MASTER OF COMPUTER APPLICATIONS

PRACTICAL RECORD WORK

ON

20MCA135 DATA STRUCTURES LAB

Submitted

By

SAMUEL J
(VDA20MCA-2049)

DEPARTMENT OF COMPUTER APPLICATIONS
COLLEGE OF ENGINEERING VADAKARA
(CAPE - GOVT. OF KERALA)

APRIL - 2021

DEPARTMENT OF COMPUTER APPLICATIONS
COLLEGE OF ENGINEERING VADAKARA
(CAPE - GOVT. OF KERALA)

CERTIFICATE

Certified that this is a bonafide record of the practical work on the course 20MCA135 DATA STRUCTURES LAB done by Mr. SAMUEL J (VDA20MCA-2049) First Semester MCA student of Department of Computer Applications at College of Engineering Vatakara in the partial fulfillment for the award of the degree of Master of Computer Applications (MCA) of APJ Abdul Kalam Technological University (KTU)

(Mr. Vijith T K)
FACULTY-IN-CHARGE **HEAD OF THE DEPARTMENT**

CEV
22/04/2021

EXAMINERS:

INDEX			
SL.NO	**PROGRAMS**	**PAGE NO**	**REMARKS**
1	STACK USING ARRAY	2	
2	STACK USING LINKED LIST	6	
3	QUEUE USING ARRAY	11	
4	QUEUE USING LINKED LIST	15	
5	QUEUE USING STACK	20	
6	MERGE 2 SORTED ARRAY	25	
7	LINEAR SEARCH	30	
8	BINARY SEARCH	34	
9	BINARY SEACH TREE	39	
10	CIRCULAR QUEUE	49	
11	DOUBLY LINKED LIST	56	
12	SINGLY LINKED LIST	70	
13	AVL TREE	84	
14	B-TREE	92	
15	BREADTH FIRST SEARCH	100	
16	DEPTH FIRST SEARCH	108	
17	DIJKSTRAS ALGORITHM	114	
18	KRUSKALS ALGORITHM	120	
19	PRIMS ALGORITHM	126	
20	RED-BLACK TREE	132	
21	TOPOLOGICAL SORTING	145	
22			

/* STACK USING ARRAY */

```
#include<stdio.h>
#define SIZE 10

void push(int a[],int *top)
{
   *top=*top+1;
   printf("\nEnter a number:");
   scanf("%d",&a[*top]);
   printf("\n%d is pushed to the stack",a[*top]);

}
int pop(int a[],int *top)
{
   printf("\n%d is poped from the stack",a[*top]);
   *top-=1;
}
void display(int a[],int *top){
   printf("\n the stak elements are:");
   for(int i=*top;i>= 0;i--)
      printf("%d ",a[i]);
}
int main()
{
   int arr[SIZE],ch,e=1;
   int top=-1;

   while(e)
   {
      printf("\nSTACK OPERATIONS");
      printf("\n__________________MENU___________________\n");
      printf("\n\t 1. push\n\t 2. pop\n\t 3. Display\n\t 4. Exit\n");
      printf("\n____________________________________________\n");
      printf("\nEnter your choice:");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1: if(top>=SIZE-1)
                 {
```

```
                printf("\nSTACK overflow\n");
                break;
            }
            push(arr, &top);
            break;
        case 2:if(top<0)
            {
                printf("\nSTACK underflow\n");
                break;
            }
            pop(arr,&top);
            break;
        case 3:display(arr,&top);
        break;
        case 4:e=0;
            printf("\nExiting from the programe");
            break;
        default:printf("\n please enter valid choice");
      }
   }
   return 0;

}
```

Output:

STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

Enter your choice:1

Enter a number:12

12 is pushed to the stack
STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

Enter your choice:1

Enter a number:8

8 is pushed to the stack
STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

Enter your choice:3

the stak elements are:8 12
STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

____

Enter your choice:2

8 is poped from the stack
STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

____

Enter your choice:

*/*STACK USING LINKED LIST*/*

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

struct Node
{
   int data;
   struct Node *next;
} *top = NULL;

void push()
{
   struct Node *temp;
   int val;
   printf("\nEnter a value:");
   scanf("%d", &val);
   temp = (struct Node *)malloc(sizeof(struct Node));
   if (temp)
   {
      temp->data = val;
      if (top == NULL)
         temp->next = NULL;
      else
         temp->next = top;
      top = temp;
      printf("\nOne value inserted into the STACK\n");
   }
   else
   {
      printf("\nSTACK overflow");
   }

}
int pop()
{
   if (top == NULL)
      printf("\nSTACK underflow\n");
   else
   {
      struct Node *temp = top;
      printf("\nDeleted element :%d", temp->data);
      top = temp->next;
```

```
        free(temp);
    }
}
void display()
{
    if (top == NULL)
    {
        printf("\nSTACK is empty\n");
    }
    else
    {
        struct Node *temp = top;
        printf("\n");
        while (temp->next != NULL)
        {
            printf("%d-->", temp->data);
            temp = temp->next;
        }
        printf("%d-->NULL\n", temp->data);
    }
}
void main()
{
    int ch, e = 1;

    while (e)
    {
        printf("\nSTACK OPERATIONS");
        printf("\
n______________________MENU________________________\n");
        printf("\n\t 1. push\n\t 2. pop\n\t 3. Display\n\t 4. Exit\n");
        printf("\
n_______________________________________________________\n");
        printf("\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
```

```
        case 3:
            display();
            break;
        case 4:
            e = 0;
            printf("\nExiting from the programe");
            break;
        default:
            printf("\n please enter valid choice");
        }
    }
}
```

Output:

STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

Enter your choice:1

Enter a value:12

One value inserted into the STACK

STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

Enter your choice:1

Enter a value:24

One value inserted into the STACK

STACK OPERATIONS
____________________MENU______________________

1. push
2. pop
3. Display
4. Exit

Enter your choice:3

24-->12-->NULL

STACK OPERATIONS
______________________MENU________________________

1. push
2. pop
3. Display
4. Exit

____

Enter your choice:2

Deleted element :24
STACK OPERATIONS
______________________MENU________________________

1. push
2. pop
3. Display
4. Exit

____

Enter your choice:

*/*QUEUE USING ARRAY*/*

```
#include <stdio.h>

void enqueue(int a[], int *front, int *rear)
{
   int e;
   printf("\nEnter number:");
   scanf("%d", &e);
   if ((*front == -1) && (*rear == -1))
   {
      *front = 0;
      *rear = 0;
   }
   else
   {
      *rear += 1;
   }
   a[*rear] = e;
   printf("\nThe entered element %d is inserted in to the QUEUE\n",
e);
}

void dequeue(int a[], int *front, int *rear)
{
   if(*front>*rear)
   {
      printf("\nQUEUE underflow\n");
   }
   else
   {
   int e;
   e = a[*front];
   printf("\nThe element %d deleted from QUEUE", e);
   *front += 1;
   }

}
void display(int a[], int *front, int *rear)
{
   if (((*front == -1) && (*rear == -1)) || *front>*rear)
   {
      printf("Queue is empty");
   }
```

```
    else
    {
    int i;
    printf("\nthe QUEUE elements are:");
    for (i = *front; i <= *rear; i++)
        printf("\t%d", a[i]);
    }
}
int main()
{
    int arr[10], front=-1, rear=-1, ch, e = 1;
    while (e)
    {
        printf("\nQUEUE OPERATIONS");
        printf("\
n____________________MENU______________________\n");
        printf("\n\t 1. insert\n\t 2. delete\n\t 3. Display\n\t 4. Exit\n");
        printf("\
n__________________________________________________\n");
        printf("\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            enqueue(arr, &front, &rear);
            break;
        case 2:
            dequeue(arr, &front, &rear);
            break;
        case 3:
            display(arr, &front, &rear);
            break;
        case 4:
            e = 0;
            printf("\nExiting from the programe");
            break;
        default:
            printf("\n please enter valid choice");
        }
    }
    return 0;
}
```

Output:

QUEUE OPERATIONS
______________________MENU________________________

1. insert
2. delete
3. Display
4. Exit

____

Enter your choice:1
Enter number:24

The entered element 24 is inserted in to the QUEUE

QUEUE OPERATIONS
______________________MENU________________________

1. insert
2. delete
3. Display
4. Exit

____

Enter your choice:1
Enter number:36

The entered element 36 is inserted in to the QUEUE

QUEUE OPERATIONS
______________________MENU________________________

1. insert
2. delete
3. Display
4. Exit

____

Enter your choice:1
Enter number:48

The entered element 48 is inserted in to the QUEUE

QUEUE OPERATIONS
____________________MENU______________________

1. insert
2. delete
3. Display
4. Exit

____

Enter your choice:3

the QUEUE elements are: 24 36 48
QUEUE OPERATIONS
____________________MENU______________________

1. insert
2. delete
3. Display
4. Exit

____

Enter your choice:2

The element 24 deleted from QUEUE
QUEUE OPERATIONS
____________________MENU______________________

1. insert
2. delete
3. Display
4. Exit

____

Enter your choice:

/*QUEUE USING LINKED LIST*/

```
#include <stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;

void insert()
{
   struct node *temp;
   int val;
   temp = (struct node*)malloc(sizeof(struct node));
   if(temp == NULL)
   {
      printf("\n Queue Overflow\n");
      return;
   }
   else
   {
      printf("\n Enter the value:");
      scanf("%d",&val);
      temp -> data = val;
      temp -> next = NULL;
      if(front == NULL)
         front = rear = temp;
      else
      {
         rear -> next = temp;
         rear = temp;
      }
      printf("\n One value is inserted into the queue\n");
   }
}
void delete()
{
   struct node *temp;
   if(front == NULL)
   {
```

```
        printf("\n Underflow\n");
        return;
    }
    else
    {
        temp = front;
        front = front -> next;
        printf("\n %d is deleted from the queue\n", temp -> data);
        free(temp);

    }
}
void display()
{
    struct node *temp;
    temp = front;
    if(front == NULL)
    {
        printf("\n Empty Queue\n");
        return;
    }
    else
    {
        printf("\n Queue elements are\n");
        while(temp != NULL)
        {
            printf("%d ", temp -> data);
            temp = temp -> next;
        }
    }
}
int main()
{
    int ch, e=1;

    while(e)
    {
        printf("\n QUEUE USING LINKED LIST");
          printf("\n______________MENU______________");
          printf("\n 1.INSERT \n 2.DELETE \n 3.DISPLAY \n 4.EXIT");
          printf("\n________________________________");
          printf("\n Enter your choice:");
          scanf("%d",&ch);
```

```
        switch(ch)
        {
            case 1:
                insert();
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                e=0;
                printf("\n exiting...");
                break;
            default: printf("\n please enter valid choice\n");
            break;
        }
    }

    return 0;
}
```

Output:

```
QUEUE USING LINKED LIST
______________MENU______________
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
________________________________
Enter your choice:1

Enter the value:12

One value is inserted into the queue

QUEUE USING LINKED LIST
______________MENU______________
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
________________________________
Enter your choice:1

Enter the value:24

One value is inserted into the queue

QUEUE USING LINKED LIST
______________MENU______________
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
________________________________
Enter your choice:1

Enter the value:36

One value is inserted into the queue
```

```
 QUEUE USING LINKED LIST
______________MENU______________
 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
________________________________
 Enter your choice:3

 Queue elements are
12 24 36
 QUEUE USING LINKED LIST
______________MENU______________
 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
________________________________
 Enter your choice:2

 12 is deleted from the queue

 QUEUE USING LINKED LIST
______________MENU______________
 1.INSERT
 2.DELETE
 3.DISPLAY
 4.EXIT
________________________________
 Enter your choice:
```

/*QUEUE USING STACK*/

```
#include <stdio.h>

void push(int stack[],int *top, int ele)
{
   *top = *top + 1;
   stack[*top] = ele;
}

int pop(int stack[], int *top)
{
   int ele;
   ele = stack[*top];
   *top = *top - 1;
   return(ele);
}

void enqueue(int stack1[], int *top1)
{
   int i, ele;
   printf("Enter the element:");
   scanf("%d", &ele);
   push(stack1, top1, ele);
}

void dequeue(int stack1[], int *top1, int stack2[], int *top2)
{
   int i;
   int count = *top1;
   for (i = 0;i <= count;i++)
   {
      push(stack2,top2,pop(stack1,top1));
   }
   printf("\nThe element %d is deleted from queue\n",
pop(stack2,top2));

   count = *top2;
   for (i = 0;i <= count;i++)
   {
```

```
        push(stack1,top1,pop(stack2,top2));
    }
}

/* Display the elements in the stack1*/

void display(int stack[], int *top)
{
    int i;

    for (i = 0;i <= *top; i++)
    {
        printf(" %d ", stack[i]);
    }
}

void main()
{
   int stack1[20], stack2[20];
   int top1 = -1, top2 = -1;
   int ch;
   int e = 1;

   printf("\nQUEUE using STACKS\n");
   while( e )
   {
        printf("\
n______________________MENU________________________\n");
        printf("\n\t1. Enqueue\n\t2. Dequeue\n\t3. Display\n\t4. Exit\
n");
        printf("\
n________________________________________________________\
n");
        printf("Enter your choice:");
        scanf("%d", &ch);
        switch( ch )
        {
            case 1: enqueue(stack1, &top1);
                    break;
            case 2: dequeue(stack1,&top1,stack2,&top2);
```

```
            break;
        case 3: display(stack1,&top1);
            break;
        case 4: e = 0;
            printf("\nExiting from the program\n");
            break;
        default: printf("\nPlease enter valid choice\n");
      }
    }
}
```

Output:

QUEUE using STACKS

_____________________MENU_______________________

1. Enqueue
2. Dequeue
3. Display
4. Exit

____

Enter your choice:1
Enter the element:12

_____________________MENU_______________________

1. Enqueue
2. Dequeue
3. Display
4. Exit

____

Enter your choice:1
Enter the element:8

_____________________MENU_______________________

1. Enqueue
2. Dequeue
3. Display
4. Exit

____

Enter your choice:1
Enter the element:24

_____________________MENU_______________________

1. Enqueue
2. Dequeue

3. Display
4. Exit

Enter your choice:3
12 8 24
_____________________MENU_______________________

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice:2

The element 12 is deleted from queue

_____________________MENU_______________________

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice:

/*MERGE 2 SORTED ARRAY*/

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

void read(int a[],int *limit)
{
   int i;
   printf("\nEnter the values in sorted order:");
   for ( i = 0; i < *limit; i++)
   {
      scanf("%d", &a[i]);
   }
}
void merge(int arr1[], int arr2[], int *s1, int *s2, int marr[])
{
   int i=0, j=0,k=0;
   while (k < *s1 + *s2)
   {
      if (j < *s2 && i < *s1)
      {
         if (arr1[i] <= arr2[j])
         {
            marr[k] = arr1[i];
            k++;
            i++;
         }
         else
         {
            marr[k] = arr2[j];
            k++;
            j++;
         }
      }
      else
      {
         if (j >= *s2 && i < *s1)
         {
            marr[k] = arr1[i];
            i++;
            k++;
```

```
        }
        else if (i >= *s1 && j < *s2)
        {
            marr[k] = arr2[j];
            j++;
            k++;
        }
    }
  }
  printf("\nmerged successfully\n");
}

void display(int arr1[], int arr2[], int *s1, int *s2, int marr[])
{
  int i;
  printf("\nThe elements in first array:\n");
  for(i=0;i<*s1;i++)
  {
    printf(" %d",arr1[i]);
  }
  printf("\nThe elements in second array:\n");
  for(i=0;i<*s2;i++)
  {
    printf(" %d",arr2[i]);
  }
  printf("\nThe array elements after merging:\n");
  for(i=0;i<*s1+*s2;i++)
  {
    printf(" %d",marr[i]);
  }
}

int main()
{
  int arr1[50], arr2[50], marr[100], s1,s2,e=0,ch;
  printf("\nMERGE TWO SORTED ARRAYS\n");
  do
  {
    printf("\n________________MENU________________\n");
```

```
        printf("\n\t1.Read sorted arrays\n\t2.Merge array\n\
t3.Display\n\t4.Exit\n");
        printf("\n___________________________________\n");
        printf("\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
           case 1:printf("\nEnter the size of the  array1:");
                  scanf("%d", &s1);
                  read(arr1,&s1);
                  printf("\nEnter the size of the  array2:");
                  scanf("%d", &s2);
                  read(arr2,&s2);
                  break;
           case 2:
                  merge(arr1,arr2,&s1,&s2,marr);
                  break;
           case 3:display(arr1,arr2,&s1,&s2,marr);
                  break;
           case 4:printf("Exiting from the programme");
                  break;
           default:
                  printf("Enter the valid option:");
        }
    }while(ch!=4);


    return 0;
}
```

Output:

MERGE TWO SORTED ARRAYS

________________MENU________________

1.Read sorted arrays
2.Merge array
3.Display
4.Exit

____

Enter your choice:1

Enter the size of the array1:4

Enter the values in sorted order:10 12 18 25

Enter the size of the array2:4

Enter the values in sorted order:9 17 20 24

________________MENU________________

1.Read sorted arrays
2.Merge array
3.Display
4.Exit

____

Enter your choice:2

merged successfully

________________MENU________________

1.Read sorted arrays

2.Merge array
3.Display
4.Exit

____

Enter your choice:3

The elements in first array:
 10 12 18 25
The elements in second array:
 9 17 20 24
The array elements after merging:
 9 10 12 17 18 20 24 25
________________MENU________________

1.Read sorted arrays
2.Merge array
3.Display
4.Exit

____

Enter your choice:

/*LINEAR SEARCH*/

```
#include<stdio.h>
#define SIZE 10

void read(int a[],int *n)
{
   printf("Enter the number of elements:");
   scanf("%d",n);
   printf("\nEnter the elements:");
   for(int i=0;i<*n;i++)
   {
      scanf("%d",&a[i]);
   }

}
void search(int a[],int *n)
{
   int e,i;
   printf("\nEnter the element to be searched:");
   scanf("%d",&e);
   for(i=0;i<*n;i++)
   {
      if(a[i] == e)
      {
         printf("\n%d is located at position %d\n",e,i+1);
         return;
      }
   }
   printf("\nEntered element is not in the data\n");
}
void display(int a[],int *n){
   printf("\n the  elements are:");
   for(int i=0;i<*n;i++)
      printf("%d ",a[i]);
}
int main()
{
   int arr[SIZE],ch,e=1;
   int n=-1;
```

```
    while(e)
    {
       printf("\n\
n______________________MENU________________________\n");
       printf("\n\t 1. read\n\t 2. search\n\t 3. Display\n\t 4. Exit\n");
       printf("\
n_______________________________________________________\
n");
       printf("\nEnter your choice:");
       scanf("%d",&ch);
       switch(ch)
       {
          case 1:read(arr, &n);
                break;
          case 2:search(arr,&n);
                break;
          case 3:display(arr,&n);
          break;
          case 4:e=0;
             printf("\nExiting from the programe\n");
             break;
          default:printf("\n please enter valid choice\n");
       }
    }
    return 0;

}
```

Output:

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

____

Enter your choice:1
Enter the number of elements:4

Enter the elements:10 21 38 44

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

____

Enter your choice:2

Enter the element to be searched:38

38 is located at position 3

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

Enter your choice:3

the elements are:10 21 38 44

______________________MENU________________________

1. read
2. search
3. Display
4. Exit

Enter your choice:

/*BINARY SEARCH*/

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 10

void read(int a[],int *top)
{
   int n;
   printf("\nEnter the number of elements:");
   scanf("%d",&n);
   printf("\nEnter the array elements in sorted order:");
   for (int i=0;i<n;i++)
   {
      *top+=1;
      scanf("%d",&a[*top]);
   }
}

void search(int a[],int *top)
{
   int ele,first,last,mid;
   printf("\nEnter the element to be searched:");
   scanf("%d",&ele);
   first=0;
   mid=*top/2;
   last=*top;
   while(first<=last)
   {
      if(a[mid] == ele)
      {
         printf("\nThe location of entered element is %d",mid+1);
         return;
      }
      else if (a[mid]>ele)
      {
         last=mid-1;
      }
      else
      {
```

```
            first=mid+1;
        }
        mid=(first+last)/2;

    }
    printf("\n Entered element is not in the list");
}

void display(int a[],int *n){
    printf("\n the  elements are:");
    for(int i=0;i<=*n;i++)
        printf("%d ",a[i]);
}

int main()
{
    int arr[SIZE],ch,e=1;
    int n=-1;
    while(e)
    {
        printf("\n\
n____________________MENU______________________\n");
        printf("\n\t 1. read\n\t 2. search\n\t 3. Display\n\t 4. Exit\n");
        printf("\
n__________________________________________________\
n");
        printf("\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:read(arr, &n);
                    break;
            case 2:search(arr,&n);
                    break;
            case 3:display(arr,&n);
            break;
            case 4:e=0;
                printf("\nExiting from the programe\n");
                break;
            default:printf("\n please enter valid choice\n");
```

```
        }
    }
    return 0;

}
```

Output:

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

____

Enter your choice:1

Enter the number of elements:5

Enter the array elements in sorted order:10 14 19 21 27

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

____

Enter your choice:2

Enter the element to be searched:19

The location of entered element is 3

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

Enter your choice:2

Enter the element to be searched:18

Entered element is not in the list

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

Enter your choice:3

the elements are:10 14 19 21 27

_____________________MENU_______________________

1. read
2. search
3. Display
4. Exit

Enter your choice:

/*BINARY SEARCH TREE*/

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

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

node *createNode(int val)
{
   node *newnode;
   newnode = (node *)malloc(sizeof(node));
   newnode->data = val;
   newnode->left = newnode->right = NULL;
   return newnode;
}

node *findNode(node *temp, node *tree)
{
   if ((temp->data > tree->data) && (tree->right == NULL))
   {
      tree->right = temp;
   }
   else if ((temp->data > tree->data) && (tree->right != NULL))
   {
      tree->right = findNode(temp, tree->right);
   }
   else if ((temp->data < tree->data) && (tree->left == NULL))
   {
      tree->left = temp;
   }
   else if ((temp->data < tree->data) && (tree->left != NULL))
   {
      tree->left = findNode(temp, tree->left);
   }
```

```
}

node *insert(int val, node *tree)
{
   node *temp = createNode(val);
   if (tree == NULL)
   {
      tree = temp;
   }
   else
   {
      findNode(temp, tree);
   }
}

node *inorderTraversal(node *tree)
{
   if (tree == NULL)
   {
      return NULL;
   }
   if (tree->left != NULL)
      inorderTraversal(tree->left);
   printf("\t%d\t", tree->data);
   if (tree->right != NULL)
      inorderTraversal(tree->right);
}

node *preorderTraversal(node *tree)
{
   if (tree == NULL)
   {
      return NULL;
   }
   printf("\t%d\t", tree->data);
   preorderTraversal(tree->left);
   preorderTraversal(tree->right);
}

node *postorderTraversal(node *tree)
```

```
{
    if (tree == NULL)
    {
        return NULL;
    }
    postorderTraversal(tree->left);
    postorderTraversal(tree->right);
    printf("\t%d\t", tree->data);
}

node *minValueNode(node *tree)
{
    node *current = tree;
    while (current && current->left != NULL)
    {
        current = current->left;
    }
    return current;
}

node *deleteNode(int val, node *tree)
{
    if (tree == NULL)
    {
        printf("\nNot such value in the bst");
    }
    if ((val < tree->data))
    {
        tree->left = deleteNode(val, tree->left);
    }
    else if (val > tree->data)
    {
        tree->right = deleteNode(val, tree->right);
    }
    else
    {
        if ((tree->left == NULL))
        {
            node *temp = tree->right;
            tree == NULL;
```

```
            return temp;
        }
        else if ((tree->right == NULL))
        {
            node *temp = tree->left;
            tree == NULL;
            return temp;
        }
        node *temp = minValueNode(tree->right);
        tree->data = temp->data;
        tree->right = deleteNode(temp->data, tree->right);
    }
    return NULL;
}

node *searchNode(int val, node *tree)
{
    if (tree == NULL)
    {
        printf("\nSearch is unsuccessfull!!!");
    }
    if ((val == tree->data))
    {
        printf("\nSearch successfull");
    }
    else if (tree->data < val)
    {
        searchNode(val, tree->right);
    }
    else
    {
        searchNode(val, tree->left);
    }
}

int main()
{
    int ch, e = 1, op, val;
    node *root = NULL;
    printf("\n BST OPERATION");
```

```
    while (e)
    {
       printf("\n_______________MENU_______________");
       printf("\n 1.INSERT \n 2.DELETE \n 3.SEARCH \n 4.IN-
ORDER TRAVERSAL \n 5.PRE-ORDER TRAVERSAL\n
6.POST-ORDER TRAVERSAL\n 7.EXIT");
       printf("\n____________________________________\n");
       printf("\n Enter your choice:");
       scanf("%d", &ch);
       switch (ch)
       {
       case 1:
          printf("\nEnter the value to be inserted:");
          scanf("%d", &val);
          root = insert(val, root);
          break;
       case 2:
          printf("\nEnter the value to be deleted:");
          scanf("%d", &val);
          deleteNode(val, root);
          printf("one value  is deleted");
          break;
       case 3:
          printf("\nEnter the value to be searched:");
          scanf("%d", &val);
          searchNode(val, root);
          break;
       case 4:
          printf("\nIn-order traversal of elements");
          inorderTraversal(root);
          break;
       case 5:
          printf("\npre-order traversal of elements");
          preorderTraversal(root);
          break;
       case 6:
          printf("\npost-order traversal of elements");
          postorderTraversal(root);
          break;
       case 7:
```

```
            e = 0;
            printf("\n exiting");
            break;
        default:
            printf("\n please enter valid choice\n");
            break;
        }
    }
    return 0;
}
```

Output:

BST OPERATION

______________MENU______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:1

Enter the value to be inserted:12

______________MENU______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:1

Enter the value to be inserted:8

______________MENU______________

1.INSERT
2.DELETE
3.SEARCH

4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:1

Enter the value to be inserted:24

_______________MENU_______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:3

Enter the value to be searched:24

Search successfull

_______________MENU_______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:4

In-order traversal of elements 8		12		24

______________MENU______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:5

pre-order traversal of elements 12		8		24

______________MENU______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:6

post-order traversal of elements	8		24		12

______________MENU______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL

6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:2

Enter the value to be deleted:24
one value is deleted

_______________MENU_______________

1.INSERT
2.DELETE
3.SEARCH
4.IN-ORDER TRAVERSAL
5.PRE-ORDER TRAVERSAL
6.POST-ORDER TRAVERSAL
7.EXIT
____

Enter your choice:

/*CIRCULAR QUEUE */

```
#include <stdio.h>
#define SIZE 5

void enqueue(int a[], int *front, int *rear)
{
   int e;
   printf("\nEnter number:");
   scanf("%d", &e);
   if ((*rear + 1) % SIZE == *front)
   {
      printf("\nQUEUE overflow");
      return;
   }
   else if (*front > 0 && *rear == SIZE - 1)
   {
      *rear = 0;
   }
   else if ((*front == -1) && (*rear == -1))
   {
      *front = 0;
      *rear = 0;
   }
   else
   {
      printf("then");
      *rear += 1;
   }
   a[*rear] = e;
   printf("\nThe entered element %d is inserted in to the QUEUE\
n", e);
}

void dequeue(int a[], int *front, int *rear)
{
   if (*front == -1)
   {
      printf("\nQUEUE underflow\n");
```

```
    }
    else if (*front == SIZE - 1)
    {
        *front = 0;
    }
    else
    {
        int e;
        e = a[*front];
        printf("\nThe element %d deleted from QUEUE", e);
        *front += 1;
    }
}

void display(int a[], int *front, int *rear)
{
    if (((*front == -1) && (*rear == -1)))
    {
        printf("Queue is empty");
    }
    else
    {
        int i;
        printf("\nthe QUEUE elements are:");
        if(*front>*rear)
        {
            for (i = *front; i<=(*rear+SIZE) ; i++)
                printf("\t%d", a[i%SIZE]);
        }
        else{
            for (i = *front; i<=(*rear) ; i++)
                printf("\t%d", a[i]);
        }

    }
}

void search(int a[], int *front, int *rear,int ele)
{
    if (((*front == -1) && (*rear == -1)))
```

```
    {
       printf("Queue is empty");
    }
    else
    {
       if(*front>*rear)
       {
       for (int i=*front;i<=(*rear+SIZE);i++)
       {
          if(a[i%SIZE]==ele)
          {
             printf("Item found!!!");
             return;
          }

       }
       printf("Item not found!!!");
       }
       else
       {
          for (int i=*front;i<=(*rear);i++)
       {
          if(a[i]==ele)
          {
             printf("Item found!!!");
             return;
          }

       }
       printf("Item not found!!!");
       }

    }
}

    int main()
{
    int arr[SIZE], front = -1, rear = -1, ch, e = 1, val;
    while (e)
    {
```

```
        printf("\nCIRCULAR QUEUE OPERATIONS");
        printf("\
n____________________MENU______________________\n");
        printf("\n\t 1. insert\n\t 2. delete\n\t 3. Display\n\t 4. Search\
n\t 5. Exit\n");
        printf("\
n___________________________________________________\
n");
        printf("\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            enqueue(arr, &front, &rear);
            break;
        case 2:
            dequeue(arr, &front, &rear);
            break;
        case 3:
            display(arr, &front, &rear);
            break;
        case 4:
            printf("\nEnter the data to be searched:");
            scanf("%d", &val);
            search(arr, &front, &rear, val);
            break;
        case 5:
            e = 0;
            printf("\nExiting from the programe");
            break;
        default:
            printf("\n please enter valid choice");
        }
    }
    return 0;
}
```

Output:

CIRCULAR QUEUE OPERATIONS
____________________MENU______________________

1. insert
2. delete
3. Display
4. Search
5. Exit

Enter your choice:1

Enter number:12

The entered element 12 is inserted in to the QUEUE

CIRCULAR QUEUE OPERATIONS
____________________MENU______________________

1. insert
2. delete
3. Display
4. Search
5. Exit

Enter your choice:1

Enter number:24
then
The entered element 24 is inserted in to the QUEUE

CIRCULAR QUEUE OPERATIONS
____________________MENU______________________

1. insert
2. delete
3. Display
4. Search
5. Exit

Enter your choice:1

Enter number:36
then
The entered element 36 is inserted in to the QUEUE

CIRCULAR QUEUE OPERATIONS
______________________MENU________________________

1. insert
2. delete
3. Display
4. Search
5. Exit

Enter your choice:3

the QUEUE elements are: 12 24 36

CIRCULAR QUEUE OPERATIONS
______________________MENU________________________

1. insert
2. delete
3. Display
4. Search
5. Exit

Enter your choice:2

The element 12 deleted from QUEUE

CIRCULAR QUEUE OPERATIONS
______________________MENU________________________

1. insert
2. delete
3. Display
4. Search
5. Exit

Enter your choice:4

Enter the data to be searched:36
Item found!!!

CIRCULAR QUEUE OPERATIONS
______________________MENU________________________

1. insert
2. delete
3. Display
4. Search
5. Exit

Enter your choice:

/*DOUBLY LINKED LIST*/

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

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

node *head;

// inserting value to the doubly linked list
void insert(int *op)
{
   // checking if the entered option is invalid
   if (*op > 3)
   {
      printf("\nEnter a valid option!!!\n\n");
      return;
   }
   int pos, i, val;
   node *temp = (node *)malloc(sizeof(node *));
   // checking overflow condition
   if (temp == NULL)
   {
      printf("\nList Overflow\n");
   }
   else
   {
      printf("\nEnter the value to be inserted:");
      scanf("%d", &val);
      temp->data = val;
      // inserting value in the front of the doubly linked list
      if (*op == 1)
      {
```

```
        if (head == NULL)
        {
            temp->next = NULL;
            temp->prev = NULL;
            head = temp;
        }
        else
        {
            temp->next = head;
            temp->prev = NULL;
            head->prev = temp;
            head = temp;
        }
        printf("\none value entered at front of Doubly linked list\
n");
    }
    // inserting value in the last position of the doubly linked list
    else if (*op == 2)
    {

        if (head == NULL)
        {
            temp->next = NULL;
            temp->prev = NULL;
            head = temp;
        }
        else
        {
            node *ptr = head;
            while (ptr->next != NULL)
            {
                ptr = ptr->next;
            }
            ptr->next = temp;
            temp->next = NULL;
            temp->prev = ptr;
        }
        printf("\none value entered at last of Doubly linked list\
n");
```

```
        }
        // inserting value in the specified position of the doubly
linked list
        else if (*op == 3)
        {
            printf("\nEnter the position where you want to insert the
data:");
            scanf("%d", &pos);
            if (pos == 1 && head == NULL)
            {
                temp->next = NULL;
                temp->prev = NULL;
                head = temp;
            }
            else
            {
                node *ptr = head;
                i = 1;
                while (i < pos - 1 && ptr != NULL)
                {
                    ptr = ptr->next;
                    i++;
                }
                if (ptr == NULL)
                {
                    printf("\nNumber of values in the linked list is
smaller than the value you entered\n");
                }
                else
                {
                    temp->next = ptr->next;
                    ptr->next->prev = temp;
                    ptr->next = temp;
                    temp->prev = ptr;
                    printf("\nValue entered at position %d ", pos);
                }
            }
        }
    }
}
```

```
// function for deleting elements from the doubly linked list
void delete (int *op)
{
   // checking whether the entered choice is valid or not
   if (*op > 3)
   {
      printf("\nplease Enter a valid option!!!\n");
      return;
   }
   int pos, i;
   node *temp = head;
   // checking underflow condition
   if (temp == NULL)
   {
      printf("\nUnderflow!!!\n");
      return;
   }
   else
   {

      // deleting an element from first position of the doubly
linked list
      if (*op == 1)
      {
         if (temp->next == NULL)
         {
            head = NULL;
         }
         else
         {
            head = temp->next;
            head->next = temp->next->next;
         }
         printf("\none value deleted from front of Doubly linked
list\n");
      }

      // deleting an element from last position of the doubly linked
list
```

```
        else if (*op == 2)
        {
           node *ptr = head;
           while (ptr->next != NULL)
           {
              ptr = ptr->next;
           }
           ptr->prev->next = NULL;
           printf("\none value deleted from the last position of
Doubly linked list\n");
        }

        // deleting an element from specified position of the doubly
linked list
        else if (*op == 3)
        {
           printf("\nEnter the position where you want to delete the
data:");
           scanf("%d", &pos);
           node *ptr = head;
           if (pos == 1 && ptr->next == NULL)
           {
              head = NULL;
              printf("\nValue deleted in position %d ", pos);
           }
           else
           {
              i = 1;
              while (i < pos - 1 && ptr != NULL)
              {
                 printf("%d\n", i);
                 ptr = ptr->next;
                 i++;
              }
              if (ptr->next == NULL)
              {
                 printf("\nNumber of values in the linked list is
smaller than the value you entered\n");
              }
              else
```

```
            {
               ptr->next = ptr->next->next;
               printf("\nValue deleted in position %d\n ", pos);
            }
         }
      }
   }
}

// function to display the elements in the doubly linked list
void display()
{
   printf("display function\n");
   if (head == NULL)
   {
      printf("\nlist is empty\n");
   }
   else
   {
      node *temp = head;
      while (temp->next != NULL)
      {
         printf("%d-->", temp->data);
         temp = temp->next;
      }
      printf("%d-->NULL", temp->data);
   }
}

void search(int ele)
{
   if(head==NULL)
   {
      printf("\nList is empty!!!");
      return;
   }
   node *temp= head;
   while(temp!=NULL)
   {
      if(temp->data==ele)
```

```
        {
            printf("%d FOUND",ele);
            return;
        }
        temp=temp->next;
    }
    printf("%d NOT FOUND!!!",ele);
}

void sort()
{
    struct node *current,*index;
    for(current=head;current->next!=NULL;current=current-
     >next)
    {
        for(index=current->next;index!=NULL;index=index->next)
        {
            if(current->data>index->data)
            {
                int temp=current->data;
                current->data=index->data;
                index->data=temp;
            }
        }
    }
    printf("\nsorted the list successfully\n");
}

int main()
{
    int ch, e = 1, op,data;
    while (e)
    {
        printf("\n DOUBLY LINKED LIST");
        printf("\n_______________MENU_______________");
        printf("\n 1.INSERT \n 2.DELETE \n 3.DISPLAY \n
4.SEARCH \n 5.SORT\n 6.EXIT");
        printf("\n____________________________________\n");
        printf("\n Enter your choice:");
        scanf("%d", &ch);
```

```
        switch (ch)
        {
        case 1:
            printf("\n___________Insertion option___________\n");
            printf("\n1.Front\n2.Last\n3.In between\n");
            printf("____________________________________\n");
            printf("\nchose your option:");
            scanf("%d", &op);
            insert(&op);
            break;
        case 2:
            printf("\n___________Deletion option___________\n");
            printf("\n1.Front\n2.Last\n3.In between\n");
            printf("____________________________________\n");
            printf("\nchose your option:");
            scanf("%d", &op);
            delete (&op);
            break;
        case 3:
            display();
            break;
        case 4:printf("Enter the data you want to search:");
        scanf("%d",&data);
            search(data);
            break;
        case 5:
            sort();
            break;
        case 6:
            e = 0;
            printf("\n exiting........");
            break;
        default:
            printf("\n please enter valid choice\n");
            break;
        }
    }
    printf("\n\n\n\t\t\t----------successfully exited----------\n\n");
    return 0;
}
```

Output:

DOUBLY LINKED LIST

_______________MENU_______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:1

___________Insertion option___________

1.Front
2.Last
3.In between
____

chose your option:1

Enter the value to be inserted:24

one value entered at front of Doubly linked list

DOUBLY LINKED LIST

_______________MENU_______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:1

__________Insertion option__________

1.Front
2.Last
3.In between
____

chose your option:2

Enter the value to be inserted:48

one value entered at last of Doubly linked list

DOUBLY LINKED LIST

______________MENU______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:1

__________Insertion option__________

1.Front
2.Last
3.In between
____

chose your option:3

Enter the value to be inserted:10

Enter the position where you want to insert the data:2

Value entered at position 2
DOUBLY LINKED LIST

_______________MENU_______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:3
display function
24-->10-->48-->NULL

DOUBLY LINKED LIST

_______________MENU_______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:4
Enter the data you want to search:10
10 FOUND

DOUBLY LINKED LIST

_______________MENU_______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:5

sorted the list successfully

DOUBLY LINKED LIST

_________________MENU_________________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:3
display function
10-->24-->48-->NULL

DOUBLY LINKED LIST

_________________MENU_________________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:2

___________Deletion option___________

1.Front
2.Last
3.In between
____

chose your option:3

Enter the position where you want to delete the data:3
1

Value deleted in position 3

DOUBLY LINKED LIST

_______________MENU_______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.SORT
6.EXIT
____

Enter your choice:3
display function
10-->24-->NULL

DOUBLY LINKED LIST

_______________MENU_______________

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH

5.SORT
6.EXIT

Enter your choice:

/*SINGLY LINKED LIST*/

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

/* Model of list structure creation */
struct link_list
{
    int data;
    struct link_list *next;
};

typedef struct link_list node;

/* Function for create a list */
void Create(node *p)
{
    int i, num;
    node *temp;
    char ch;

    if( p -> data != -999)
    {
       printf("\n The list already exist.\n");
       printf(" Do you want to continue? (Y for yes, N for no)\n");
       scanf(" %c ", &ch);
       if( ch == 'N'|| ch == 'n')
          return;
        else
        {
            free(p);
            p->data =-999;
            p->next = NULL;
            Create(p);
        }

    }
    printf("\n Enter number of nodes:");
    scanf("%d", &num);
    printf( "\n Enter the elements: " );
```

```
    for( i = 0; i < num; i++ )
    {
       temp = ( node * ) malloc( sizeof ( node ) );
       if( temp )
       {
           scanf( "%d", &temp -> data );
           temp -> next = NULL;

           if( p -> data != -999 )
           {
               while( p -> next )
                   p = p -> next;
               p -> next = temp;
           }
           else
               p -> data = temp -> data;
       }
       else
           printf( "\n Memory overflow\n" );
    }
}

/* Function for add a node to the list */
node *Insert( node *p )
{
    node *q,*temp;
    int pos, count = 0;
    q = p;

    if( p -> data == -999)
    {
       printf("\n The list is empty. Please create a list first\n");
       return p;
    }


    temp = ( node * ) malloc( sizeof ( node ) );
    if( temp )
    {
       while( q -> next )
```

```
{
    count++;
    q = q -> next;
}
count++;
printf("\n Enter the position to insert between <1 and
 %d>:", count + 1);
scanf("%d", &pos);
if( ( pos < 0 ) || ( pos > (count + 2 ) ) )
{
   printf("\n It is not possible to insert the element at the
      given position. Position beyond the limit\n");
   return p;
}

 printf( "\n Enter the element: " );
 scanf( "%d", &temp -> data );
 temp -> next = NULL;

 if( pos == 1 )
 {
    printf("\n Inserting the element at the first position.");
    temp -> next = p;
    p = temp;
    return p;
 }
 else
 {
    q = p;
    count = 1;
    while( q -> next )
    {
       count++;
       if( pos == count )
       {
          printf("\n Inserting the element in between nodes\
         n");
          temp -> next = q -> next;
          q -> next = temp;
          return p;
```

```
            }
            q = q -> next;
        }
        printf("\n Inserting the element as last node\n");
        q -> next = temp;
        return p;

    }
  }
  else
      printf( "\n Memory overflow\n" );
}

/* Function for list all list elements */
void Display( node *p )
{
  if( p -> data != -999 )
  {
      printf( "\n The list elements are: " );
      while( p )
      {
          printf( " %d ", p -> data );
          p = p -> next;
      }
      printf( "\n" );
  }
  else
      printf("\n List is empty \n");
}


/* Function for delete an element from the list */
node * Delete( node *start )
{
  int ele;
  node *p, *q;

  if( start -> data != -999 )
  {
      printf( "\n Enter the element to be dalete:" );
```

```
    scanf( "%d", &ele );

    if( start -> data == ele )
    {
        p = start;
        printf( " \nThe element %d is deleted from the list\n ",
        p -> data );
        if( start -> next == NULL )
        {
            q = (node * ) malloc( sizeof( node ) );
            q -> data = -999;
            q -> next = NULL;
            free( p );
            return q;
        }
        start = start -> next;
        free( p );
        return start;
    }
    else
    {
        p = start;
        while( p -> next )
        {
            q = p -> next;
            if( q -> data == ele )
            {
                p -> next = q -> next;
                printf( " \nThe element %d is deleted from the
                list\n ", q -> data );
                free( q );
                return start;
            }
            p = p -> next;
        }
        printf( " \nThe element %d is not present in the list\n ",
            ele );
        return start;
    }
}
```

```
    else
        printf( "\n Memory underflow\n" );

    return start;

}

/* Function for reverse elements of the list */
node * Reverse( node *start )
{
    node *q, *r, *s;
    q = start;
    r = NULL;
    while( q )
    {
        s = r;
        r = q;
        q = q -> next;
        r -> next = s;
    }
    return r;


}

/* Function for search an element from the list */
void Search( node *p )
{
   int ele, count = 0;
    if( p -> data != -999 )
    {
        printf( "\n Enter the element to search: " );
        scanf("%d", &ele);
        while( p )
        {
           count++;
           if( p-> data == ele )
           {
              printf("\n The element %d is present in the list at %d
                 position", ele,count);
```

```
                return;
            }
            p = p -> next;
        }
        printf("\n The element is not present in the list\n" );
    }
    else
        printf("\n List is empty \n");
}
```

```
/* Function for sort the list */
node *Sort(node *start)
{
    node *fnode= start ;
    node *pre1= start;
    node *pre,*t1,*temp;
    if( !start )
        printf("\n The list is empty.");
    else
    {
        node *pre1= start,*pre,*t1,*temp;
        while( start -> next )
        {
            pre = start;
            temp = start -> next;
            while(temp)
            {
                if( start -> data > temp -> data )
                {
                    t1 = temp -> next;
                    temp -> next = start -> next;
                    start -> next = t1;
                    if(pre != start )
                        pre -> next = start;
                    else
                        temp -> next = start ;
                    if( start == fnode)
                        fnode = temp;
                    else
```

```
                    pre1 -> next = temp;
                t1 = start;
                start = temp;
                temp = t1;
            }
        pre = temp;
        temp = temp -> next;
        }
    pre1 = start;
    start = start -> next;
    }
  }
  start = fnode;
  return start;
}

/* Main function */
int main()
{

  node *start = ( node * ) malloc( sizeof( node ) );
  start -> data = -999;
  start -> next = NULL;

  int e = 1, ch;

  while( e )
  {
    printf("\n________________MENU________________");
    printf( "\n\t1. Create\n\t2. Insert\n\t3. Display\n\t4. Delete\
    n\t5. Reverse\n\t6. Search\n\t7. Sort\n\t8. Exit\n" );
    printf("\n_____________________________________\n");
    printf( "\n Enter your choice:" );
    scanf( "%d", &ch );

    switch( ch )
    {
      case 1: Create( start );
          break;
      case 2 : start = Insert( start );
```

```
                break;
            case 3 : Display( start );
                break;
            case 4 : start = Delete( start );
                break;
            case 5 : start = Reverse( start );
                break;
         case 6: Search(start);
             break;
         case 7: start = Sort( start );
             break;
            case 8 : e = 0;
                break;
            default: printf( "\n Invalid choice \n" );
         }

    }
    return 0;
}
```

Output:

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:1

Enter number of nodes:3

Enter the elements: 12 24 8

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:2

Enter the position to insert between <1 and 4>:2

Enter the element: 48

Inserting the element in between nodes

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:3

The list elements are: 12 48 24 8

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:5

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:3

The list elements are: 8 24 48 12

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:6

Enter the element to search: 48

The element 48 is present in the list at 3 position

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:7

_________________MENU_________________

1. Create

2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:3

The list elements are: 8 12 24 48

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

____

Enter your choice:4

Enter the element to be dalete:24

The element 24 is deleted from the list

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort

8. Exit

Enter your choice:3

The list elements are: 8 12 48

_________________MENU_________________

1. Create
2. Insert
3. Display
4. Delete
5. Reverse
6. Search
7. Sort
8. Exit

Enter your choice:

/*AVL TREE*/

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *left;
   struct Node *right;
   int height;
};
typedef struct Node Node;
int max(int a, int b)
{
   return (a > b) ? a : b;
}
int height(Node *N)
{
   if (N == NULL)
      return 0;
   return N->height;
}
Node *newNode(int ele)
{
   Node *node = (Node *)malloc(sizeof(Node));
   node->data = ele;
   node->left = NULL;
   node->right = NULL;
   node->height = 1;
   return (node);
}
Node *rightRotate(Node *y)
{
   Node *x = y->left;
   Node *T2 = x->right;
   x->right = y;
   y->left = T2;
   y->height = max(height(y->left), height(y->right)) + 1;
   x->height = max(height(x->left), height(x->right)) + 1;
```

```
    return x;
}
Node *leftRotate(Node *x)
{
    Node *y = x->right;
    Node *T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}
int getBalance(Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}
Node *Insert(Node *node, int ele)
{
    int balance;
    if (node == NULL)
        return (newNode(ele));
    if (ele < node->data)
        node->left = Insert(node->left, ele);
    else if (ele > node->data)
        node->right = Insert(node->right, ele);
    else
        return node;
    node->height = 1 + max(height(node->left), height(node-
>right));
    balance = getBalance(node);
    if (balance > 1 && ele < node->left->data)
        return rightRotate(node);
    if (balance < -1 && ele > node->right->data)
        return leftRotate(node);
    if (balance > 1 && ele > node->left->data)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
```

```
    }
    if (balance < -1 && ele < node->right->data)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
Node *Create(Node *root)
{
    int num, i, ele;
    printf("\n Enter number of nodes:");
    scanf("%d", &num);
    printf("\n Enter elements:");
    for (i = 0; i < num; i++)
    {
        scanf("%d", &ele);
        root = Insert(root, ele);
    }
    return root;
}
Node *minValueNode(Node *node)
{
    Node *current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}
Node *Delete(Node *root, int ele)
{
    int balance;
    if (root == NULL)
    {
        printf("\nTree is empty. Please create tree\n");
        return root;
    }
    if (ele < root->data)
        root->left = Delete(root->left, ele);
    else if (ele > root->data)
        root->right = Delete(root->right, ele);
```

```
else
{
   if ((root->left == NULL) || (root->right == NULL))
   {
      Node *temp = root->left ? root->left : root->right;
      if (temp == NULL)
      {
         temp = root;
         root = NULL;
      }
      else
         *root = *temp;
      free(temp);
   }
   else
   {
      Node *temp = minValueNode(root->right);
      root->data = temp->data;
      root->right = Delete(root->right, temp->data);
   }
}
if (root == NULL)
   return root;
root->height = 1 + max(height(root->left), height(root->right));
balance = getBalance(root);
if (balance > 1 && getBalance(root->left) >= 0)
   return rightRotate(root);
if (balance > 1 && getBalance(root->left) < 0)
{
   root->left = leftRotate(root->left);
   return rightRotate(root);
}
if (balance < -1 && getBalance(root->right) <= 0)
   return leftRotate(root);
if (balance < -1 && getBalance(root->right) > 0)
{
   root->right = rightRotate(root->right);
   return leftRotate(root);
}
return root;
```

```
}
void Inorder(Node *root)
{
   if (root != NULL)
   {
      Inorder(root->left);
      printf("%d ", root->data);
      Inorder(root->right);
   }
}
int main()
{
   Node *root = NULL;
   int ele;
   int e = 1, ch;
   while (e)
   {
      printf("\n--------------MENU--------------\n");
      printf("\n\t1. Create\n\t2. Insert\n\t3. Inorder Traversal\n\t4.
        Delete\n\t5. Exit\n");
      printf("\n--------------------------------\n");
      printf("\n Enter your choice:");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         root = Create(root);
         break;
      case 2:
         printf("\n Enter the element to insert:");
         scanf("%d", &ele);
         root = Insert(root, ele);
         break;
      case 3:
         Inorder(root);
         break;
      case 4:
         printf("\n Enter the element to delete :");
         scanf("%d", &ele);
         root = Delete(root, ele);
```

```
            break;
        case 5:
            e = 0;
            break;
        default:
            printf("\n Invalid choice \n");
        }
    }
    return 0;
}
```

Output:

--------------MENU--------------

1. Create
2. Insert
3. Inorder Traversal
4. Delete
5. Exit

Enter your choice:1

Enter number of nodes:3

Enter elements:12 8 24

--------------MENU--------------

1. Create
2. Insert
3. Inorder Traversal
4. Delete
5. Exit

Enter your choice:2

Enter the element to insert:36

--------------MENU--------------

1. Create
2. Insert
3. Inorder Traversal
4. Delete
5. Exit

```
--------------------------------

 Enter your choice:3
8 12 24 36
--------------MENU--------------

      1. Create
      2. Insert
      3. Inorder Traversal
      4. Delete
      5. Exit

--------------------------------

 Enter your choice:4

 Enter the element to delete :24

--------------MENU--------------

      1. Create
      2. Insert
      3. Inorder Traversal
      4. Delete
      5. Exit

--------------------------------

 Enter your choice:3
8 12 36
--------------MENU--------------

      1. Create
      2. Insert
      3. Inorder Traversal
      4. Delete
      5. Exit

--------------------------------
 Enter your choice:
```

/*B-TREE*/

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
#define MIN 2
struct BTreeNode
{
   int val[MAX + 1], count;
   struct BTreeNode *link[MAX + 1];
};
typedef struct BTreeNode BTreeNode;
BTreeNode *createNode(BTreeNode *root, int val, BTreeNode
*child)
{
   BTreeNode *newNode;
   newNode = (BTreeNode *)malloc(sizeof(BTreeNode));
   newNode->val[1] = val;
   newNode->count = 1;
   newNode->link[0] = root;
   newNode->link[1] = child;
   return newNode;
}
void insertNode(int val, int pos, BTreeNode *node, BTreeNode
*child)
{
   int j = node->count;
   while (j > pos)
   {
      node->val[j + 1] = node->val[j];
      node->link[j + 1] = node->link[j];
      j--;
   }
   node->val[j + 1] = val;
   node->link[j + 1] = child;
   node->count++;
}
void splitNode(int val, int *pval, int pos, BTreeNode *node,
BTreeNode *child,
           BTreeNode **newNode)
```

```
{
   int median, j;
   if (pos > MIN)
      median = MIN + 1;
   else
      median = MIN;
   *newNode = (BTreeNode *)malloc(sizeof(BTreeNode));
   j = median + 1;
   while (j <= MAX)
   {
      (*newNode)->val[j - median] = node->val[j];
      (*newNode)->link[j - median] = node->link[j];
      j++;
   }
   node->count = median;
   (*newNode)->count = MAX - median;
   if (pos <= MIN)
   {
      insertNode(val, pos, node, child);
   }
   else
   {
      insertNode(val, pos - median, *newNode, child);
   }
   *pval = node->val[node->count];
   (*newNode)->link[0] = node->link[node->count];
   node->count--;
}
int setValue(int val, int *pval, BTreeNode *node, BTreeNode
**child)
{
   int pos;
   if (!node)
   {
      *pval = val;
      *child = NULL;
      return 1;
   }
   if (val < node->val[1])
   {
```

```
        pos = 0;
    }
    else
    {
        for (pos = node->count; (val < node->val[pos] && pos > 1);
pos--)
            ;
        if (val == node->val[pos])
        {
            printf("\nSorry, duplicates are not permitted\n");
            return 0;
        }
    }
    if (setValue(val, pval, node->link[pos], child))
    {
        if (node->count < MAX)
        {
            insertNode(*pval, pos, node, *child);
        }
        else
        {
            splitNode(*pval, pval, pos, node, *child, child);
            return 1;
        }
    }
    return 0;
}
BTreeNode *Insert(BTreeNode *root, int val)
{
    int flag, i;
    BTreeNode *child;
    flag = setValue(val, &i, root, &child);
    if (flag)
        root = createNode(root, i, child);
    return root;
}
BTreeNode *Create(BTreeNode *root)
{
    int num, i, ele;
    printf("\n Enter the number of elements:");
```

```
    scanf("%d", &num);
    printf("\n Enter elements:");
    for (i = 0; i < num; i++)
    {
        scanf("%d", &ele);
        root = Insert(root, ele);
    }
    return root;
}
void search(int val, int *pos, BTreeNode *myNode)
{
    if (!myNode)
    {
        return;
    }
    if (val < myNode->val[1])
    {
        *pos = 0;
    }
    else
    {
        for (*pos = myNode->count; (val < myNode->val[*pos] &&
*pos > 1);
             (*pos)--)
            ;
        if (val == myNode->val[*pos])
        {
            printf("\nThe element %d is present in the B - Tree\n",
val);
            return;
        }
    }
    search(val, pos, myNode->link[*pos]);
    return;
}
void displayTree(BTreeNode *myNode)
{
    int i;
    if (myNode)
    {
```

```
        for (i = 0; i < myNode->count; i++)
        {
           displayTree(myNode->link[i]);
           printf("%d ", myNode->val[i + 1]);
        }
        displayTree(myNode->link[i]);
    }
}
int main()
{
    BTreeNode *root = NULL;
    int pos;
    int ele;
    int e = 1, ch;
    while (e)
    {
        printf("\n--------------MENU--------------\n");
printf( "\n\t1. Create\n\t2. Insert\n\t3. Display\n\t4.Search\n\t5.
Exit\n" );
printf( "\n-------------------------------\n" );printf( "\n Enter your
choice:" );
scanf( "%d", &ch );
switch( ch )
{
        case 1:
           root = Create(root);
           break;
        case 2:
           printf("\n Enter the element to insert:");
           scanf("%d", &ele);
           root = Insert(root, ele);
           break;
        case 3:
           displayTree(root);
           break;
        case 4:
           printf("\n Enter the element to search :");
           scanf("%d", &ele);
           search(ele, &pos, root);
           break;
```

```
        case 5:
            e = 0;
            break;
        default:
            printf("\n Invalid choice \n");
}
    }
    return 0;
}
```

Output:

--------------MENU--------------

1. Create
2. Insert
3. Display
4.Search
5. Exit

Enter your choice:1

Enter the number of elements:3

Enter elements:12 8 16

--------------MENU--------------

1. Create
2. Insert
3. Display
4.Search
5. Exit

Enter your choice:2

Enter the element to insert:24

--------------MENU--------------

1. Create
2. Insert
3. Display
4.Search
5. Exit

Enter your choice:2

Enter the element to insert:36

--------------MENU--------------

1. Create
2. Insert
3. Display
4.Search
5. Exit

Enter your choice:3
8 12 16 24 36
--------------MENU--------------

1. Create
2. Insert
3. Display
4.Search
5. Exit

Enter your choice:4

Enter the element to search :24

The element 24 is present in the B - Tree

--------------MENU--------------

1. Create
2. Insert
3. Display
4.Search

5. Exit

Enter your choice:

/*BREADTH FIRST SEARCH ALGORITHM*/

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40
struct node
{
   int vertex;
   struct node *next;
};
typedef struct node node;
struct Graph
{
   int numVertices;
   struct node **adjLists;
   int *visited;
};
typedef struct Graph Graph;
struct queue
{
   int items[SIZE];
   int front;
   int rear;
};
typedef struct queue queue;
queue *createQueue()
{
   queue *q = malloc(sizeof(struct queue));
   q->front = -1;
   q->rear = -1;
   return q;
}
int isEmpty(queue *q)
{
   if (q->rear == -1)
      return 1;
   else
      return 0;
}
void enqueue(queue *q, int value)
```

```
{
   if (q->rear == SIZE - 1)
      printf("\nMemory overflow. Queue is full...\n");
   else
   {
      if (q->front == -1)
         q->front = 0;
      q->rear++;
      q->items[q->rear] = value;
   }
}
int dequeue(queue *q)
{
   int item;
   if (isEmpty(q))
   {
      printf("\nQueue is empty\n");
      item = -1;
   }
   else
   {
      item = q->items[q->front];
      q->front++;
      if (q->front > q->rear)
      {
         q->front = q->rear = -1;
      }
   }
   return item;
}
node *createNode(int v)
{
   node *newNode = (node *)malloc(sizeof(node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}
void addEdge(Graph *graph, int src, int dest)
{
   node *newNode = createNode(dest);
```

```
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}
Graph *createGraph(int vertices, int edges)
{
    int i;
    int src, dest;
    Graph *graph = (Graph *)malloc(sizeof(Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(node *));
    graph->visited = malloc(vertices * sizeof(int));
    for (i = 0; i < vertices; i++)
    {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    printf("\nEnter Edges...\n");
    printf("\n<source,destination> (Between 0 to %d)", vertices -
1);
    for (i = 0; i < edges; i++)
    {
        printf("\nEnter edge %d:", i + 1);
        scanf("%d%d", &src, &dest);
        addEdge(graph, src, dest);
    }
    return graph;
}
void BFS(Graph *graph, int start)
{
    queue *q = createQueue();
    graph->visited[start] = 1;
    enqueue(q, start);
    while (!isEmpty(q))
    {
        int currentVertex = dequeue(q);
        printf(" %d -> ", currentVertex);
        node *temp = graph->adjLists[currentVertex];
```

```
        while (temp)
        {
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0)
            {
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp->next;
        }
    }
}
void displayGraph(Graph *graph)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        node *temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while (temp)
        {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}
int main()
{
    Graph *graph = NULL;
    int nv, ne;
    int start = 0;
    int e = 1, ch;
    while (e)
    {
        printf("\n--------------MENU-------------\n");
printf( "\n\t1. Create Graph\n\t2. Display\n\t3. Breadth First
Search (BFS) Algorithm\n\t4. Exit\n" );
printf( "\n-------------------------------\n" );
printf( "\n Enter your choice:" );
```

```
scanf( "%d", &ch );
switch( ch )
{
        case 1:
            printf("\nEnter number of verices and edges: ");
            scanf("%d%d", &nv, &ne);
            graph = createGraph(nv, ne);
            break;
        case 2:
            displayGraph(graph);
            break;
        case 3:
            printf("\nSearched in the order (from the vertex0) : ");
                        BFS(graph, start);
            break;
        case 4:
            e = 0;
            break;
        default:
            printf("\n Invalid choice \n");
}
    }
    return 0;
}
```

Output:

```
--------------MENU--------------

        1. Create Graph
        2. Display
        3. Breadth First Search (BFS) Algorithm
        4. Exit

--------------------------------

 Enter your choice:1

Enter number of verices and edges: 3 3

Enter Edges...

<source,destination> (Between 0 to 2)
Enter edge 1:0 2

Enter edge 2:2 1

Enter edge 3:1 0

--------------MENU--------------

        1. Create Graph
        2. Display
        3. Breadth First Search (BFS) Algorithm
        4. Exit

--------------------------------

 Enter your choice:2

 Adjacency list of vertex 0
 1 -> 2 ->

 Adjacency list of vertex 1
```

```
 0 -> 2 ->

 Adjacency list of vertex 2
 1 -> 0 ->

--------------MENU--------------

        1. Create Graph
        2. Display
        3. Breadth First Search (BFS) Algorithm
        4. Exit

--------------------------------

 Enter your choice:3

Searched in the order (from the vertex0) :  0 ->  1 ->  2 ->
--------------MENU--------------

        1. Create Graph
        2. Display
        3. Breadth First Search (BFS) Algorithm
        4. Exit

--------------------------------

 Enter your choice:
```

/*DEPTH FIRST ALGORITHM*/

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int vertex;
   struct node *next;
};
typedef struct node node;
struct Graph
{
   int numVertices;
   int *visited;
   node **adjLists;
};
typedef struct Graph Graph;
node *createNode(int v)
{
   node *newNode = (node *)malloc(sizeof(node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}
void addEdge(Graph *graph, int src, int dest)
{
   node *newNode = createNode(dest);
   newNode->next = graph->adjLists[src];
   graph->adjLists[src] = newNode;
   newNode = createNode(src);
   newNode->next = graph->adjLists[dest];
   graph->adjLists[dest] = newNode;
}
Graph *createGraph(int vertices, int edges)
{
   int i;
   int src, dest;
   Graph *graph = (Graph *)malloc(sizeof(Graph));
   graph->numVertices = vertices;
   graph->adjLists = malloc(vertices * sizeof(node *));
```

```
    graph->visited = malloc(vertices * sizeof(int));
    for (i = 0; i < vertices; i++)
    {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    printf("\nEnter Edges...\n");
    printf("\n<source,destination> (Between 0 to %d)", vertices -
1);
    for (i = 0; i < edges; i++)
    {
        printf("\nEnter edge %d:", i + 1);
        scanf("%d%d", &src, &dest);
        addEdge(graph, src, dest);
    }
    return graph;
}
void DFS(Graph *graph, int vertex)
{
    node *adjList = graph->adjLists[vertex];
    node *temp = adjList;
    graph->visited[vertex] = 1;
    printf("%d -> ", vertex);
    while (temp != NULL)
    {
        int connectedVertex = temp->vertex;
        if (graph->visited[connectedVertex] == 0)
        {
            DFS(graph, connectedVertex);
        }
        temp = temp->next;
    }
}
void displayGraph(Graph *graph)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        node *temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
```

```
        while (temp)
        {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}
int main()
{
    Graph
        *graph = NULL;
    int nv, ne;
    int start = 0;
    int e = 1, ch;
    while (e)
    {
        printf("\n--------------MENU--------------\n");
printf( "\n\t1. Create Graph\n\t2. Display\n\t3. Depth First Search
(DFS) Algorithm\n\t4. Exit\n" );
printf( "\n--------------------------------\n" );
printf( "\n Enter your choice:" );
scanf( "%d", &ch );
switch( ch )
{
        case 1:
            printf("\nEnter number of verices and edges: ");
            scanf("%d%d", &nv, &ne);
            graph = createGraph(nv, ne);
            break;
        case 2:
            displayGraph(graph);
            break;
        case 3:
            printf("\nSearched in the order (from the vertex 0) : ");
                        DFS(graph, start);
            break;
        case 4:
            e = 0;
            break;
```

```
        default:
            printf("\n Invalid choice \n");
}
    }
    return 0;
}
```

Output:

--------------MENU--------------

1. Create Graph
2. Display
3. Depth First Search (DFS) Algorithm
4. Exit

Enter your choice:1

Enter number of verices and edges: 3 3

Enter Edges...

<source,destination> (Between 0 to 2)
Enter edge 1:0 2

Enter edge 2:2 1

Enter edge 3:1 0

--------------MENU--------------

1. Create Graph
2. Display
3. Depth First Search (DFS) Algorithm
4. Exit

Enter your choice:2

Adjacency list of vertex 0
1 -> 2 ->

Adjacency list of vertex 1

0 -> 2 ->

Adjacency list of vertex 2
1 -> 0 ->

--------------MENU--------------

1. Create Graph
2. Display
3. Depth First Search (DFS) Algorithm
4. Exit

Enter your choice:3

Searched in the order (from the vertex 0) : 0 -> 1 -> 2 ->
--------------MENU--------------

1. Create Graph
2. Display
3. Depth First Search (DFS) Algorithm
4. Exit

Enter your choice:

/*DIJKSTRA'S ALGORITHM*/

```
#include <stdio.h>
#define SIZE 10
#define INFINITY 999
void read_graph(int *nv, int adj[][SIZE])
{
   int i, j;
   printf("\nEnter the number of vertices : ");
   scanf("%d", nv);
   printf("\nEnter the adjecency matrix (order %d x %d) :\n", *nv,
*nv);
   for (i = 0; i < *nv; i++)
      for (j = 0; j < *nv; j++)
         scanf("%d", &adj[i][j]);
}
void Dijkstra(int adj[][SIZE], int *nv, int start, int distance[])
{
   int cost[SIZE][SIZE], pred[SIZE];
   int visited[SIZE], count, mindistance, nextnode, i, j;
   if (!*nv)
   {
      printf("\nPlease read a graph...\n");
      return;
   }
   for (i = 0; i < *nv; i++)
      for (j = 0; j < *nv; j++)
         if (adj[i][j] == 0)
            cost[i][j] = INFINITY;
         else
            cost[i][j] = adj[i][j];
   for (i = 0; i < *nv; i++)
   {
      distance[i] = cost[start][i];
      pred[i] = start;
      visited[i] = 0;
   }
   distance[start] = 0;
   visited[start] = 1;
   count = 1;
```

```
    while (count < *nv - 1)
    {
        mindistance = INFINITY;
        for (i = 0; i < *nv; i++)
            if (distance[i] < mindistance && !visited[i])
            {
                mindistance = distance[i];
                nextnode = i;
            }
        visited[nextnode] = 1;
        for (i = 0; i < *nv; i++)
            if (!visited[i])
                if (mindistance + cost[nextnode][i] < distance[i])
                {
                    distance[i] = mindistance + cost[nextnode][i];
                    pred[i] = nextnode;
                }
        count++;
    }
printf("\nSuccessfully created shortest path vector beased on the
given start vertex %d \n", start);
for(i = 0; i < *nv; i++)
if(i != start)
{
        printf("\nDistance from source to %d: %d", i, distance[i]);
}
}
void display(int adj[][SIZE], int *nv, int flag, int distance[], int
start)
{
    int i, j;
    if (!*nv)
    {
        printf("\nPlease read a graph...\n");
        return;
    }
    printf("\nThe given graph (adjacency matrix) is:\n");
    for (i = 0; i < *nv; i++)
    {
        for (j = 0; j < *nv; j++)
```

```
            printf("%d ", adj[i][j]);
        printf("\n");
    }
    if (flag)
    {
        for (i = 0; i < *nv; i++)
            if (i != start)
            {
                printf("\nDistance from source to %d: %d", i,
distance[i]);
            }
    }
}
int main()
{
    int adj[SIZE][SIZE],distance[SIZE];
   int nv;
    int start = 0;
    int flag = 0;
    int e = 1, ch;
    while (e)
    {
        printf("\n--------------MENU--------------\n");
printf( "\n\t1. Read Graph\n\t2. Display\n\t3. Dijksta's Algorithm-
Shortest path(Single source)\n\t4. Exit\n" );
printf( "\n--------------------------------\n" );
printf( "\n Enter your choice:" );
scanf( "%d", &ch );
switch( ch )
{
        case 1:
            read_graph(&nv, adj);
            break;
        case 2:
            display(adj, &nv, flag, distance, start);
            break;
        case 3:
            flag = 1;
            Dijkstra(adj, &nv, start, distance);
            break;
```

```
        case 4:
            e = 0;
            break;
        default:
            printf("\n Invalid choice \n");
}
    }
    return 0;
}
```

Output:

```
--------------MENU--------------

        1. Read Graph
        2. Display
        3. Dijksta's Algorithm- Shortest path(Single source)
        4. Exit

--------------------------------

 Enter your choice:1

Enter the number of vertices : 5

Enter the adjecency matrix (order 5 x 5) :
0 3 1 0 0
3 0 7 5 1
1 7 0 0 0
0 5 2 0 7
0 1 0 7 0

--------------MENU--------------

        1. Read Graph
        2. Display
        3. Dijksta's Algorithm- Shortest path(Single source)
        4. Exit

--------------------------------

 Enter your choice:2

The given graph (adjacency matrix) is:
0 3 1 0 0
3 0 7 5 1
1 7 0 0 0
0 5 2 0 7
0 1 0 7 0
```

--------------MENU--------------

1. Read Graph
2. Display
3. Dijksta's Algorithm- Shortest path(Single source)
4. Exit

Enter your choice:3

Successfully created shortest path vector beased on the given start vertex 0

Distance from source to 1: 3
Distance from source to 2: 1
Distance from source to 3: 8
Distance from source to 4: 4
--------------MENU--------------

1. Read Graph
2. Display
3. Dijksta's Algorithm- Shortest path(Single source)
4. Exit

Enter your choice:

/*KRUSKAL ALGORITHM*/

```
#include <stdio.h>
#define SIZE 20
#define infinity 999
void read_graph(int *nv, int adj[][SIZE])
{
   int i, j;
   printf("\nEnter the number of vertices : ");
   scanf("%d", nv);
   printf("\nEnter the adjecency matrix (order %d x %d) :\n", *nv,
*nv);
   for (i = 1; i <= *nv; i++)
      for (j = 1; j <= *nv; j++)
         scanf("%d", &adj[i][j]);
}
int find(int i, int parent[])
{
   while (parent[i])
      i = parent[i];
   return i;
}
int uni(int i, int j, int parent[])
{
   if (i != j)
   {
      parent[j] = i;
      return 1;
   }
   return 0;
}
void Kruskal(int adj[][SIZE], int *nv)
{
   int i, j, a, b, u, v, ne = 1;
   int min, mincost = 0;
   int parent[SIZE] = {0};
   int adj_temp[SIZE][SIZE];
   if (!*nv)
   {
      printf("\nPlease read a graph...\n");
```

```
        return;
    }
    for (i = 1; i <= *nv; i++)
        for (j = 1; j <= *nv; j++)
        {
            adj_temp[i][j] = adj[i][j];
            if (adj_temp[i][j] == 0)
                adj_temp[i][j] = infinity;
        }
    printf("The edges of Minimum Cost Spanning Tree are\n");
    while (ne < *nv)
    {
        for (i = 1, min = infinity; i <= *nv; i++)
        {
            for (j = 1; j <= *nv; j++)
            {
                if (adj_temp[i][j] < min)
                {
                    min = adj_temp[i][j];
                    a = u = i;
                    b = v = j;
                }
            }
        }
        u = find(u, parent);
        v = find(v, parent);
        if (uni(u, v, parent))
        {
            printf("%d edge (%d,%d) = %d\n", ne++, a, b, min);
            mincost += min;
        }
        adj_temp[a][b] = adj_temp[b][a] = infinity;
    }
printf("\nSuccessfully created a spanning tree and its minimum
cost is %d\n", mincost);
}
void display(int adj[][SIZE], int *nv, int flag)
{
    int i, j;
    if (!*nv)
```

```
    {
        printf("\nPlease read a graph...\n");
        return;
    }
    printf("\nThe given graph (adjacency matrix) is:\n");
    for (i = 1; i <= *nv; i++)
    {
        for (j = 1; j <= *nv; j++)
            printf("%d ", adj[i][j]);
        printf("\n");
    }
    if (flag)
        Kruskal(adj, nv);
}
int main()
{
    int adj[SIZE][SIZE];
    int nv;
    int flag = 0;
    int e = 1, ch;
    while (e)
    {
        printf("\n--------------MENU--------------\n");
printf( "\n\t1. Read Graph\n\t2. Display\n\t3. Kruskal's
Algorithm- Spanning Tree\n\t4. Exit\n" );
printf( "\n-------------------------------\n" );
printf( "\n Enter your choice:" );
scanf( "%d", &ch );
switch( ch )
{
        case 1:
            read_graph(&nv, adj);
            break;
        case 2:
            display(adj, &nv, flag);
            break;
        case 3:
            flag = 1;
            Kruskal(adj, &nv);
            break;
```

```
        case 4:
            e = 0;
            break;
        default:
            printf("\n Invalid choice \n");
}
    }
    return 0;
}
```

Output:

```
--------------MENU--------------

        1. Read Graph
        2. Display
        3. Kruskal's Algorithm- Spanning Tree
        4. Exit

-------------------------------

 Enter your choice:1

Enter the number of vertices : 5

Enter the adjecency matrix (order 5 x 5) :
0 3 1 0 0
3 0 7 5 1
1 7 0 0 0
0 5 2 0 7
0 1 0 7 0

--------------MENU--------------

        1. Read Graph
        2. Display
        3. Kruskal's Algorithm- Spanning Tree
        4. Exit

-------------------------------

 Enter your choice:2

The given graph (adjacency matrix) is:
0 3 1 0 0
3 0 7 5 1
1 7 0 0 0
0 5 2 0 7
0 1 0 7 0
```

```
--------------MENU--------------

      1. Read Graph
      2. Display
      3. Kruskal's Algorithm- Spanning Tree
      4. Exit

--------------------------------

 Enter your choice:
3
The edges of Minimum Cost Spanning Tree are
1 edge (1,3) = 1
2 edge (2,5) = 1
3 edge (4,3) = 2
4 edge (1,2) = 3

Successfully created a spanning tree and its minimum cost is 7

--------------MENU--------------

      1. Read Graph
      2. Display
      3. Kruskal's Algorithm- Spanning Tree
      4. Exit

--------------------------------

 Enter your choice:
```

/*PRIMS ALGORITHM*/

```
#include <stdio.h>
#define SIZE 20
#define infinity 999
void read_graph(int *nv, int adj[][SIZE])
{
   int i, j;
   printf("\nEnter the number of vertices : ");
   scanf("%d", nv);
   printf("\nEnter the adjecency matrix (order %d x %d) :\n", *nv,
*nv);
   for (i = 0; i < *nv; i++)
      for (j = 0; j < *nv; j++)
         scanf("%d", &adj[i][j]);
}
void display(int adj[][SIZE], int st[][SIZE], int *nv, int flag, int
cost)
{
   int i, j;
   if (!*nv)
   {
      printf("\nPlease read a graph...\n");
      return;
   }
   printf("\nThe given graph (adjacency matrix) is:\n");
   for (i = 0; i < *nv; i++)
   {
      for (j = 0; j < *nv; j++)
         printf("%d ", adj[i][j]);
      printf("\n");
   }
   if (flag)
   {
      printf("\nSpanning Tree is: \n");
      for (i = 0; i < *nv; i++)
      {
         for (j = 0; j < *nv; j++)
            printf("%d ", st[i][j]);
         printf("\n");
```

```
        }
        printf("\nThe minimum cost is %d ", cost);
    }
}
int Prims(int adj[][SIZE], int st[][SIZE], int *nv)
{
    int cost[SIZE][SIZE];
    int u, v, min_distance, distance[SIZE], from[SIZE];
    int visited[SIZE], no_of_edges, i, min_cost, j;
    if (!*nv)
    {
        printf("\nPlease read a graph...\n");
        return 0;
    }
    for (i = 0; i < *nv; i++)
        for (j = 0; j < *nv; j++)
        {
            if (adj[i][j] == 0)
                cost[i][j] = infinity;
            else
                cost[i][j] = adj[i][j];
            st[i][j] = 0;
        }
    distance[0] = 0;
    visited[0] = 1;
    for (i = 1; i < *nv; i++)
    {
        distance[i] = cost[0][i];
        from[i] = 0;
        visited[i] = 0;
    }
    min_cost = 0;
    no_of_edges = *nv - 1;
    while (no_of_edges > 0)
    {
        min_distance = infinity;
        for (i = 1; i < *nv; i++)
            if (visited[i] == 0 && distance[i] < min_distance)
            {
                v = i;
```

```
            min_distance = distance[i];
          }
        u = from[v];
        st[u][v] = distance[v];
        st[v][u] = distance[v];
        no_of_edges--;
        visited[v] = 1;
        for (i = 1; i < *nv; i++)
          if (visited[i] == 0 && cost[i][v] < distance[i])
          {
            distance[i] = cost[i][v];
            from[i] = v;
          }
        min_cost = min_cost + cost[u][v];
    }
    return (min_cost);
}
int main()
{
    int adj[SIZE][SIZE],st[SIZE][SIZE];
    int nv;
    int cost = 0;
    int flag = 0;
    int e = 1, ch;
    while (e)
    {
        printf("\n--------------MENU--------------\n");
printf( "\n\t1. Read Graph\n\t2. Display\n\t3. Prim's Algorithm -
Spanning Tree\n\t4. Exit\n" );
printf( "\n--------------------------------\n" );
printf( "\n Enter your choice:" );
scanf( "%d", &ch );
switch( ch )
{
        case 1:
          read_graph(&nv, adj);
          break;
        case 2:
          display(adj, st, &nv, flag, cost);
          break;
```

```
        case 3:
            flag = 1;
            cost = Prims(adj, st, &nv);
            if (cost)
printf("\nSuccessfully created a spanning tree and its minimum
cost is %d \n", cost );
break;
case 4 : e = 0;
break;
default: printf( "\n Invalid choice \n" );
}
    }
    return 0;
}
```

Output:

--------------MENU--------------

1. Read Graph
2. Display
3. Prim's Algorithm - Spanning Tree
4. Exit

Enter your choice:1

Enter the number of vertices : 4

Enter the adjecency matrix (order 4 x 4) :
0 4 2 0
4 0 5 0
0 5 0 3
2 0 3 0

--------------MENU--------------

1. Read Graph
2. Display
3. Prim's Algorithm - Spanning Tree
4. Exit

Enter your choice:2

The given graph (adjacency matrix) is:
0 4 2 0
4 0 5 0
0 5 0 3
2 0 3 0

--------------MENU--------------

1. Read Graph
2. Display
3. Prim's Algorithm - Spanning Tree
4. Exit

Enter your choice:3

Successfully created a spanning tree and its minimum cost is 9

--------------MENU--------------

1. Read Graph
2. Display
3. Prim's Algorithm - Spanning Tree
4. Exit

Enter your choice:

/*RED-BLACK TREE*/

```
#include <stdio.h>
#include <stdlib.h>
struct rbNode
{
   int data;
   int color;
   struct rbNode *link[2];
};
typedef struct rbNode rbNode;
enum nodeColor
{
   RED,
   BLACK
};
rbNode *createNode(int data)
{
   rbNode *newnode;
   newnode = (rbNode *)malloc(sizeof(rbNode));
   newnode->data = data;
   newnode->color = RED;
   newnode->link[0] = newnode->link[1] = NULL;
   return newnode;
}
rbNode *Insert(rbNode *root, int data)
{
   rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
   int dir[98], ht = 0, index;
   ptr = root;
   if (!root)
   {
      root = createNode(data);
      return root;
   }
   stack[ht] = root;
   dir[ht++] = 0;
   while (ptr != NULL)
   {
      if (ptr->data == data)
```

```
        {
            printf("\nSorry , duplicates not allowed...\n");
            return root;
        }
        index = (data - ptr->data) > 0 ? 1 : 0;
        stack[ht] = ptr;
        ptr = ptr->link[index];
        dir[ht++] = index;
    }
    stack[ht - 1]->link[index] = newnode = createNode(data);
    while ((ht >= 3) && (stack[ht - 1]->color == RED))
    {
        if (dir[ht - 2] == 0)
        {
            yPtr = stack[ht - 2]->link[1];
            if (yPtr != NULL && yPtr->color == RED)
            {
                stack[ht - 2]->color = RED;
                stack[ht - 1]->color = yPtr->color = BLACK;
                ht = ht - 2;
            }
            else
            {
                if (dir[ht - 1] == 0)
                {
                    yPtr = stack[ht - 1];
                }
                else
                {
                    xPtr = stack[ht - 1];
                    yPtr = xPtr->link[1];
                    xPtr->link[1] = yPtr->link[0];
                    yPtr->link[0] = xPtr;
                    stack[ht - 2]->link[0] = yPtr;
                }
                xPtr = stack[ht - 2];
                xPtr->color = RED;
                yPtr->color = BLACK;
                xPtr->link[0] = yPtr->link[1];
                yPtr->link[1] = xPtr;
```

```
        if (xPtr == root)
        {
            root = yPtr;
        }
        else
        {
            stack[ht - 3]->link[dir[ht - 3]] = yPtr;
        }
        break;
    }
}
else
{
    yPtr = stack[ht - 2]->link[0];
    if ((yPtr != NULL) && (yPtr->color == RED))
    {
        stack[ht - 2]->color = RED;
        stack[ht - 1]->color = yPtr->color = BLACK;
        ht = ht - 2;
    }
    else
    {
        if (dir[ht - 1] == 1)
        {
            yPtr = stack[ht - 1];
        }
        else
        {
            xPtr = stack[ht - 1];
            yPtr = xPtr->link[0];
            xPtr->link[0] = yPtr->link[1];
            yPtr->link[1] = xPtr;
            stack[ht - 2]->link[1] = yPtr;
        }
        xPtr = stack[ht - 2];
        yPtr->color = BLACK;
        xPtr->color = RED;
        xPtr->link[1] = yPtr->link[0];
        yPtr->link[0] = xPtr;
        if (xPtr == root)
```

```
            {
               root = yPtr;
            }
            else
            {
               stack[ht - 3]->link[dir[ht - 3]] = yPtr;
            }
            break;
         }
      }
   }
   root->color = BLACK;
   return root;
}
rbNode *Create(rbNode *root)
{
   int num, i, ele;
   printf("\n Enter number of nodes:");
   scanf("%d", &num);
   printf("\n Enter elements:");
   for (i = 0; i < num; i++)
   {
      scanf("%d", &ele);
      root = Insert(root, ele);
   }
   return root;
}
rbNode *Delete(rbNode *root, int data)
{
   rbNode *stack[98], *ptr, *xPtr, *yPtr;
   rbNode *pPtr, *qPtr, *rPtr;
   int dir[98], ht = 0, diff, i;
   enum nodeColor color;
   if (!root)
   {
      printf("\nEmpty tree\n");
      return root;
   }
   ptr = root;
   while (ptr != NULL)
```

```
{
   if ((data - ptr->data) == 0)
      break;
   diff = (data - ptr->data) > 0 ? 1 : 0;
   stack[ht] = ptr;
   dir[ht++] = diff;
   ptr = ptr->link[diff];
}
if (ptr->link[1] == NULL)
{
   if ((ptr == root) && (ptr->link[0] == NULL))
   {
      free(ptr);
      root = NULL;
   }
   else if (ptr == root)
   {
      root = ptr->link[0];
      free(ptr);
   }
   else
   {
      stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];
   }
}
else
{
   xPtr = ptr->link[1];
   if (xPtr->link[0] == NULL)
   {
      xPtr->link[0] = ptr->link[0];
      color = xPtr->color;
      xPtr->color = ptr->color;
      ptr->color = color;
      if (ptr == root)
      {
         root = xPtr;
      }
      else
      {
```

```
            stack[ht - 1]->link[dir[ht - 1]] = xPtr;
        }
        dir[ht] = 1;
        stack[ht++] = xPtr;
    }
    else
    {
        i = ht++;
        while (1)
        {
            dir[ht] = 0;
            stack[ht++] = xPtr;
            yPtr = xPtr->link[0];
            if (!yPtr->link[0])
                break;
            xPtr = yPtr;
        }
        dir[i] = 1;
        stack[i] = yPtr;
        if (i > 0)
            stack[i - 1]->link[dir[i - 1]] = yPtr;
        yPtr->link[0] = ptr->link[0];
        xPtr->link[0] = yPtr->link[1];
        yPtr->link[1] = ptr->link[1];
        if (ptr == root)
        {
            root = yPtr;
        }
        color = yPtr->color;
        yPtr->color = ptr->color;
        ptr->color = color;
    }
}
if (ht < 1)
    return root;
if (ptr->color == BLACK)
{
    while (1)
    {
        pPtr = stack[ht - 1]->link[dir[ht - 1]];
```

```
if (pPtr && pPtr->color == RED)
{
   pPtr->color = BLACK;
   break;
}
if (ht < 2)
   break;
if (dir[ht - 2] == 0)
{
   rPtr = stack[ht - 1]->link[1];
   if (!rPtr)
      break;
   if (rPtr->color == RED)
   {
      stack[ht - 1]->color = RED;
      rPtr->color = BLACK;
      stack[ht - 1]->link[1] = rPtr->link[0];
      rPtr->link[0] = stack[ht - 1];
      if (stack[ht - 1] == root)
      {
         root = rPtr;
      }
      else
      {
         stack[ht - 2]->link[dir[ht - 2]] = rPtr;
      }
      dir[ht] = 0;
      stack[ht] = stack[ht - 1];
      stack[ht - 1] = rPtr;
      ht++;
      rPtr = stack[ht - 1]->link[1];
   }
   if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK)
        &&
      (!rPtr->link[1] || rPtr->link[1]->color == BLACK))
   {
      rPtr->color = RED;
   }
   else
   {
```

```
        if (!rPtr->link[0] || rPtr->link[0]->color == BLACK)
        {
            qPtr->color = RED;
            rPtr->color = BLACK;
            qPtr->link[0] = qPtr->link[1];
            rPtr->link[1] = rPtr;
            qPtr = stack[ht - 1]->link[1] = qPtr;
        }
        rPtr->color = stack[ht - 1]->color;
        stack[ht - 1]->color = BLACK;
        rPtr->link[1]->color = BLACK;
        stack[ht - 1]->link[1] = rPtr->link[0];
        rPtr->link[0] = stack[ht - 1];

        if (stack[ht - 1] == root)
        {
            root = rPtr;
        }
        else
        {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
        }
        break;
    }
}
else
{
    rPtr = stack[ht - 1]->link[0];
    if (!rPtr)
        break;
    if (rPtr->color == RED)
    {
        stack[ht - 1]->color = RED;
        rPtr->color = BLACK;
        stack[ht - 1]->link[0] = rPtr->link[1];
        rPtr->link[1] = stack[ht - 1];
        if (stack[ht - 1] == root)
        {
            root = rPtr;
        }
```

```
        else
        {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
        }
        dir[ht] = 1;
        stack[ht] = stack[ht - 1];
        stack[ht - 1] = rPtr;
        ht++;
        rPtr = stack[ht - 1]->link[0];
    }
    if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK)
&& (!rPtr
                                    ->link[1] ||
                                rPtr->link[1]-
>color == BLACK))
    {
        rPtr->color = RED;
    }
    else
    {
        if (!rPtr->link[0] || rPtr->link[0]->color == BLACK)
        {
            qPtr = rPtr->link[1];
            rPtr->color = RED;
            qPtr->color = BLACK;
            rPtr->link[1] = qPtr->link[0];
            qPtr->link[0] = rPtr;
            rPtr = stack[ht - 1]->link[0] = qPtr;
        }
        rPtr->color = stack[ht - 1]->color;
        stack[ht - 1]->color = BLACK;
        rPtr->link[0]->color = BLACK;
        stack[ht - 1]->link[0] = rPtr->link[1];
        rPtr->link[1] = stack[ht - 1];
        if (stack[ht - 1] == root)
        {
            root = rPtr;
        }
        else
        {
```

```
                stack[ht - 2]->link[dir[ht - 2]] = rPtr;
              }
              break;
            }
          }
          ht--;
        }
    }
    return root;
}

void Inorder(rbNode *root)
{
    if (root != NULL)
    {
        Inorder(root->link[0]);
        printf("%d ->", root->data);
        Inorder(root->link[1]);
    }
}

int main()
{
    rbNode *root = NULL;
    int ele;
    int e = 1, ch;
    while (e)
    {
        printf("\n--------------MENU--------------\n");
        printf("\n\t1. Create\n\t2. Insert\n\t3. Inorder Traversal\n\
t4.Delete\n\t5. Exit\n");
        printf("\n--------------------------------\n");
        printf("\n Enter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            root = Create(root);
            break;
        case 2:
```

```
            printf("\n Enter the element to insert:");
            scanf("%d", &ele);
            root = Insert(root, ele);
            break;
        case 3:
            Inorder(root);
            break;
        case 4:
            printf("\n Enter the element to delete :");
            scanf("%d", &ele);
            root = Delete(root, ele);
            break;
        case 5:
            e = 0;
            break;
        default:
            printf("\n Invalid choice \n");
        }
    }
    return 0;
}
```

Output:

--------------MENU--------------

1. Create
2. Insert
3. Inorder Traversal
4. Delete
5. Exit

Enter your choice:1

Enter number of nodes:9

Enter elements:4 2 6 1 3 5 8 7 9

--------------MENU--------------

1. Create
2. Insert
3. Inorder Traversal
4. Delete
5. Exit

Enter your choice:2

Enter the element to insert:13

--------------MENU--------------

1. Create
2. Insert
3. Inorder Traversal
4. Delete
5. Exit

```
--------------------------------

 Enter your choice:3
1 ->2 ->3 ->4 ->5 ->6 ->7 ->8 ->9 ->13 ->
--------------MENU--------------

        1. Create
        2. Insert
        3. Inorder Traversal
        4.Delete
        5. Exit

--------------------------------

 Enter your choice:4

 Enter the element to delete :3

--------------MENU--------------

        1. Create
        2. Insert
        3. Inorder Traversal
        4.Delete
        5. Exit

--------------------------------

 Enter your choice:3
1 ->2 ->4 ->5 ->6 ->7 ->8 ->9 ->13 ->
--------------MENU--------------

        1. Create
        2. Insert
        3. Inorder Traversal
        4.Delete
        5. Exit
--------------------------------
 Enter your choice:
```

/*TOPOLOGICAL SORTING*/

```
#include <stdio.h>
#define SIZE 20
void read_graph(int *nv, int adj[][SIZE])
{
   int i, j;
   printf("\nEnter the number of vertices : ");
   scanf("%d", nv);
   printf("\nEnter the adjecency matrix (order %d x %d) :\n", *nv,
*nv);
   for (i = 0; i < *nv; i++)
      for (j = 0; j < *nv; j++)
         scanf("%d", &adj[i][j]);
}
int indegree(int v, int *nv, int adj[][SIZE])
{
   int i, id = 0;
   for (i = 0; i < *nv; i++)
      if (adj[i][v] == 1)
         id++;
   return id;
}
int delete_queue(int queue[], int *front, int *rear)
{
   int del_item;
   if (*front == -1 || *front > *rear)
   {
      printf("\nQueue underflow\n");
      return 0;
   }
   else
   {
      del_item = queue[*front];
      *front = *front + 1;
      return del_item;
   }
}
void insert_queue(int vertex, int queue[], int *front, int *rear)
{
```

```
    if (*rear == SIZE - 1)
        printf("\nQueue overflow\n");
    else
    {
        if (*front == -1)
            *front = 0;
        *rear = *rear + 1;
        queue[*rear] = vertex;
    }
}
int isEmpty_queue(int *front, int *rear)
{
    if (*front == -1 || *front > *rear)
        return 1;
    else
        return 0;
}
void topo_sort(int *nv, int adj[][SIZE], int topo_order[], int *flag)
{
    int i, v;
    int count = 0;
    int indeg[SIZE];
    int queue[SIZE], front, rear;
    front = rear = -1;
    *flag = 1;
    if (!*nv)
    {
        printf("\nPlease read a graph \n");
        return;
    }
    for (i = 0; i < *nv; i++)
    {
        indeg[i] = indegree(i, nv, adj);
        if (indeg[i] == 0)
            insert_queue(i, queue, &front, &rear);
    }
    while (!isEmpty_queue(&front, &rear) && count < *nv)
    {
        v = delete_queue(queue, &front, &rear);
        topo_order[++count] = v + 1;
```

```
        for (i = 0; i < *nv; i++)
        {
            if (adj[v][i] == 1)
            {
                adj[v][i] = 0;
                indeg[i] = indeg[i] - 1;
                if (indeg[i] == 0)
                    insert_queue(i, queue, &front, &rear);
            }
        }
    }
    if (count < *nv)
    {
        printf("\nNo topological ordering possible, graph contains
cycle\n");
        *flag = 0;
        return;
    }
    printf("\nTopological ordering of vertices successfully
conducted\n");
}
void display(int *nv, int adj[][SIZE], int topo_order[], int *flag)
{
    int i, j;
    if (*nv)
    {
        printf("\nThe given adjecency matrix (order %d x %d) is :\
n", *nv, *nv);
        for (i = 0; i < *nv; i++)
        {
            for (j = 0; j < *nv; j++)
                printf("%d ", adj[i][j]);
            printf("\n");
        }
        if (*flag)
        {
            printf("\nVertices in topological order are :\n");
            for (i = 1; i <= *nv; i++)
                printf("%d ", topo_order[i]);
            printf("\n");
```

```
        }
    }
    else
    {
        printf("\nPlease read a graph \n");
        return;
    }
}
int main()
{
    int adj[SIZE][SIZE], topo_order[SIZE];
    int nv = 0;
    int flag = 0;
    int ele = 1, ch;
    while (ele)
    {
        printf("\n--------------MENU--------------\n");
printf( "\n\t1. Read Graph\n\t2. Topological Sort\n\t3. Display\n\
t4. Exit\n" );
printf( "\n-------------------------------\n" );
printf( "\n Enter your choice:" );
scanf( "%d", &ch );
switch( ch )
{
        case 1:
            read_graph(&nv, adj);
            break;
        case 2:
            topo_sort(&nv, adj, topo_order, &flag);
            break;
        case 3:
            display(&nv, adj, topo_order, &flag);
            break;
        case 4:
            ele = 0;
            printf("\nExit from the program\n");
            break;
        default: printf( "\n Invalid choice. Please enter a valid
choice... \n" );
}
```

```
    }
    return 0;
}
```

Output:

```
--------------MENU--------------

        1. Read Graph
        2. Topological Sort
        3. Display
        4. Exit

--------------------------------

 Enter your choice:1

Enter the number of vertices : 6

Enter the adjecency matrix (order 6 x 6) :
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 0
1 1 0 0 0 0
1 0 1 0 0 0

--------------MENU--------------

        1. Read Graph
        2. Topological Sort
        3. Display
        4. Exit

--------------------------------

 Enter your choice:2

Topological ordering of vertices successfully conducted

--------------MENU--------------

        1. Read Graph
        2. Topological Sort
```

3. Display
4. Exit

Enter your choice:3

The given adjecency matrix (order 6 x 6) is :
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 0
1 1 0 0 0 0
1 0 1 0 0 0

Vertices in topological order are :
5 6 1 3 4 2

--------------MENU--------------

1. Read Graph
2. Topological Sort
3. Display
4. Exit

Enter your choice: