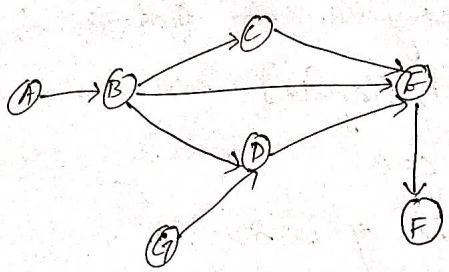
20MCA135 - DATA STRUCTURES LAB
FIRST SEMESTER MCA

SAKTHI PRIYA M
MCA 131

QUESTION

1) Consider a directed acyclic graph G given in following figure:



Develop a program to implement topological sorting

2) Write a program for creating doubly LL and perform the following operations:

a) Insert an element at particular position

b) Search an element

c) Delete an element at the end of the list.

ALGORITHM :

Algorithm:

1) Step 1 : Start

Step 2 : Initialize the necessary variables such as i, j, k, a[10][10], deg[10], flag[10]

Step 3 : Initialize count c=0

Step 4 : Read the number of vertices, n

Step 5 : for i in range 0 to n

Step 5.1 : for j in range 0 to n

Step 5.3 : Read the adjacency matrix, a[i][j]

Step 6 : for i in range 0 to n

Step 6.1 : Assign deg[i] = 0, flag[i] = 0

Step 7 : for i in range 0 to n

Step 7.1 : for j in range 0 to n

Step 7.2 : Assign deg[i] = deg[i] + a[j][i]

Step 8 : while c < n

Step 8.1 : for k in range 0 to n

Step 8.1.1 : if deg[k] == 0 and flag[k] == 0

Step 8.1.2 : print vertices, k+1 and Assign flag[k] = 1

Step 8.2 : for i in range 0 to n

Step 8.2.1 : if a[i][k] == 1 then deg[k] --

Step 8.3 : Increment c

Step 9 : Stop.

2) CREATE LIST

Step 1 : if n >= 1

Step 1.1 : print "Enter data of node 1 "

Step 1.2 : Read data

Step 1.3 : head → data = data

head → prev = NULL

head → next = NULL

last = head

Step 1.4 : for i in range 2 to n

Step 1.4.1 : Read the data to be inserted

Step 1.4.2 : newNode → data = data

newNode → prev = last

newNode → next = NULL

last → next = newNode

last = newNode.

Step 2 : print "Doubly Linked List created Successfully"

INSERT AT ANY POSITION

Step 1 : if head == NULL

Step 1.1 : print " List is Empty "

Step 2 : else

Step 2.1 : Assign temp = head

i = 1

Step 2.2 : while i < position - 1 && temp != NULL

Step 2.3 : temp = temp -> next
i++.

Step 2.4 : if temp != NULL

Step 2.4.1 : newNode -> data = data
newNode -> next = temp -> next
newNode -> prev = temp.

Step 2.4.2 : if temp -> next != NULL then
temp -> next -> prev = newNode.

Step 2.4.3 : temp -> next = newNode.
print "Node inserted successfully"

Step 2.4.4 : else print "Invalid"

SEARCH ()

Step 1 : Assign ptr = head

Step 2 : if ptr == NULL then print "Empty List"

Step 3 : else
print the value of node to be inserted.

Step 3.1 : while & ptr != NULL

Step 3.1.1 : if ptr -> data == item
print "Node found" and assign flag = 0

Step 3.1.2 : else assign flag = 1

Step 3.2 : Increment i
ptr = ptr -> next.

Step 3.3 : if flag == 1 , print " Node not found"

DELETE AT END

Step 1 : if head == NULL print " Cannot delete "

Step 2 : else if head → next == NULL

Step 2.1 : head = NULL
free (head)
print " Node deleted"

Step 3 : else

Step 3.1 : ptr = head

Step 3.2 : if (ptr → next != NULL then ,
ptr = ptr → next

Step 3.3 : ptr → prev => next = NULL
free (ptr)
print "Node deleted"

DISPLAY

Step 1 : if head == NULL then print "List empty"

Step 2 : else

Step 2.1 : ~~print~~ " Assign ptr = head

Step 2.2 : while ptr != NULL

Step 2.2.1 : print the data and assign ptr = ptr → next

PROGRAM CODE:

1) TOPOLOGICAL SORTING

```
#include<stdio.h>

int main()
{
	int n,i,j,k,a[10][10],deg[10],flag[10];
	int c=0;

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

	printf("Enter the adjacency matrix of the graph: ");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			scanf("%d",&a[i][j]);
		}
	}

	for(i=0;i<n;i++)
	{
		deg[i]=0;
		flag[i]=0;
	}

	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			deg[i]=deg[i]+a[j][i];
		}
	}
```

```
	printf("\n The topological order of vertices in the given graph is : \n");

	while(c<n)
	{
		for(k=0;k<n;k++)
		{
			if((deg[k]==0) && (flag[k]==0))
			{
				printf(" %d\t",(k+1));
				flag[k]=1;
			}
		for(i=0;i<n;i++)
		{
			if(a[i][k]==1)
				deg[k]--;
		}
		}
		c++;
	}
	return 0;
}
```

2) DOUBLY LINKED LIST

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

struct node *head,*last;

void createList(int n);
void insert_at_anyposition(int data,int position);
void search();
void delete_at_end();
void display();

void main ()
{
        int choice=0,n,data;
        while(choice!=6)
        {
                printf("-----DOUBLY LINKED LIST-----");
                printf("\n1.Create the list \n2.Insert at particular position \n3.Search an
element \n4.Delete an element at end \n5.Display \n6.Exit\n");
                printf("Enter your choice : ");
                scanf("\n%d",&choice);

                switch(choice)
                {
                        case 1:printf("Enter number of nodes to be inserted: ");
                                scanf("%d",&n);
                                createList(n);
```

```
                        break;
                    case 2:printf("Enter the position where you want to insert new node:
");
                    scanf("%d", &n);
                    printf("Enter data of %d node : ", n);
                    scanf("%d", &data);
                        insert_at_anyposition(data,n);
                        break;
                    case 3:search();
                        break;
                    case 4:delete_at_end();
                        break;
                    case 5:display();
                        break;
                    case 6:exit(0);
                        break;
                    default: printf("Enter a valid choice");
                }
        }
}

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

   if(n >= 1)
   {
      head = (struct node *)malloc(sizeof(struct node));

      printf("Enter data of node 1: ");
      scanf("%d", &data);

      head->data = data;
      head->prev = NULL;
      head->next = NULL;
```

```
        last = head;
        for(i=2; i<=n; i++)
        {
            newNode = (struct node *)malloc(sizeof(struct node));

            printf("Enter data of node %d: ", i);
            scanf("%d", &data);

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

            last->next = newNode;
            last = newNode;
        }

        printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
    }
}
void insert_at_anyposition(int data,int position)
{
          int i;
    struct node * newNode, *temp;

    if(head == NULL)
    {
        printf(" List is empty!\n");
    }
    else
    {
        temp = head;
        i=1;

        while(i<position-1 && temp!=NULL)
        {
            temp = temp->next;
            i++;
```

```
        }

       if(temp!=NULL)
        {
           newNode = (struct node *)malloc(sizeof(struct node));

           newNode->data = data;
           newNode->next = temp->next;
           newNode->prev = temp;
           if(temp->next != NULL)
           {
             temp->next->prev = newNode;
           }

           temp->next = newNode;

           printf("NODE INSERTED SUCCESSFULLY AT %d POSITION\n", position);
       }
       else
       {
           printf(" Invalid position\n");
       }
    }
}

void search()
{
          struct node *ptr;
          int item,i=0,flag;
          ptr = head;
          if(ptr == NULL)
          {
                    printf("\nEmpty list\n");
          }
          else
          {
                    printf("\nEnter the value of node you want to search:\n");
```

```
            scanf("%d",&item);
            while (ptr!=NULL)
            {
                if(ptr->data == item)
                {
                    printf("\nNode found at %d position\n ",i+1);
                    flag=0;
                    break;
                }
                else
                {
                    flag=1;
                }
            i++;
            ptr = ptr -> next;
            }
            if(flag==1)
            {
                printf("\nNode not found\n");
            }
        }
}

void delete_at_end()
{
        struct node *ptr;
        if(head == NULL)
        {
            printf("\nCannot delete");
        }
        else if(head->next == NULL)
        {
            head = NULL;
            free(head);
            printf("\nNode deleted\n");
        }
        else
```

```
        {
                ptr = head;
                if(ptr->next != NULL)
                {
                ptr = ptr -> next;
                }
                ptr -> prev -> next = NULL;
                free(ptr);
                printf("\nNode deleted\n");
        }
 }

void display()
{
        struct node *ptr;
        if(head == NULL)
        {
        printf("List is empty");
        }
        else
        {
        printf("\n The nodes in DoublyLL : \n");
        ptr = head;
        while(ptr != NULL)
        {
                printf("%d\n",ptr->data);
                ptr=ptr->next;
        }
        }
}
```

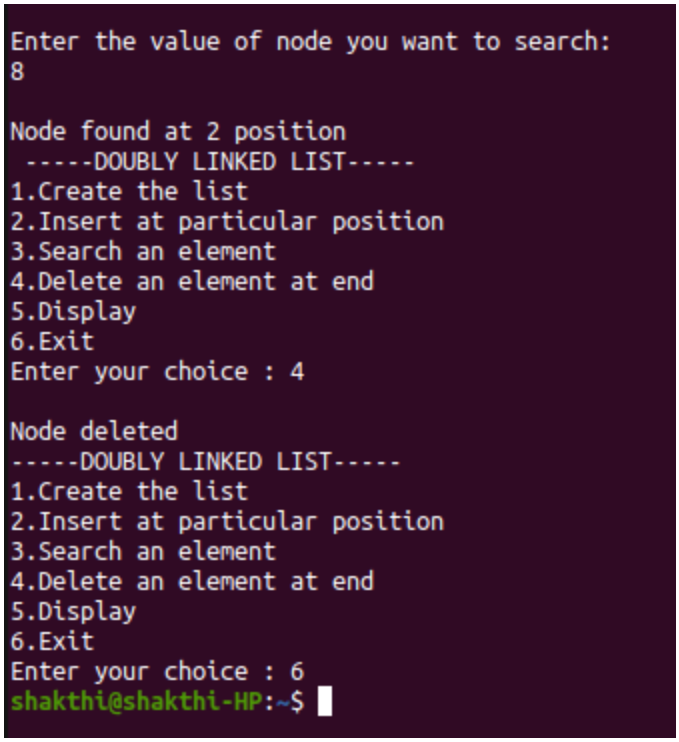
OUTPUT :

1)

```
shakthi@shakthi-HP:~$ gcc topo.c -o t
shakthi@shakthi-HP:~$ ./t
Enter the number of vertices in the given graph: 7
Enter the adjacency matrix of the graph:
0 1 0 0 0 0 0
0 0 1 1 1 0 0
0 0 0 0 1 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0

 The topological order of vertices in the given graph is :
 1       7       2       3       4       5       6       shakthi@shakthi-HP:~$
```

2)

```
shakthi@shakthi-HP:~$ gcc doublyLL.c -o d
shakthi@shakthi-HP:~$ ./d
-----DOUBLY LINKED LIST-----
1.Create the list
2.Insert at particular position
3.Search an element
4.Delete an element at end
5.Display
6.Exit
Enter your choice : 1
Enter number of nodes to be inserted: 3
Enter data of node 1: 4
Enter data of node 2: 5
Enter data of node 3: 6

DOUBLY LINKED LIST CREATED SUCCESSFULLY
-----DOUBLY LINKED LIST-----
1.Create the list
2.Insert at particular position
3.Search an element
4.Delete an element at end
5.Display
6.Exit
Enter your choice : 2
Enter the position where you want to insert new node: 1
Enter data of 1 node : 8
NODE INSERTED SUCCESSFULLY AT 1 POSITION
-----DOUBLY LINKED LIST-----
1.Create the list
2.Insert at particular position
3.Search an element
4.Delete an element at end
5.Display
6.Exit
Enter your choice : 5

 The nodes in DoublyLL :
4
8
5
6
-----DOUBLY LINKED LIST-----
1.Create the list
2.Insert at particular position
3.Search an element
4.Delete an element at end
5.Display
6.Exit
Enter your choice : 3
```



GIT REPOSITORY LINK :

https://github.com/sakthi-priya-m/DS_LAB