MANUAL FOR DATA STRUCTURE LAB

of the

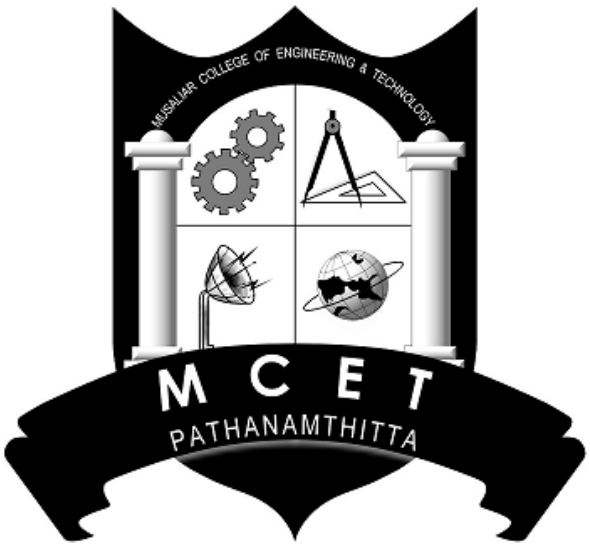
MCA COURSE

in

COMPUTER SCIENCE AND ENGINEERING

under

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY



MUSALIAR COLLEGE OF ENGINEERING AND TECHNOLOGY, PATHANAMTHITTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course No.	Course Name	L-T-P	Credits
03 CS 6821	Datastructure Lab using C	1-3-2	2

Experiment No	Description
I	Merge two sorted arrays and store in a third array
II	Write a C program for implementing Stack Operations
III	Write a C program for implementing Queue Operation
IV	Circular Queue - Add, Delete, Search
V	Set Data Structure and set operations (Union, Intersection and Difference) using Bit String
VI	Disjoint Sets and the associated operations (create, union, find)
VII	B Trees and its operations
VIII	Graph Traversal techniques (BFS)
IX	Graph Traversal techniques (DFS)
X	Prim's Algorithm for finding the minimum cost spanning tree
XI	Kruskal's algorithm using the Disjoint set data structure

PROGRAM NO: 1

AIM: Write a program for merging two sorted arrays and store in a third array.

ALGORITHM

1. Start

2. Input the two sorted arrays
3. Find and add the size of the given arrays and declare a third array of the same size.
4. Loop through the index of the third array using for loop:
    a)Check which array has the smaller element and fill the third array with the same.
    b)If the first array has the smaller element (i.e. arr1[i] < arr2[j]) then fill the third array with the element of the first array (i.e. arr3[k++] = arr1[i++])
    otherwise fill with the element of the second array(i.e. arr3[k++] = arr2[j++])
5. End loop
6. Output the third array.
7. Stop

PROGRAM:

```
#include <stdio.h>
void main()
{
int array1[50], array2[50], array3[100], m, n, i, j, k = 0;
printf("\n Enter size of  Array 1: ");
scanf("%d", &m);
printf("\n Enter sorted elements of array 1: \n");
for (i = 0; i < m; i++)
{
scanf("%d", &array1[i]);
}
printf("\n Enter size of Array 2: ");
scanf("%d", &n);

printf("\n Enter sorted elements of array 2: \n");
for (i = 0; i < n; i++)
{
scanf("%d", &array2[i]);
}

i = 0;
j = 0;
```

```
while (i < m && j < n)
{
if (array1[i] < array2[j])
{
array3[k] = array1[i];
i++;
}

else
{
array3[k] = array2[j];
j++;
}
k++;
}

if (i >= m)
{
while (j < n)
{
array3[k] = array2[j];
j++;
```

```
k++;
}
}

if (j >= n)
{
while (i < m)
{
array3[k] = array1[i];
i++;
k++;
}
}

printf("\n After merging: \n");
for (i = 0; i < m + n; i++)
{
printf("\n%d", array3[i]);
}

}
```

OUT PUT

Enter the size of Array 1

3

Enter the sorted elements of array1

1 2 3

Enter the size of Array 2

3

Enter the sorted elements of array2

4 5 6

After merging

1 2 3 4 5 6

PROGRAM NO: 2

AIM

Write a C program for implementing Stack Operations.

ALGORITHM

1. Start

2. Check top=size-1

3. Display stack overflow

4. Otherwise read the element to be inserted

5. Set top=top+1

6.Check top=-1

7.Display stack under flow

8.Otherwise display popped element

9.Stop

PROGRAM

```
#include<stdio.h>

#include<stdlib.h>

#define Size 4

int Top=-1, array[Size];
void Push();
void Pop();
void show();

int main()
{
   int choice;

   while(1)
   {
      printf("\nOperations performed by Stack");
      printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");
      printf("\n\nEnter the choice:");
      scanf("%d",&choice);

      switch(choice)
      {
         case 1: Push();
               break;
         case 2: Pop();
               break;
         case 3: show();
               break;
         case 4: exit(0);

         default: printf("\nInvalid choice!!");
      }
   }
}

void Push()
```

```
{
   int x;

   if(Top==Size-1)
   {
      printf("\nOverflow!!");
   }
   else
   {
      printf("\nEnter element to be inserted to the stack:");
      scanf("%d",&x);
      Top=Top+1;
      array[Top]=x;
   }
}

void Pop()
{
   if(Top== -1)
   {
      printf("\nUnderflow!!");
   }
   else
   {
      printf("\nPopped element:  %d",inp_array[Top]);
      Top=Top-1;
   }
}

void show()
{


   if(Top==-1)
   {
      printf("\nUnderflow!!");
   }
   else
   {
      printf("\nElements present in the stack: \n");
```

```
        for(int i=Top;i>=0;--i)
            printf("%d\n",inp_array[i]);
    }
}
```

OUTPUT:

```
Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4.End

Enter the choice:1

Enter element to be inserted to the stack:10

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4.End

Enter the choice:3

Elements present in the stack:
10

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
```

4.End

Enter the choice:2
Popped element: 10

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4.End

Enter the choice:3
Underflow!

PROGRAM NO:3

AIM

Write a C program for implementing Queue Operation

ALGORITHM

1.Start

2.Check rear = MAX-1

3.Display Queue overflow

4.Otherwise Check front= -1

5.Read the element in the Queue

6.Set rear=rear+1

and add element to rear

7.Check front= -1 or front >rear

8.Display Queue overflow

9.Otherwise delete from queue

10.Stop

PROGRAM

```
#include <stdio.h>

define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
   int choice;
   while (1)
   {
      printf("1.Insert element to queue \n");
      printf("2.Delete element from queue \n");
      printf("3.Display all elements of queue \n");
      printf("4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
         display();
         break;
         case 4:
         exit(1);
        default:
         printf("Wrong choice \n");
     } /* End of switch */
      } /* End of while */
```

```
} /* End of main() */

void insert()
{
   int add_item;
   if (rear == MAX - 1)
   printf("Queue Overflow \n");
   else
   {
      if (front == - 1)
      /*If queue is initially empty */
      front = 0;
      printf("Inset the element in queue : ");
      scanf("%d", &add_item);
      rear = rear + 1;
      queue_array[rear] = add_item;
}
} /* End of insert() */

void delete()
{
   if (front == - 1 || front > rear)
   {
      printf("Queue Underflow \n");
      return ;
   }
   else
   {
      printf("Element deleted from queue is : %d\n", queue_array[front]);
      front = front + 1;
}
} /* End of delete() */

void display()
{
   int i;
   if (front == - 1)
      printf("Queue is empty \n");
   else
   {
```

```
            printf("Queue is : \n");
            for (i = front; i <= rear; i++)
                printf("%d ", queue_array[i]);
            printf("\n");
        }
    }
```

OUTPUT

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 10
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 15
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 20
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 30
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2
```

Element deleted from queue is : 10
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3
Queue is :
15 20 30
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 4

PROGRAM N0:4

AIM: Implement a circular queue and its operations.

ALGORITHM

INSERT AN ELEMENT IN A CIRCULAR QUEUE

1. Start

2. Check front=0 & rear=size-1

3. Display queue is full

4. Otherwise rear=-1 then increment value of rear and front by 1

5 .Insert first items to front

6. Check front>rear ,display the items of queue.

7.Stop

DELETE A ELEMENT IN A CIRCULAR QUEUE

1. Start

2. Check front = -1 ,display queue is empty

3. Otherwise Check front = rear ,delete the element from front

4. Otherwise delete element from front and increment the value of front

5.Stop

PROGRAM

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

void insertq(int[], int);
void deleteq(int[]);
void display(int[]);

int front =  - 1;
int rear =  - 1;

int main()
{
   int n, ch;
   int queue[size];
   do
   {
      printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0. Exit");
      printf("\nEnter Choice 0-3? : ");
      scanf("%d", &ch);
      switch (ch)
      {
         case 1:
```

```
            printf("\nEnter number: ");
            scanf("%d", &n);
            insertq(queue, n);
            break;
        case 2:
            deleteq(queue);
            break;
        case 3:
            display(queue);
            break;
      }
   }while (ch != 0);
}



void insertq(int queue[], int item)
{
   if ((front == 0 && rear == size - 1) || (front == rear + 1))
   {
      printf("queue is full");
      return;
   }
   else if (rear ==  - 1)
```

```
    {
        rear++;
        front++;
    }
    else if (rear == size - 1 && front > 0)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    queue[rear] = item;
}

void display(int queue[])
{
    int i;
    printf("\n");
    if (front > rear)
    {
        for (i = front; i < size; i++)
        {
```

```
            printf("%d ", queue[i]);
        }
        for (i = 0; i <= rear; i++)
            printf("%d ", queue[i]);
    }
    else
    {
        for (i = front; i <= rear; i++)
            printf("%d ", queue[i]);
    }
}

void deleteq(int queue[])
{
    if (front ==  - 1)
    {
        printf("Queue is empty ");
    }
    else if (front == rear)
    {
        printf("\n %d deleted", queue[front]);
        front =  - 1;
        rear =  - 1;
```

```
    }
    else
    {
        printf("\n %d deleted", queue[front]);
        front++;
    }
}
```

OUTPUT

```
Circular Queue
1.Insert
2.Delete
3.Display
4.Exit
Enter choice 0-3?: 1
Enter number :4
Circular Queue
1.Insert
2.Delete
3.Display
4.Exit
Enter choice 0-3?: 1
Enter number :5
1.Insert
```

2.Delete

3.Display

4.Exit

Enter choice 0-3?: 1

Enter number :6

1.Insert

2.Delete

3.Display

4.Exit

Enter choice 0-3?: 2

6 is deleted

Circular Queue

1.Insert

2.Delete

3.Display

4.Exit

Enter choice 0-3?: 3

PROGRAM N0:5

AIM: Implement a Stack using linked list and its operations.

ALGORITHM

1.Start

2. Create a new node with given data.

3. Make the new node points to the head node.

4. Now make the new node as the head node.

5.Check whether the head node is NULL

6..if head == NULL

The stack is empty. we can't pop the element.

7.Otherwise,

Move to head node to the next node. head = head->next;

Free the head node's memory.

8.Stop

PROGRAM

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

// A linked list node
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

/* Given a reference (pointer to pointer) to the head of a
   list and an int, inserts a new node on the front of the
   list. */
void push(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    /* 2. put in the data  */
    new_node->data = new_data;
```

```
    /* 3. Make next of new node as head and previous as NULL
     */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* 4. change prev of head node to new node */
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    /* 5. move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Given a node as prev_node, insert a new node after the
 * given node */
void insertAfter(struct Node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL) {
        printf("the given previous node cannot be NULL");
        return;
    }
```

```
    /* 2. allocate new node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    /* 3. put in the data  */
    new_node->data = new_data;

    /* 4. Make next of new node as next of prev_node */
    new_node->next = prev_node->next;

    /* 5. Make the next of prev_node as new_node */
    prev_node->next = new_node;

    /* 6. Make prev_node as previous of new_node */
    new_node->prev = prev_node;

    /* 7. Change previous of new_node's next node */
    if (new_node->next != NULL)
        new_node->next->prev = new_node;
}

/* Given a reference (pointer to pointer) to the head
   of a DLL and an int, appends a new node at the end  */
```

```
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    struct Node* last = *head_ref; /* used in step 5*/

    /* 2. put in the data  */
    new_node->data = new_data;

    /* 3. This new node is going to be the last node, so
          make next of it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new
          node as head */
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }
```

```
    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;

    /* 7. Make last node as previous of new node */
    new_node->prev = last;

    return;
}

// This function prints contents of linked list starting
// from the given node
void printList(struct Node* node)
{
    struct Node* last;
    printf("\nTraversal in forward direction \n");
    while (node != NULL) {
        printf(" %d ", node->data);
        last = node;
        node = node->next;
```

```
    }

    printf("\nTraversal in reverse direction \n");
    while (last != NULL) {
        printf(" %d ", last->data);
        last = last->prev;
    }
}

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    // Insert 6.  So linked list becomes 6->NULL
    append(&head, 6);

    // Insert 7 at the beginning. So linked list becomes
    // 7->6->NULL
    push(&head, 7);

    // Insert 1 at the beginning. So linked list becomes
```

```
    // 1->7->6->NULL
    push(&head, 1);

    // Insert 4 at the end. So linked list becomes
    // 1->7->6->4->NULL
    append(&head, 4);

    // Insert 8, after 7. So linked list becomes
    // 1->7->8->6->4->NULL
    insertAfter(head->next, 8);

    printf("Created DLL is: ");
    printList(head);

    getchar();
    return 0;
}
```

OUTPUT

```
Created DLL is
Traversal in forward direction
1 7 8 6 4
Traversal in reverse direction
4 6 8 7 1
```

PROGRAM N0: 6

AIM: Implement a Set Data Structure and set operations (Union, Intersection and Difference) using Bit String.

ALGORITHM

1. Start
2. Read the no: of elements in first set
3. Read the elements of first set
4. Read the no: of elements in second set
5. Read the elements of second set
6. Check k=0 display resultant set is null
7. Otherwise display element of resultant set
8. Stop

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
void Union(int set1[10],int set2[10],int m,int n);
void Intersection(int set1[10],int set2[10],int m,int n);
void main()
{
 int a[10],b[10],m,n,i,j;
 int ch;
printf("\nEnter the number of elements in first set:");
 scanf("%d",&m);
 printf("\nEnter the elements:");
 for(i=0;i<m;i++)
 {
  scanf("%d",&a[i]);
 }
 printf("\nElement of First set:");
 for(i=0;i<m;i++)
 {
  printf("%d",a[i]);
```

```
}
printf("\nEnter th//e number of elements in second set:");
scanf("%d",&n);
printf("\nEnter the elements:n");
for(i=0;i<n;i++)
{
 scanf("%d",&b[i]);
}
printf("\nElement of second set");
for(i=0;i<n;i++)
{
 printf("%d",b[i]);
}
for(;;)
{
 printf("\n Menunn\n 1.Unionn2.Intersection");
 printf("\n     3.exit");
 printf("\n Enter your choice:");
 scanf("%d",&ch);
 switch(ch) {
 case 1:
```

```
   Union(a,b,m,n);
   break;
  case 2:
   Intersection(a,b,m,n);
   break;
  case 3:
   exit(0);
  }
}
}

void Union(int a[10],int b[10],int m,int n)
{
 int c[20],i,j,k=0,flag=0;
 for(i=0;i<m;i++)
 {
  c[k]=a[i];
  k++;
 }
 for(i=0;i<n;i++)
 {
```

```
 flag=0;
 for(j=0;j<m;j++)
 {
  if(b[i]==c[j])
  {
   flag=1;
   break;
  }
 }
 if(flag==0)
 {
  c[k]=b[i];
  k++;
 }
}
printf("\nElement of resultant set\n");
for(i=0;i<k;i++)
{
 printf("t%d",c[i]);
}
}
```

```
void Intersection(int a[10],int b[10],int m,int n)
{
 int c[20],i,j,k=0,flag=0;
 for(i=0;i<m;i++)
 {
  flag=0;
  for(j=0;j<n;j++)
  {
   if(a[i]==b[j])
   {
    flag=1;
    break;
   }
  }
  if(flag==1)
  {
   c[k]=a[i];
   k++;
  }
 }
 if(k==0)
```

```
{
 printf("\n Resultant set is null set!");
}else{
 printf("\n Element of resultant set");
 for(i=0;i<k;i++)
 {
  printf("\t%d",c[i]);
 }  }}
```

OUTPUT

```
Enter the number of elements in first set:  3
Enter the Elements: 1 2 3
Enter the number of elements in second set : 3
Enter the Elements:4 5 3
Menu
1. Union 2.Intersection 3.exit
Enter your choice: 1
Element of resultant set  1 2 3 4 5
Menu
1.Union 2.Intersection 3.exit
Enter your choice: 2
Element of resultant set  :3
```

PROGRAM NO: 7

AIM: Implement B Trees and its operations.

ALGORITHM

1. Start

2. If the tree is empty, allocate a root node and insert the key.
3.Update the allowed number of keys in the node.
4.Search the appropriate node for insertion.
5.If the node is full, follow the steps below.
6.Insert the elements in increasing order.
7.Now, there are elements greater than its limit. So, split at the median.
8.Push the median key upwards and make the left keys as a left child and the right keys as a right child.
9.If the node is not full, follow the steps below.
10 Insert the node in increasing order.

11.Stop

PROGRAM

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

#define MAX 3
#define MIN 2

struct BTreeNode {
  int val[MAX + 1], count;
  struct BTreeNode *link[MAX + 1];
};

struct BTreeNode *root;

// Create a node
struct BTreeNode *createNode(int val, struct BTreeNode *child) {
  struct BTreeNode *newNode;
  newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
  newNode->val[1] = val;
  newNode->count = 1;
  newNode->link[0] = root;
  newNode->link[1] = child;
  return newNode;
}

// Insert node
void insertNode(int val, int pos, struct BTreeNode *node,
        struct 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++;
}

// Split node
void splitNode(int val, int *pval, int pos, struct BTreeNode *node,
        struct BTreeNode *child, struct BTreeNode **newNode) {
  int median, j;

  if (pos > MIN)
    median = MIN + 1;
  else
    median = MIN;

  *newNode = (struct BTreeNode *)malloc(sizeof(struct 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--;
}
```

```
// Set the value
int setValue(int val, int *pval,
          struct BTreeNode *node, struct 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("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;
}

// Insert the value
void insert(int val) {
```

```
  int flag, i;
  struct BTreeNode *child;

  flag = setValue(val, &i, root, &child);
  if (flag)
    root = createNode(i, child);
}

// Search node
void search(int val, int *pos, struct 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("%d is found", val);
      return;
    }
  }
  search(val, pos, myNode->link[*pos]);

  return;
}

// Traverse then nodes
void traversal(struct BTreeNode *myNode) {
  int i;
  if (myNode) {
    for (i = 0; i < myNode->count; i++) {
```

```
      traversal(myNode->link[i]);
      printf("%d ", myNode->val[i + 1]);
    }
    traversal(myNode->link[i]);
  }
}

int main() {
  int val, ch;

  insert(8);
  insert(9);
  insert(10);
  insert(11);
  insert(15);
  insert(16);
  insert(17);
  insert(18);
  insert(20);
  insert(23);

  traversal(root);

  printf("\n");
  search(11, &ch, root);
}
```

OUTPUT

```
 8 9 10 11 15 16 7 18 20 23 11
```

PROGRAM NO: 8

AIM: Implement Graph Traversal techniques BFS.

ALGORITHM

1:Start

2:Enter the number of

vertices Step

3:Read the graph in matrix

form

4:Enter the starting vertex

5:Repeat the step4&5 until the queue is empty

6:Take the front item of the queue and add it to the visited list

7:Create a list of that vertex adjacent node.Add the one which

aren't in the Visited list to the back of the queue

8:Stop

PROGRAM

```
#include<stdio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v) {
for (i=1;i<=n;i++)
if(a[v][i] && !visited[i])
q[++r]=i;
if(f<=r) {
visited[q[f]]=1;
bfs(q[f++]);
}
}
void main() {
int v;

printf("\n Enter the number of vertices:");
scanf("%d",&n);
for (i=1;i<=n;i++) {
q[i]=0;
visited[i]=0;
}
printf("\n Enter graph data in matrix form:\n");
for (i=1;i<=n;i++)
for (j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n");
for (i=1;i<=n;i++)
if(visited[i])
printf("%d\t",i); else
printf("\n Bfs is not possible");
getch();
}
```

OUTPUT

```
Enter the number of vertices :3
Enter graph data in matrix form
1 2 3
4 5 6
7 8 9
Enter Starting vertex:2
The node which are reachable
1  2  3
```

PROGRAM NO :9

AIM: Implement Graph Traversal techniques DFS.

ALGORITHM

1. Start
2. Create a structure node with element vertex

   and *next

3.Read number of vertices and assign to v

4.Initialize adj[] with a null

5 .Read edges and insert them in adj[]

6.Represent the edges into adjacency list

7. Acquire only for the new node

8. Insert the node in the new node

9. Go to end of the linked list

10. Keep an array visited[] to keep track of the visited vertices

11. if visited[i]=1 represent that vertex is has been visited before and the DFS function for some already visited node need not be called

12. Repeat the step until all the vertex are visited

13.Print the vertex

14.Stop

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
/*        ADJACENCY MATRIX                    */
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
   int j;
   visited[i]=1;
   printf(" %d->",i+1);
   for(j=0;j<V;j++)
   {
      if(G[i][j]==1&&visited[j]==0)
         DFS(j);
   }
}
int main()
{
   int i,j,v1,v2;
   printf("\t\t\tGraphs\n");
   printf("Enter the no of edges:");
   scanf("%d",&E);
   printf("Enter the no of vertices:");
   scanf("%d",&V);
   for(i=0;i<V;i++)
   {
      for(j=0;j<V;j++)
         G[i][j]=0;
   }
   /*    creating edges :P    */
   for(i=0;i<E;i++)
   {
      printf("Enter the edges (format: V1 V2) : ");
      scanf("%d%d",&v1,&v2);
      G[v1-1][v2-1]=1;

   }
```

```
    for(i=0;i<V;i++)
    {
       for(j=0;j<V;j++)
          printf(" %d ",G[i][j]);
       printf("\n");
    }
    printf("Enter the source: ");
    scanf("%d",&source);
       DFS(source-1);
    return 0;
}
```

OUTPUT

```
Enter the number of edges: 5
Enter the number of Vertices:4
Enter the edges(format :V1 V2: 1 3
Enter the edges(format :V1 V2: 4  5
Enter the edges(format :V1 V2: 6 7
Enter the edges(format :V1 V2: 8 9
Enter the edges(format :V1 V2: 10 11
0 0 1 0
0 0 0 0
0 0 0 0
0 0 0 0
Enter Source vertices :2
2 ⟶
```

PROGRAM NO:10

AIM: Implement Prim's Algorithm for finding the minimum cost spanning tree using C

ALGORITHM

1.Begin
2.Create edge list of given graph, with their weights.
3.Draw all nodes to create skeleton for spanning tree.
4.Select an edge with lowest weight and add it to skeleton and delete edge from edge list.
5.Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.
6.Repeat step 5 until n-1 edges are added.
7.Return.

PROGRAM

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

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
        int i,j,total_cost;
        printf("Enter no. of vertices:");
        scanf("%d",&n);

        printf("\nEnter the adjacency matrix:\n");

        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&G[i][j]);

        total_cost=prims();
        printf("\nspanning tree matrix:\n");
```

```
	for(i=0;i<n;i++)
	{
		printf("\n");
		for(j=0;j<n;j++)
			printf("%d\t",spanning[i][j]);
	}

	printf("\n\nTotal cost of spanning tree=%d",total_cost);
	return 0;
}

int prims()
{
	int cost[MAX][MAX];
	int u,v,min_distance,distance[MAX],from[MAX];
	int visited[MAX],no_of_edges,i,min_cost,j;

	//create cost[][] matrix,spanning[][]
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
		{
			if(G[i][j]==0)
				cost[i][j]=infinity;
			else
				cost[i][j]=G[i][j];
				spanning[i][j]=0;
		}
```

```
//initialise visited[],distance[] and from[]
distance[0]=0;
visited[0]=1;

for(i=1;i<n;i++)
{
        distance[i]=cost[0][i];
        from[i]=0;
        visited[i]=0;
}

min_cost=0;         //cost of spanning tree
no_of_edges=n-1;          //no. of edges to be added

while(no_of_edges>0)
{
        //find the vertex at minimum distance from the tree
        min_distance=infinity;
        for(i=1;i<n;i++)
                if(visited[i]==0&&distance[i]<min_distance)
                {
                        v=i;
                        min_distance=distance[i];
                }

        u=from[v];
```

```
            //insert the edge in spanning tree
            spanning[u][v]=distance[v];
            spanning[v][u]=distance[v];
            no_of_edges--;
            visited[v]=1;

            //updated the distance[] array
            for(i=1;i<n;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);
}
```

OUTPUT

Enter number of vertices:2

Enter the adjacency matrix

1 2 3

4 5 6

7 8 9

Spanning tree matrix

0 2 3

2 0 0

3 0 0

Total cost of spanning tree: 5

EXPERIMENT NO:11

AIM: .Write a C program to implement Kruskal's algorithm using the Disjoint set data structure.

ALGORITHM

1.Start

2.Read the elements of the

graph

3.if graph[i][j]!=0 then

4.Create a edge list of given graph with their weight

5.Sort the edge list according to their weights in

ascending order

6.Pick up the edge at the top of the edge list

7.Remove this edgr from edge list

8.Connect the edge .If by connecting the vertices a cycle is created then

discard the edge

9.Repeat the step 6 to 8 list of edge is visited

10.Print the minimum

cost

11.Stop

PROGRAM

```
#include <stdio.h>

#define MAX 30

typedef struct edge {
  int u, v, w;
} edge;

typedef struct edge_list {
  edge data[MAX];
  int n;
} edge_list;

edge_list elist;

int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();
```

```
// Applying Krushkal Algo
void kruskalAlgo() {
  int belongs[MAX], i, j, cno1, cno2;
  elist.n = 0;

  for (i = 1; i < n; i++)
    for (j = 0; j < i; j++) {
      if (Graph[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = Graph[i][j];
        elist.n++;
      }
    }

  sort();

  for (i = 0; i < n; i++)
    belongs[i] = i;

  spanlist.n = 0;

  for (i = 0; i < elist.n; i++) {
    cno1 = find(belongs, elist.data[i].u);
    cno2 = find(belongs, elist.data[i].v);

    if (cno1 != cno2) {
```

```
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
    }
  }
}

int find(int belongs[], int vertexno) {
  return (belongs[vertexno]);
}

void applyUnion(int belongs[], int c1, int c2) {
  int i;

  for (i = 0; i < n; i++)
    if (belongs[i] == c2)
      belongs[i] = c1;
}

// Sorting algo
void sort() {
  int i, j;
  edge temp;

  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w) {
```

```
        temp = elist.data[j];
        elist.data[j] = elist.data[j + 1];
        elist.data[j + 1] = temp;
       }
}

// Printing the result
void print() {
  int i, cost = 0;

  for (i = 0; i < spanlist.n; i++) {
    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v,
spanlist.data[i].w);
    cost = cost + spanlist.data[i].w;
  }

  printf("\nSpanning tree cost: %d", cost);
}

int main() {
  int i, j, total_cost;

  n = 6;

  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
```

```
Graph[0][3] = 0;
Graph[0][4] = 0;
Graph[0][5] = 0;
Graph[0][6] = 0;

Graph[1][0] = 4;
Graph[1][1] = 0;
Graph[1][2] = 2;
Graph[1][3] = 0;
Graph[1][4] = 0;
Graph[1][5] = 0;
Graph[1][6] = 0;

Graph[2][0] = 4;
Graph[2][1] = 2;
Graph[2][2] = 0;
Graph[2][3] = 3;
Graph[2][4] = 4;
Graph[2][5] = 0;
Graph[2][6] = 0;

Graph[3][0] = 0;
Graph[3][1] = 0;
Graph[3][2] = 3;
Graph[3][3] = 0;
Graph[3][4] = 3;
Graph[3][5] = 0;
```

```
        Graph[3][6] = 0;

        Graph[4][0] = 0;
        Graph[4][1] = 0;
        Graph[4][2] = 4;
        Graph[4][3] = 3;
        Graph[4][4] = 0;
        Graph[4][5] = 0;
        Graph[4][6] = 0;

        Graph[5][0] = 0;
        Graph[5][1] = 0;
        Graph[5][2] = 2;
        Graph[5][3] = 0;
        Graph[5][4] = 3;
        Graph[5][5] = 0;
        Graph[5][6] = 0;
kruskalAlgo();
        print();
    }
```

OUTPUT

```
2-1   :2
5-2   : 2
3-2   :3
4-3    : 3
1-0   :4
Spanning tree cost :14
```