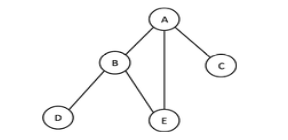
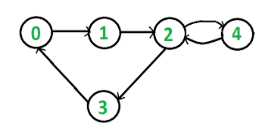
**SSN College of Engineering, Kalavakkam Department of Computer Science and Engineering III Semester - CSE 'A ',’B’ & ‘C’**  
**UCS 1312 Data Structures Lab**

Aditya Krishna P

185001010 || CSE A

**Exercise 11: Graph Traversal**

(i) Represent the graphs using adjacency matrix or adjacency list.  
(ii) Perform Depth first search(DFS) and Breadth first search(BFS) traversals by considering the start vertices as ‘A’ and ‘0’ for Graph 1 and Graph 2 respectively.



GRAPH 1 GRAPH 2

**//stack.h**

struct \_snode

{ int data;

struct \_snode \*next;

} \*top =NULL;

typedef struct \_snode snode;

int isempty()

{

if(top==NULL)

return 1;

else

return 0; }

void push (int x)

{ snode \*new = (snode \*) malloc (sizeof(snode));

new->data=x;

if(top==NULL)

new->next=NULL;

else

new->next=top;

top=new; }

int pop ()

{ int x=-1;

if(top==NULL)

printf("Empty stack\n");

else

{ snode \* temp=(snode\*) malloc(sizeof(snode));

temp=top;

top=temp->next;

x=temp->data;

free(temp); }

return x; }

void sdisplay ()

{ printf("STACk\n");

if(top==NULL)

printf("Empty stack\n");

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

temp=top;

while (temp!= NULL)

{ printf("%c \t", temp->data);

temp=temp->next; }

printf("\n"); }

void clearstack ()

{ while (top!=NULL)

pop(); }

**//queue.h**

typedef struct \_qnode

{

int data;

struct \_qnode \* next;

} qnode;

qnode \*front =NULL, \*rear= NULL;

int qisempty()

{ if(front == NULL)

return 1;

else

return 0; }

void enqueue (int x)

{ qnode \* new= (qnode \*) malloc (sizeof(qnode));

new->data=x;

new->next=NULL;

if(front == NULL)

{ front=rear=new; }

else

{ rear->next= new;

rear=new; } }

int dequeue ()

{ int x=-1;

if(front ==NULL)

printf ("Empty Queue \n");

else

{ qnode\* temp=front;

front=front->next;

x=temp->data;

free(temp); }

return x; }

void qdisplay(int x)

{ printf("---QUEUE ---\n");

if (front==NULL)

printf("Empty Queue\n");

else

{ qnode \* temp= front;

while(temp!=NULL)

{ printf("%d\n", temp->data);

temp=temp->next;

} } }

void clearqueue()

{ while(!qisempty())

dequeue(); }

**//graph.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "queue.h"

#include "stack.h"

#define NO1 5

#define NO2 5

char inttochar (int x)

{ char a;

switch (x)

{ case 0: a='A'; break;

case 1: a='B'; break;

case 2: a='C'; break;

case 3: a='D'; break;

case 4: a='E'; break;

} return a; }

int chartoint (char x)

{ int a;

switch (x)

{ case 'A': a=0; break;

case 'B': a=1; break;

case 'C': a=2; break;

case 'D': a=3; break;

case 'E': a=4; break; }

return a; }

void clearv (int visited[])

{ for (int i=0; i<5; i++)

visited[i]=0; }

void markandvisit (int node, int gno, int visited[])

{ if(gno==1) {

printf (" %c\t", inttochar(node));

visited[node]=1; }

if(gno==2) {

printf ("%d\t", node);

visited[node]=1; }

}

void dfs (int x, int gno, int g[5][5], int visited[])

{ clearstack();

clearv(visited);

int node;

int NO;

if(gno==1) NO=NO1;

if(gno==2) NO=NO2;

push(x);

while(!isempty())

{ node=pop();

if(visited[node]==0)

markandvisit(node, gno, visited);

for(int i=NO-1; i >=0; i--)

{ //adjacent to node and not been visited

if(g[node][i]==1 && visited[i]==0)

push(i);

} } }

int hasunvisitednode (int gno, int visited[]) {

int ind=0;

int NO;

if(gno==1) NO=NO1;

if(gno==2) NO=NO2;

for(int i=0; i<NO; i++)

if (visited[i]==0)

ind =1;

return ind; }

void bfs (int x, int gno, int g[5][5], int visited[]) {

clearv(visited);

clearqueue();

int node;

int NO;

if(gno==1) NO=NO1;

if(gno==2) NO=NO2;

markandvisit(x,gno, visited);

enqueue(x);

while (!qisempty())

{

node=dequeue();

for(int i=0; i <NO; i++)

{ //adjacent to node and not been visited

if(g[node][i]==1 && visited[i]==0) {

markandvisit(i, gno, visited);

enqueue(i);

} } }

}

int main()

{

int visited1[5], visited2[5];

int g1[5][5] = { {0,1,1,0,1} , {1,0,0,1,1} , {1,0,0,0,0}, {0,1,0,0,0}, {1,1,0,0,0}};

int g2[5][5] = { {0,1,0,0,0}, {0,0,1,0,0}, {0,0,0,1,1}, {1,0,0,0,0}, {0,0,1,0,0}};

printf("\nAdjacency Matrix of Graph 1 \n");

for(int i=0; i<NO1; i++) {

for(int j=0; j<NO1; j++)

printf("%d ", g1[i][j]);

printf("\n"); }

printf("\nDepth-First Traversal of Graph 1\n");

dfs(chartoint('A'),1, g1, visited1);

printf("\n\nBreadth-First Traversal of Graph 1\n");

bfs(chartoint('A'),1, g1, visited1);

printf("\n--------------------------------------------\n");

printf("\nAdjacency Matrix of Graph 2 \n");

for(int i=0; i<NO2; i++) {

for(int j=0; j<NO2; j++)

printf("%d ", g2[i][j]);

printf("\n"); }

printf("\nDepth-First Traversal of Graph 2\n");

dfs(0,2, g2, visited2);

printf("\n\nBreadth-First Traversal of Graph 2\n");

bfs(0,2, g2, visited2);

printf("\n");

return 0;

}

**OUTPUT**

Adjacency Matrix of Graph 1

0 1 1 0 1

1 0 0 1 1

1 0 0 0 0

0 1 0 0 0

1 1 0 0 0

Depth-First Traversal of Graph 1

A B D E C

Breadth-First Traversal of Graph 1

A B C E D

-----------------------------------------------

Adjacency Matrix of Graph 2

0 1 0 0 0

0 0 1 0 0

0 0 0 1 1

1 0 0 0 0

0 0 1 0 0

Depth-First Traversal of Graph 2

0 1 2 3 4

Breadth-First Traversal of Graph 2

0 1 2 3 4