VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SHRADDHA (1BM22CS357)

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
Dec 2023- March 2024

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 NAME **(USN)**, who is a 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 2023-24. 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.

Prof. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

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:

(a) Write a program to swap two numbers using pointers.

```
#include<stdio.h>
void swapping(int*x,int*y);
int main()
{
   int x,y;
   printf("Enter two numbers \n");
   scanf("%d%d",&x,&y);
   printf("The value of x and y before swapping is %d and%d\n",x,y);
   swapping(&x,&y);
   return 0;
}
void swapping(int*x,int*y)
{
   int temp=*x;
   *x=*y;
   *y=temp;
   printf("The value of x and y after swapping is %d and %d",*x,*y);
}
```

Output:

```
Enter two numbers
5 9
The value of x and y before swapping is 5 and9
The value of x and y after swapping is 9 and 5
```

(b) Write a program to demonstrate memory allocation and deallocation in C using malloc , calloc , realloc and free.

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

int main() {
   int *arr1, *arr2;
   int size;

   // Using malloc to allocate memory for an array
   printf("Enter the size of the array: ");
   scanf("%d", &size);

   arr1 = (int *)malloc(size * sizeof(int));
```

```
    if (arr1 == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    printf("Memory allocated successfully using malloc.\n");

    // Using free to release the allocated memory
    free(arr1);
    printf("Memory freed using free.\n");

    // Using calloc to allocate and initialize memory for an array
    arr2 = (int *)calloc(size, sizeof(int));
    if (arr2 == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    printf("Memory allocated and initialized to zero using calloc.\n");

    // Using realloc to resize the allocated memory
    int newSize;
    printf("Enter the new size for the array: ");
    scanf("%d", &newSize);

    arr2 = (int *)realloc(arr2, newSize * sizeof(int));
    if (arr2 == NULL) {
        printf("Memory reallocation failed.\n");
        return 1;
    }

    printf("Memory reallocated successfully using realloc.\n");

    // Using free to release the reallocated memory
    free(arr2);
    printf("Memory freed after reallocation using free.\n");

    return 0;
}
```

Output:

```
Enter the size of the array: 3
Memory allocated successfully using malloc.
Memory freed using free.
Memory allocated and initialized to zero using calloc.
Enter the new size for the array: 6
Memory reallocated successfully using realloc.
Memory freed after reallocation using free.
```

(c)Write a program to simulate the working of stack using an array with the following Push , Pop & Display

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int top = -1;
int inp_array[SIZE];
void push();
void pop();
void show();

void main()
{
   int ch;
   while (1)
   {
      printf("Operations on the stack:\n");
      printf("1.Push the element\n2.Pop the element\n3.Show\n4.End\n");
      printf("Enter the choice:\n ");
      scanf("%d",&ch);

      switch (ch)
      {
      case 1:
         push();
         break;
      case 2:
         pop();
         break;
      case 3:
         show();
         break;
      case 4:
         exit(0);
```

```
        default:
            printf("Invalid choice\n");
        }
    }
}

void push()
{
    int x;
    if (top == SIZE - 1)
    {
        printf("Overflow\n");
    }
    else
    {
        printf("Enter the element to be added in the stack:\n ");
        scanf("%d", &x);
        top = top + 1;
        inp_array[top] = x;
    }
}
void pop()
{
    if (top == -1)
    {
        printf("Underflow\n");
    }
    else
    {
        printf("Popped element: %d\n", inp_array[top]);
        top = top - 1;
    }
}
void show()
{
    if (top == -1)
    {
        printf("Underflow\n");
    }
    else
    {
        int i;
        printf("Elements in the stack are: \n");
        for (i = top; i >= 0; --i)
            printf("%d\n", inp_array[i]);
    }
}
```

Output:

```
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
1
Enter the element to be added in the stack:
 20
 Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
 1
 Enter the element to be added in the stack:
 24
 Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
 1
 Enter the element to be added in the stack:
 45
 Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
 2
 Popped element: 45
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
 3
 Elements in the stack are:
24
20
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
 4
```

Lab program 2:

(a) Write a program to converts an infix expression to a postfix expression using a stack.

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50

char stack[SIZE];
int top = -1;

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

char pop() {
   return (stack[top--]);
}

int pr(char symbol) {
   if (symbol == '^') {
      return (3);
   } else if (symbol == '*' || symbol == '/') {
      return (2);
   } else if (symbol == '+' || symbol == '-') {
      return (1);
   } else {
      return (0);
   }
}

int main() {
   char infix[50], postfix[50], ch, ele;
   int i = 0, k = 0;

   printf("Enter the infix expression:");
   scanf("%s", infix);

   push('#');

   while ((ch = infix[i++]) != '\0') {
      if (ch == '(')
         push(ch);
      else if (isalnum(ch))
         postfix[k++] = ch;
      else if (ch == ')') {
         while (stack[top] != '(')
            postfix[k++] = pop();
         ele = pop();
```

```
        } else {
            while (pr(stack[top]) >= pr(ch))
                postfix[k++] = pop();
            push(ch);
        }
    }

    while (stack[top] != '#')
        postfix[k++] = pop();

    postfix[k] = '\0';
    printf("\nPostfix expression = %s\n", postfix);
    return 0;
}
```

Output:

```
Enter the infix expression:A*B+C*D-E
Postfix expression = AB*CD*+E-
```

(b) Write a program to evaluates a postfix expression using a stack.

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

#define MAX_SIZE 100

// Stack structure
struct Stack {
    int top;
    int items[MAX_SIZE];
};

// Function prototypes
void initialize(struct Stack *s);
void push(struct Stack *s, int value);
int pop(struct Stack *s);
int isOperator(char ch);
int compute(char operator, int operand1, int operand2);
int evaluatePostfix(char postfixExpression[]);

int main() {
    char postfixExpression[MAX_SIZE];

    // Input the postfix expression
```

```
    printf("Enter a postfix expression: ");
    gets(postfixExpression);

    // Evaluate and print the result
    printf("Result: %d\n", evaluatePostfix(postfixExpression));

    return 0;
}

void initialize(struct Stack *s) {
    s->top = -1;
}

void push(struct Stack *s, int value) {
    if (s->top == MAX_SIZE - 1) {
        printf("Stack overflow\n");
        exit(EXIT_FAILURE);
    }
    s->items[++(s->top)] = value;
}

int pop(struct Stack *s) {
    if (s->top == -1) {
        printf("Stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return s->items[(s->top)--];
}

int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}

int compute(char operator, int operand1, int operand2) {
    switch (operator) {
    case '+':
        return operand1 + operand2;
    case '-':
        return operand1 - operand2;
    case '*':
        return operand1 * operand2;
    case '/':
        if (operand2 != 0) {
            return operand1 / operand2;
        } else {
            printf("Error: Division by zero\n");
            exit(EXIT_FAILURE);
        }
```

```
    default:
        printf("Error: Invalid operator\n");
        exit(EXIT_FAILURE);
    }
}

int evaluatePostfix(char postfixExpression[]) {
    struct Stack s;
    initialize(&s);

    for (int i = 0; postfixExpression[i] != '\0'; i++) {
        if (isdigit(postfixExpression[i])) {
            // Operand
            push(&s, postfixExpression[i] - '0');
        } else if (isOperator(postfixExpression[i])) {
            // Operator
            int op2 = pop(&s);
            int op1 = pop(&s);
            int result = compute(postfixExpression[i], op1, op2);
            push(&s, result);
        }
        // Ignore other characters
    }

    return pop(&s);
}
```

Output:

```
Enter a postfix expression: 52+83-*4/
Result: 8
```

Lab program 3:

(a) Write a program to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display.

```
#include<stdio.h>
#define size 30

int queue[size];
int front=-1;
int rear=-1;

void insert(int a)
{
   if(rear==size-1)
   {
      printf("Queue overflow\n");
      return;
   }
   else
   {
      if(front==-1)
         front=0;
      queue[++rear]=a;
   }
}

void delete()
{
   if(front==-1||front>rear)
   {
      printf("Queue Empty\n");
   }
   else
   {
      front++;
   }
}

void display()
{
   int i;
   if(front==-1)
   {
      printf("Queue Empty\n");
      return;
   }
   printf("Queue:");
   for(i=front;i<=rear;i++)
```

```
    {
        printf("%d ",queue[i]);
    }
}


void main()
{
    int choice;
    int a;
    while(1)
    {
        printf("\n1.Insert\n2.Delete\n3.Display\nChoice:");
        scanf("%d",&choice);

        switch (choice)
        {
        case 1:printf("Enter an element:");
               scanf("%d",&a);
               insert(a);
               display();
               break;

        case 2:delete();
               display();
               break;

        case 3:display();
               break;
        }
    }

}
```

Output:

```
1.Insert
2.Delete
3.Display
Choice:1
Enter an element:24
Queue:24
1.Insert
2.Delete
3.Display
```

```
Choice:1
Enter an element:57
Queue:24 57
1.Insert
2.Delete
3.Display
Choice:1
Enter an element:92
Queue:24 57 92
1.Insert
2.Delete
3.Display
Choice:2
Queue:57 92
1.Insert
2.Delete
3.Display
Choice:3
Queue:57 92
1.Insert
2.Delete
3.Display
Choice:
```

(b) Write a program to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display.

```
#include<stdio.h>
#define pf printf
#define sf scanf
#define MAX 100
int cq[MAX];
int front=-1,rear=-1;

void enqueue(){
   int data;
   pf("\nEnter the element:");
   sf("%d",&data);

   if(front==-1 && rear==-1){
      front=rear=0;
      cq[rear]=data;
```

```
    }
    else if((rear+1)%MAX==front)
        pf("\nQueue Overflow");
    else{
        rear=(rear+1)%MAX;
        cq[rear]=data;
    }
}
void dequeue()
{
    if(front==-1 && rear==-1){
        pf("\nQueue is empty");
    }
    else if(front==rear){
        front=rear=-1;
    }
    else{
        front=(front+1)%MAX;
    }
}

void display()
{
    int i=front;
    pf("\nThe circular queue is :\n");
    while(i!=rear){
        pf("\t%d",cq[i]);
        i=(i+1)%MAX;
    }
    pf("\t%d",cq[i]);
}
int main()
{
    int ch;
    while(1){
        pf("\n----Enter----\n1.Insertion\n2.Deletion\n3.Display\t");
        sf("%d",&ch);
        switch(ch){
            case 1: enqueue();
                    display();
                    break;
            case 2: dequeue();
                    display();
                    break;
            case 3: display();
                    return 0;

        }
```

```
    }
}
```

Output:

```
----Enter----
1.Insertion
2.Deletion
3.Display   1
Enter the element:22
The circular queue is :
    22
----Enter----
1.Insertion
2.Deletion
3.Display   1
Enter the element:56
The circular queue is :
    22  56
----Enter----
1.Insertion
2.Deletion
3.Display   1
Enter the element:72
The circular queue is :
    22  56  72
----Enter----
1.Insertion
2.Deletion
3.Display   2
The circular queue is :
    56  72
----Enter----
1.Insertion
2.Deletion
3.Display   3
The circular queue is :
    56  72
```

Lab program 4:

Write a program 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.
c) Display the contents of the linked list.

```
#include<stdio.h>
#include<stdlib.h>
#define pf printf
#define sf scanf

struct node{
   int data;
   struct node *next;
};
struct node *head,*newnode,*temp,*last;


void push()
{
   newnode=(struct node *)malloc(sizeof(struct node));
   pf("Enter the data of the node to be inserted at the beginning:");
   sf("%d",&newnode->data);
   newnode->next=head;
   head=newnode;
   pf("\n---The element %d has been inserted at beg---",newnode->data);
   /*display*/
   pf("\nThe linked list you have created is: \n");
   temp=head;
   while(temp!=NULL)
   {

      pf("%d\t",temp->data);
      temp=temp->next;
   }


}
void append()
{
   newnode=(struct node *)malloc(sizeof(struct node));
   pf("Enter the data of the node 1:");
   sf("%d",&newnode->data);
   newnode->next=NULL;
   temp=head;
   while(temp->next!=NULL)
   {
      temp=temp->next;
```

```
    }
    temp->next=newnode;
    pf("\nThe element %d has been inserted at the end:",newnode->data);

    /*display*/

    pf("The linked list you have created is: \n");
    temp=head;
    while(temp!=NULL)
    {

        pf("%d\t",temp->data);
        temp=temp->next;
    }


}

void insert_pos()
{
    int pos, i=0;
    newnode=(struct node *)malloc(sizeof(struct node));
    pf("Enter the data of the node :");
    sf("%d",&newnode->data);
    pf("\nEnter the pos:");
    sf("%d",&pos);
    temp=last=head;

    for(i=0;i<pos;i++)
    {
        last=temp;
        temp=temp->next;
    }
    last->next=newnode;
    newnode->next=temp;

        pf("The linked list you have created is: \n");
    temp=head;
    while(temp!=NULL)
    {

        pf("%d\t",temp->data);
        temp=temp->next;
    }
}
void display()
{
     pf("The linked list you have created is: \n");
```

```
    temp=head;
    while(temp!=NULL)
    {

        pf("%d\t",temp->data);
        temp=temp->next;
    }
}
int main()
{

   int ch;
   while(1){
      pf("\n\nEnter your choice:\n1.Insert at begining \n2.Insert at End \n3.Insert at a pos
\n4.Display: ");
   sf("%d",&ch);
   switch(ch){
   case 1:
            push();
            break;
    case 2: append();
            break;
    case 3:
            insert_pos();

            break;
    case 4:
            display();
            return 0;
   }
}
```

Output:

```
1.Insert at begining
2.Insert at End
3.Insert at a pos
4.Display: 2
Enter the data of the appending node :6

The element 6 has been inserted at the end:The linked list you have created is:
2       4       6

Enter your choice:
1.Insert at begining
2.Insert at End
3.Insert at a pos
4.Display: 3
Enter the data of the node :0
```

```
Enter the pos:2
The linked list you have created is:
2       4       0       6

Enter your choice:
1.Insert at begining
2.Insert at End
3.Insert at a pos
4.Display: 4
The linked list you have created is:
2       4       0       6
--------------------------------
Process exited after 49.16 seconds with return value 0
Press any key to continue . . .
```

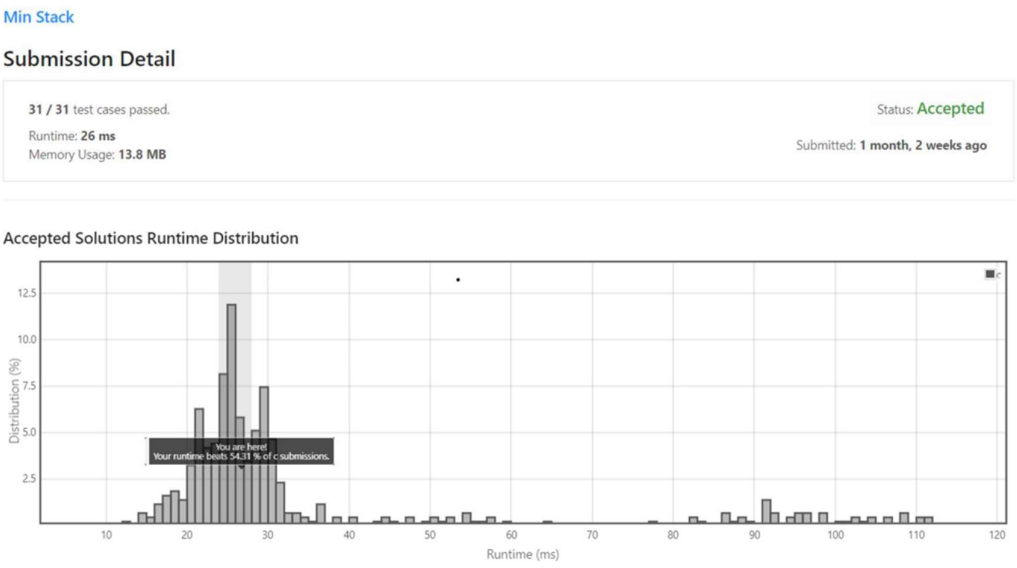
Leetcode Problem-1:

Min Stack Problem

```
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
   int value;
   int min;
} StackNode;
typedef struct
{
   StackNode *array;
   int capacity;
   int top;
}MinStack;
MinStack* minStackCreate()
{
   MinStack* stack=(MinStack*)malloc(sizeof(MinStack));
   stack->capacity = 10;
   stack->array = (StackNode*)malloc(stack->capacity * sizeof(StackNode));
   stack->top = -1;
   return stack;
 }
void minStackPush(MinStack* obj, int val)
{
   if (obj->top == obj->capacity - 1)
   {
      obj->capacity *= 2;
      obj->array =(StackNode*)realloc(obj->array,obj->capacity*size(StackNode));
   }
   StackNode newNode;
```

```
    newNode.value = val;
    newNode.min = (obj->top == -1) ? val : (val < obj->array[obj->top].min) ? val : obj-
>array[obj->top].min;
    obj->array[++(obj->top)] = newNode;
}
void minStackPop(MinStack* obj)
{
    if (obj->top != -1)
    {
        obj->top--;
    }
}
int minStackTop(MinStack* obj)
{
    if (obj->top != -1)
    {
         return obj->array[obj->top].value;
    }
     return -1;
}
int minStackGetMin(MinStack* obj)
{
    if (obj->top != -1)
    {
        return obj->array[obj->top].min;
    }
    return -1;
}
void minStackFree(MinStack* obj)
{
    free(obj->array);
    free(obj);
}
```

Output:



Lab program 5:

1.Write a program 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>
#define pf printf
#define sf scanf

struct node{
   int data;
   struct node *next;
};
struct node *head,*newnode,*temp,*temp1;


void create()
{
   newnode=(struct node *)malloc(sizeof(struct node));
   pf("Enter the data of the node to be inserted at the beginning:");
   sf("%d",&newnode->data);
   newnode->next=head;
   head=newnode;
   pf("\n---The element %d has been inserted at beg---",newnode->data);
   /*display*/
   pf("\nThe linked list you have created is: \n");
   temp=head;
   while(temp!=NULL)
   {

      pf("%d\t",temp->data);
      temp=temp->next;
   }
}

void delete_at_beg(){
   temp=head;
   if(head==NULL){
      pf("\nThe linked list is empty");
   }
   else if(head->next==NULL)
   {
      head=NULL;
      pf("\nThe only node is deleted, the list is now empty")  ;
   }
   else{
```

```
        head=temp->next;
        free(temp);
    }
}
void delete_at_end()
{
        if(head==NULL)
    {
        pf("\nThe linked list is empty");
    }
    else if(head->next==NULL)
    {
        head=NULL;
        pf("\nThe only node is deleted, the list is now empty")  ;
    }
    else
        {
            temp=head;
        while(temp->next!=NULL)
        {
                temp1=temp;
                temp=temp->next;

        }
        temp1->next=NULL;
        free(temp);
}
}

void delete_atpos()
{
    temp=head;
    int pos,i;


    if(head==NULL){
        pf("\nThe linked list is empty");
    }
    else if(head->next==NULL)
    {
        head=NULL;
        pf("\nThe only node is deleted, the list is now empty");
    }
    else{
        pf("\nEnter the position of element to be deleted:");
        sf("%d",&pos);

    for(i=1;i<pos;i++)
    {
```

```
        temp1=temp;
        temp=temp->next;
        if(temp==NULL){
            pf("\nThere are less than %d elements in the list",pos);
            return 0;
            }
    }
    temp1->next=temp->next;
    free(temp);
    }

}

void display()
{
     pf("The linked list you have created is: \n");
    temp=head;
    while(temp!=NULL)
    {
        pf("%d\t",temp->data);
        temp=temp->next;
    }


}

int main()
{
    int ch;
    while(1)
    {
         pf("\n----Menu----\n 1.Create\n2.Delete_at_beg\n3.delete at end\n4.Delete at
pos\n5.Display: ");
        sf("%d",&ch);
    switch(ch)
    {
    case 1:create();
            break;
    case 2:delete_at_beg();
            break;
    case 3:delete_at_end();
            break;
    case 4:delete_atpos();
            break;
    case 5: display();
           return 0;
        }
    }
}
```

Output:

```
---The element 80 has been inserted at beg---
The linked list you have created is:
80      60      40      20
----Menu----
 1.Create
2.Delete_at_beg
3.delete at end
4.Delete at pos
5.Display: 2

----Menu----
 1.Create
2.Delete_at_beg
3.delete at end
4.Delete at pos
5.Display: 3

----Menu----
 1.Create
2.Delete_at_beg
3.delete at end
4.Delete at pos
5.Display: 4

Enter the position of element to be deleted:2

----Menu----
 1.Create
2.Delete_at_beg
3.delete at end
4.Delete at pos
5.Display: 5
The linked list you have created is:
60
--------------------------------
Process exited after 48.74 seconds with return value 0
Press any key to continue . . .
```

Leetcode Problem-2:

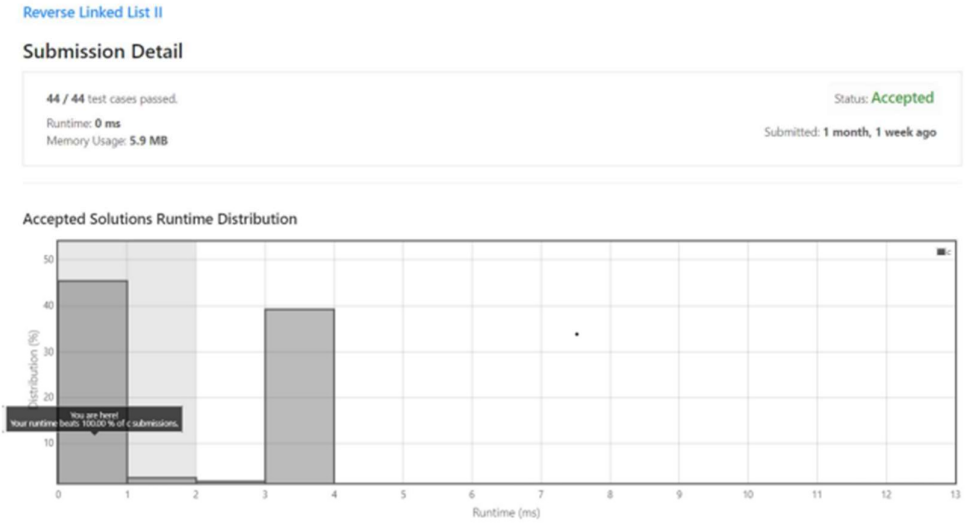
Reversal Linked List-II

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 * int val;
 * struct ListNode *next;
```

```
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
if (head == NULL) return NULL;
if (left == right) return head;
struct ListNode* prev = NULL;
struct ListNode* curr = head;
int index = 1;
while (index < left)
{
   prev = curr;
   curr = curr->next;
   index++;
 }
struct ListNode* leftMinusOneNode = prev;
struct ListNode* leftNode = curr;
struct ListNode* next = NULL;
while (left <= right)
{
   next = curr->next;
   curr->next = prev;
   prev = curr;
   curr = next;
   left++;
}
if (leftMinusOneNode == NULL) // means head changes
head = prev;
else
leftMinusOneNode->next = prev;
```

```
leftNode->next = curr;
return head;
}
```

Output:



Lab program 6:

(a) Write a program 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;
};

typedef struct Node Node;

Node *createNode(int value)
{
   Node *newNode=(Node*)malloc(sizeof(Node));
   newNode->data=value;
   newNode->next=NULL;
   return newNode;

}

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

Node *sortList(Node *head)
{
   if(head==NULL || head->next==NULL)
   {
      return head;
   }


   int swapped;
   Node *temp;
   Node *end=NULL;

   do
   {
```

```
        swapped=0;
        temp=head;
        while(temp->next !=end)
        {
           if(temp->data > temp->next->data)
           {
              int tempData=temp->data;
              temp->data=temp->next->data;
              temp->next->data=tempData;
              swapped=1;

           }
           temp=temp->next;
        }
        end=temp;
    }while(swapped);

    return head;
}

Node *reverseList(Node *head)
{
    Node *prev=NULL;
    Node *current=head;
    Node *nextNode=NULL;

    while(current!=NULL)
    {
       nextNode=current->next;
       current->next=prev;
       prev=current;
       current=nextNode;
    }
    return prev;

}

Node *concatenatedLists(Node *list1,Node *list2)
{

    if(list1==NULL)
    {
      return list2;
    }

    Node *temp=list1;
    while(temp->next!=NULL)
```

```
    {
        temp=temp->next;
    }
    temp->next=list2;
    return list1;

}

int main()
{
    Node *list1=createNode(3);
    list1->next=createNode(1);
    list1->next->next=createNode(4);

    Node *list2=createNode(2);
    list2->next=createNode(5);

    printf("Original list 1:");
    displayList(list1);

    printf("Original list 2:");
    displayList(list2);

    list1=sortList(list1);
    printf("Sorted list 1:");
    displayList(list1);

    list1=reverseList(list1);
    printf("Reversed list 1:");
    displayList(list1);

    Node *Concatenated= concatenatedLists(list1,list2);
    printf("Concatenated List:");
    displayList(Concatenated);
    return 0;
}
```

Output:

```
Original list 1:3->1->4->NULL
Original list 2:2->5->NULL
Sorted list 1:1->3->4->NULL
Reversed list 1:4->3->1->NULL
Concatenated List:4->3->1->2->5->NULL
```

(b) Write a program to Implement Single Link List to simulate Stack & Queue Operations.

Stack Implementation

```
#include<stdio.h>
#include<stdlib.h>
#define pf printf
#define sf scanf

struct node{
   int data;
   struct node *next;
};
struct node *head,*newnode,*temp,*temp1;


void push()
{
   newnode=(struct node *)malloc(sizeof(struct node));
   pf("Enter the data to be pushed into the stack:");
   sf("%d",&newnode->data);
   newnode->next=head;
   head=newnode;

}
 void display()
 {
    pf("\nThe stack you have created is:\n");
    temp=head;
    while(temp!=NULL)
   {
      pf("%d\n",temp->data);
      temp=temp->next;
   }
}

void pop()
{
   if(head==NULL)
      pf("\nThe stack is empty");
   else
   {
      temp=head;
      head=temp->next;
      free(temp);
   }
 }

 int main()
```

```
{
int ch;

while(1)
  {
     pf("\n1.Push\n2.Pop\n3.Display\n4.Exit Menu\nChoice:");
    sf("%d",&ch);

switch(ch)
     {
  case 1: push();
          display();
          break;
  case 2: pop();
          display();
          break;
  case 3: display();
          break;
  case 4: return 0;
  default:
          pf("\nInvalid input--try again\n\n\n");
     }
  }
}
```

Output:

```
1.Push
2.Pop
3.Display
4.Exit Menu
Choice:1
Enter the data to be pushed into the stack:20
The stack you have created is:
20

1.Push
2.Pop
3.Display
4.Exit Menu
Choice:1
Enter the data to be pushed into the stack:45
The stack you have created is:
45
20
```

```
1.Push
2.Pop
3.Display
4.Exit Menu
Choice:1
Enter the data to be pushed into the stack:37
The stack you have created is:
37
45
20

1.Push
2.Pop
3.Display
4.Exit Menu
Choice:2
The stack you have created is:
45
20

1.Push
2.Pop
3.Display
4.Exit Menu
Choice:4
```

Queue Implementation

```
#include<stdio.h>
#include<stdlib.h>
#define pf printf
#define sf scanf

struct node{
   int data;
   struct node *next;
};
struct node *head,*newnode,*temp,*temp1;


void pushqueue()
{
   newnode=(struct node *)malloc(sizeof(struct node));
   pf("Enter the data to be pushed into the queue:");
   sf("%d",&newnode->data);
```

```
   newnode->next=head;
   head=newnode;

}
 void display()
 {
    pf("\nThe queue you have created is:\n");
    temp=head;
    while(temp!=NULL)
   {
      pf("\t%d",temp->data);
      temp=temp->next;
   }
}

void popqueue()
{
   if(head==NULL)
      pf("\nThe queue is empty");
   else
   {
      temp=temp1=head;
      while(temp->next!=NULL)
          {
          temp1=temp;
          temp=temp->next;
          }
      temp1->next=NULL;
      free(temp);
   }
 }

 int main()
 {
 int ch;

 while(1)
   {
       pf("\n1.Push\n2.Pop\n3.Display\n4.Exit Menu\nChoice:");
          sf("%d",&ch);

 switch(ch)
       {
   case 1: pushqueue();
          display();
          break;
   case 2: popqueue();
          display();
```

```
            break;
    case 3: display();
            break;
    case 4: return 0;
    default:
            pf("\nInvalid input--try again\n\n\n");
        }
    }
}
```

Output:

```
1.Push
2.Pop
3.Display
4.Exit Menu
Choice:1
Enter the data to be pushed into the queue:35
The queue you have created is:
    35
1.Push
2.Pop
3.Display
4.Exit Menu
Choice:1
Enter the data to be pushed into the queue:89
The queue you have created is:
    89  35
1.Push
2.Pop
3.Display
4.Exit Menu
Choice:1
Enter the data to be pushed into the queue:76
The queue you have created is:
    76  89  35
1.Push
2.Pop
3.Display
4.Exit Menu
Choice:2
The queue you have created is:
    76  89
```

```
1.Push
2.Pop
3.Display
4.Exit Menu
Choice:4
```

Lab program 7:

1.Write a program 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* createNode(int data)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL)
   {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}

void insertNodeToLeft(struct Node* head, struct Node* target, int data)
{
   struct Node* newNode = createNode(data);
   if (target->prev != NULL)
   {
      target->prev->next = newNode;
      newNode->prev = target->prev;
   }
   else
   {
      head = newNode;
   }
   newNode->next = target;
   target->prev = newNode;
}

void deleteNode(struct Node* head, int value)
{
```

```
    struct Node* current = head;
    while (current != NULL)
    {
        if (current->data == value)
        {
            if (current->prev != NULL)
            {
                current->prev->next = current->next;
            }
            else
            {
                head = current->next;
            }
            if (current->next != NULL)
            {
                current->next->prev = current->prev;
            }
            free(current);
            return;
        }
        current = current->next;
    }
    printf("Node with value %d not found\n", value);
}

void displayList(struct Node* head)
{
    printf("Doubly Linked List: ");
    while (head != NULL)
    {
        printf("%d <-> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

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


    head = createNode(1);
    head->next = createNode(2);
    head->next->prev = head;
    head->next->next = createNode(3);
    head->next->next->prev = head->next;

    displayList(head);

    insertNodeToLeft(head, head->next, 10);
```

```
    printf("After insertion:\n");
    displayList(head);

    deleteNode(head, 2);
    printf("After deletion:\n");
    displayList(head);

    return 0;
}
```

Output:

```
Doubly Linked List: 1 <-> 2 <-> 3 <-> NULL
After insertion:
Doubly Linked List: 1 <-> 10 <-> 2 <-> 3 <-> NULL
After deletion:
Doubly Linked List: 1 <-> 10 <-> 3 <-> NULL
```

Leetcode Problem-3:

Split Linked List in Parts

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
/**
 * Note: The returned array must be malloced, assume caller calls free().
*/ int getLength(struct ListNode* head) {
    int length = 0;
    while (head != NULL) {
        length++;
        head = head->next;
    }
    return length;
 }
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    int length = getLength(head);
    int partSize = length / k;
    int remainder = length % k;

    struct ListNode** result = (struct ListNode**)malloc(k * sizeof(struct ListNode*)); *returnSize = k;
    for (int i = 0; i < k; i++) {
        int currentPartSize = partSize + (i < remainder ? 1 : 0);
        if (currentPartSize == 0) {
            result[i] = NULL;
```

```
        } else {
            result[i] = head;
            for (int j = 0;
            j < currentPartSize - 1; j++) {
                head = head->next;
            }
            struct ListNode* temp = head->next;
            head->next = NULL;
            head = temp;
        }
    }
    return result;
}
```

Output:

Split Linked List in Parts

Submission Detail

43 / 43 test cases passed.

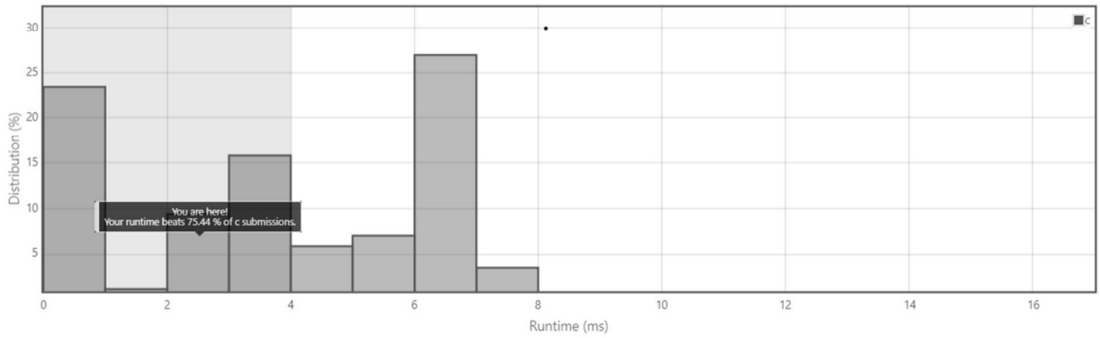
Runtime: **2 ms**

Memory Usage: **6.2 MB**

Status: Accepted

Submitted: **1 month ago**

Accepted Solutions Runtime Distribution



Lab program 8:

Write a program
a) To construct a Binary Search Tree.
b) To traverse the tree using all the methods i.e., in-order, preorder and post order
c) To display the elements in the tree.

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

typedef struct BST
{
   int data;
   struct BST *left;
   struct BST *right;
}node;

node *create()
{
   node *temp;
   printf("Enter data:\n");
   temp=(node*)malloc(sizeof(node));
   scanf("%d",&temp->data);
   temp->left=temp->right=NULL;
   return temp;
}

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

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

void postorder(node *root)
{
   if(root!=NULL)
```

```
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d  ",root->data);
    }
}

void insert(node *root,node *temp)
{
    if(temp->data<root->data)
    {
        if (root->left!=NULL)
            insert(root->left,temp);
        else
            root->left=temp;
    }
    else if (temp->data>root->data)
    {
        if (root->right!=NULL)
            insert(root->right,temp);
        else
            root->right=temp;
    }
    else
    {
        printf("Ignoring duplicate data %d\n",temp->data);
    }
}

int main()
{
    char ch;
    int choice;
    node *root=NULL,*temp;
    do
    {
        temp=create();
        if(root==NULL)
            root=temp;
        else
            insert(root,temp);
        printf("\nDo you want to continue? (y/n): ");
        getchar();
        scanf("%c",&ch);
    }
    while(ch=='y'|ch=='Y');

    printf("Choose traversal type\n");
    printf("1.Preorder traversal\n");
    printf("2.Postorder traversal\n");
```

```
    printf("3.Inorder traversal\n");
    printf("Enter your choose(1-3)\n");
    scanf("%d",&choice);

    switch(choice){
        case 1:
            printf("\nBST in preorder traversal\n");
            preorder(root);
            break;
        case 2:
            printf("\nBST in postorder traversal\n");
            postorder(root);
            break;
        case 3:
            printf("BST in inorder traversal\n");
            inorder(root);
            break;
        default:
            printf("Invalid choice");
    }
    return 0;
}
```

Output:

```
Enter data:
15
Do you want to continue? (y/n): y
Enter data:
5
Do you want to continue? (y/n): y
Enter data:
20
Do you want to continue? (y/n): y
Enter data:
37
Do you want to continue? (y/n): y
Enter data:
24
Do you want to continue? (y/n): y
Enter data:
3
Do you want to continue? (y/n): n
```

```
Choose traversal type
1.Preorder traversal
2.Postorder traversal
3.Inorder traversal
Enter your choose(1-3)
2

BST in postorder traversal
3  5  24  37  20  15
```

Leetcode Problem-4:

Rotate List

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* rotateRight(struct ListNode* head, int k) {
   if (head == NULL || head->next == NULL || k == 0)
      return head;
   int len = 1;
   struct ListNode *tail = head;
   while (tail->next != NULL) {
      tail = tail->next;
      len++;
   }
   k = k % len;
   if (k == 0)
   return head;
   struct ListNode *p = head;
   for (int i = 0; i < len - k - 1; i++) {
      p = p->next;
    }
   tail->next = head;
   head = p->next
   p->next = NULL;
   return head;
}
```

Output:

Rotate List

Submission Detail

232 / 232 test cases passed.

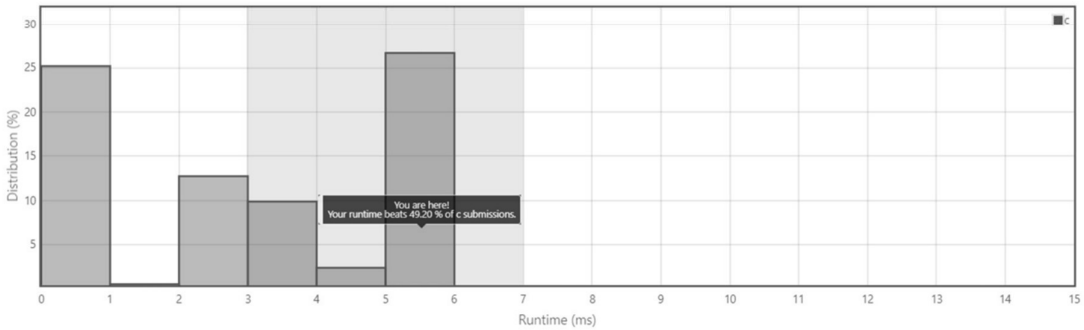
Status: Accepted

Runtime: **5 ms**

Memory Usage: **6.1 MB**

Submitted: **2 weeks, 3 days ago**

Accepted Solutions Runtime Distribution



Lab program 9:

(a) Write a program to traverse a graph using BFS method.

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

#define MAX_SIZE 100

{
   int items[MAX_SIZE];
   int front;
   int rear;
};

struct Queue* createQueue()
{
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = -1;
   queue->rear = -1;
   return queue;
}

int isEmpty(struct Queue* queue)
{
   if (queue->rear == -1)
      return 1;
   else
      return 0;
}

void enqueue(struct Queue* queue, int value)
{
   if (queue->rear == MAX_SIZE - 1)
      printf("\nQueue is Full!!");
   else
   {
      if (queue->front == -1)
         queue->front = 0;
      queue->rear++;
      queue->items[queue->rear] = value;
   }
}

int dequeue(struct Queue* queue)
{
   int item;
   if (isEmpty(queue))
   {
      printf("Queue is empty");
      item = -1;
```

```
    }
    else
    {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear)
        {
            queue->front = queue->rear = -1;
        }
    }
    return item;
}

struct Graph
{
    int vertices;
    int** adjMatrix;
};

struct Graph* createGraph(int vertices)
{
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->vertices = vertices;

    graph->adjMatrix = (int*)malloc(vertices * sizeof(int));
    for (int i = 0; i < vertices; i++)
    {
        graph->adjMatrix[i] = (int*)malloc(vertices * sizeof(int));
        for (int j = 0; j < vertices; j++)
            graph->adjMatrix[i][j] = 0;
    }
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest)
{
    graph->adjMatrix[src][dest] = 1;
    graph->adjMatrix[dest][src] = 1; // Uncomment if the graph is undirected
}

// Breadth First Search traversal
void BFS(struct Graph* graph, int startVertex)
{
    int visited[MAX_SIZE] = {0}; // Initialize all vertices as not visited
    struct Queue* queue = createQueue();

    visited[startVertex] = 1;
    enqueue(queue, startVertex);

    printf("Breadth First Search Traversal: ");
```

```
    while (!isEmpty(queue))
    {
        int currentVertex = dequeue(queue);
        printf("%d ", currentVertex);

        for (int i = 0; i < graph->vertices; i++)
        {
            if (graph->adjMatrix[currentVertex][i] == 1 && visited[i] == 0)
            {
                visited[i] = 1;
                enqueue(queue, i);
            }
        }
    }
    printf("\n");
}

int main()
{
    int vertices, edges, src, dest;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    struct Graph* graph = createGraph(vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (int i = 0; i < edges; i++)
    {
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d%d", &src, &dest);
        addEdge(graph, src, dest);
    }

    int startVertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);

    BFS(graph, startVertex);

    return 0;
}
```

Output:

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 2
Enter edge 3 (source destination): 1 3
Enter edge 4 (source destination): 2 3
Enter the starting vertex for BFS: 2
Breadth First Search Traversal: 2 0 3 1
```

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

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

#define MAX_SIZE 100

// Graph implementation
struct Graph
{
   int vertices;
   int** adjMatrix;
};

// Create a graph with 'vertices' number of vertices
struct Graph* createGraph(int vertices)
{
   struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
   graph->vertices = vertices;

   graph->adjMatrix = (int*)malloc(vertices * sizeof(int));
   for (int i = 0; i < vertices; i++)
   {
      graph->adjMatrix[i] = (int*)malloc(vertices * sizeof(int));
      for (int j = 0; j < vertices; j++)
         graph->adjMatrix[i][j] = 0;
   }
   return graph;
}

// Add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest)
{
   graph->adjMatrix[src][dest] = 1;
   graph->adjMatrix[dest][src] = 1; // Uncomment if the graph is undirected
}
```

```
// Depth First Search traversal
void DFS(struct Graph* graph, int startVertex, int visited[])
{
    visited[startVertex] = 1;

    for (int i = 0; i < graph->vertices; i++)
    {
        if (graph->adjMatrix[startVertex][i] == 1 && visited[i] == 0)
            DFS(graph, i, visited);
    }
}

// Check if the graph is connected
int isConnected(struct Graph* graph)
{
    int* visited = (int*)malloc(graph->vertices * sizeof(int));

    for (int i = 0; i < graph->vertices; i++)
        visited[i] = 0;

    DFS(graph, 0, visited);

    for (int i = 0; i < graph->vertices; i++)
    {
        if (visited[i] == 0)
            return 0; // Graph is not connected
    }
    return 1; // Graph is connected
}

int main()
{
    int vertices, edges, src, dest;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    struct Graph* graph = createGraph(vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (int i = 0; i < edges; i++)
    {
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d%d", &src, &dest);
        addEdge(graph, src, dest);
    }
```

```
    if (isConnected(graph))
        printf("The graph is connected.\n");
    else
        printf("The graph is not connected.\n");

    return 0;
}
```

Output:

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 1 2
Enter edge 3 (source destination): 2 3
Enter edge 4 (source destination): 3 1
The graph is connected.
```

HackerRank Problem

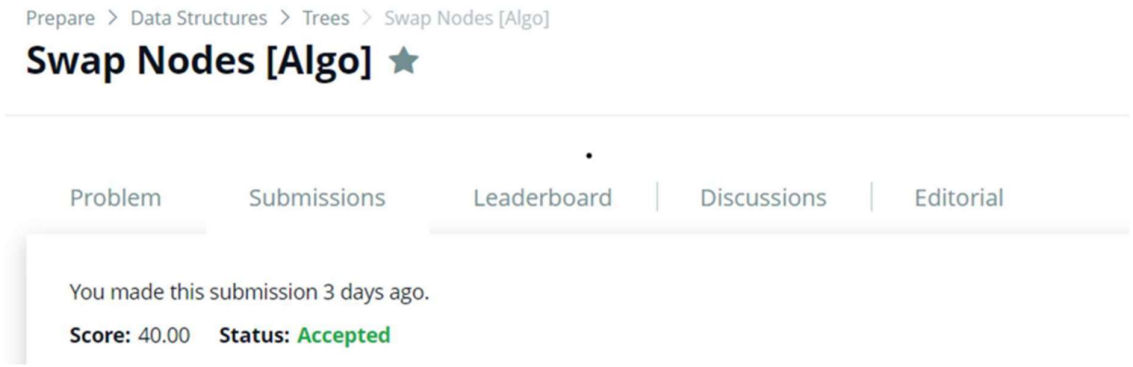
Swap Nodes

```
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void inOrderTraversal(Node* root, int* result, int* index) {
    if (root == NULL) return;
    inOrderTraversal(root->left, result, index);
    result[(*index)++] = root->data;
    inOrderTraversal(root->right, result, index);
```

```
}

void swapAtLevel(Node* root, int k, int level) {
    if (root == NULL) return;
    if (level % k == 0) {
        Node* temp = root->left;
        root->left = root->right;
        root->right = temp;
    }
    swapAtLevel(root->left, k, level + 1);
    swapAtLevel(root->right, k, level + 1);
}

int** swapNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count,
                int* queries, int* result_rows, int* result_columns) {
    // Build the tree
    Node** nodes = (Node**)malloc((indexes_rows + 1) * sizeof(Node*));
    for (int i = 1; i <= indexes_rows; i++) {
        nodes[i] = createNode(i);
    }
    for (int i = 0; i < indexes_rows; i++) {
        int leftIndex = indexes[i][0];
        int rightIndex = indexes[i][1];
        if (leftIndex != -1) nodes[i + 1]->left = nodes[leftIndex];
        if (rightIndex != -1) nodes[i + 1]->right = nodes[rightIndex];
    }

    // Perform swaps and store results
    int** result = (int**)malloc(queries_count * sizeof(int*));
    *result_rows = queries_count;
    *result_columns = indexes_rows;
    for (int i = 0; i < queries_count; i++) {
        swapAtLevel(nodes[1], queries[i], 1);
        int* traversalResult = (int*)malloc(indexes_rows * sizeof(int));
        int index = 0;
        inOrderTraversal(nodes[1], traversalResult, &index);
        result[i] = traversalResult;
    }
    free(nodes);
    return result;
}

int main() {
    int n;
    scanf("%d", &n);
    int** indexes = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        indexes[i] = malloc(2 * sizeof(int));
        scanf("%d %d", &indexes[i][0], &indexes[i][1]);
    }
```

```
    int queries_count;
    scanf("%d", &queries_count);
    int* queries = malloc(queries_count * sizeof(int));
    for (int i = 0; i < queries_count; i++) {
        scanf("%d", &queries[i]);
    }

    int result_rows;
    int result_columns;
    int** result = swapNodes(n, 2, indexes, queries_count, queries, &result_rows,
                             &result_columns);
    for (int i = 0; i < result_rows; i++) {
        for (int j = 0; j < result_columns; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
        free(result[i]);  // Free memory allocated for each row
    }
    free(result);  // Free memory allocated for the result array

    // Free memory allocated for indexes and queries arrays
    for (int i = 0; i < n; i++) {
        free(indexes[i]);
    }
    free(indexes);
    free(queries);
    return 0;
}
```

Output:



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 -> L as H(K)=K mod 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 TABLE_SIZE 10

int hashFunction(int key) {
   return key % TABLE_SIZE;
}

void insertValue(int hashTable[], int key) {
   int i = 0;
   int hkey = hashFunction(key);
   int index;
   do {
      index = (hkey + i) % TABLE_SIZE;
      if (hashTable[index] == -1) {
         hashTable[index] = key;
         printf("Inserted key %d at index %d\n", key, index);
         return;
      }
      i++;
   } while (i < TABLE_SIZE);
   printf("Unable to insert key %d. Hash table is full.\n", key);
}

int searchValue(int hashTable[], int key) {
   int i = 0;
   int hkey = hashFunction(key);
   int index;
   do {
      index = (hkey + i) % TABLE_SIZE;
      if (hashTable[index] == key) {
         printf("Key %d found at index %d\n", key, index);
         return index;
      }
      i++;
```

```
    } while (i < TABLE_SIZE);
    printf("Key %d not found in hash table.\n", key);
    return -1;
}

int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = -1;
    }
    insertValue(hashTable, 1234);
    insertValue(hashTable, 5678);
    insertValue(hashTable, 9876);
    searchValue(hashTable, 5678);
    searchValue(hashTable, 1111);
    return 0;
}
```

Output:

```
Inserted key 1234 at index 4
Inserted key 5678 at index 8
Inserted key 9876 at index 6
Key 5678 found at index 8
Key 1111 not found in hash table.

Process returned 0 (0x0)   execution time : 0.074 s
Press any key to continue.
```