# Data Structure –II
# SLIP

****************************SLIP 1*******************************

Q 1. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

```
 #include<stdio.h>

#include<stdlib.h>

void create(int m[10][10],int n)

{

   int i,j;

   char ans;

   for(i=0;i<n;i++)

   for(j=0;j<n;j++)

   {

      m[i][i]=0;

      if(i!=j)

      {

         printf("\n Is there an edge between %d and %d : ",i+1,j+1);

         scanf("%d",&m[i][j]);

      }

   }

}

void display(int m[10][10],int n)

{

   int i,j;

   printf("\n \t Adjacency matrix is : \n");
```

```
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        printf("%5d",m[i][j]);
        printf("\n");
    }
}
void main()
{

    int m[10][10],n;
    printf("\n \t Enter vertices : ");
    scanf("%d",&n);
    create(m,n);
    display(m,n);
}
```

Q 2. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, preorder. Write a menu driven program that performs the above operations

```
#include<stdio.h>
#include<stdlib.h>
//#include"btree14.h"
typedef struct bnode
{
    int data;
    struct bnode *left,*right;
```

```
}bnode;
    bnode *insert(bnode *,int);
    bnode *create();
    void preorder(bnode *T);
    bnode *create()
    {
            int n,x,i;
            bnode *root;
            root=NULL;
            printf("\nEnter no. of nodes :");
            scanf("%d",&n);
            printf("\nEnter tree values :");
            for(i=0;i<n;i++)
            {
                    scanf("%d",&x);
                    root=insert(root,x);
            }
            return(root);
}
void preorder(bnode *T)
{
    if(T!=NULL)
    {
        printf("%d\t",T->data);
        preorder(T->left);
        preorder(T->right);
    }
}
bnode *insert(bnode *T,int x)
{
    bnode *temp;
    if(T==NULL)
    {
            temp=(bnode*)malloc(sizeof(bnode));
            temp->data=x;
            temp->left=NULL;
            temp->right=NULL;
            return(temp);
        }
        if(x>T->data)
        {
                T->right=insert(T->right,x);
                return(T);
        }
        else
                if(x<T->data)
                {
                        T->left=insert(T->left,x);
                        return(T);
                }
                return(T);
```

```
}




void main()
{
      bnode *root=NULL,*p;
      int x,ch;
    do
    {
        printf("\n\t1.create");
        printf("\n\t2.insert");
        printf("\n\t3.Display");
        printf("\n\t4.Exit)");
        printf("\n enter your choice:-->");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: root=create();break;
            case 2: printf("\n enter the key to thr inserted:-->");
                scanf("%d",&x);
                root=insert(root,x);
                  break;
            case 3: preorder(root);
                break;
```

```
                    case 4: exit(0);
                }
        }while(ch!=4);
}
```

Q 2. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm

```
#include<stdio.h>
int main()
{
int adj_mat[10][10],visited[10]= {0},i,j,n,no_e=1,min,a,b,min_cost=0;
printf("Enter number of nodes ");
scanf("%d",&n);
printf("Enter adj_mat in form of adjacency matrix\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
        {
                scanf("%d",&adj_mat[i][j]);
                if(adj_mat[i][j]==0)
                                adj_mat[i][j]=1000;
                }
        }
        visited[1]=1; // visited first node
        while(no_e<n)
        {
                min=1000;
```

```
                              // in each cycle find minimum adj_mat
                    for(i=1;i<=n;i++)
                    {
                              for(j=1;j<=n;j++)
{
if(adj_mat[i][j]<min)
{
if(visited[i]!=0)
{
min=adj_mat[i][j];
a=i;
b=j;
}
}
}
}
 //if node is not visited
if(visited[b]==0)
{
printf("\n%d to %d adj_mat=%d",a,b,min);
min_cost=min_cost+min;
no_e++;
}
visited[b]=1;
adj_mat[a][b]=adj_mat[b][a]=1000;
}
```

```
printf("\nminimum weight is %d",min_cost);
return 0;
}
```

******************************SLIP 2*******************************

Q1. Write a C program for the implementation of Topological sorting

```
#include<stdio.h>
int main()
{
        int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;
        printf("Enter the no of vertices:\n");
        scanf("%d",&n);
        printf("Enter the adjacency matrix:\n");
        for(i=0;i<n;i++)
        {
                printf("Enter row %d\n",i+1);
                for(j=0;j<n;j++)
                scanf("%d",&a[i][j]);
        }
        for(i=0;i<n;i++)
        {
                indeg[i]=0;
                flag[i]=0;
         }

 for(i=0;i<n;i++)
 for(j=0;j<n;j++)
 indeg[i]=indeg[i]+a[j][i];
printf("\nThe topological order is:");
while(count<n)
{
        for(k=0;k<n;k++)
        {
                if((indeg[k]==0) && (flag[k]==0))
                {
                    printf("%d ",(k+1));
                    flag [k]=1;
                }
                for(i=0;i<n;i++)
                {
                        if(a[i][k]==1)
                        indeg[k]--;
                }
        }
        count++;
}
return 0;
}
```

Q2. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

Repeat SLIP 1 Q1.

Q3. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Depth First Search (DFS) traversal.

```
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;
}
```

*****************************SLIP  3********************************

Q 1. Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm

Repeat SLIP1 Q2

Q2 .Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

Repeat SLIP1 Q1

Q2. Write a C program for the implementation of Floyd Warshall’s algorithm for finding all pairs shortest path using adjacency cost matrix.

```
#include<stdio.h>
void floyd(int a[4][4], int n)
{
        for(int k=0;k<n;k++)
        {
                for(int i=0;i<n;i++)
                {
                        for(int j=0;j<n;j++)
                        {
                                if(a[i][j]>a[i][k]+a[k][j])
                                {
                                        a[i][j]=a[i][k]+a[k][j];
                                }
                    }
            }
        }
        printf("All Pairs Shortest Path is :\n");
                for(int i=0;i<n;i++)
            {
                for(int j=0;j<n;j++)
                {
                        printf("%d ",a[i][j]);
                        }
                printf("\n");
                }
}
int main()
{
        int cost[4][4] = {{0, 3, 999, 5}, {2, 0,999,4}, {999,1, 0, 999}, {999, 999,2,
0}};
        int n = 4;

        floyd(cost,n);
}
```

******************************SLIP 4**********************************

Q 1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        int vertex;
        struct node *next;
        } NODE;
        NODE *head[10];
        void createlist(int n)
        {
        int i, j, x;
        NODE *temp, *newnode;
        for(i=0; i<n; i++)
        {
                head[i]=NULL;
                for(j=0; j<n; j++)
                {
                printf("is there any edge between %d and %d (1/0): ", i+1, j+1);
                scanf("%d", &x);
                if (x==1)
                {
                        newnode=(NODE*)malloc(sizeof(NODE));
                        newnode->vertex=j+1;
                        newnode->next=NULL;
```

```
            if (head[i]==NULL)
                head[i]=temp=newnode;
            else
            {
                temp->next=newnode;
                temp=newnode;
            }
        }
    }
    }
}


void displaylist(int n)
{
    NODE *temp;
    printf("\nThe Adjacency list is: \n");
    for(int i=0; i<n; i++)
    {
        printf("\nv%d -> ", i+1);
        temp=head[i];
    while(temp)
    {
        printf("v%d -> ", temp->vertex);
        temp=temp->next;
    }
```

```
                    printf("NULL");
          }
                    printf("\n");
}
void main()
{
          int n;
          printf("Enter how many vertices: ");
          scanf("%d", &n);
          createlist(n);
          displaylist(n);
}
```

2. Write a program which uses binary search tree library and counts the total nodes and total leaf nodes in the tree. int count Leaf(T) – returns the total number of leaf nodes from BST.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *LC,*RC;
   int data;
};
struct node *create(struct node *root,int item)
{
   if(root==NULL)
   {
        root=(struct node *)malloc(sizeof(struct node));
        root->LC=root->RC=NULL;
        root->data=item;
        return root;
    }
    else
    {
        if(item<root->data)
            root->LC=create(root->LC,item);
            else if(item>root->data)
                 root->RC=create(root->RC,item);
            else
                printf("Invalid item");
        return root;
    }
}
int totalnodes(struct node *root)
{
    if(root==NULL)
        return 0;
    else
        return(1+totalnodes(root->LC)+totalnodes(root->RC));
}
int count=0;
int leafnodes(struct node* newnode)
{
    if(newnode != NULL)
    {
        leafnodes(newnode->LC);
        if((newnode->LC == NULL) && (newnode->RC == NULL))
        {
            count++;
        }
        leafnodes(newnode->RC);
    }
    return count;

}
void main()
{
    struct node *root=NULL;
   int choice,i,n,item;
   printf("1.Create\n");
   printf("2.totalnodes\n");
   printf("3.countLeaf\n");
   printf("4.Exit\n");
   while(1)
    {
        printf("\nEnter your choice : \n");
        scanf("%d",&choice);
        switch(choice)
    {
      case 1: root=NULL;
          printf("Enter total nodes : ");
          scanf("%d",&n);
          for(i=1;i<=n;i++)
          {
            printf("Enter %d data for nodes : ",i);
            scanf("%d",&item);
            root=create(root,item);
          }
          break;
      case 2: printf("Number of nodes in a tree is %d",totalnodes(root));
          totalnodes(root);
          break;
      case 3: printf("Number of leaf nodes in a tree are %d",leafnodes(root));
          leafnodes(root);
          break;
      case 4: exit(0);
    }
  }
}
```

Q 2. Write a C program for the implementation of Topological sorting.

Repeat SLIP 2 Q1.

*****************************&#42;**SLIP 5**********************************

Q1 .Write a C program which uses Binary search tree library and displays nodes at each level, count of node at each level.

```
#include<stdio.h>

#include<stdlib.h>


struct treenode

{

        int data;

        struct treenode *leftchild, *rightchild;

};


struct Queue{

        struct treenode *node;

        struct Queue *next;

}*front=NULL,*rear=NULL; //initQ




int isEmpty()

{

        return front == NULL;

}


void add_Q(struct treenode *item)

{

        struct Queue *newnode;

        if(front == NULL)

        {

                front = (struct Queue *)malloc(sizeof(struct Queue));
        rear = front;
    }
    else{
        rear->next = (struct Queue *)malloc(sizeof(struct Queue));
        rear = rear->next;
    }
    rear->node = item;
    rear->next = NULL;
}

struct treenode * remove_Q()
{
    if(front!=NULL)
    {
        struct Queue *temp = front;
        struct treenode *returnnode = front->node;
        if(front == rear) rear = NULL;
        front = front->next;
        free(temp);
        return returnnode;
    }
    else return NULL;
}

void level_wise_traversal(struct treenode *root)
{
        struct treenode *current_node = NULL;


        //creating a delimiter variable to keep track of new level

        struct treenode *delimiter = (struct treenode *)malloc(sizeof(struct
treenode));

        delimiter->data = -1;

        delimiter->leftchild = NULL;

        delimiter->rightchild = NULL;


        if(root == NULL) return;


        add_Q(root);

        add_Q(delimiter);


        while(!(front->node == delimiter && front == rear))  //operating queue till
we are left with only delimiter in queue

   {

      current_node = remove_Q();

      if (current_node == delimiter)

      {

        add_Q(delimiter);  //new level

        printf("\n");

      }

      else

                {        //adding children of current node in the queue

        if(current_node->leftchild  !=NULL)
        add_Q(current_node->leftchild);

    if(current_node->rightchild!=NULL)
        add_Q(current_node->rightchild);

    printf(" %d",current_node->data);
   }
 }
}

struct treenode * addnode(struct treenode *root, int data)
{
    struct treenode *s, *temp1, *temp2;

    s = (struct treenode *)malloc(sizeof(struct treenode));
    s->data = data;
    s->leftchild = s->rightchild = NULL;

    if(root == NULL)
        root = s;
    else
    {
        temp1 = root;
        while(temp1 != NULL)
        {
            temp2 = temp1;
            if(data <= temp1->data)
                temp1 = temp1->leftchild;
            else
                temp1 = temp1->rightchild;
        }
        if(data<=temp2->data)
            temp2->leftchild = s;
        else
            temp2->rightchild = s;
    }
    return root;
}

void inorder(struct treenode *t)
{
    if(t!=NULL)
    {
        inorder(t->leftchild);
        printf("%d  ",t->data);
        inorder(t->rightchild);
    }
}

int main(void)
{
    int n,data,i,choice,height;
    struct treenode *root = NULL;

    printf("\nHow many nodes in tree?  ");
    scanf("%d",&n);

    for(i=0; i<n; i++)
    {
        printf("\nEnter node %d:  ",i+1);
        scanf("%d",&data);
        root = addnode(root,data);
    }

    printf("\n\nLevel order traversal: \n");
    level_wise_traversal(root);
}
```

Q2. Write a program to sort n randomly generated elements using Heapsort method.

```
#include<stdio.h>
void main()
{
   int heap[10],n,i,j,c,root,temp;
   printf("\nEnter no of total elements : ");
   scanf("%d",&n);
   printf("\nEnter the numbers : ");
```

```
for(i=0;i<n;i++)
scanf("%d",&heap[i]);
for(i=1;i<n;i++)
{
   c=i;
   do
   {
      root=(c-1)/2;
      if(heap[root]<heap[c])
      {
         temp=heap[root];
         heap[root]=heap[c];
         heap[c]=temp;
      }
      c=root;
   }while(c!=0);
}
printf("Heap array is : ");
for(i=0;i<n;i++)
   printf("%d\t",heap[i]);
for(j=n-1;j>=0;j--)
{
   temp=heap[0];
   heap[0]=heap[j];
   heap[j]=temp;
   root=0;
```

```
        do
        {
            c=2*root+1;
            if((heap[c]<heap[c+1]) && c<j-1)
                c++;
            if(heap[root]<heap[c] && c<j)
            {
                temp=heap[root];
                heap[root]=heap[c];
                heap[c]=temp;
            }
            root=c;
        }while(c<j);
    }
    printf("\nThe sorted array is  : ");
    for(i=0;i<n;i++)
        printf("\t%d",heap[i]);
}
```

Q3 Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal.

```
#include<stdio.h>
#include<stdlib.h>
struct q
{
    int data[20];
    int front,rear;
}q1;
void add(int n)
{
    q1.rear++;
    q1.data[q1.rear]=n;
}
int del()
{
    q1.front++;
    return q1.data[q1.front];
}
void initq()
{
    q1.front=q1.rear=-1;
}
int emptyq()
{
    return (q1.rear==q1.front);
}
void create(int m[10][10],int n)
{
    int i,j;
    char ans;
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    {
        m[i][i]=0;
        if(i!=j)
        {
            printf("\n Is there an edge between %d and %d : ",i+1,j+1);
            scanf("%d",&m[i][j]);
        }
    }
}
void display(int m[10][10],int n)
{
    int i,j;
    printf("\n \t Adjacency matrix is : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        printf("%5d",m[i][j]);
        printf("\n");
    }
}

void bfs(int m[10][10],int n)
{
    int i,j,v,w;
    int visited[20];
    initq();
    for(i=0;i<n;i++)
    visited[i]=0;
    printf("\n \t The Bfs is: \n");
    v=0;
    visited[v]=1;
    add(v);
    while(! emptyq())
    {
        v=del();
        printf("\n v%d ",v+1);
        printf("\n");
        for(w=0;w<n;w++)
        if((m[v][w]==1) &&(visited[w]==0))
        {
            add(w);
            visited[w]=1;
        }
    }
}
void main()
{

    int m[10][10],n;
    printf("\n \t Enter vertices : ");
    scanf("%d",&n);
    create(m,n);
```

```
   display(m,n);

   bfs(m,n);

}
```

***************************SLIP 6******************************

Q1 .Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm.

**Repeat SLIP 1 Q2**

Q 2. Write a C program for the implementation of Dijkstra’s shortest path algorithm for finding shortest path from a given source vertex using adjacency cost matrix.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
   int G[MAX][MAX],i,j,n,u;
   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]);
   printf("\nEnter the starting node:");
   scanf("%d",&u);
   dijkstra(G,n,u);
   return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
   int cost[MAX][MAX],distance[MAX],pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,j;
```

```
    //give
//pred[] stores the predecessor of each node count givesththe number of nodes seen
so far
//matrix//create the cost matrix
    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];
                    for(i=0;i<n;i++)
                    {
                        distance[i]=cost[startnode][i];
                        pred[i]=startnode;
                        visited[i]=0;
                    }
                    distance[startnode]=0;
                    visited[startnode]=1;
                    count=1;
                    while(count<n-1)
                    {
                            mindistance=INFINITY;
                                    //distance//nextnode gives the node at minimum distance
                            for(i=0;i<n;i++)
                                    if(distance[i]<mindistance&&!visited[i])
                                    {
                                        mindistance=distance[i];
                                        nextnode=i;
                                    }

                                    visited[nextnode]=1;
                                    for(i=0;i<n;i++)
                                            if(!visited[i])
                                                    if(mindistance+cost[nextnode][i]<distance[i])
                                                    {
                                                        distance[i]=mindistance+cost[nextnode][i];
                                                        pred[i]=nextnode;
                                                    }
                                                    count++;
                                            }
                                        //print the path and distance of each node
                                        for(i=0;i<n;i++)
                                            if(i!=startnode)
```

```
       {
          printf("\nDistance of node%d=%d",i,distance[i]);
          printf("\nPath=%d",i);
          j=i;
          do
          {
            j=pred[j];
            printf("<-%d",j);
            }while(j!=startnode);
          }
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

**Repeat SLIP 1 Q1**

*************************SLIP 7***********************************

Write a C program for the implementation of Floyd Warshall’s algorithm for finding all pairs shortest path using adjacency cost matrix.

**Repeat SLIP 3 Q2**

Q2. Write a program to sort n randomly generated elements using Heap sort method.

**Repeat SLIP 5 Q2**

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level, and total levels in the tree.

**Repeat SLIP 5 Q1**

*************************SLIP 8***********************************

Q 1. Write a program to sort n randomly generated elements using Heapsort method.

**Repeat SLIP 5 Q2**

Q 2. Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm.

**Repeat SLIP 1 Q2**

Q2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement functions to print indegree of all vertices of graph.

```
#include<stdio.h>
#define MAX 10
int adj[MAX][MAX];
int n;
int main()
{
        int max_edges, i, j, origin, destin, ind;
        int graph_type;
        printf("\n1. Undirected Graph\n2. Directed Graph\n");
        printf("Enter your choice: ");
        scanf("%d", &graph_type);
        printf("\nEnter number of vertices: ");
        scanf("%d", &n);
        if(graph_type==1)
        {
                max_edges = n*(n-1)/2;
        }
```

```
else
{
    max_edges = n*(n-1);
}

for(i=1; i<=max_edges; i++)
{
    printf("\nEnter edge[%d](-1 -1 to quit): ", i);
    scanf("%d %d", &origin, &destin);
    if((origin == -1)&&(destin == -1))
    {
        break;
    }
    if(origin >= n || destin >= n || origin < 0 || destin < 0)
    {
        printf("\nInvalid vertex!\n");
        i--;
    }
    else
    {
        adj[origin][destin] = 1;
        if(graph_type == 1)
        {
            adj[destin][origin] = 1;
        }
    }
```

```
        }


        printf("\nThe adjacency matrix is:\n");
        for(i=0; i<=n-1; i++)
        {
                for(j=0; j<=n-1; j++)
                {
                        printf("%4d", adj[i][j]);
                }
                printf("\n");
        }


        printf("\nIndegree, ");
        for(i=0; i<n; i++)
        {
                for(j=0, ind=0; j<n; j++)
                {
                        if(adj[j][i] == 1)
                                ind++;
                }
                printf("\nv%d %5d \n",i+1,ind);
        }
        }
```

****************************SLIP 9*************************************

Q1.Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.

**Repeat SLIP 4 Q1.**

Q2. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, postorder. Write a menu driven program that performs the above operations.

**Repeat SLIP 1 Q2.**

Q2.Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Depth First Search (DFS) traversal.

**Repeat SLIP 2 Q2.**

***************************SLIP 10************************************

Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, inorder. Write a menu driven program that performs the above operations.

```
#include<stdio.h>

#include<stdlib.h>

//#include"btree14.h"

typedef struct bnode

{

   int data;

   struct bnode *left,*right;

}bnode;

   bnode *insert(bnode *,int);

   bnode *create();

   void inorder(bnode *T);

   bnode *create()

   {
        int n,x,i;
        bnode *root;
        root=NULL;
        printf("\nEnter no. of nodes :");
        scanf("%d",&n);
        printf("\nEnter tree values :");
        for(i=0;i<n;i++)
        {
              scanf("%d",&x);
              root=insert(root,x);
        }
        return(root);
}
void inorder(bnode *T)
{
   if(T!=NULL)
   {

      inorder(T->left);
      printf("%d\t",T->data);
      inorder(T->right);
   }
}
bnode *insert(bnode *T,int x)
{
   bnode *temp;
```

```
  if(T==NULL)
  {
      temp=(bnode*)malloc(sizeof(bnode));
      temp->data=x;
      temp->left=NULL;
      temp->right=NULL;
      return(temp);
    }
    if(x>T->data)
    {
        T->right=insert(T->right,x);
        return(T);
    }
    else
        if(x<T->data)
        {
            T->left=insert(T->left,x);
            return(T);
        }
        return(T);
}

void main()
{
      bnode *root=NULL,*p;
      int x,ch;
```

```
        do
        {
                printf("\n\t1.create");
                printf("\n\t2.insert");
                printf("\n\t3.Inorder");
                printf("\n\t4.Exit)");
                printf("\n enter your choice:-->");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: root=create();break;
                        case 2: printf("\n Enter the Node to be  inserted:-->");
                                scanf("%d",&x);
                                root=insert(root,x);
                                    break;
                        case 3: inorder(root);
                                break;
                        case 4: exit(0);
                }
    }while(ch!=4);
}
```

Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.

**Repeat SLIP 4 Q1.**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal.

**Repeat SLIP 5 Q2.**

*************************SLIP 11**********************************

Q1. Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix.

**Repeat SLIP 3 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.

**Repeat SLIP 4 Q1**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Depth First Search (DFS) traversal.

Q 2. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.

**Repeat SLIP 2 Q1**

****************************SLIP 12*******************************

Q 1. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, preorder. Write a menu driven program that performs the above operations.

**Repeat SLIP 1 Q2**

Q 2. Write a C program for the implementation of Topological sorting.

**Repeat SLIP 2 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement functions to print indegree, outdegree and total degree of all vertices of graph.

```
#include<stdio.h>
#define MAX 10
int adj[MAX][MAX];
int n;
int main()
{
        int max_edges, i, j, origin, destin, ind, outd, total;
        int graph_type;
        printf("\n1. Undirected Graph\n2. Directed Graph\n");
        printf("Enter your choice: ");
        scanf("%d", &graph_type);
        printf("\nEnter number of vertices: ");
        scanf("%d", &n);
        if(graph_type==1)
        {
                max_edges = n*(n-1)/2;
        }
        else
        {
                max_edges = n*(n-1);
        }
        for(i=1; i<=max_edges; i++)
        {
```

```
        printf("\nEnter edge[%d](-1 -1 to quit): ", i);
        scanf("%d %d", &origin, &destin);
        if((origin == -1)&&(destin == -1))
        {
            break;
        }
        if(origin >= n || destin >= n || origin < 0 || destin < 0)
        {
            printf("\nInvalid vertex!\n");
            i--;
        }
        else
        {
            adj[origin][destin] = 1;
            if(graph_type == 1)
            {
                adj[destin][origin] = 1;
            }
        }
    }

    printf("\nThe adjacency matrix is:\n");
    for(i=0; i<=n-1; i++)
    {
        for(j=0; j<=n-1; j++)
        {
            printf("%4d", adj[i][j]);
        }
        printf("\n");
    }

    printf("\nIndegree, Outdegreee & Total: ");
    for(i=0; i<n; i++)
    {
        for(j=0, ind=0, outd=0; j<n; j++)
        {
            if(adj[i][j] == 1)
                outd++;
            if(adj[j][i] == 1)
                ind++;
            total = ind+outd;
        }
        printf("\nv%d %5d %5d %5d\n",i+1,ind,outd,total);
    }
    }
```

****************************SLIP 13*******************************

Write a C program for the Implementation of Kruskal’s Minimum spanning tree algorithm.

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

```
#include <stdlib.h>

int i, j, k, a, b, u, v, n, ne = 1;

int min, mincost = 0, cost[9][9], parent[9];

int find(int);

int uni(int, int);

void main()

{

        printf("Kruskal's algorithm in C\n");

        printf("========================\n");

        printf("Enter the no. of vertices:\n");

        scanf("%d", &n);

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

        for (i = 1; i <= n; i++)

        {

                for (j = 1; j <= n; j++)

                {

                        scanf("%d", &cost[i][j]);

                        if (cost[i][j] == 0)

                        cost[i][j] = 999;

                }

        }

        printf("The edges of Minimum Cost Spanning Tree are\n");

        while (ne < n)

        {
```

```
        for (i = 1, min = 999; i <= n; i++)
        {
            for (j = 1; j <= n; j++)
            {
                if (cost[i][j] < min)
                {
                    min = cost[i][j];
                    a = u = i;
                    b = v = j;
                }
        }
}
u  =  find(u);
v  =  find(v);
if (uni(u, v))
{
    printf("%d edge (%d,%d) =%d\n", ne++, a, b, min);
    mincost = min;
}
cost[a][b] = cost[b][a] = 999;
}
printf("\nMinimum cost = %d\n", mincost);
}
int find(int i)
```

```
{
        while (parent[i])
                i = parent[i];
                return i;
        }
int uni(int i, int j)
{
        if (i != j)
        {
                parent[j] = i;
                return 1;
}
return 0;
}
```

Q 2. Write a program which uses binary search tree library and counts the total nodes and total leaf nodes in the tree. int countLeaf(T) – returns the total number of leaf nodes from BST

**SLIP 4 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal.

**SLIP 5 Q2**

****************************SLIP 14*******************************

Q 1. Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix.

**Repeat SLIP 3 Q2**

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level, and total levels in the tree.

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

struct node
{
	struct node *lchild;
	int info;
	struct node *rchild;
};


struct node *insert(struct node *ptr, int ikey);
void display(struct node *ptr,int level);
int NodesAtLevel(struct node *ptr, int level) ;

int main()
{
	struct node *root=NULL,*root1=NULL,*ptr;
	int choice,k,item,level;

	while(1)
	{
		printf("\n");
		printf("1.Insert Tree \n");
		printf("2.Display Tree \n");
		printf("3.Number of Nodes \n");
		printf("4.Quit\n");
		printf("\nEnter your choice : ");
		scanf("%d",&choice);

		switch(choice)
		{

		case 1:
```

```
                        printf("\nEnter the key to be inserted : ");
                        scanf("%d",&k);
                        root = insert(root, k);
                        break;

        case 2:
            printf("\n");
            display(root,0);
            printf("\n");
            break;

        case 3:
            printf("\n");
            printf("Enter any level :: ");
            scanf("%d",&level);
            printf("\nNumber of nodes at [ %d ] Level ::
%d\n",level,NodesAtLevel(root,level));
            break;

        case 4:
                        exit(1);

                 default:
                        printf("\nWrong choice\n");

                }/*End of switch */
        }/*End of while */

        return 0;

}/*End of main( )*/


struct node *insert(struct node *ptr, int ikey )
{
        if(ptr==NULL)
        {
                ptr = (struct node *) malloc(sizeof(struct node));
                ptr->info = ikey;
                ptr->lchild = NULL;
                ptr->rchild = NULL;
        }
        else if(ikey < ptr->info) /*Insertion in left subtree*/
```

```
            ptr->lchild = insert(ptr->lchild, ikey);
        else if(ikey > ptr->info) /*Insertion in right subtree */
            ptr->rchild = insert(ptr->rchild, ikey);
        else
            printf("\nDuplicate key\n");
        return(ptr);
}/*End of insert( )*/

void display(struct node *ptr,int level)
{
        int i;
        if(ptr == NULL )/*Base Case*/
            return;
        else
      {
            display(ptr->rchild, level+1);
            printf("\n");
            for (i=0; i<level; i++)
                printf("    ");
            printf("%d", ptr->info);
            display(ptr->lchild, level+1);
        }
}/*End of display()*/


int NodesAtLevel(struct node *ptr, int level)
{
        if(ptr==NULL)
            return 0;
        if(level==0)
            return 1;
        return NodesAtLevel(ptr->lchild,level-1) + NodesAtLevel(ptr->rchild,level-1);
}/*End of NodesAtLevel()*/
```

*****************************SLIP 15********************************

Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm.

**Repeat SLIP 1 Q2**

Q 2. Write a C program for the implementation of Dijkstra's shortest path algorithm for finding shortest path from a given source vertex using adjacency cost matrix.

**Repeat SLIP 3 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency list. Implement function to traverse the graph using Breadth First Search (BFS) traversal

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 20
typedef struct node
{
int vertex;
struct node *next;
}NODE;
NODE *head[10];
typedef struct
{
int data[MAXSIZE];
int front, rear;
}QUEUE;
void createlist(int n)
{
int i, j, x;
NODE *temp, *newnode;
for(i=0; i<n; i++)
{
head[i]=NULL;
for(j=0; j<n; j++)
{
printf("is there any edge between %d and %d (1/0): ", i+1, j+1);
scanf("%d", &x);
if(x==1)
{
newnode=(NODE*)malloc(sizeof(NODE));
newnode->vertex=j+1;
newnode->next=NULL;
if(head[i]==NULL)
head[i]=temp=newnode;
else
{
```

```
temp->next=newnode;
temp=newnode;
}
}
}
}
}

void addq(QUEUE *pq, int num)
{
pq->data[++pq->rear]=num;
}
int removeq(QUEUE *pq)
{
return pq->data[++pq->front];
}
void initq(QUEUE *pq)
{
pq->front=pq->rear=-1;
}
int isempty(QUEUE *pq)
{
return (pq->rear==pq->front);
}
void bfs(NODE *list[10], int n)
{
int v;
int visited[20]={0}, x;
NODE *temp;
QUEUE pq;
initq(&pq);
v=0;
visited[v]=1;
addq(&pq, v); while(!
isempty(&pq))
{
v=removeq(&pq);
printf(" v%d ", v+1);
temp=list[v];
while(temp)
{
x=temp->vertex-1;
if(visited[x]==0)
```

```
{
addq(&pq, x);
visited[x]=1;
}
temp=temp->next;
}
}
}

void displaylist(int n)
{
NODE *temp;
printf("\nThe Adjacency list is: \n");
for(int i=0; i<n; i++)
{
printf("\nv%d -> ", i+1);
temp=head[i];
while(temp)
{
printf("v%d -> ", temp->vertex);
temp=temp->next;
}
printf("NULL");
}
printf("\n");
}
void main()
{
int n;
printf("Enter how many vertices: ");
scanf("%d", &n);
printf("\nEnter data for adjacency list:\n");
createlist(n);
displaylist(n);
printf("\nBFS Traversal is:");
bfs(head, n);
}
```

******************************SLIP 16*********************************

Q 1. Write a C program for the implementation of Floyd Warshall’s algorithm for finding all pairs shortest path using adjacency cost matrix.

**Repeat SLIP 3 Q2**

Q2. Write a program to sort n randomly generated elements using Heapsort method
**Repeat SLIP 5 Q2**

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level, and total levels in the tree.
**Repeat SLIP 14 Q2**

****************************SLIP 18*******************************

Q 1. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix
**Repeat SLIP 1 Q2**

Q 2. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, in order. Write a menu driven program that performs the above operations.
**Repeat SLIP 10 Q2**

Q 2. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.
**Repeat SLIP 1 Q2**

****************************SLIP 19*******************************

1. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, inorder. Write a menu driven program that performs the above operations.

**Repeat SLIP 10 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.
**Repeat SLIP 4 Q1**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Depth First Search (DFS) traversal
**Repeat SLIP 9 Q2**

****************************SLIP 20*******************************

Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, inorder. Write a menu driven program that performs the above operations.

**Repeat SLIP 10 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list

**Repeat SLIP 4 Q1**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal.

**Repeat SLIP 5 Q2**

****************************SLIP 21*******************************

Q 1. Write a C program for the implementation of Dijkstra's shortest path algorithm for finding shortest path from a given source vertex using adjacency cost matrix.

**Repeat SLIP 6 Q2**

Q 2. Write a program which uses binary search tree library and counts the total nodes and total leaf nodes in the tree. int count Leaf(T) – returns the total number of leaf nodes from BST

**Repeat SLIP 4 Q2**

Q 2. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm

**Repeat SLIP 1 Q2**

****************************SLIP 22*******************************

Q 1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.

**Repeat SLIP 4 Q1**

Q 2. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, postorder. Write a menu driven program that performs the above operations.

**Repeat SLIP 9 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Depth First Search (DFS) traversal.

**Repeat SLIP 2 Q2**

****************************SLIP 23********************************

Q 1. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

**Repeat SLIP 1 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

**Repeat SLIP 1 Q1**

Q 2. Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix.

**Repeat SLIP 3 Q2**

****************************SLIP 24*******************************

1. Write a program to sort n randomly generated elements using Heap sort method.

**Repeat SLIP 7 Q2**

Q 2. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

**Repeat SLIP 3 Q2**

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement functions to print indegree of all vertices of graph.

**Repeat SLIP 8 Q2**

****************************SLIP 25********************************

Q 1. Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix.
**Repeat SLIP 3 Q2**

Q2. Write a program to sort n randomly generated elements using Heapsort method.
**Repeat SLIP 7 Q2**

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level, and total levels in the tree.
**Repeat SLIP 14 Q2**