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

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*lchild;

int data;

struct node \*rchild;

};

typedef struct node NODE;

NODE \*getnode()

{

NODE \*temp;

temp=(NODE\*)malloc(sizeof(NODE));

printf("\n\n Enter the data : ");

scanf("%d",&temp->data);

temp->lchild=NULL;

temp->rchild=NULL;

return(temp);

}

NODE \*create()

{

NODE \*temp,\*ptr,\*root;

char ch;

root=NULL;

do

{

temp=getnode();

if(root==NULL)

root=temp;

else

{

ptr=root;

while(ptr!=NULL)

{

if(temp->data<ptr->data)

{

if(ptr->lchild==NULL)

{

ptr->lchild=temp;

break;

}

else

ptr=ptr->lchild;

}

else

{

if(ptr->rchild==NULL)

{

ptr->rchild=temp;

break;

}

else

ptr=ptr->rchild;

}

}//while

} //else

printf("\n Add More (Y/N)? : ");

scanf(" %c",&ch);

}while(ch=='Y' || ch=='y');

return(root);

}

int search(int num,NODE \*ptr)

{

while(ptr!=NULL)

{

if(num==ptr->data)

return 1;

if(num<ptr->data)

{

ptr=ptr->lchild;

}

if(num>ptr->data)

{

ptr=ptr->rchild;

}

}

return 0;

}

NODE \*insert(NODE \*temp, NODE \*root)

{

NODE \*ptr;

ptr=root;

if(ptr==NULL)

{

root=ptr=temp;

}

else

{

ptr=root;

while(ptr!=NULL)

{

if(temp->data<ptr->data)

{

if(ptr->lchild==NULL)

{

ptr->lchild=temp;

break;

}

else

ptr=ptr->lchild;

}

else

{

if(ptr->rchild==NULL)

{

ptr->rchild=temp;

break;

}

else

ptr=ptr->rchild;

}

}

return(root);

}

return(root);

}

void inorder(NODE \*ptr)

{

if(ptr!=NULL)

{

inorder(ptr->lchild);

printf(" %d",ptr->data);

inorder(ptr->rchild);

}

}

void preorder(NODE \*ptr)

{

if(ptr!=NULL)

{

printf(" %d",ptr->data);

preorder(ptr->lchild);

preorder(ptr->rchild);

}

}

void postorder(NODE \*ptr)

{

if(ptr!=NULL)

{

postorder(ptr->lchild);

postorder(ptr->rchild);

printf(" %d",ptr->data);

}

}

main()

{

int ch,num,t,abc;

NODE \*root;

NODE \*temp;

while(1)

{

printf("\nMain Menu");

printf("\n1: Create Binary search tree");

printf("\n2: Inorder traversal");

printf("\n3: Preorder traversal");

printf("\n4: postorder traversal");

printf("\n5: Search a value");

printf("\n6: Insert a value");

printf("\n7: Exit");

printf("\n Enter the choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: root=create();

break;

case 2:

printf("\nInorder traversal");

inorder(root);

break;

case 3:

printf("\nPreorder traversal: ");

preorder(root);

break;

case 4:

printf("\nPostorder Traversal: ");

postorder(root);

break;

case 5: printf("\nEnter the Value to be searched: ");

scanf("%d",&num);

t=search(num,root);

if(t==1)

printf("\nValue is found");

else

printf("\nValue is not found");

break;

case 6:

temp=getnode();

root=insert(temp,root);

break;

case 7:

exit(1);

}

}

}

b) Write a program which uses binary search tree library and counts the total nodes and total leaf nodes in the tree.

Int count(T) – returns the total number of nodes from BST

int countLeaf(T) – returns the total number of leaf nodes from BST

#include<stdio.h>

#include<stdlib.h>

int nodetotal=0,leaftotal=0;

struct node

{

struct node \*lchild;

int data;

struct node \*rchild;

};

typedef struct node NODE;

NODE \*getnode()

{

NODE \*temp;

temp=(NODE\*)malloc(sizeof(NODE));

printf("\n\n Enter the data : ");

scanf("%d",&temp->data);

temp->lchild=NULL;

temp->rchild=NULL;

return(temp);

}

NODE \*create()

{

NODE \*temp,\*ptr,\*root;

char ch;

root=NULL;

do

{

temp=getnode();

if(root==NULL)

root=temp;

else

{

ptr=root;

while(ptr!=NULL)

{

if(temp->data<ptr->data)

{

if(ptr->lchild==NULL)

{

ptr->lchild=temp;

break;

}

else

ptr=ptr->lchild;

}

else

{

if(ptr->rchild==NULL)

{

ptr->rchild=temp;

break;

}

else

ptr=ptr->rchild;

}

}//while

} //else

printf("\n Add More (Y/N)? : ");

scanf(" %c",&ch);

}while(ch=='Y' || ch=='y');

return(root);

}

int totalnode(NODE \*ptr)

{

if(ptr!=NULL)

{

nodetotal++;

totalnode(ptr->lchild);

totalnode(ptr->rchild);

}

return nodetotal;

}

int leafcount(NODE \*ptr)

{

if(ptr!=NULL)

{

leafcount(ptr->lchild);

if(ptr->lchild==NULL && ptr->rchild==NULL)

leaftotal++;

leafcount(ptr->rchild);

}

return leaftotal;

}

main()

{

NODE \*root;

root=create();

printf("\n");

nodetotal=totalnode(root);

leaftotal=leafcount(root);

printf("\n The total no. of nodes are : %d",nodetotal);

printf("\n\n The total no. of leaf nodes are : %d",leaftotal);

}

Set B

1. Write a C program which uses Binary search tree library and implements following function with recursion:

T copy(T) – create another BST which is exact copy of BST which is passed as parameter.

int compare(T1, T2) – compares two binary search trees and returns 1 if they are equal and 0 otherwise.

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*lchild;

int data;

struct node \*rchild;

};

typedef struct node NODE;

NODE \*getnode()

{

NODE \*temp;

temp=(NODE\*)malloc(sizeof(NODE));

printf("\n\n Enter the data : ");

scanf("%d",&temp->data);

temp->lchild=NULL;

temp->rchild=NULL;

return(temp);

}

NODE \*create()

{

NODE \*temp,\*ptr,\*root;

char ch;

root=NULL;

do

{

temp=getnode();

if(root==NULL)

root=temp;

else

{

ptr=root;

while(ptr!=NULL)

{

if(temp->data<ptr->data)

{

if(ptr->lchild==NULL)

{

ptr->lchild=temp;

break;

}

else

ptr=ptr->lchild;

}

else

{

if(ptr->rchild==NULL)

{

ptr->rchild=temp;

break;

}

else

ptr=ptr->rchild;

}

}//while

} //else

printf("\n Add More (Y/N)? : ");

scanf(" %c",&ch);

}while(ch=='Y' || ch=='y');

return(root);

}

NODE\* copy(NODE \*ptr)

{

NODE \*temp;

if(ptr!=NULL)

{

temp=(NODE\*)malloc(sizeof(NODE));

temp->data=ptr->data;

temp->lchild=copy(ptr->lchild);

temp->rchild=copy(ptr->rchild);

return(temp);

}

return NULL;

}

int compare(NODE \*ptr,NODE \*ptr1)

{

if(!ptr && !ptr1)

return 1;

if(ptr!=NULL && ptr1!=NULL && ptr->data==ptr1->data && compare(ptr->lchild,ptr1->lchild) && compare(ptr->rchild,ptr1->rchild))

return 1;

else

return 0;

}

void display(NODE \*ptr)

{

if(ptr!=NULL)

{

display(ptr->lchild);

printf(" %d",ptr->data);

display(ptr->rchild);

}

}

main()

{

NODE \*root,\*root1;

int t;

root=create();

printf("\nBinary search after creation: ");

display(root);

root1=copy(root);

printf("\nBinary search tree after copying: ");

display(root1);

root1=NULL;

root1=create(root1);

t=compare(root,root1);

if(t==1)

printf("\nBoth the trees are same.");

else

printf("\nTrees are not same.");

}

Set C

1. Write a C program which uses Binary search tree library and implements following two functions:

int sumodd(T) – returnssum of all odd numbers from BST //should be done by student

int sumeven(T) – returnssum of all even numbers from BST //should be done by student

mirror(T) – converts given tree into its mirror image

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*lchild;

int data;

struct node \*rchild;

};

typedef struct node NODE;

NODE \*getnode()

{

NODE \*temp;

temp=(NODE\*)malloc(sizeof(NODE));

printf("\n\n Enter the data : ");

scanf("%d",&temp->data);

temp->lchild=NULL;

temp->rchild=NULL;

return(temp);

}

NODE \*create()

{

NODE \*temp,\*ptr,\*root;

char ch;

root=NULL;

do

{

temp=getnode();

if(root==NULL)

root=temp;

else

{

ptr=root;

while(ptr!=NULL)

{

if(temp->data<ptr->data)

{

if(ptr->lchild==NULL)

{

ptr->lchild=temp;

break;

}

else

ptr=ptr->lchild;

}

else

{

if(ptr->rchild==NULL)

{

ptr->rchild=temp;

break;

}

else

ptr=ptr->rchild;

}

}//while

} //else

printf("\n Add More (Y/N)? : ");

scanf(" %c",&ch);

}while(ch=='Y' || ch=='y');

return(root);

}

/\*

void mirror(NODE \*ptr)

{

NODE \*temp;

if(ptr)

{

if(ptr->lchild)

mirror(ptr->lchild);

if(temp->rchild)

mirror(ptr->rchild);

temp=ptr->lchild;

ptr->lchild=ptr->rchild;

ptr->rchild=temp;

}

}

\*/

void display(NODE \*ptr)

{

if(ptr!=NULL)

{

display(ptr->lchild);

printf(" %d",ptr->data);

display(ptr->rchild);

}

}

void mirror(NODE \*h)

{

NODE \*ptr=h,\*temp;

if(ptr!=NULL)

{

if(ptr->lchild!=NULL)

mirror(ptr->lchild);

if(ptr->rchild!=NULL)

mirror(ptr->rchild);

temp1=ptr->lchild;

ptr->lchild=ptr->rchild;

ptr->rchild=temp1;

}

}

main()

{

NODE \*root,\*root1;

int t;

root=create();

printf("\nBinary search after creation: ");

display(root);

mirror(root);

printf("\nBinary search tree after mirror image: ");

display(root);

}

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>

int nov,a[20][20];

void creatematrix()

{

int i,j;

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

//accept the matrix

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

{

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

{

printf("\nIs there egde between V[%d] and V[%d]: ",i,j);

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

}

}

}

void display(int a[20][20]) //print the matrix

{

int i,j;

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

{

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

printf("\t%d",a[i][j]);

printf("\n");

}

}

main()

{

int ch;

creatematrix();

printf("\n\t\*\*\*Adjacency Matrix\*\*\*\*\n");

display(a);

}

1. 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>

int nov,a[20][20];

void creatematrix()

{

int i,j;

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

//accept the matrix

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

{

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

{

printf("\nIs there egede between V[%d] and V[%d]: ",i,j);

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

}

}

}

void display(int a[20][20]) //print the matrix

{

int i,j;

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

{

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

printf("\t%d",a[i][j]);

printf("\n");

}

}

void degree()

{

int i,j,indegree,outdegree;

printf("\nVertex\t Indegree\t Outdegree\t Totaldegree \n");

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

{

indegree=0;

outdegree=0;

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

{

if(a[i][j]==1)

outdegree+=1;

if(a[j][i]==1)

indegree+=1;

}

printf("\nV%d \t\t %d \t\t %d \t\t %d",i,indegree,outdegree,indegree+outdegree);

}

}

main()

{

int ch;

creatematrix();

printf("\n\t\*\*\*Adjacency Matrix\*\*\*\*\n");

display(a);

degree();

}

Set B

1. 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.

//bfs

#include<stdio.h>

struct queue

{

int front,rear;

int Q[20];

};

typedef struct queue QUEUE;

int nov,a[20][20];

int visited[20];

void initqueue(QUEUE \*q)

{

int i;

for(i=0;i<20;i++)

q->Q[i]=0;

q->rear=-1;

q->front=-1;

printf("\nQueue created");

}

void add(QUEUE \*q,int data)

{

q->Q[++q->rear]=data;

}

int delet(QUEUE \*q)

{

return(q->Q[++q->front]);

}

int isempty(QUEUE \*q)

{

if (q->rear==q->front)

return 1;

else

return 0;

}

void bfs(int a[20][20],int nov)

{

int ver,j;

QUEUE q;

initqueue(&q);

printf("\n \t BFS seq. :\n ");

ver=1;

add(&q,ver);

visited[ver]=1;

while(!isempty(&q))

{

ver=delet(&q);

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

{

if(a[ver][j]==1 && visited[j]==0)

{

add(&q,j);

visited[j]=1;

}

}

printf("\t V%d ",ver);

}

}

void creatematrix()

{

int i,j;

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

//accept the matrix

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

{

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

{

printf("\nIs there egde between V[%d] and V[%d]: ",i,j);

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

}

}

}

void display(int a[20][20]) //print the matrix

{

int i,j;

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

{

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

printf("\t%d",a[i][j]);

printf("\n");

}

}

main()

{

int ch,i;

creatematrix();

printf("\n\t\*\*\*Adjacency Matrix\*\*\*\*\n");

display(a);

printf("\nThe Depth First search Traversal(DFS) is:");

bfs(a,nov);

}

b) 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>

int nov,a[20][20];

int visited[20];

void creatematrix()

{

int i,j;

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

//accept the matrix

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

{

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

{

printf("\nIs there egde between V[%d] and V[%d]: ",i,j);

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

}

}

}

void display(int a[20][20]) //print the matrix

{

int i,j;

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

{

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

printf("\t%d",a[i][j]);

printf("\n");

}

}

void recdfs(int a[20][20],int nov,int ver)

{

int i;

visited[ver]=1;

printf(" V%d",ver);

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

{

if((a[ver][i]==1) && (visited[i]==0))

recdfs(a,nov,i);

}

}

main()

{

int ch,i;

creatematrix();

printf("\n\t\*\*\*Adjacency Matrix\*\*\*\*\n");

display(a);

printf("\nThe Depth First search Traversal(DFS) is:");

recdfs(a,nov,1);

}

1. Write a C program which uses Binary search tree library and displays nodes at each level, count of node at each level and total levels in the tree.

#include<stdio.h>

#include<stdlib.h>

int height;

struct node

{

struct node \*lchild;

int data;

struct node \*rchild;

};

typedef struct node NODE;

NODE \*getnode()

{

NODE \*temp;

temp=(NODE\*)malloc(sizeof(NODE));

printf("\n\n Enter the data : ");

scanf("%d",&temp->data);

temp->lchild=NULL;

temp->rchild=NULL;

return(temp);

}

NODE \*create()

{

NODE \*temp,\*ptr,\*root;

char ch;

root=NULL;

do

{

temp=getnode();

if(root==NULL)

root=temp;

else

{

ptr=root;

while(ptr!=NULL)

{

if(temp->data<ptr->data)

{

if(ptr->lchild==NULL)

{

ptr->lchild=temp;

break;

}

else

ptr=ptr->lchild;

}

else

{

if(ptr->rchild==NULL)

{

ptr->rchild=temp;

break;

}

else

ptr=ptr->rchild;

}

}//while

} //else

printf("\n Add More (Y/N)? : ");

scanf(" %c",&ch);

}while(ch=='Y' || ch=='y');

return(root);

}

int tree\_height(NODE \* ptr)

{

if (!ptr)

return 0;

else {

int left\_height = tree\_height(ptr->lchild);

int right\_height = tree\_height(ptr->rchild);

if (left\_height >= right\_height)

return left\_height + 1;

else

return right\_height + 1;

}

}

void print\_level(NODE \* ptr, int level)

{

if (!ptr)

return;

if (level == 0)

{

printf("%d -> ", ptr->data);

}

else

{

print\_level(ptr->lchild, level - 1);

print\_level(ptr->rchild, level - 1);

}

}

void print\_tree\_level\_order(NODE\* ptr)

{

int i;

if (!ptr)

return;

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

{

printf("\nLevel %d: ", i);

print\_level(ptr, i);

printf("\n");

}

printf("\n\n-----Complete Level Order Traversal:-----\n");

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

{

print\_level(ptr, i);

}

printf("\n");

}

int countnodelevel(NODE \*ptr,int level)

{

if(ptr==NULL)

return 0;

if(level==0)

return 1;

return countnodelevel(ptr->lchild,level-1) + countnodelevel(ptr->rchild,level-1);

}

main()

{

NODE \*root;

int i;

root=create();

printf("\n");

height=tree\_height(root);

printf("\nTotal Levels in the tree: %d",height);

print\_tree\_level\_order(root);

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

printf("\nNumber of nodes at [ %d ] Level :: %d\n",i,countnodelevel(root,i));

}

Set B

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

// Heap Sort in C

#include <stdio.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 left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < N && arr[left] > arr[largest])

largest = left;

if (right < N && arr[right] > arr[largest])

largest = right;

if (largest != i)

{

swap(&arr[i], &arr[largest]);

heapify(arr, N, largest);

}

}

void printheap(int arr[], int N)

{

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

void heapsort(int arr[], int N)

{

int i,pass=1;

// Build max heap

for (i = N / 2 - 1; i >= 0; i--)

heapify(arr, N, i);

printf("\nArray After Building Max Heap: ");

printheap(arr, N);

// swap 1st and last element

for (i = N - 1; i >= 0; i--)

{

swap(&arr[0], &arr[i]);

if(pass<N)

{

printf("\nSorted array after Pass %d: ",pass++);

printheap(arr, N);

}

heapify(arr, i, 0);

}

}

int main()

{

int arr[] = {26,5,77,1,61,11,59,15};

int N = sizeof(arr) / sizeof(arr[0]);

heapsort(arr, N);

printf("Sorted array is\n");

printheap(arr, N);

}

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

#include<stdio.h>

#include<stdlib.h>

struct NODE

{

int data;

struct NODE \*next;

};

typedef struct NODE node;

node \*list[10];

int nov;

node \*getnodenum(int vno)

{

node \*temp;

temp=(node\*) malloc(sizeof(node));

temp->data=vno;

temp->next=NULL;

return temp;

}

void display(node \*list[10])

{

int i;

node \*ptr;

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

{

printf("V%d ",i);

for(ptr=list[i];ptr!=NULL;ptr=ptr->next)

printf("%d->",ptr->data);

printf("NULL");

printf("\n");

}

}

void creatadjacencylist()

{

int i,j;

char ch;

node \*temp,\*last;

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

list[i]=NULL;

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

{

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

{

printf("\nIs there edge between V[%d] and V[%d] (Choose (y/n): ",i,j);

scanf(" %c",&ch);

if(ch=='Y' || ch=='y')

{

temp=getnodenum(j);

if(list[i]==NULL)

list[i]=temp;

else

{

for(last=list[i];last->next!=NULL;last=last->next);

last->next=temp;

}

}

}

}

//display(list);

}

main()

{

int i,j,a[20][20];

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

creatadjacencylist();

printf("\n\*\*\*\*\*\*Adjacency List\*\*\*\*\*\*\*\n");

display(list);

}

1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list. Implement functions to print indegree, outdegree and total degree of all vertex of graph.

#include<stdio.h>

#include<stdlib.h>

struct NODE

{

int data;

struct NODE \*next;

};

typedef struct NODE node;

node \*list[10];

int nov;

node \*getnodenum(int vno)

{

node \*temp;

temp=(node\*) malloc(sizeof(node));

temp->data=vno;

temp->next=NULL;

return temp;

}

void display(node \*list[10])

{

int i;

node \*ptr;

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

{

printf("V%d ",i);

for(ptr=list[i];ptr!=NULL;ptr=ptr->next)

printf("%d->",ptr->data);

printf("NULL");

printf("\n");

}

}

void creatadjacencylist()

{

int i,j;

char ch;

node \*temp,\*last;

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

list[i]=NULL;

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

{

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

{

printf("\nIs there edge between V[%d] and V[%d] (choose : y/n): ",i,j);

scanf(" %c",&ch);

if(ch=='Y' || ch=='y')

{

temp=getnodenum(j);

if(list[i]==NULL)

list[i]=temp;

else

{

for(last=list[i];last->next!=NULL;last=last->next);

last->next=temp;

}

}

}

}

}

void degree()

{

int i,cnt,outcnt[10],incnt[10]={0};

node \*ptr;

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

{

for(ptr=list[i],cnt=0;ptr!=NULL;ptr=ptr->next,cnt++)

incnt[ptr->data]+=1;

outcnt[i]=cnt;

}

printf("\nVertex\t Indegree\t Outdegree\t Totaldegree");

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

printf("\nV%d \t\t %d \t\t %d \t\t %d",i,incnt[i],outcnt[i],incnt[i]+outcnt[i]);

}

main()

{

int i,j,a[20][20];

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

creatadjacencylist();

printf("\n\*\*\*\*\*\*Adjacency List\*\*\*\*\*\*\*\n");

display(list);

degree();

}

Set B

1. 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>

struct NODE

{

int data;

struct NODE \*next;

};

typedef struct NODE node;

struct queue

{

int front,rear;

int Q[20];

};

typedef struct queue QUEUE;

int nov,a[20][20];

int visited[20];

void initqueue(QUEUE \*q)

{

int i;

for(i=0;i<20;i++)

q->Q[i]=0;

q->rear=-1;

q->front=-1;

printf("\nQueue created");

}

void add(QUEUE \*q,int data)

{

q->Q[++q->rear]=data;

}

int delet(QUEUE \*q)

{

return(q->Q[++q->front]);

}

int isempty(QUEUE \*q)

{

if (q->rear==q->front)

return 1;

else

return 0;

}

void bfs(node \*list[10],int nov)

{

int ver;

node \*ptr;

QUEUE q;

initqueue(&q);

printf("\n \t BFS seq. :\n ");

ver=1;

add(&q,ver);

visited[ver]=1;

while(!isempty(&q))

{

ver=delet(&q);

for(ptr=list[ver];ptr!=NULL;ptr=ptr->next)

{

if(visited[ptr->data]==0)

{

add(&q,ptr->data);

visited[ptr->data]=1;

}

}

printf("\t V%d ",ver);

}

}

node \*list[10];

int nov,visited[20];

node \*getnodenum(int vno)

{

node \*temp;

temp=(node\*) malloc(sizeof(node));

temp->data=vno;

temp->next=NULL;

return temp;

}

void display(node \*list[10])

{

int i;

node \*ptr;

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

{

printf("V%d ",i);

for(ptr=list[i];ptr!=NULL;ptr=ptr->next)

printf("%d->",ptr->data);

printf("NULL");

printf("\n");

}

}

void creatadjacencylist()

{

int i,j;

char ch;

node \*temp,\*last;

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

list[i]=NULL;

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

{

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

{

printf("\nIs there edge between V[%d] and V[%d] (Choose (y/n): ",i,j);

scanf(" %c",&ch);

if(ch=='Y' || ch=='y')

{

temp=getnodenum(j);

if(list[i]==NULL)

list[i]=temp;

else

{

for(last=list[i];last->next!=NULL;last=last->next);

last->next=temp;

}

}

}

}

}

main()

{

int i,j,a[20][20];

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

creatadjacencylist();

printf("\n\*\*\*\*\*\*Adjacency List\*\*\*\*\*\*\*\n");

display(list);

printf("\nDFS Sequence: ");

bfs(list,nov);

}

b) 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 Depth First Search (DFS) traversal.

#include<stdio.h>

#include<stdlib.h>

struct NODE

{

int data;

struct NODE \*next;

};

typedef struct NODE node;

node \*list[10];

int nov,visited[20];

node \*getnodenum(int vno)

{

node \*temp;

temp=(node\*) malloc(sizeof(node));

temp->data=vno;

temp->next=NULL;

return temp;

}

void display(node \*list[10])

{

int i;

node \*ptr;

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

{

printf("V%d ",i);

for(ptr=list[i];ptr!=NULL;ptr=ptr->next)

printf("%d->",ptr->data);

printf("NULL");

printf("\n");

}

}

void creatadjacencylist()

{

int i,j;

char ch;

node \*temp,\*last;

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

list[i]=NULL;

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

{

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

{

printf("\nIs there edge between V[%d] and V[%d] (Choose (y/n): ",i,j);

scanf(" %c",&ch);

if(ch=='Y' || ch=='y')

{

temp=getnodenum(j);

if(list[i]==NULL)

list[i]=temp;

else

{

for(last=list[i];last->next!=NULL;last=last->next);

last->next=temp;

}

}

}

}

}

void recdfs(node \*list[10],int nov,int ver)

{

int i;

node \*ptr;

visited[ver]=1;

printf(" V%d",ver);

//for(i=1;i<=nov;i++)

for(ptr=list[ver];ptr!=NULL;ptr=ptr->next)

{

if((visited[ptr->data]==0))

recdfs(list,nov,ptr->data);

}

}

main()

{

int i,j,a[20][20];

printf("\nEnter no. of vertices: ");

scanf("%d",&nov);

creatadjacencylist();

printf("\n\*\*\*\*\*\*Adjacency List\*\*\*\*\*\*\*\n");

display(list);

printf("\nDFS Sequence: ");

recdfs(list,nov,1);

}