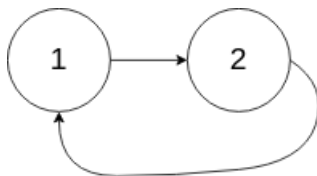


Input: `head = [3,2,0,-4]`, `pos = 1`

Output: `true`

Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:

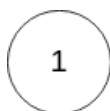


Input: `head = [1,2]`, `pos = 0`

Output: `true`

Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:



Input: `head = [1]`, `pos = -1`

Output: `false`

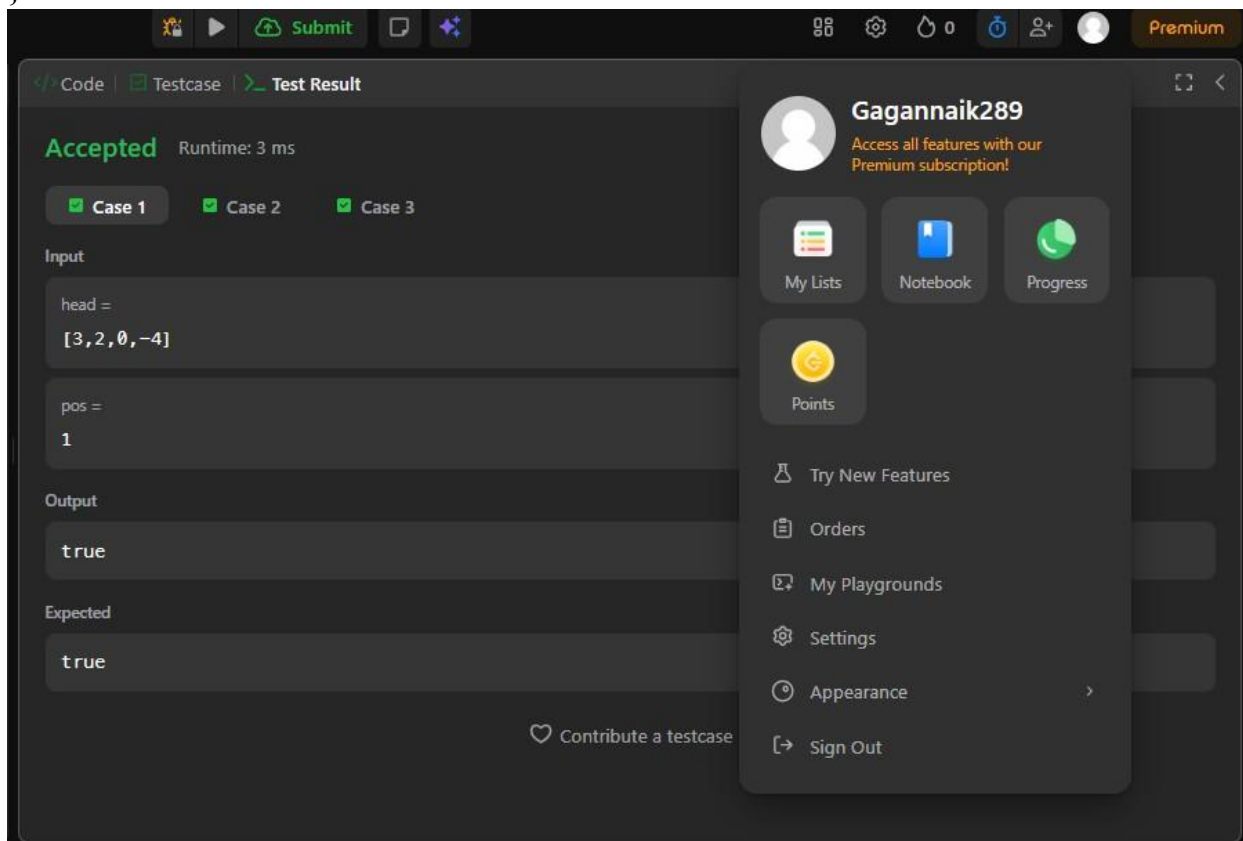
Explanation: There is no cycle in the linked list.

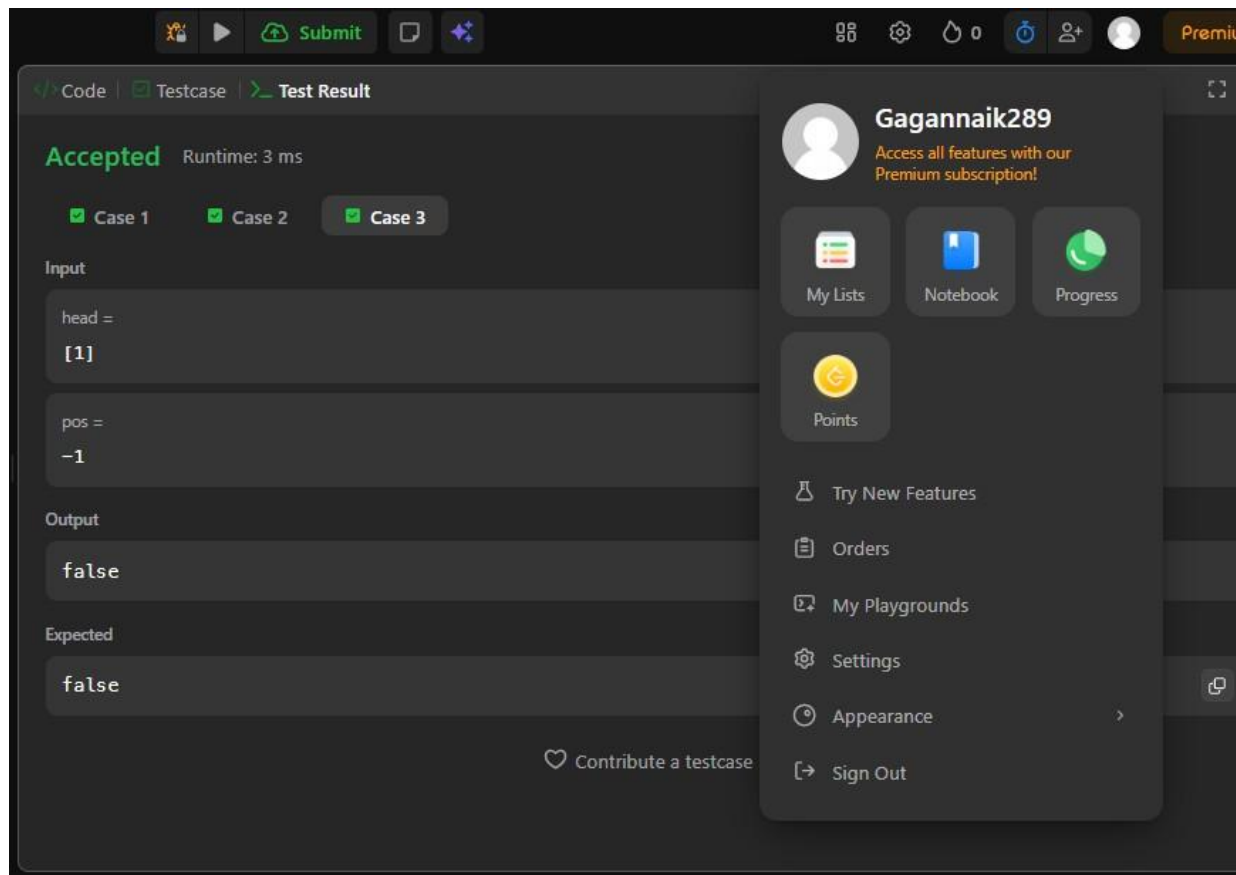
Constraints:

- The number of the nodes in the list is in the range $[0, 10^4]$.
- $-10^5 \leq \text{Node.val} \leq 10^5$
- `pos` is `-1` or a **valid index** in the linked-list.

Follow up: Can you solve it using $O(1)$ (i.e. constant) memory?

```
bool hasCycle(struct ListNode *head) {  
    struct ListNode* slow = head;  
    struct ListNode* fast = head;  
  
    while (fast && fast->next) {  
        slow = slow->next;  
        fast = fast->next->next;  
        if (slow == fast)  
            return true;  
    }  
  
    return false;  
}
```





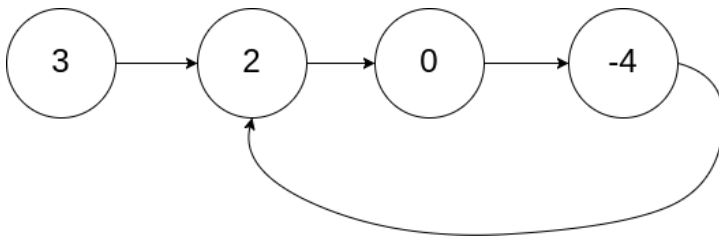
Leetcode Problem 6

Given the `head` of a linked list, return *the node where the cycle begins*. If there is no cycle, return `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that `pos` is not passed as a parameter**.

Do not modify the linked list.

Example 1:

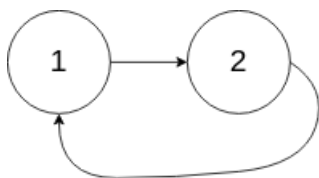


Input: `head = [3,2,0,-4]`, `pos = 1`

Output: tail connects to node index 1

Explanation: There is a cycle in the linked list, where tail connects to the second node.

Example 2:

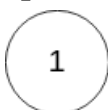


Input: `head = [1,2]`, `pos = 0`

Output: tail connects to node index 0

Explanation: There is a cycle in the linked list, where tail connects to the first node.

Example 3:



Input: `head = [1]`, `pos = -1`

Output: no cycle

Explanation: There is no cycle in the linked list.

Constraints:

- The number of the nodes in the list is in the range `[0, 104]`.
- `-105 <= Node.val <= 105`
- `pos` is `-1` or a **valid index** in the linked-list.

```
struct ListNode *detectCycle(struct ListNode *head) {
    struct ListNode* slow = head;
    struct ListNode* fast = head;

    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast)
            break;
    }

    if (!fast || !fast->next)
        return NULL;

    slow = head;
    while (slow != fast) {
        slow = slow->next;
        fast = fast->next;
    }

    return slow;
}
```

Code

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

head =
[3,2,0,-4]

pos =
1

Output

tail connects to node index 1

Expected

tail connects to node index 1

Contribute a testcase

Gagannaik289

Access all features with our Premium subscription!

My Lists

Notebook

Progress

Points

Try New Features

Orders

My Playgrounds

Settings

Appearance

Sign Out

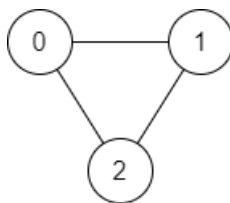
Leetcode Problem 7

There is a **bi-directional** graph with n vertices, where each vertex is labeled from 0 to $n - 1$ (**inclusive**). The edges in the graph are represented as a 2D integer array `edges`, where each `edges[i] = [ui, vi]` denotes a bi-directional edge between vertex u_i and vertex v_i . Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex `source` to vertex `destination`.

Given `edges` and the integers `n`, `source`, and `destination`, return `true` if there is a **valid path** from `source` to `destination`, or `false` otherwise.

Example 1:



Input: `n = 3`, `edges = [[0,1],[1,2],[2,0]]`, `source = 0`, `destination = 2`

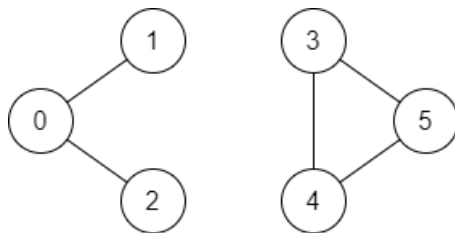
Output: `true`

Explanation: There are two paths from vertex 0 to vertex 2:

- $0 \rightarrow 1 \rightarrow 2$

- $0 \rightarrow 2$

Example 2:



Input: `n = 6`, `edges = [[0,1],[0,2],[3,5],[5,4],[4,3]]`, `source = 0`, `destination = 5`

Output: `false`

Explanation: There is no path from vertex 0 to vertex 5.

Constraints:

- $1 \leq n \leq 2 * 10^5$
- $0 \leq \text{edges.length} \leq 2 * 10^5$
- $\text{edges}[i].\text{length} == 2$
- $0 \leq u_i, v_i \leq n - 1$
- $u_i \neq v_i$
- $0 \leq \text{source}, \text{destination} \leq n - 1$
- There are no duplicate edges.
- There are no self edges.

The screenshot displays a coding platform interface with a dark theme. At the top, there's a navigation bar with icons for a flag, play, submit, and other features. The main content area shows a 'Test Result' for a submission that was 'Accepted' with a runtime of 0 ms. Below this, there are two test cases, 'Case 1' and 'Case 2', both marked as passed. The input section shows the following values: n = 3, edges = [[0,1], [1,2], [2,0]], source = 0, and destination = 2. The output section shows 'true', and the expected section also shows 'true'. On the right side, there's a user profile for 'Gagannaik289' with a prompt to 'Access all features with our Premium subscription!'. Below the profile, there are several menu items: 'My Lists', 'Notebook', 'Progress', 'Points', 'Try New Features', 'Orders', 'My Playgrounds', 'Settings', 'Appearance', and 'Sign Out'.

Submit

Code

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Input

n =
6

edges =
[[0,1],[0,2],[3,5],[5,4],[4,3]]

source =
0

destination =
5

Output
false

Expected
false

Gagannaik289

Access all features with our Premium subscription!

My Lists

Notebook

Progress

Points

Try New Features

Orders

My Playgrounds

Settings

Appearance

Sign Out

59 | Page