K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: **A3** **Roll. No.: 16010121051**

Experiment:

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementation of Graph - insertion, search and traversal

Objective: To understand graph as data structure and methods of traversing Graph

Expected Outcome of Experiment:

CO	Outcome
CO2	Apply linear and non-linear data structure in application development

Websites/books referred:

Abstract: - (Definition of Graph, types of graphs, and difference and similarity between graph & tree)

Graph:

Types of Graph:

1. Null Graph

A graph is known as null graph if there are no edges in the graph.

2. Trivial Graph

Graph having only a single vertex, it is the smallest graph possible.

3. Undirected Graph

A graph in which edges do not have any direction. That is the nodes are unordered pairs in the definition of every edge.

4. Directed Graph

A graph in which edge has direction. That is the nodes are ordered pairs in the definition of every edge.

5. Connected Graph

The graph in which from one node we can visit any other node in the graph is known as a connected graph.

6. Disconnected Graph

The graph in which at least one node is not reachable from a node is known as a disconnected graph.

7. Regular Graph

The graph in which the degree of every vertex is equal to the other vertices of the graph.

Let the degree of each vertex be K then the graph is called K-regular.

8. Complete Graph

The graph in which from each node there is an edge to each other node.

9. Cycle Graph

The graph in which the graph is a cycle in itself, the degree of each vertex is 2.

10. Cyclic Graph

A graph containing at least one cycle is known as a Cyclic graph.

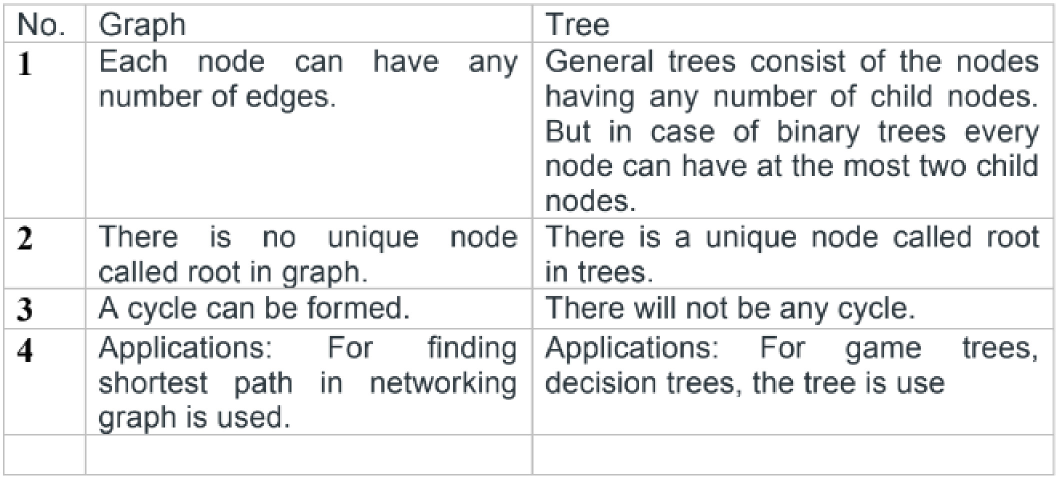
11. Directed Acyclic Graph

A Directed Graph that does not contain any cycle.

12. Bipartite graph

A graph in which vertex can be divided into two sets such that vertex in each set does not contain any edge between them.

Difference between graph and tree:



No.	Graph	Tree
1	Each node can have any number of edges.	General trees consist of the nodes having any number of child nodes. But in case of binary trees every node can have at the most two child nodes.
2	There is no unique node called root in graph.	There is a unique node called root in trees.
3	A cycle can be formed.	There will not be any cycle.
4	Applications: For finding shortest path in networking graph is used.	Applications: For game trees, decision trees, the tree is use

Similarity between graph and tree:

Both graph are non linear data structures consisting of nodes and vertices/edges.

Every node in both graph and tree has at least one data variable and a pointer.

.

Algorithm for DFS/BFS:

DFS

- **Mark all the visited array elements as false.**
- **Call DFS for first node.**
- **Mark the node as visited and print it**
- **Iterate through adjacency list (node) and if they are not visited then recursively call dfs for them.**
- **End**

BFS

- **Mark all the visited array elements as false.**
- **Call BFS for first node.**
- **Mark the node as visited and print it**
- **Iterate through adjacency list (node), if then they are not visited then mark them true and add then to the queue.**
- **If queue has elements the remove the top one and call BFS for that node.**
- **End.**

Code and output screenshots:

```
#include <stdio.h>

#include<stdlib.h>

int visit[20]={0};

int v[20]={0};

typedef struct node{

int data;

struct node *prev;

struct node *link;

}node;
```

```
typedef struct queue{

struct node *rr;

struct node *fr;

}que;




int dequeue(que *q){

node *temp;

if(q->rr!=NULL){

  temp=q->rr;

  int d=temp->data;

  q->rr=temp->prev;

  if(q->rr!=NULL)

  q->rr->link=NULL;
  else

  q->fr=NULL;

  return d;

}

 return 0;

}



void enqueue(int ch,que *q ){

node *nnode;

nnode=(node*)malloc(sizeof(node));

nnode->data=ch;

nnode->link=NULL;
```

```
nnode->prev=NULL;
if(q->fr==NULL){
q->fr=nnode;
q->rr=nnode;
}
else{
nnode->link=q->fr;
q->fr->prev=nnode;
q->fr=nnode;
}
}

void display(que *q){
```

```
node *temp;

temp=q->fr;

 while(temp!=NULL){

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

  temp=temp->link;

}

}



void dfs(int t,int a[20][20],int n){

int i,j;

printf("%d->",t);

visit[t-1]=1;
```

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

 if(a[t-1][i]==1 && visit[i]==0){

  dfs(i+1,a,n);

 }

}

}




void bfs(int t,int a[20][20],int n,que *q){

int i,j;

printf("%d->",t);

int temp;

enqueue(t,q);

v[t-1]=1;
```

```
while(q->fr!=NULL){

  temp=dequeue(q);

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

 if(a[temp-1][i]==1 && v[i]==0){

    enqueue(i+1,q);

    printf("%d->",i+1);

    v[i]=1;

 }




}

}

}
```

```
}
```

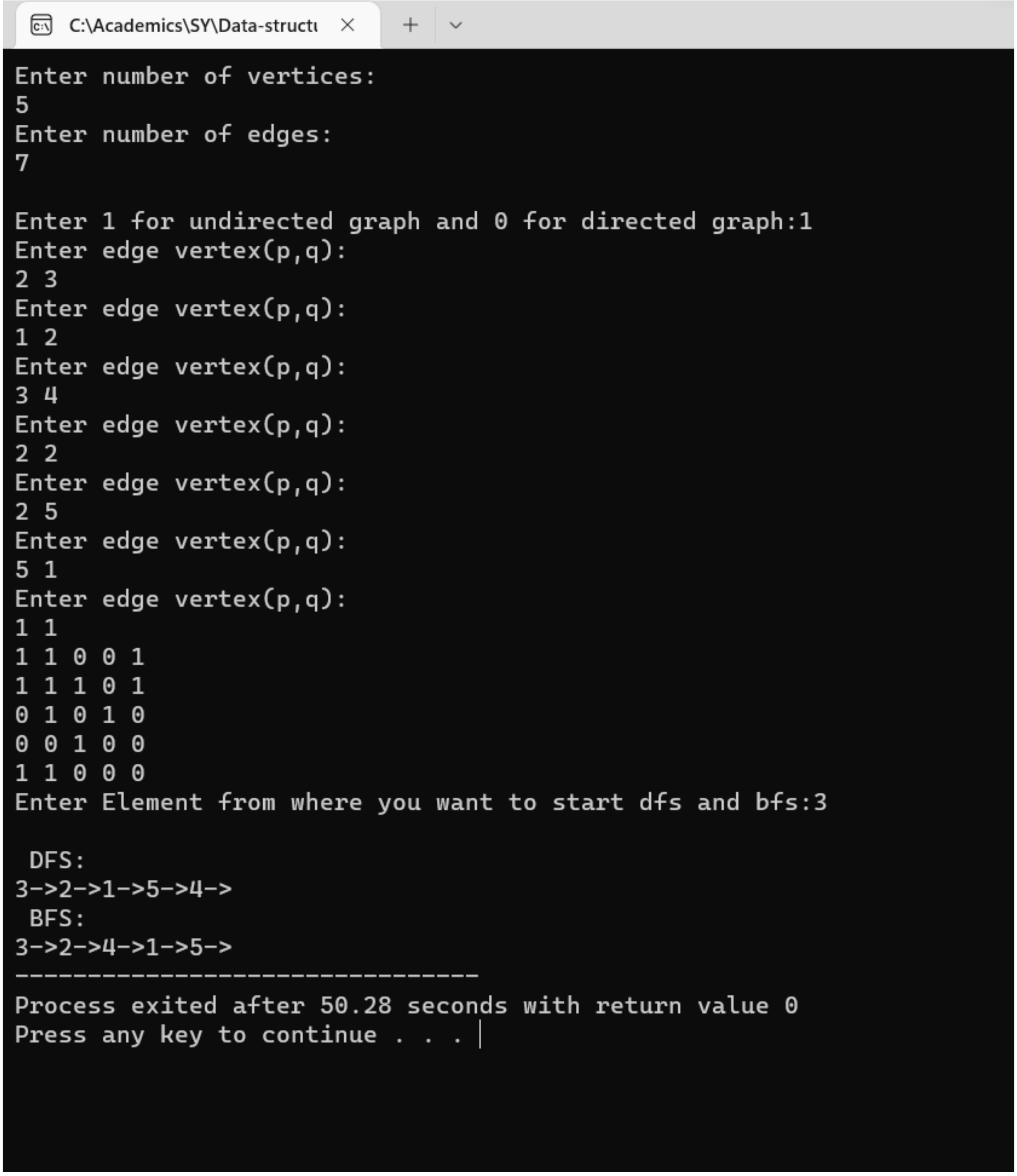
```
int main(void) {

printf("Enter number of vertices:\n");

int n,i,j,e,p,q;

scanf("%d",&n);

int a[20][20];

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

     visit[i]=0;

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

      a[i][j]=0;

    }

}
```

```
printf("Enter number of edges:\n");

scanf("%d",&e);

printf("\nEnter 1 for undirected graph and 0 for directed graph:");

int t;

scanf("%d",&t);

for(i=0;i<e;i++){

printf("Enter edge vertex(p,q):\n");

scanf("%d%d",&p,&q);

a[p-1][q-1]=1;

if(t==1)

a[q-1][p-1]=1;

}
```

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

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

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

  }

  printf("\n");

}

printf("Enter Element from where you want to start dfs and bfs:");

int d;

scanf("%d",&d);

printf("\n DFS:\n");

 dfs(d,a,n);
```

```
que q1;

q1.fr=q1.rr=NULL;

printf("\n BFS:\n");

bfs(d,a,n,&q1);

return 0;

}
```

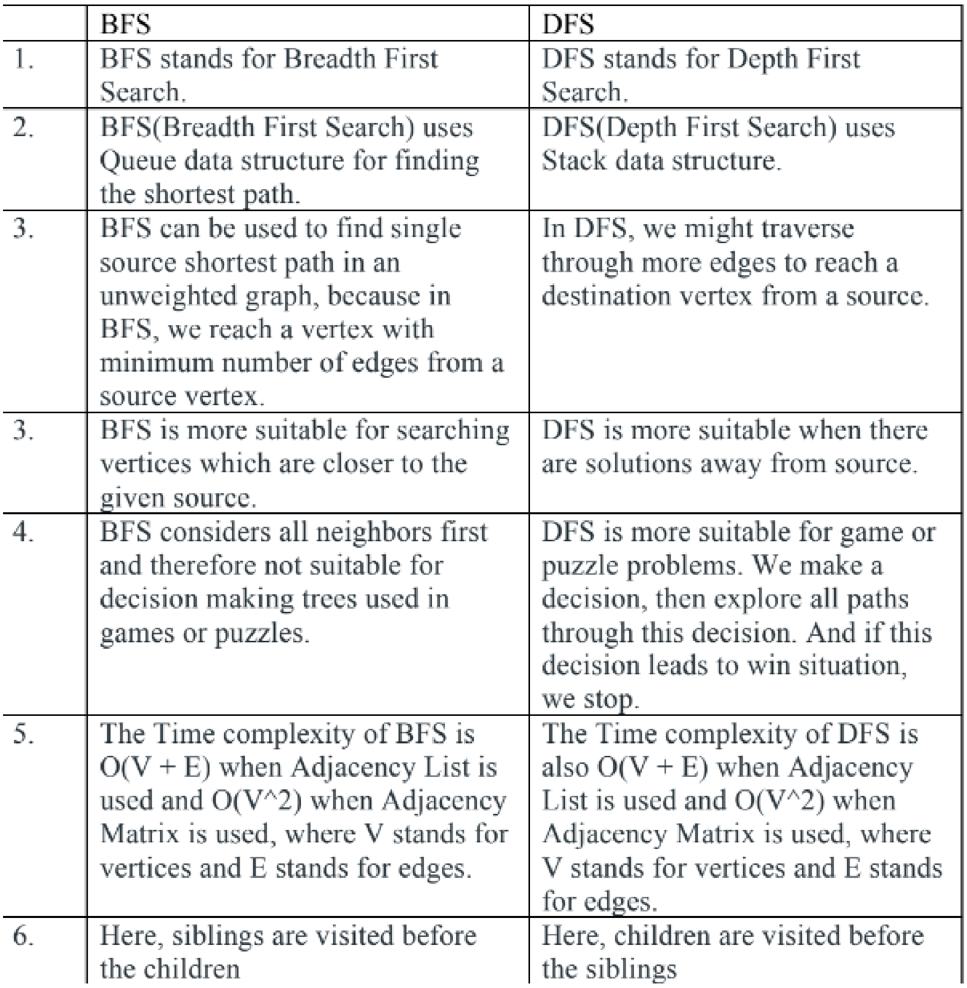
Output:



```
Enter number of vertices:
5
Enter number of edges:
7

Enter 1 for undirected graph and 0 for directed graph:1
Enter edge vertex(p,q):
2 3
Enter edge vertex(p,q):
1 2
Enter edge vertex(p,q):
3 4
Enter edge vertex(p,q):
2 2
Enter edge vertex(p,q):
2 5
Enter edge vertex(p,q):
5 1
Enter edge vertex(p,q):
1 1
1 1 0 0 1
1 1 1 0 1
0 1 0 1 0
0 0 1 0 0
1 1 0 0 0
Enter Element from where you want to start dfs and bfs:3

 DFS:
3->2->1->5->4->
 BFS:
3->2->4->1->5->
--------------------------------
Process exited after 50.28 seconds with return value 0
Press any key to continue . . .
```

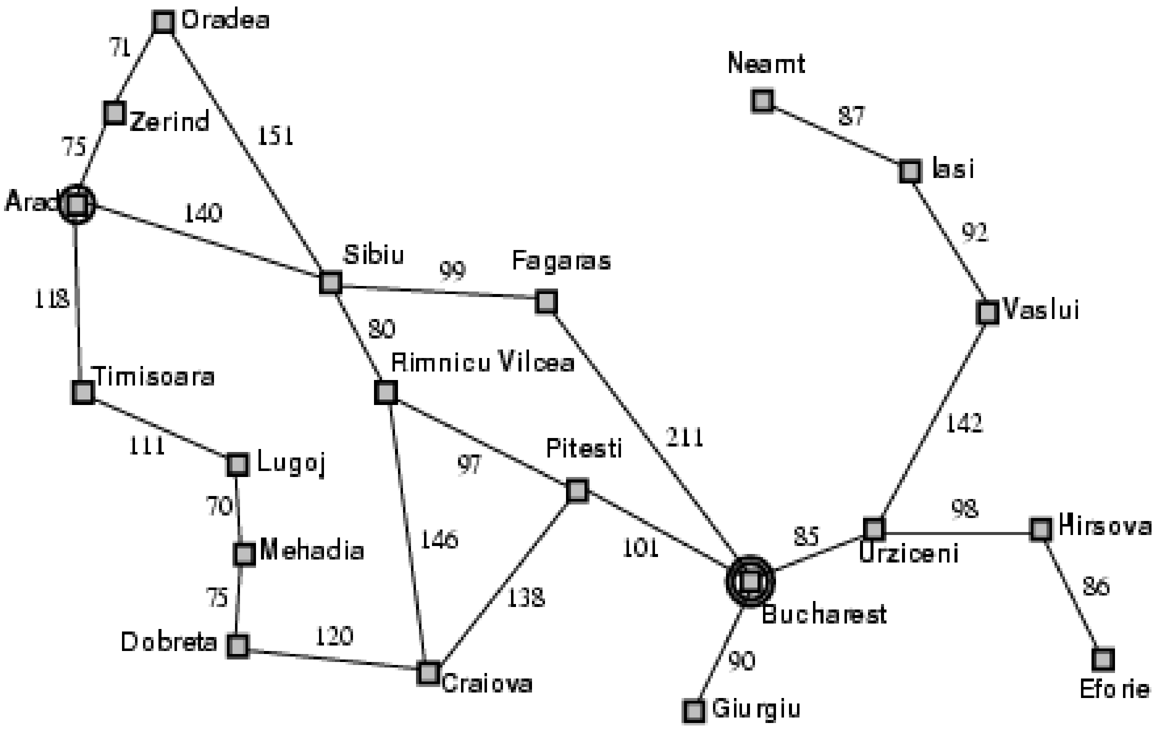
Post lab questions-

a. **Differentiate between BFS and DFS.**



	BFS	DFS
1.	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
2.	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
3.	BFS can be used to find single source shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
3.	BFS is more suitable for searching vertices which are closer to the given source.	DFS is more suitable when there are solutions away from source.
4.	BFS considers all neighbors first and therefore not suitable for decision making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
5.	The Time complexity of BFS is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.	The Time complexity of DFS is also O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.
6.	Here, siblings are visited before the children	Here, children are visited before the siblings

b. **Give sequence of the nodes visited as per BFS and DFS strategy for following example. Source- Arad, Destination- Bucharest (Traversal would stop after destination is reached)**



BFS – Arad->Zerind -> Sibiu -> Timisoara -> Oradea ->Fagaras -> Rimnicu Vilcea -> Lugoj -> Bucharest
DFS - Arad->Zerind->Oradea->Sibiu->Fagaras->Bucharest

Conclusion: -
In this experiment we learnt about two types of traversals in a graph and implemented them.