Slip Solutions

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 main()
{
 int m[10][10],n,v,i,j;
printf("How many vertices:");
scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 for(j=0;j<n;j++)
 {
printf("Is there edge between %d->%d(1/0):",i+1,j+1);
scanf("%d",&m[i][j]);
 }
 }
printf("Adjacency matrix is\n");
 for(i=0;i<n;i++)
 {
 for(j=0;j<n;j++)
 {
printf("%d\t",m[i][j]);
 }
printf("\n");
 }
}
```

Solution 2 ->

```
#include<stdio.h>
 #include<stdlib.h>
 typedef struct node
 {
    int vertex;
    struct node *next;
 }NODE;
 NODE *list[20];
 void createmat(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][j] = 0;
       if(i!= j)
       {
          printf("\nIs there Edge between %d and %d (1/0)", i+1,j+1);
          scanf("%d",&m[i][j]);
       }
    }
 }
 void dispmat(int m[10][10], int n)
 {
    int i,j;
    printf("\nThe 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,i,j,ind,outd,total;
   printf("\nEnter the no. of vertices :");
   scanf("%d", &n);
   createmat(m,n);
   dispmat(m,n);
}
```

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>
#define NEWNODE (struct node *)malloc(sizeof(struct node))
struct node
{
          struct node *left;
          int data;
          struct node *right;
};
struct node *root;
void init()
{
          root=NULL;
}
```

```
void insert(int item)
{	struct node *t1,*t2,*t;
	t=NEWNODE;
	t->data=item;
	t->left=NULL;
	t->right=NULL;
	if(root==NULL)
	{
		root=t;
	}
	else
	{
		t1=root;
		while(t1!=NULL)
		{
			t2=t1;
			if(item<=t1->data)
				t1=t1->left;
			else
				t1=t1->right;
		}
		if(item<=t2->data)
			t2->left=t;
		else
			t2->right=t;
	}
}
void preorder(struct node *t)
{
    if(t!=NULL)
    {
printf(" %d ",t->data);
		preorder(t->left);
preorder(t->right);
    }
}
int main()
{
	int n,i,item;
	printf("\nHow many item u want to store =");
	scanf("%d",&n);
	init();
	for(i=1;i<=n;i++)
	{
		printf("\nEnter data = ");
		scanf("%d",&item);
		insert(item);
	}
```

```
        printf("\nDisplaying tree PREORDER WISE = ");
   preorder(root);
        return 0;

}
```

Solution 2 ->

```
#include<stdio.h>
#include<malloc.h>
#define MAX 20
typedef struct node
{
int info;
struct node *left,*right;
} NODE;
/***TREE FUNCTION***/
NODE * createbst(NODE * root)
{
NODE *newnode,*temp;
int i,n,num;
printf("How many nodes:");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
newnode=(NODE*)malloc(sizeof(NODE));
printf("Enter the elements:");
scanf("%d",&num);
newnode->info=num;
newnode->left=newnode->right=NULL;
if(root==NULL)
root=newnode;
else
{
temp=root;
while(1)
{
if(num<temp->info)
if(temp->left==NULL) //temp does not have left child//
{
temp->left=newnode; //attach node//
break;
}
else
temp=temp->left; //move temp left//
else
if(temp->right==NULL)
{
```

```
temp->right=newnode; //attach newnode//
break;
}
else
temp=temp->right;
}
}
}
return(root);
}
void preorder(NODE * root)
{
NODE *temp=root;
if(temp!=NULL)
{
printf("%d\n",temp->info); //Data
preorder(temp->left); //left
preorder(temp->right); //right
}
}
int main()
{
int n,choice;
NODE *root=NULL;
do
{
printf("\n1:CREATE");
printf("\n2:TRAVERSALS");
printf("\n Enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:root=createbst(root);
     break;
case 2:printf("\npreorder traversal is :\n");
     preorder(root);
     break;
}
} while(choice<=2);
}
```

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

```
#include<stdio.h>
int a,b,u,v,n,i,j,e=1;
int visited[10]= {0},min,mincost=0,cost[10][10];
void main()
```

```
{
printf("Enter the number of vertices:");
scanf("%d",&n);
printf("Enter the 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;
}
visited[1]=1;
printf("\n");
while(e<n)
{
for(i=1,min=999;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]<min)
if(visited[i]!=0)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
if(visited[u]==0 || visited[v]==0)
{
printf("\n Edge %d:(%d %d) cost:%d",e++,a,b,min);
mincost+=min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n Minimun cost=%d",mincost);
}
```

Solution 2

```
#include<stdio.h>
int
cost[7][7]={{0,5,3,999,999,999,999},{5,0,4,6,2,999,999},{3,4,0,5,999,6,999},{999,6,5,0,8,6,999},{999,2,99
9,8,0,3,5},{999,999,6,6,3,0,4},{999,999,999,999,5,4,0}};
int n=7;
void main()
{
int a,b,u,v,i,j,e;
int visited[10]={0},min,mincost=0;
visited[0]=1;
printf("\n");
```

```
for(e=0;e<n;e++)
{
for(i=0,min=999;i<n;i++)
for(j=0;j<n;j++)
{
if(cost[i][j]==0)cost[i][j]=999;
if(cost[i][j]<min)
if(visited[i]!=0)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
}
if(visited[u]==0 || visited[v]==0)
{
printf("\n edge%d :(%d%d)cost:%d",e+1,a+1,b+1,min);
mincost+=min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n minimum cost=%d",mincost);
}
```

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

```
#include<stdio.h>
#define MAX 20
typedef struct
{
int data[MAX];
int top;
}STACK;
void init(STACK * ps)
{
ps->top=-1;
}
void push(STACK *ps, int num)
{
ps->data[++ps->top]=num;
}
int pop(STACK *ps)
{
return(ps->data[ps->top--]);
}
int isempty(STACK *ps)
{
```

```
return(ps->top==-1);
}
int isfull(STACK *ps)
{
return (ps->top==MAX-1);
}
int topoSort(int m[4][4], int n)
{
int i,v,j,w;
int visited[10]={0};
int indeg[10]={0};
/calculate the indegree/
for(i=0; i<n; i++)
for(j=0; j<n; j++)
indeg[i] = indeg[i] + m[j][i];
STACK s;
init(&s);
while(1)
{
for(v=0; v<n; v++)
if( (visited[v]==0) && (indeg[v]==0) )
{
visited[v]=1;
push(&s, v);
printf("v%d\n", v+1);
}
if(isempty(&s))
break;
v = pop(&s);
for(w=0; w<n; w++)
if(m[v][w]==1)
indeg[w] = indeg[w]-1;//reduce indegrees of adjacent vertices
}
}
int main()
{
int m[4][4] ={ {0,1,1},{0,0,0,1},{0,0,0,1},{0,0,0,0} };
topoSort(m,4);
}
```

5. 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>
#define MAX 10
typedef struct
{
```

```
int data[MAX];
int top;
}STACK;
void initstack(STACK * ps)
{
ps->top=-1;
}
void push(STACK *ps, int num)
{
ps->data[++ps->top]=num;
}
int pop(STACK *ps)
{
return(ps->data[ps->top--]);
}
int isempty(STACK *ps)
{
return(ps->top==-1);
}
int isfull(STACK *ps)
{
return (ps->top==MAX-1);
}
void dfs(int m[10][10], int n)
{
int i, v, w, found;
int visited[10]={0};
STACK s;
initstack(&s);
v=0;
visited[v]=1;
push(&s,v);
printf("v%d" ,v+1);
while(1)
{
found=0;
for(w=0; w<n; w++)
{
if((m[v][w]==1)&&(visited[w]==0))
{
push(&s, w);
printf("v%d", w+1);
visited[w]=1;
v = w;
found = 1;
break;
}
}
```

```
if(found == 0)// did not find an adjacent unvisited vertex
if(isempty(&s))
break;
else
v = pop(&s);
}
}
int recdfs(int m[10][10], int n, int v)
{
int w;
static int visited[10]={0};
visited[v]=1;
printf("v%d" ,v+1);
for(w=0; w<n; w++)
{
if( (m[v] [w]==1) && (visited[w]==0) )
recdfs(m,n,w);
}
}
int main()
{
int m[10][10], n, i, j, w;
printf("\nHow many vertices:");
scanf("%d", &n);
for(i=0; i<n; i++)
for(j=0; j<n; j++)
{
if(i!=j)
{
printf("Is there edge between vertex %d and %d
(1/0):", i+1, j+1);
scanf("%d", &m[i][j]);
}
}
printf("\nNon recursive depth first search is :");
dfs(m,n);
printf("\nRecursive depth first search is :");
recdfs(m,n,0);
}
```

6. Write a C program Which uses Binary Search tree library and Implements following function:
Int sumeven(T) – returns sum of all even numbers from BST.

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

```
{
   int info;
   struct node *left;
   struct node *right;
};
struct node *createnode(int key)
{
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
   newnode->info = key;
   newnode->left = NULL;
   newnode->right = NULL;
   return(newnode);
}
int sumeven(struct node *root)
{
   int rightsubtree, leftsubtree, sum = 0;
   if(root != NULL)
   {
      leftsubtree = sumeven(root->left);
      rightsubtree = sumeven(root->right);
      sum = (root->info) + leftsubtree + rightsubtree;
      return sum;
   }
}
int main()
{
   // first bst
   struct node *newnode = createnode(22);
   newnode->left = createnode(28);
   newnode->right = createnode(18);
   newnode->left->left = createnode(10);
   newnode->left->right = createnode(90);
   newnode->right->left = createnode(12);
   newnode->right->right = createnode(56);
   printf("Returning sum of all even numbers from BST  1 = %d", sumeven(newnode));
   printf("\n");

// creating 2nd bst//
   struct node *node = createnode(14);
   node->right = createnode(2);
   node->right->right = createnode(2);
   node->right->right->right = createnode(4);
   node->right->right->right->right = createnode(6);
   printf("Returning sum of all even numbers from BST   2 = %d", sumeven(node));
   printf("\n");

 // third bst
   struct node *root = createnode(16);
```

```
   printf("Returning sum of all even numbers from BST   3 = %d", sumeven(root));
   printf("\n");
   return 0;
}
```

7. 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 *list[10];
void createmat(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 present between %d and %d(1/0) :",i+1,j+1);
         scanf("%d",&m[i][j]);
      }
   }
}
void dispmat(int m[10][10],int n)
{
   int i,j;
   printf("\nThe adjacency matrix is :\n");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      printf("%5d",m[i][j]);
      printf("\n");
   }
}
void createlist(int m[10][10],int n)
{
   int i,j;
```

```
  struct node *temp,*newnode;
  for(i=0;i<n;i++)
  {
     list[i]=NULL;
     for(j=0;j<n;j++)
     {
        if(m[i][j]==1)
        {
           newnode=(NODE*)malloc(sizeof(NODE));
           newnode->vertex=j+1;
           if(list[i]==NULL)
           list[i]=temp=newnode;
           else
           {
              temp->next=newnode;
              temp=newnode;
           }
        }
     }
  }
}
 void displist(int n)
 {
    struct node *temp;
    int i;
    printf("\n The adjcency list is :\n");
    for(i=0;i<n;i++)
    {
       printf("\nv%d->",i+1);
       temp=list[i];
       while(temp)
       {
          printf("v%d->",temp->vertex);
          temp=temp->next;
       }
       printf("NULL");
    }
 }
 void main()
 {
    int m[10][10],n;
    printf("Enter the no of vertices :");
    scanf("%d",&n);
    createmat(m,n);
```

```
    dispmat(m,n);
    createlist(m,n);
    displist(n);
 }
```

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

```
#include<stdio.h>
#include<stdlib.h>
//node structure
struct node
{
int data;
struct node *right;
struct node *left;
}*root;
//create function
struct node *create(struct node *root,int item)
{
if(root==NULL)
{
//creating newnode
root=(struct node *)malloc(sizeof(struct node));
root->right = root->left = NULL;
root->data=item;
return root;
}
else
{
if(root->data <item)
root->right=create(root->right,item);
else if(root->data > item)
root->left=create(root->left,item); else
printf("\nDuplicate elements are not allowed in BST");
return(root);
}
}
int countleaf(struct node *root)
{
   static int  leaf = 0;
   if(root!=NULL)
   {
```

```
      if((root->left==NULL) && (root->right==NULL))
      leaf++;
      countleaf(root->left);
      countleaf(root->right);
   }
   return leaf;
}
struct node *create(struct node * , int);
void inorder(struct node *);
int countleaf(struct node *);
int main()
{
int i,j,n,item,ch,key,cnt;
printf("\nBinary search tree");
printf("\n 1.create");
printf("\n 2.count leaf nodes");
printf("\n 3.Exit");
while(1)
{
printf("\nEnter your choice :");
scanf("%d",&ch);
switch(ch)
{
case 1:
  root=NULL;
  printf("\nEnter how many nodes:");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
{
  printf("\nEnter data for nodes:");
  scanf("%d",&item);
   root=create(root,item); //calling the create function
}
break;
case 2 :cnt = countleaf(root);
           printf("\n Leaf nodes are : %d",cnt);
           break;
case 3:exit(0);
 default:
 printf("\nWrong choice");
}
}
return 0;
}
```

Solution 2

```
#include<stdio.h>
#include<malloc.h>
#define MAX 3
typedef struct node
{
int info;
struct node *left,*right;
} NODE;
NODE * createbst(NODE * root)
{
NODE *newnode,*temp, *parent;
int i,n,num;
printf("How many nodes:");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
newnode=(NODE*)malloc(sizeof(NODE));
printf("Enter the elements:");
scanf("%d",&num);
newnode->info=num;
newnode->left=newnode->right=NULL;
if(root==NULL)
{
root = newnode;
continue;
}
temp=root;
while(temp!=NULL)
{
parent=temp;
if(num < temp->info)
temp=temp->left;
else
temp=temp->right;
}
if(num < parent->info)
parent->left=newnode;
else
parent->right=newnode;
}
return(root);
```

```
}
int Cnodes(NODE * root)
{
static int count = 0;
NODE * temp = root;
if(temp!=NULL)
{
count++;
Cnodes(temp->left);
Cnodes(temp->right);
}
return count;
}
int Cleaf(NODE * root)
{
static int leaf = 0;
NODE * temp = root;
if(temp!=NULL)
{
if((temp->left==NULL) && (temp->right==NULL))
leaf++;
Cleaf(temp->left);
Cleaf(temp->right);
}
return leaf;
}
int main()
{
int n,choice,key,count;
NODE *root=NULL, *root1=NULL, *root2=NULL;
do
{
printf("\n1:CREATE");
printf("\n2:Count Leaf Nodes");
printf("\nEnter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:root=createbst(root);
          break;
case 2:printf("Total leaf nodes in Tree = %d",Cleaf(root));
         break;
}
} while(choice!=3);
```

}

9. 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 node
{
      struct node *left;
      int info;
      struct node *right;
};
struct node *insert(struct node *ptr, int key )
{
  if(ptr==NULL)
{
ptr = (struct node *) malloc(sizeof(struct node));
ptr->info = key;
ptr->left = NULL;
ptr->right = NULL;
}
   else if(key <ptr->info)
ptr->left = insert(ptr->left, key);
   else if(key >ptr->info)
ptr->right = insert(ptr->right, key);
   else
printf("\nDuplicate key\n");
      return(ptr);
}
int NodesAtLevel(struct node *ptr, int level)
{
      if(ptr==NULL)
       return 0;
      if(level==0)
        return 1;
       return NodesAtLevel(ptr->left,level-1) + NodesAtLevel(ptr->right,level-1);
}
int main()
{
   struct node *root=NULL,*root1=NULL,*ptr;
   int choice,k,item,level,i;
   while(1)
{
```

```
printf("\n");
printf("1.Insert Tree \n");
printf("2.Number of Nodes present at any level\n");
printf("\nEnter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("\nEnter number of nodes : ");
scanf("%d",&k);
for(i=1;i<=k;i++)
{
printf(“enter data for node”,i);
scanf("%d",&item);
root = insert(root, item);
}
break;
case 2: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");
            }
      }
      return 0;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
void swap(int* a, int* b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}
void heapify(int arr[], int n, int i) {
   int largest = i;
   int l = 2 * i + 1;
   int r = 2 * i + 2;
```

```
    if (l < n && arr[l] > arr[largest]) {
        largest = l;
    }
    if (r < n && arr[r] > arr[largest]) {
        largest = r;
    }
    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
    for (int i = n - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Generating %d random numbers...\n", n);
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100;
    }
    printf("Unsorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    heapSort(arr, n);
    printf("\nSorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

11. 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>
#define MAX 10
typedef struct
{
int data[MAX];
int front, rear;
}QUEUE;
void initq(QUEUE *pq)
{
pq->front = pq->rear = -1;
}
void addq(QUEUE *pq, int num)
{
pq->rear++;
pq->data[pq->rear] = num;
}
int removeq(QUEUE *pq)
{
int num;
pq->front++;
num=pq->data[pq->front];
return(num);
}
int isempty(QUEUE *pq)
{
return(pq->front == pq->rear);
}
int isfull(QUEUE *pq)
{
return(pq->rear==MAX-1);
}
void bfs(int m[10][10], int n)
{
int i, v, w;
int visited[10]={0};
QUEUE q;
initq(&q);
v=0;
visited[v]=1;
addq(&q,v);
```

```
while(!isempty(&q))
{
v=removeq(&q);
printf("v%d",v+1);
for(w=0; w<n; w++)
{
if((m[v][w]==1)&&(visited[w]==0))
{
addq(&q, w);
visited[w]=1;
}
}
}
}
int main()
{
int m[10][10], n, i, j, w;
printf("\nHow many vertices:");
scanf("%d", &n);
for(i=0; i<n; i++)
for(j=0; j<n; j++)
{
if(i!=j)
{
printf("Is there edge between vertex %d and %d
(1/0):", i+1, j+1);
scanf("%d", &m[i][j]);
}
}
printf("\nNon recursive breadth first search is :");
bfs(m,n);
}
```

12. 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>
#define MAX 20
void djks(int c[10][10], int n)
{
int i, j, v, w, u, count, min;
int dist[10];
int visited[10]={0};
```

```
printf("\nEnter the source vertex:");
scanf("%d", &v);
v=v-1;
for(i=0; i<n; i++)
dist[i] = c[v][i];
visited[v] = 1;
count = 1;
while(count < n)
{
min = 999;
for(i=0; i<n; i++)
if( (visited[i]==0) && (dist[i] < min))
{
min = dist[i];
u = i;
}
visited[u] = 1;
for(w=0; w<n; w++)
{
if(!visited[w])
if(dist[u] + c[u][w] < dist[w])
dist[w] = dist[u] + c[u][w];
count++;
}
}
printf("\nThe sortest path are:\n");
for(i=0;i<n;i++)
printf("from v%d to v%d = %d\n",v+1,i+1,dist[i]);
}
int main()
{
int c[10][10]={0},n,i,j;
printf("\nHow many vertices:");
scanf("%d",&n);
printf("\nEnter the Adjacency cost matrix:\n");
for(i=0; i<n; i++)
for(j=0; j<n; j++)
{
if(i<j)
{
printf("\nEnter the cost of edge %d->%d :", i+1, j+1);
scanf("%d", &c[i][j]);
c[j][i]= c[i][j];
}
```

```
}
djks(c,n);
}
```

13. Write a C program Which uses Binary Search tree library and Implements following function:
Int sumodd(T) – returns sum of all even numbers from BST.

```
#include <stdio.h>
#include <stdlib.h>
struct node
 {
   int data;
   struct node* left;
   struct node* right;
};
struct node* newNode(int data)
{
   struct node* node = (struct node*)malloc(sizeof(struct node));
   node->data = data;
   node->left = NULL;
   node->right = NULL;
   return node;
}

// ****Insert a new node into the Binary Search Tree****//
struct node* insert(struct node* node, int data)
{
   if (node == NULL) {
      return newNode(data);
   }
   if (data < node->data) {
      node->left = insert(node->left, data);
   }
   else if (data > node->data) {
      node->right = insert(node->right, data);
   }
   return node;
}
int sumodd(struct node* node)
{
   if (node == NULL) {
      return 0;
   }
```

```
   int sum = 0;
   if (node->data % 2 != 0)
{
      sum += node->data;
   }
   sum += sumodd(node->left);
   sum += sumodd(node->right);
   return sum;
}
int main()
{
   struct node* root = NULL;
   root = insert(root, 5);
        insert(root, 3);
        insert(root, 7);
        insert(root, 2);
        insert(root, 4);
        insert(root, 6);
        insert(root, 8);
   printf("Sum of all odd numbers in the Binary Search Tree: %d", sumodd(root));
   return 0;
}
```

Solution 2

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *left;
struct node *right;
};
struct node *createnode(int key)
{
struct node *newnode = (struct node*)malloc(sizeof(struct node));
newnode->info = key;
newnode->left = NULL;
newnode->right = NULL;
return(newnode);
}
int sumodd(struct node *root)
{
int rightsubtree, leftsubtree, sum = 0;
```

```
if(root != NULL)
{
leftsubtree = sumodd(root->left);
rightsubtree = sumodd(root->right);
sum = (root->info) + leftsubtree + rightsubtree;
return sum;
   }
}
int main()
{    // first bst//
struct node *newnode = createnode(25);
newnode->left = createnode(27);
newnode->right = createnode(19);
newnode->left->left = createnode(17);
newnode->left->right = createnode(91);
newnode->right->left = createnode(13);
newnode->right->right = createnode(55);
printf("Returning sum of all odd numbers from BST  1 = %d", sumodd(newnode));
printf("\n");
   // creating 2nd bst
struct node *node = createnode(1);
node->right = createnode(2);
node->right->right = createnode(3);
node->right->right->right = createnode(4);
node->right->right->right->right = createnode(5);
printf("Returning sum of all odd numbers from BST   2 = %d", sumodd(node));
printf("\n");
 // third bst
struct node *root = createnode(15);
printf("Returning sum of all odd numbers from BST   3 = %d", sumodd(root));
printf("\n");
return 0;
}
```

14. 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.

OR

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>
void main()
{
int m[10][10],r,c,sumin,sumout,n,v,i;
printf("how many vertices:");
scanf("%d",&n);
for(r=0;r<n;r++)
for(c=0;c<n;c++)
{
m[r][c]=0;
if(r!=c)
{
printf("is there an edge between %d and %d(1/0):",r+1,c+1);
scanf("%d",&m[r][c]);
}
}
printf("\n\nVertex Indegree Outdegree Total degree\n");
for(v=0;v<n;v++)
{
sumin=sumout=0;
for(i=0;i<n;i++)
{
sumin=sumin+m[i][v];
sumout=sumout+m[v][i];
}
printf("%d\t\t%d\t\t%d\t\t%d\n",v+1,sumin,sumout,sumin+sumout);
}
}
```

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

```
#include<stdio.h>
#include<stdlib.h>
#define NEWNODE (struct node *)malloc(sizeof(struct node))
struct node
{
        struct node *left;
        int data;
        struct node *right;
};
struct node *root;
void init()
{
```

```
        root=NULL;
}
void insert(int item)
{       struct node *t1,*t2,*t;
        t=NEWNODE;
        t->data=item;
        t->left=NULL;
        t->right=NULL;
        if(root==NULL)
        {
                root=t;
        }
        else
        {
                t1=root;
                while(t1!=NULL)
                {
                        t2=t1;
                        if(item<=t1->data)
                                t1=t1->left;
                        else
                                t1=t1->right;
                }
                if(item<=t2->data)
                        t2->left=t;
                else
                        t2->right=t;
        }
}
void postorder(struct node *t)
{
   if(t!=NULL)
{
postorder(t->left);
postorder(t->right);
printf(" %d ",t->data);
      }
}
int main()
{
        int n,i,item;
        printf("\nHow many item u want to store =");
        scanf("%d",&n);
        init();
        for(i=1;i<=n;i++)
        {
                printf("\nEnter data = ");
                scanf("%d",&item);
```

```
                insert(item);
        }
        printf("\nDisplaying tree POSTORDER WISE = ");
        postorder(root);
        return 0;
}
```

16. 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>
#define NEWNODE (struct node *)malloc(sizeof(struct node))
struct node
{
        struct node *left;
        int data;
        struct node *right;
};
struct node *root;
void init()
{
        root=NULL;
}
void insert(int item)
{       struct node *t1,*t2,*t;
        t=NEWNODE;
        t->data=item;
        t->left=NULL;
        t->right=NULL;
        if(root==NULL)
        {
                root=t;
        }
        else
        {
                t1=root;
                while(t1!=NULL)
                {
                        t2=t1;
                        if(item<=t1->data)
                                t1=t1->left;
                        else
                                t1=t1->right;
                }
                if(item<=t2->data)
                        t2->left=t;
                else
                        t2->right=t;
```

```
            }
}
void inorder(struct node *t)
{
   if(t!=NULL)
{
inorder(t->left);
printf(" %d ",t->data);
inorder(t->right);
      }
}
int main()
{
            int n,i,item;
            printf("\n How many item u want to store =");
            scanf("%d",&n);
            init();
            for(i=1;i<=n;i++)
            {
                        printf("\n Enter data = ");
                        scanf("%d",&item);
                        insert(item);
            }
            printf("\n Displaying tree INORDER WISE = ");
   inorder(root);
            return 0;

}
```

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

```
#include <stdio.h>
int i,j,k,a,b,u,v,n,e=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
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;
}
void main()
{
printf("Enter the no. of vertices:");
scanf("%d",&n);
printf("Enter the 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("\n");
while(e<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",e++,a,b,min);
mincost+=min;
}
cost[a][b]=cost[b][a]=999;
```

```
}
printf("\nMinimum cost = %d\n",mincost);
}
```

18. Write a C program which uses Binary Search tree library and implements following function:
Int compare(T1,T2) – compares two binary search trees.

```
#include <stdio.h>
#include <stdlib.h>
// BST node
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
 struct Node* newNode(int data)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));

    node->data = data;
    node->left = NULL;
    node->right =NULL;

    return node;
}
// Function to check if two BSTs are identical//
int isIdentical(struct Node* root1,struct Node* root2)
{
    // Check if both the trees are empty
    if (root1 == NULL && root2 == NULL)
        return 1;
    // If any one of the tree is non-empty and other is empty, return false//
    else if (root1 == NULL || root2 == NULL)
        return 0;
    else {
         // Check if current data of both trees equal
        // and recursively check for left and right subtrees
        if (root1->data == root2->data && isIdentical(root1->left, root2->left)&& isIdentical(root1-
>right, root2->right))
            return 1;
        else
            return 0;
    }
}
```

```
// Driver code
int main()
{
    struct Node* root1 = newNode(5);
    struct Node* root2 = newNode(5);
    root1->left = newNode(3);
    root1->right = newNode(8);
    root1->left->left = newNode(2);
    root1->left->right = newNode(4);

    root2->left = newNode(3);
    root2->right = newNode(8);
    root2->left->left = newNode(2);
    root2->left->right = newNode(4);

    if (isIdentical(root1, root2))
        printf( "Both BSTs are identical");
    else
        printf("BSTs are not identical") ;

    return 0;
}
```