SLIP-1

Q1. Implement C program to implement BST to perform following operations on BST- create,
recursive traversal - inorder.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
```

```
}
}
void inorder(NODE *root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Inorder");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
```

```
printf("\nInorder=");
inorder(root);
break;
case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice1
Enter how many nodes you want to create4
Enter the node info12
Enter the node info23
Enter the node info45
Enter the node info11
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice2
Inorder=11 12 23 45
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
```

Q2. Implementation of Dijkstra’s shortest path algorithm for finding Shortest Path from a given source
vertex using adjacency cost matrix.

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

```
int cost[8][8]={
{0,999,999,999,999,999,999,999},
{30,0,999,999,999,999,999,999},
{100,80,0,999,999,999,999,999},
{999,999,120,0,999,999,999,999},
{999,999,999,150,0,25,999,999},
{999,999,999,100,999,0,90,140},
{999,999,999,999,999,999,0,100},
{170,999,999,999,999,999,999,0}
};
void dijkstra(int v,int n)
{
int i,j,u,w,count,min;
int dist[10],visited[10]={0};
visited[v]=1;
for(i=0;i<n;i++)
dist[i]=cost[v][i];
count=2;
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(dist[u]+cost[u][w]<dist[w])
dist[w]=dist[u]+cost[u][w];
count++;
}
printf("\n shortest distance from vertex %d are:\n",v);
for(i=0;i<n;i++)
printf("%d\t",dist[i]);
}
void main()
{
dijkstra(4,8);
}
```

OUTPUT:

```
Shortest distance from vertex 4 are:
335 325 245 125 0 25 115 165
```

SLIP-2

Q1. C program to implement BST to perform following operations on BST a) Create b) delete

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
```

```
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
}
}
void inorder(NODE *root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
NODE* delete_BST (NODE *root, int n)
{
NODE *temp, *succ ;
if(root== NULL)
{
printf("\n No. not found");
return (root);
}
if (n<root->info)
root->left-delete_BST(root->left,n);
else
if (n>root->info)
root->right=delete_BST(root->right,n);
else
{
if (root->left!= NULL && root->right!=NULL)
{
succ=root->right;
```

```
while (succ->left)
succ= succ->left;
root->info=succ->info;
root->right=delete_BST(root->right, succ->info);
}
else
{
temp-root;
if(root->left != NULL)
root-root->left;
else if (root->right!= NULL)
root=root->right;
else
root=NULL;
free (temp);
}
}
return (root);
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Deleting node from bst");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
```

```
for(i=0;i<n;i++)
{
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nEnter element to be deleted from BST");
scanf("%d",&n);
temp=delete_BST(root,n);
printf("\nResult after deletion is :\n");
inorder(root);
break;
case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Deleting node from bst
3.Exit
Enter your choice1
Enter how many nodes you want to create4
ENter the node info13
ENter the node info45
ENter the node info78
ENter the node info99
BINARY SEARCH TREE
1.Create bst
2.Deleting node from bst
3.Exit
```

```
Enter your choice2
Enter element to be deleted from BST99
13 45 78
BINARY SEARCH TREE
1.Create bst
2.Deleting node from bst
3.Exit
Enter your choice3
```

Q2. C program to calculate in-degree and out-degree of all vertices in a graph .

```c
#include<stdio.h>
void main()
{
int a[10][10],n,j,i,out=0,in=0;
printf("enter the how many vertex");
scanf("%d",&n);
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
a[i][j]=0;
if(i!=j)
{
printf("is there edge bet %d and %d",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
}
printf("Vertex Indegree Outdegree\n");
for(i=0;i<n;i++)
{
in=out=0;
for(j=0;j<n;j++)
{
in=in+a[j][i]; // count for indegree at vertex i
out=out+a[i][j]; // count for outdegree from vertex i
}
printf(" v%3d%14d%10d",i+1,in,out);
printf("\n");
}
}
```

OUTPUT:

```
enter the how many vertex6
is there edge bet 1 and 2
0
is there edge bet 1 and 30
is there edge bet 1 and 40
is there edge bet 1 and 50
is there edge bet 1 and 60
is there edge bet 2 and 11
is there edge bet 2 and 3
0
is there edge bet 2 and 41
is there edge bet 2 and 50
is there edge bet 2 and 60
is there edge bet 3 and 10
is there edge bet 3 and 21
is there edge bet 3 and 40
is there edge bet 3 and 50
is there edge bet 3 and 61
is there edge bet 4 and 10
is there edge bet 4 and 20
is there edge bet 4 and 31
is there edge bet 4 and 51
is there edge bet 4 and 61
is there edge bet 5 and 11
is there edge bet 5 and 20
is there edge bet 5 and 30
is there edge bet 5 and 40
is there edge bet 5 and 61
is there edge bet 6 and 11
is there edge bet 6 and 21
is there edge bet 6 and 30
is there edge bet 6 and 40
is there edge bet 6 and 50
vertex indegree oudegree
 1 3 0
 2 2 2
 3 1 2
 4 1 3
 5 1 2
 6 3 2
```

SLIP-3

Q1. Implementation of static hash table with Linear Probing.

```
//Hashing using linear probing : C program
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
int key,index,i,flag=0,hkey;
printf("\nenter a value to insert into hash table\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE;i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index] == NULL)
{
h[index]=key;
break;
}
}
```

```
if(i == TABLE_SIZE)
printf("\nelement cannot be inserted\n");
}
void search()
{
int key,index,i,flag=0,hkey;
printf("\nenter search element\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE; i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index]==key)
{
printf("value is found at index %d",index);
break;
}
}
if(i == TABLE_SIZE)
printf("\n value is not found\n");
}
void display()
{
int i;
printf("\nelements in the hash table are \n");
for(i=0;i< TABLE_SIZE; i++)
printf("\nat index %d \t value = %d",i,h[i]);
}
main()
{
int opt,i;
while(1)
{
printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
scanf("%d",&opt);
switch(opt)
{
case 1:
insert();
break;
case 2:
display();
break;
case 3:
search();
break;
case 4:exit(0);
}
}
}
```

OUTPUT:

```
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
12
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
13
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
22
Press 1. Insert 2. Display 3. Search 4.Exit
2
elements in the hash table are
at index 0 value = 0
at index 1 value = 0
at index 2 value = 12
at index 3 value = 13
at index 4 value = 22
at index 5 value = 0
at index 6 value = 0
at index 7 value = 0
at index 8 value = 0
at index 9 value = 0
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
12
value is found at index 2
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
23
value is not found
Press 1. Insert 2. Display 3. Search 4.Exit
4*/
```

Q.2. C program to implement graph traversal method using depth first search.

```
#include<stdio.h>
#include<stdlib.h>
void recdfs(int m[5][5],int n ,int v)
{
int w;
static int visited[20]={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);
}
}
void main()
{
int m[5][5]={{0,0,1,1,0},{0,0,1,0,1},{0,1,0,0,0},{0,0,0,0,1},{0,0,0,0,0}};
printf("\n the depth first search traverse is : ");
recdfs(m,5,0);
}
```

OUTPUT:

```
the depth first search traverse is : v1v3v2v5v4
```

SLIP-4

Q1. C program to implement BST to perform following operations on BSTcreate, recursive traversalpreorder.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
```

```
else
printf("Duplicate element is not allowed");
}
return root;
}
}
void preorder(NODE *root)
{
if(root!=NULL)
{
printf("%d\t",root->info);
preorder(root->left);
preorder(root->right);
}
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Preorder");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nPreorder=");
preorder(root);
break;
case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Preorder
3.Exit
Enter your choice1
Enter how many nodes you want to create4
ENter the node info23
ENter the node info45
ENter the node info67
ENter the node info98
BINARY SEARCH TREE
1.Create bst
2.Preorder
3.Exit
Enter your choice2
Preorder=23 45 67 98
BINARY SEARCH TREE
1.Create bst
2.Preorder
3.Exit
Enter your choice3
```

Q2. Write C Program that accept the vertices and edges of a graph and store it is an adjacency Matrix.

```
#include<stdio.h>
int main()
{
int a[10][10],i,j,n;
//Create
printf("\nEnter total no. of vertices: ");
scanf("%d",&n);
printf("\n*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge
not present)*\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
a[i][j]=0;
if(i!=j)
{
printf("\nIs there edge between v%d & v%d: ",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
}
//Display
printf("\n Matrix is \n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
}
```

OUTPUT:

```
Enter total no. of vertex: 3
Is there edge between 1 & 2: 1
Is there edge between 1 & 3: 0
Is there edge between 2 & 1: 1
Is there edge between 2 & 3: 1
Is there edge between 3 & 1: 0
Is there edge between 3 & 2: 1
Matrix is
0 1 0
1 0 1
0 1 0
```

SLIP-5

Q1. C program to implement graph traversal method using breadth first search.

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 20
typedef struct
{
int data[MAXSIZE];
int front,rear;
}QUEUE;
void initq(QUEUE *pq)
{
pq->front=pq->rear=-1;
}
void addq(QUEUE *pq,int n)
{
pq->data[++pq->rear]=n;
}
int removeq(QUEUE *pq)
{
return pq->data[++pq->front];
```

```
}
int isempty(QUEUE *pq)
{
return(pq->front==pq->rear);
}
void createlist(int m[10][10],int n)
{
int i,j;
printf("\n*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge
not present)*\n");
for(i=0;i<n;i++)
{
// list[i]=NULL;
for(j=0;j<n;j++)
{
if(i!=j)
{
printf("\nIs there edge between %d & %d: ",i+1,j+1);
scanf("%d",&m[i][j]);
}
}
}
}
void bfs(int a[10][10],int n)
{
int v=0;
int visited[20]={0};
QUEUE q;
initq(&q);
printf("\nThe breadth first traversal is:\n");
visited[v]=1;
addq(&q,v);
while(!isempty(&q))
{
v=removeq(&q);
printf("v%d\t",v+1);
for(int w=0;w<n;w++)
{
if((a[v][w]==1)&&(visited[w]==0))
{
addq(&q,w);
visited[w]=1;
}
}
}
}
int main()
{
int a[10][10],n;
printf("\nEnter the no of vertex:");
scanf("%d",&n);
createlist(a,n);
bfs(a,n);
}
OUTPUT:
Enter the no of vertex:4
Type 1 for Yes and 0 for No
Is there edge between 1 and 2:1
Is there edge between 1 and 3:0
Is there edge between 1 and 4:1
Is there edge between 2 and 1:1
Is there edge between 2 and 3:0
Is there edge between 2 and 4:1
Is there edge between 3 and 1:1
Is there edge between 3 and 2:0
Is there edge between 3 and 4:0
Is there edge between 4 and 1:1
Is there edge between 4 and 2:1
Is there edge between 4 and 3:1
The breadth first traversal is:
v1 v2 v4 v3 */
```

Q2. C program to implement BST to perform following operations on BST a) create b) insert.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
}
}
void inorder(NODE *root)
{
```

```
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Inorder");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nInorder=");
inorder(root);
break;case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice1
Enter how many nodes you want to create4
ENter the node info12
ENter the node info43
ENter the node info75
ENter the node info99
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice2
Inorder=12 43 75 99
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice3
```

SLIP-6

Q1. C program to implement graph traversal method using depth first search.

```
#include<stdio.h>
#include<stdlib.h>
void recdfs(int m[5][5],int n ,int v)
{
```

```
int w;
static int visited[20]={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);
}
}
void main()
{
int m[5][5]={{0,0,1,1,0},{0,0,1,0,1},{0,1,0,0,0},{0,0,0,0,1},{0,0,0,0,0}};
printf("\n the depth first search traverse is : ");
recdfs(m,5,0);
}
```

OUTPUT:

```
the depth first search traverse is : v1v3v2v5v4
```

Q2. Implementation of Dijkstra’s shortest path algorithm for finding Shortest Path from a given source
vertex using adjacency cost matrix.

```
#include<stdio.h>
int cost[8][8]={
{0,999,999,999,999,999,999,999},
{30,0,999,999,999,999,999,999},
{100,80,0,999,999,999,999,999},
{999,999,120,0,999,999,999,999},
{999,999,999,150,0,25,999,999},
{999,999,999,100,999,0,90,140},
{999,999,999,999,999,999,0,100},
{170,999,999,999,999,999,999,0}
};
void dijkstra(int v,int n)
{
int i,j,u,w,count,min;
int dist[10],visited[10]={0};
visited[v]=1;
for(i=0;i<n;i++)
dist[i]=cost[v][i];
count=2;
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(dist[u]+cost[u][w]<dist[w])
dist[w]=dist[u]+cost[u][w];
count++;
}
printf("\n shortest distance from vertex %d are:\n",v);
for(i=0;i<n;i++)
printf("%d\t",dist[i]);
}
void main()
{
dijkstra(4,8);
}
```

OUTPUT:

```
Shortest distance from vertex 4 are:
335 325 245 125 0 25 115 165
```

SLIP-7

Q1. C program to implement BST to perform following operations on BST1) Create
2) recursive
traversal- postorder .

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
}
}
void postorder(NODE *root)
{
if(root!=NULL)
{
```

```
postorder(root->left);
postorder(root->right);
printf("%d\t",root->info);
}
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Postorder");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nPostorder=");
postorder(root);
break;
case 3:
exit(0);
default:
```

```
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Postorder
3.Exit
Enter your choice1
Enter how many nodes you want to create4
ENter the node info33
ENter the node info11
ENter the node info67
ENter the node info23
BINARY SEARCH TREE
1.Create bst
2.Postorder
3.Exit
Enter your choice2
Postorder=23 11 67 33
BINARY SEARCH TREE
1.Create bst
2.Postorder
3.Exit
```

Q2. Write C Program that accept the vertices and edges of graph. Create adjacency List and display
adjacency List.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
int v;
struct node *next;
```

```
}NODE;
NODE *list[10]; //array of pointers
void createlist(int m[10][10],int n)
{
int i,j;
printf("\n*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge
not present)*\n");
for(i=0;i<n;i++)
{
// list[i]=NULL;
for(j=0;j<n;j++)
{
if(i!=j)
{
printf("\nIs there edge between v%d & v%d : ",i+1,j+1);
scanf("%d",&m[i][j]);
}
}
}
}
void adjlist(int m[10][10],int n)
{
int i,j;
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->v=j+1;
newnode->next=NULL;
if(list[i]==NULL)
list[i]=temp=newnode;
else
{
temp->next=newnode;
temp=newnode;
}
}
}
}
}
void displaylist(int n)
{
NODE *temp;
int i;
printf("\n The adjecancy list is:\n");
for(i=0;i<n;i++)
{
printf("\nv%d->",i+1);
temp=list[i];
while(temp)
{
printf("v%d->", temp->v);
temp=temp->next;
}
printf("NULL");
}
}
void main()
{
int m[10][10],n;
printf("\nEnter the number of vertices:");
scanf("%d",&n);
createlist(m,n);
adjlist(m,n);
displaylist(n);
}
```

OUTPUT:

```
Enter the number of vertices:4
*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge not present)*
Is there edge between v1 & v2 : 1
Is there edge between v1 & v3 : 1
Is there edge between v1 & v4 : 0
Is there edge between v2 & v1 : 0
Is there edge between v2 & v3 : 1
Is there edge between v2 & v4 : 1
Is there edge between v3 & v1 : 0
Is there edge between v3 & v2 : 0
Is there edge between v3 & v4 : 1
Is there edge between v4 & v1 : 0
Is there edge between v4 & v2 : 1
Is there edge between v4 & v3 : 0
The adjecancy list is:
v1->v2->v3->NULL
v2->v3->v4->NULL
v3->v4->NULL
v4->v2->NULL
```

SLIP-8

Q1. C program to implement graph as adjacency matrix.

```
#include<stdio.h>
int main()
{
int a[10][10],i,j,n;
//Create
printf("\nEnter total no. of vertices: ");
scanf("%d",&n);
printf("\n*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge
not present)*\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
a[i][j]=0;
if(i!=j)
{
printf("\nIs there edge between v%d & v%d: ",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
}
//Display
printf("\n Matrix is \n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
}
```

OUTPUT:

```
Enter total no. of vertex: 3
Is there edge between 1 & 2: 1
Is there edge between 1 & 3: 0
Is there edge between 2 & 1: 1
Is there edge between 2 & 3: 1
Is there edge between 3 & 1: 0
Is there edge between 3 & 2: 1
Matrix is
0 1 0
1 0 1
0 1 0
```

Q2. Implementation of static hash table with Linear Probing.

```
//Hashing using linear probing : C program
```

```
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
int key,index,i,flag=0,hkey;
printf("\nenter a value to insert into hash table\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE;i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index] == NULL)
{
h[index]=key;
break;
}
}
if(i == TABLE_SIZE)
printf("\nelement cannot be inserted\n");
}
void search()
{
int key,index,i,flag=0,hkey;
printf("\nenter search element\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE; i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index]==key)
{
printf("value is found at index %d",index);
break;
}
}
if(i == TABLE_SIZE)
printf("\n value is not found\n");
}
void display()
{
int i;
printf("\nelements in the hash table are \n");
for(i=0;i< TABLE_SIZE; i++)
printf("\nat index %d \t value = %d",i,h[i]);
}
main()
{
int opt,i;
while(1)
{
printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
scanf("%d",&opt);
switch(opt)
{
case 1:
insert();
break;
case 2:
display();
break;
case 3:
search();
break;
case 4:exit(0);
}
}
}
```

OUTPUT:

```
Press 1. Insert 2. Display 3. Search 4.Exit
1
```

```
enter a value to insert into hash table
12
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
13
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
22
Press 1. Insert 2. Display 3. Search 4.Exit
2
elements in the hash table are
at index 0 value = 0
at index 1 value = 0
at index 2 value = 12
at index 3 value = 13
at index 4 value = 22
at index 5 value = 0
at index 6 value = 0
at index 7 value = 0
at index 8 value = 0
at index 9 value = 0
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
12
value is found at index 2
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
23
value is not found
Press 1. Insert 2. Display 3. Search 4.Exit
4*/
```

SLIP-9

Q1. C program to implement graph traversal method using breadth first search.

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 20
typedef struct
{
int data[MAXSIZE];
int front,rear;
}QUEUE;
void initq(QUEUE *pq)
{
pq->front=pq->rear=-1;
}
void addq(QUEUE *pq,int n)
{
pq->data[++pq->rear]=n;
}
int removeq(QUEUE *pq)
{
return pq->data[++pq->front];
}
int isempty(QUEUE *pq)
{
return(pq->front==pq->rear);
}
void createlist(int m[10][10],int n)
{
int i,j;
printf("\n*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge
not present)*\n");
for(i=0;i<n;i++)
{
// list[i]=NULL;
for(j=0;j<n;j++)
{
if(i!=j)
{
printf("\nIs there edge between %d & %d: ",i+1,j+1);
scanf("%d",&m[i][j]);
}
}
}
}
void bfs(int a[10][10],int n)
{
int v=0;
int visited[20]={0};
QUEUE q;
initq(&q);
printf("\nThe breadth first traversal is:\n");
visited[v]=1;
addq(&q,v);
while(!isempty(&q))
{
v=removeq(&q);
printf("v%d\t",v+1);
for(int w=0;w<n;w++)
{
if((a[v][w]==1)&&(visited[w]==0))
{
addq(&q,w);
visited[w]=1;
}
}
}
}
int main()
{
int a[10][10],n;
```

```
printf("\nEnter the no of vertex:");
scanf("%d",&n);
createlist(a,n);
bfs(a,n);
}
OUTPUT:
Enter the no of vertex:4
Type 1 for Yes and 0 for No
Is there edge between 1 and 2:1
Is there edge between 1 and 3:0
Is there edge between 1 and 4:1
Is there edge between 2 and 1:1
Is there edge between 2 and 3:0
Is there edge between 2 and 4:1
Is there edge between 3 and 1:1
Is there edge between 3 and 2:0
Is there edge between 3 and 4:0
Is there edge between 4 and 1:1
Is there edge between 4 and 2:1
Is there edge between 4 and 3:1
The breadth first traversal is:
v1 v2 v4 v3 */
```

Q2. C program to implement BST to perform following operations on BST1) Create
2) Recursive
traverse In-order .

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
```

```
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
}
}
void inorder(NODE *root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Inorder");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nInorder=");
inorder(root);
break;
case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice1
Enter how many nodes you want to create4
Enter the node info12
Enter the node info23
Enter the node info45
Enter the node info11
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice2
Inorder=11 12 23 45
```

SLIP-10

Q1. C program to implement graph traversal method using depth first search.

```
#include<stdio.h>
#include<stdlib.h>
void recdfs(int m[5][5],int n ,int v)
{
int w;
static int visited[20]={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);
}
}
void main()
{
int m[5][5]={{0,0,1,1,0},{0,0,1,0,1},{0,1,0,0,0},{0,0,0,0,1},{0,0,0,0,0}};
printf("\n the depth first search traverse is : ");
recdfs(m,5,0);
}
```

OUTPUT:

```
the depth first search traverse is : v1v3v2v5v4
```

Q2. C program to implement BST to perform following operations on BST- a) Create b) Counting leaf
nodes.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
```

```
}
}
int countleaf(NODE* root)
{
if(root==NULL)
return 0;
else
if((root->left==NULL)&&(root->right==NULL))
return 1;
else
return (countleaf(root->left)+countleaf(root->right));
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Count leaf nodes of bst");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nTotal leaf nodes of binary tree = %d ",countleaf(root));
break;
case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Count leaf nodes of bst
3.Exit
Enter your choice1
Enter how many nodes you want to create3
ENter the node info34
ENter the node info12
ENter the node info45
BINARY SEARCH TREE
1.Create bst
2.Count leaf nodes of bst
3.Exit
Enter your choice2
Total leaf nodes of binary tree = 2
BINARY SEARCH TREE
1.Create bst
2.Count leaf nodes of bst
3.Exit
Enter your choice
```

SLIP-11

Q1. C program to implement graph as adjacency matrix.

```
#include<stdio.h>
int main()
{
int a[10][10],i,j,n;
//Create
printf("\nEnter total no. of vertices: ");
scanf("%d",&n);
printf("\n*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge
not present)*\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
a[i][j]=0;
if(i!=j)
{
printf("\nIs there edge between v%d & v%d: ",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
}
//Display
printf("\n Matrix is \n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
}
```

OUTPUT:

```
Enter total no. of vertex: 3
Is there edge between 1 & 2: 1
Is there edge between 1 & 3: 0
Is there edge between 2 & 1: 1
Is there edge between 2 & 3: 1
Is there edge between 3 & 1: 0
Is there edge between 3 & 2: 1
Matrix is
0 1 0
1 0 1
0 1 0
```

Q2. C program to implement BST to perform following operations on BSTa) insert b) delete.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
```

```
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
}
}
void inorder(NODE *root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
NODE* delete_BST (NODE *root, int n)
{
NODE *temp, *succ ;
if(root== NULL)
{
printf("\n No. not found");
return (root);
}
if (n<root->info)
root->left-delete_BST(root->left,n);
else
if (n>root->info)
root->right=delete_BST(root->right,n);
else
{
if (root->left!= NULL && root->right!=NULL)
{
succ=root->right;
while (succ->left)
succ= succ->left;
```

```
root->info=succ->info;
root->right=delete_BST(root->right, succ->info);
}
else
{
temp-root;
if(root->left != NULL)
root-root->left;
else if (root->right!= NULL)
root=root->right;
else
root=NULL;
free (temp);
}
}
return (root);
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Deleting node from bst");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
```

```
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nEnter element to be deleted from BST");
scanf("%d",&n);
temp=delete_BST(root,n);
printf("\nResult after deletion is :\n");
inorder(root);
break;
case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Deleting node from bst
3.Exit
Enter your choice1
Enter how many nodes you want to create4
ENter the node info13
ENter the node info45
ENter the node info78
ENter the node info99
BINARY SEARCH TREE
1.Create bst
2.Deleting node from bst
3.Exit
Enter your choice2
Enter element to be deleted from BST99
```

```
13 45 78
BINARY SEARCH TREE
1.Create bst
2.Deleting node from bst
3.Exit
Enter your choice3
```

SLIP-12

Q1. C program to implement graph as adjacency List.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
int v;
struct node *next;
}NODE;
NODE *list[10]; //array of pointers
void createlist(int m[10][10],int n)
{
int i,j;
printf("\n*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge
not present)*\n");
for(i=0;i<n;i++)
{
// list[i]=NULL;
for(j=0;j<n;j++)
{
if(i!=j)
{
printf("\nIs there edge between v%d & v%d : ",i+1,j+1);
scanf("%d",&m[i][j]);
}
}
}
```

```
}
void adjlist(int m[10][10],int n)
{
int i,j;
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->v=j+1;
newnode->next=NULL;
if(list[i]==NULL)
list[i]=temp=newnode;
else
{
temp->next=newnode;
temp=newnode;
}
}
}
}
}
void displaylist(int n)
{
NODE *temp;
int i;
printf("\n The adjecancy list is:\n");
for(i=0;i<n;i++)
{
printf("\nv%d->",i+1);
temp=list[i];
while(temp)
{
printf("v%d->", temp->v);
temp=temp->next;
}
printf("NULL");
}
}
void main()
{
int m[10][10],n;
printf("\nEnter the number of vertices:");
scanf("%d",&n);
createlist(m,n);
adjlist(m,n);
displaylist(n);
}
```

OUTPUT:

```
Enter the number of vertices:4
*PRESS 1 FOR YES(edge present) & 0 FOR NO(edge not present)*
Is there edge between v1 & v2 : 1
Is there edge between v1 & v3 : 1
Is there edge between v1 & v4 : 0
Is there edge between v2 & v1 : 0
Is there edge between v2 & v3 : 1
Is there edge between v2 & v4 : 1
Is there edge between v3 & v1 : 0
Is there edge between v3 & v2 : 0
Is there edge between v3 & v4 : 1
Is there edge between v4 & v1 : 0
Is there edge between v4 & v2 : 1
Is there edge between v4 & v3 : 0
The adjecancy list is:
v1->v2->v3->NULL
v2->v3->v4->NULL
v3->v4->NULL
v4->v2->NULL
```

Q2. C program to implement BST to perform following operations on BST- a) Create
b) Counting Total
nodes

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
}
}
```

```
int countnode(NODE* root)
{
static int count;
if(root==NULL)
return 0;
else
return (1+countnode(root->left)+countnode(root->right));
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Count total nodes of bst");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nTotal nodes of BST = %d ",countnode(root));
break;
case 3:
```

```
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

```
OUTPUT:BINARY SEARCH TREE
1.Create bst
2.Count total nodes of bst
3.Exit
Enter your choice1
Enter how many nodes you want to create4
ENter the node info34
ENter the node info23
ENter the node info65
ENter the node info78
BINARY SEARCH TREE
1.Create bst
2.Count total nodes of bst
3.Exit
Enter your choice2
Total nodes of BST = 4
```

SLIP-13

Q1. C program to implement BST to perform following operations on BSTa) Create b) insert

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
int info;
struct node*left,*right;
} NODE;
```

```
NODE *createbst(NODE *root,int item)
{
if(root==NULL)
{
root=(NODE *)malloc(sizeof(NODE));
root->left=root->right=NULL;
root->info=item;
return root;
}
else
{
if(item<root->info)
root->left=createbst(root->left, item);
else
{
if(item>root->info)
root->right=createbst(root->right,item);
else
printf("Duplicate element is not allowed");
}
return root;
}
}
void inorder(NODE *root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
int main()
{
int i,n,choice,item,key;
NODE *temp,*root=NULL;
while(1)
{
printf("\nBINARY SEARCH TREE");
printf("\n1.Create bst");
printf("\n2.Inorder");
printf("\n3.Exit");
printf("\nEnter your choice");
scanf("%d",&choice);
switch (choice)
{
case 1:
printf("Enter how many nodes you want to create");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nENter the node info");
scanf("%d",&item);
root=createbst(root,item);
}
break;
case 2:
printf("\nInorder=");
inorder(root);
break;case 3:
exit(0);
default:
printf("Wrong choice entered....");
}
}
}
```

OUTPUT:

```
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice1
Enter how many nodes you want to create4
ENter the node info12
ENter the node info43
ENter the node info75
ENter the node info99
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice2
Inorder=12 43 75 99
BINARY SEARCH TREE
1.Create bst
2.Inorder
3.Exit
Enter your choice3
```

Q 2. C program to calculate in-degree and out-degree and total degree of all vertices in a graph .

```
#include<stdio.h>
void main()
{
int a[10][10],n,j,i,out=0,in=0;
printf("enter the how many vertex");
scanf("%d",&n);
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
a[i][j]=0;
if(i!=j)
{
printf("is there edge bet %d and %d",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
}
printf("Vertex Indegree Outdegree\n");
for(i=0;i<n;i++)
{
in=out=0;
for(j=0;j<n;j++)
{
in=in+a[j][i]; // count for indegree at vertex i
out=out+a[i][j]; // count for outdegree from vertex i
}
printf(" v%3d%14d%10d",i+1,in,out);
printf("\n");
}
}
```

OUTPUT:

```
enter the how many vertex6
is there edge bet 1 and 2
0
is there edge bet 1 and 30
is there edge bet 1 and 40
is there edge bet 1 and 50
is there edge bet 1 and 60
is there edge bet 2 and 11
is there edge bet 2 and 3
0
is there edge bet 2 and 41
is there edge bet 2 and 50
is there edge bet 2 and 60
is there edge bet 3 and 10
is there edge bet 3 and 21
is there edge bet 3 and 40
is there edge bet 3 and 50
is there edge bet 3 and 61
is there edge bet 4 and 10
is there edge bet 4 and 20
is there edge bet 4 and 31
is there edge bet 4 and 51
is there edge bet 4 and 61
is there edge bet 5 and 11
is there edge bet 5 and 20
is there edge bet 5 and 30
is there edge bet 5 and 40
is there edge bet 5 and 61
is there edge bet 6 and 11
is there edge bet 6 and 21
is there edge bet 6 and 30
is there edge bet 6 and 40
is there edge bet 6 and 50
vertex indegree oudegree
 1 3 0
 2 2 2
 3 1 2
 4 1 3
 5 1 2
 6 3 2
```

SLIP-14

Q 1. Implementation of static hash table with Linear Probing.

```
//Hashing using linear probing : C program
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
int key,index,i,flag=0,hkey;
printf("\nenter a value to insert into hash table\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
```

```
for(i=0;i<TABLE_SIZE;i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index] == NULL)
{
h[index]=key;
break;
}
}
if(i == TABLE_SIZE)
printf("\nelement cannot be inserted\n");
}
void search()
{
int key,index,i,flag=0,hkey;
printf("\nenter search element\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE; i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index]==key)
{
printf("value is found at index %d",index);
break;
}
}
if(i == TABLE_SIZE)
printf("\n value is not found\n");
}
void display()
{
int i;
printf("\nelements in the hash table are \n");
for(i=0;i< TABLE_SIZE; i++)
printf("\nat index %d \t value = %d",i,h[i]);
```

```
}
main()
{
int opt,i;
while(1)
{
printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
scanf("%d",&opt);
switch(opt)
{
case 1:
insert();
break;
case 2:
display();
break;
case 3:
search();
break;
case 4:exit(0);
}
}
}
```

OUTPUT:

```
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
12
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
13
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
22
```

```
Press 1. Insert 2. Display 3. Search 4.Exit
2
elements in the hash table are
at index 0 value = 0
at index 1 value = 0
at index 2 value = 12
at index 3 value = 13
at index 4 value = 22
at index 5 value = 0
at index 6 value = 0
at index 7 value = 0
at index 8 value = 0
at index 9 value = 0
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
12
value is found at index 2
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
23
value is not found
Press 1. Insert 2. Display 3. Search 4.Exit
```

Q 2. C program to calculate in-degree and out-degree and total degree of all vertices in a graph .

```
#include<stdio.h>
void main()
{
int a[10][10],n,j,i,out=0,in=0;
printf("enter the how many vertex");
scanf("%d",&n);
for(i=0;i<n;i++)
{
```

```
for(j=0;j<n;j++)
{
a[i][j]=0;
if(i!=j)
{
printf("is there edge bet %d and %d",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
}
printf("Vertex Indegree Outdegree\n");
for(i=0;i<n;i++)
{
in=out=0;
for(j=0;j<n;j++)
{
in=in+a[j][i]; // count for indegree at vertex i
out=out+a[i][j]; // count for outdegree from vertex i
}
printf(" v%3d%14d%10d",i+1,in,out);
printf("\n");
}
}
```

OUTPUT:

```
enter the how many vertex6
is there edge bet 1 and 2
0
is there edge bet 1 and 30
is there edge bet 1 and 40
is there edge bet 1 and 50
is there edge bet 1 and 60
is there edge bet 2 and 11
is there edge bet 2 and 3
0
is there edge bet 2 and 41
is there edge bet 2 and 50
is there edge bet 2 and 60
is there edge bet 3 and 10
is there edge bet 3 and 21
is there edge bet 3 and 40
is there edge bet 3 and 50
is there edge bet 3 and 61
is there edge bet 4 and 10
is there edge bet 4 and 20
is there edge bet 4 and 31
is there edge bet 4 and 51
is there edge bet 4 and 61
is there edge bet 5 and 11
is there edge bet 5 and 20
is there edge bet 5 and 30
is there edge bet 5 and 40
is there edge bet 5 and 61
is there edge bet 6 and 11
is there edge bet 6 and 21
is there edge bet 6 and 30
is there edge bet 6 and 40
is there edge bet 6 and 50
vertex indegree oudegree
 1 3 0
 2 2 2
 3 1 2
 4 1 3
 5 1 2
 6 3 2
```

SLIP-15

Q 1. C program to implement graph traversal method using depth first search.

```
#include<stdio.h>
#include<stdlib.h>
void recdfs(int m[5][5],int n ,int v)
```

```
{
int w;
static int visited[20]={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);
}
}
void main()
{
int m[5][5]={{0,0,1,1,0},{0,0,1,0,1},{0,1,0,0,0},{0,0,0,0,1},{0,0,0,0,0}};
printf("\n the depth first search traverse is : ");
recdfs(m,5,0);
}
```

OUTPUT:

```
the depth first search traverse is : v1v3v2v5v4
```

Q 2. Implementation of static hash table with Linear Probing.

```
//Hashing using linear probing : C program
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
int key,index,i,flag=0,hkey;
printf("\nenter a value to insert into hash table\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE;i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index] == NULL)
{
h[index]=key;
break;
}
}
if(i == TABLE_SIZE)
printf("\nelement cannot be inserted\n");
}
void search()
{
int key,index,i,flag=0,hkey;
printf("\nenter search element\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE; i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index]==key)
{
printf("value is found at index %d",index);
break;
}
}
if(i == TABLE_SIZE)
printf("\n value is not found\n");
}
void display()
{
int i;
printf("\nelements in the hash table are \n");
for(i=0;i< TABLE_SIZE; i++)
printf("\nat index %d \t value = %d",i,h[i]);
}
main()
{
```

```
int opt,i;
while(1)
{
printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
scanf("%d",&opt);
switch(opt)
{
case 1:
insert();
break;
case 2:
display();
break;
case 3:
search();
break;
case 4:exit(0);
}
}
}
```

OUTPUT:

```
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
12
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
13
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
22
Press 1. Insert 2. Display 3. Search 4.Exit
2
elements in the hash table are
```

```
at index 0 value = 0
at index 1 value = 0
at index 2 value = 12
at index 3 value = 13
at index 4 value = 22
at index 5 value = 0
at index 6 value = 0
at index 7 value = 0
at index 8 value = 0
at index 9 value = 0
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
12
value is found at index 2
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
23
value is not found
Press 1. Insert 2. Display 3. Search 4.Exit
```