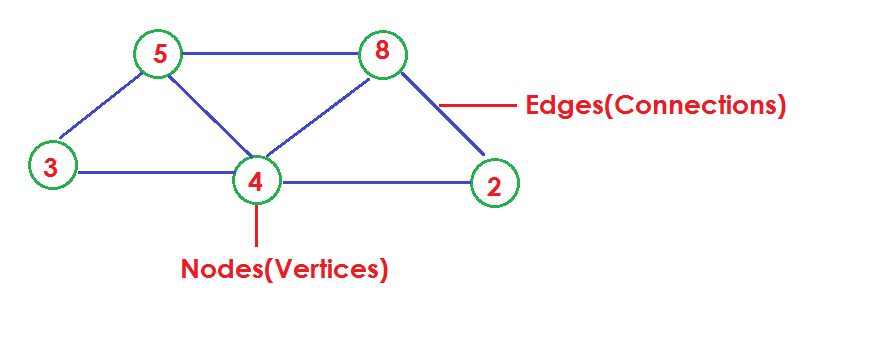
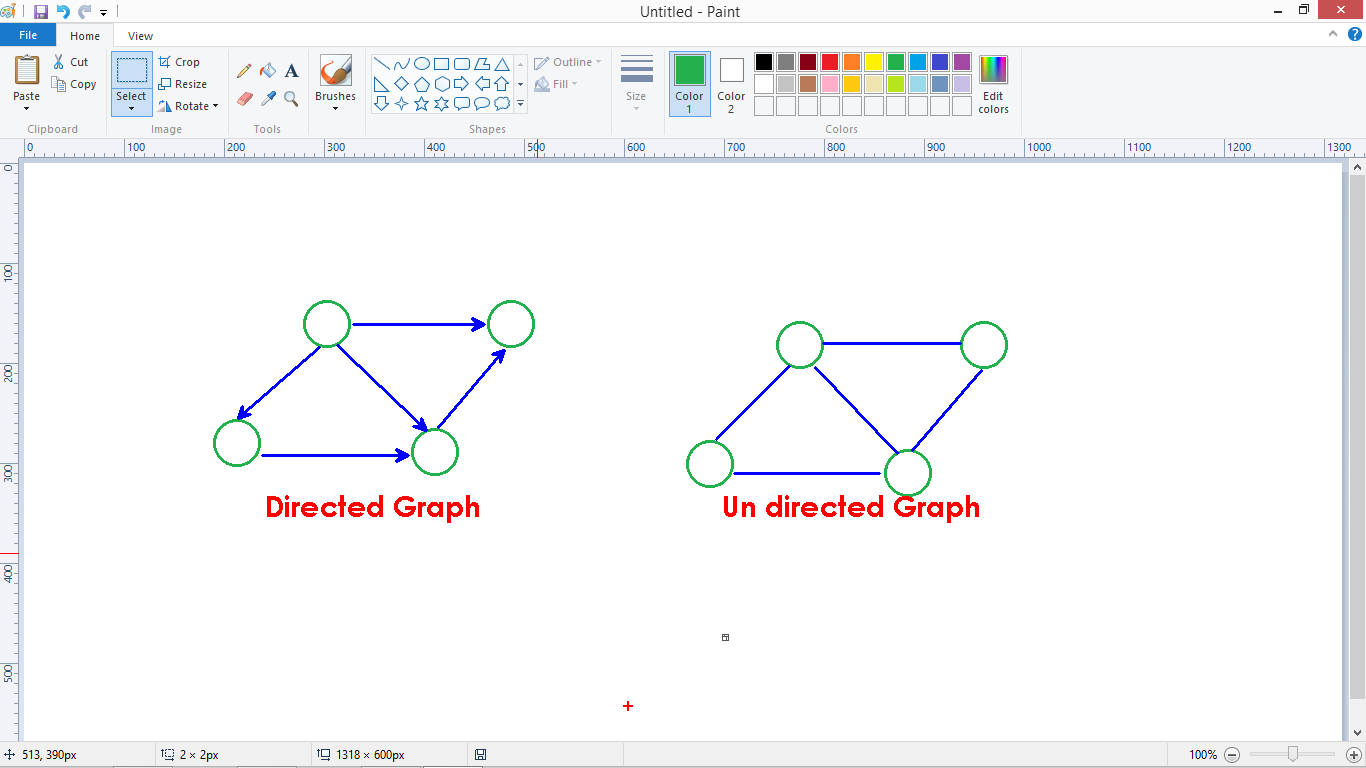
**Graphs**

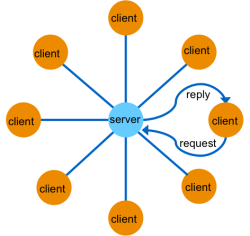
* Graph is a non linear data structure.
* A *graph* is a collection of nodes(called *vertices)*, and the connections between them, called *edges*.
* Graph is either Undirected or directed.



* When the edges in a graph have a direction, the graph is called a *directed graph*.
* The edges are called *directed edges* or *arcs*.



* Graphs are used to solve many real-life problems.
* Graphs are used to represent networks.
* The networks may include paths in a city or telephone network or circuit network.



* Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex (or node).
* Each node is a structure and contains information like person id, name, gender, locale etc.
* Let's try to understand this through an example. On facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note...anything that has data is a node.
* All of facebook is then a collection of these nodes and edges. This is because facebook uses a graph data structure to store its data.

**Graph:** A graph is a data structure {V, E} that consists of

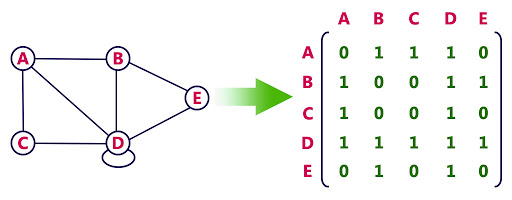
* A collection of vertices V
* A collection of edges E, represented as ordered pairs of vertices (u,v)

**Graph Terminology**

1. **Adjacency**: A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.
2. **Path**: A sequence of edges that allows you to go from vertex A to vertex B is called a path.

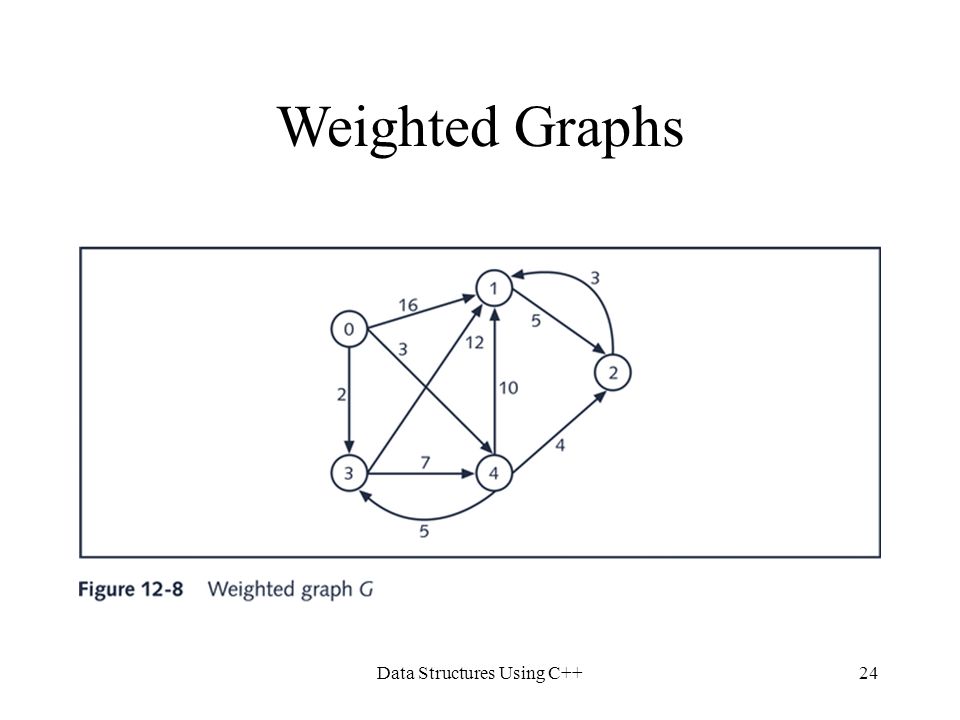
**Graph representation as matrix called Adjacency matrix:**

* An adjacency matrix is a 2D array of V x V vertices. Each row and column represents a vertex.
* If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.



**Traversal:** We use weight graphs and directions to find the BFS, DFS.

**Applications:** Using the weighted graph we can understand “Shortest path” problem” which is the application of Graph.



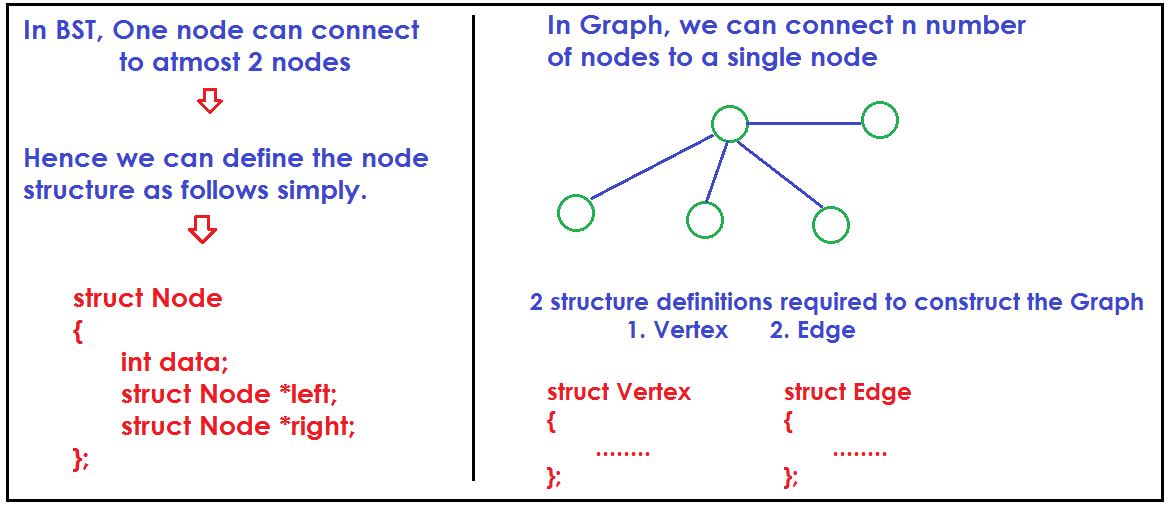
**Graph Operations:**

The most common graph operations are:

1. Check if the element is present in the graph
2. Graph Traversal
3. Add elements(vertex, edges) to graph
4. Finding the path from one vertex to another

**Structure representations:**

* We need to define 2 structures among one is representing the Node and the second one representing Edge.



struct Vertex

{

int data, status;

struct Vertex \*next;

struct Edge \*adj;

};

struct Edge

{

struct Vertex \*next;

struct Edge \*adj;

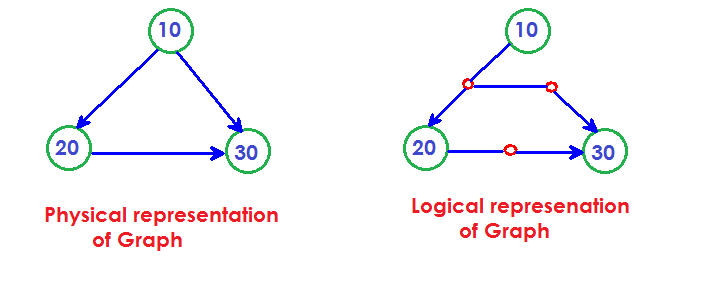
};

**Variables required constructing the Graph:**

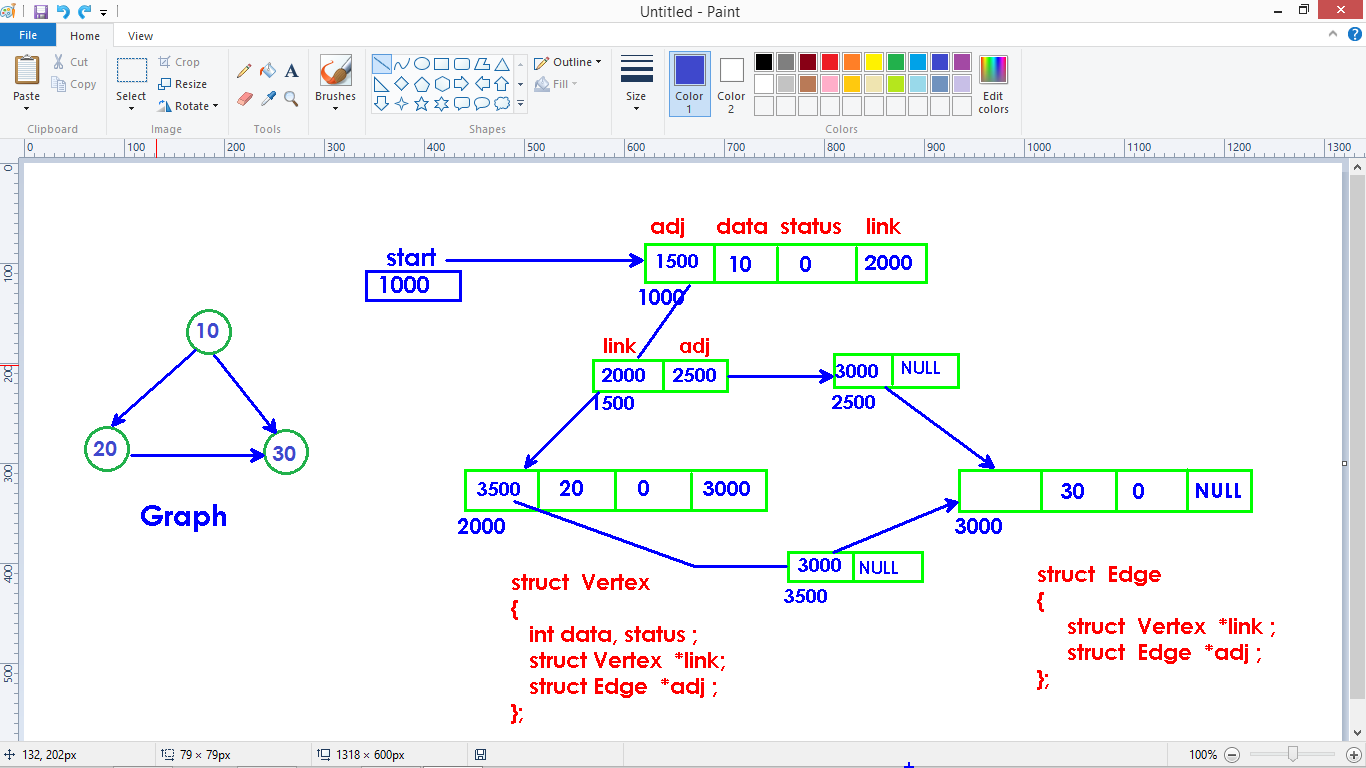
struct Vertex \*start,\*p,\*q;

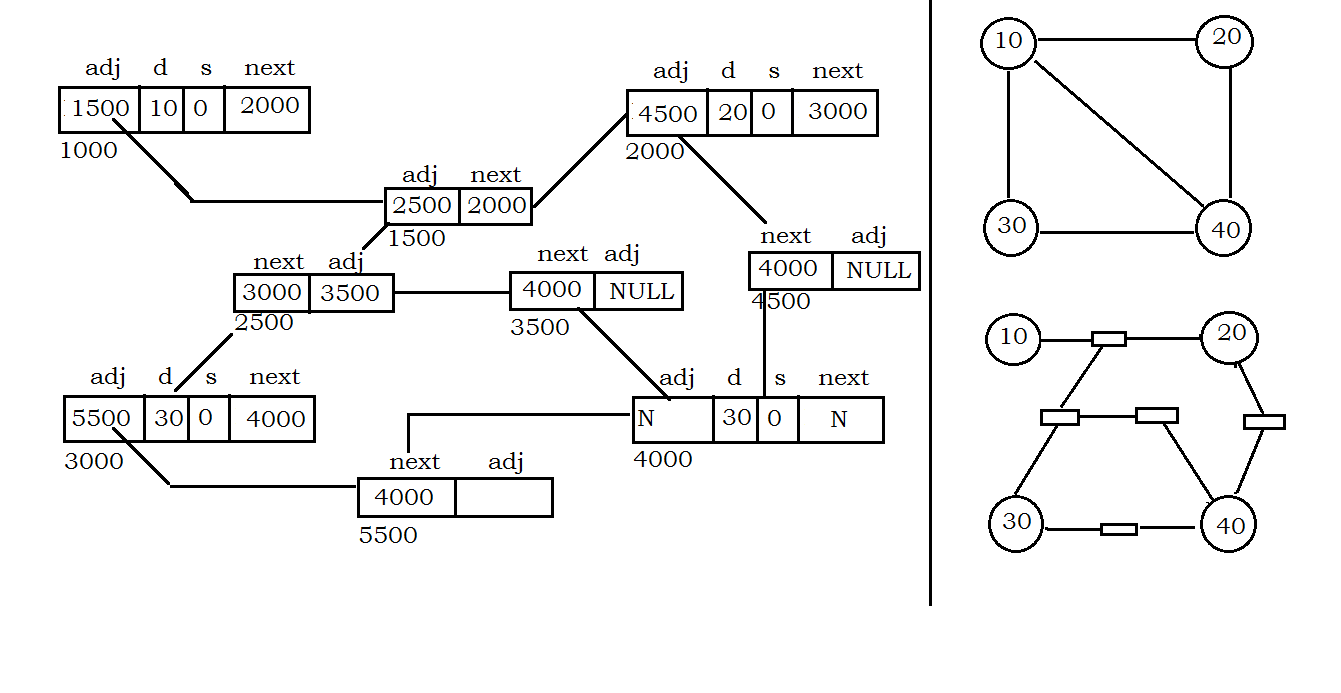
struct Edge \*l,\*k;

**Physical representation v/s Logical representation:**



**The internal representation of Graph structure:**





**void create()**

{

int data, flag=0;

start = NULL;

printf("Enter nodes (0 to stop) : \n");

while(1)

{

scanf("%d", &data);

if(data==0)

break;

p = (struct Vertex\*)malloc(sizeof(struct Vertex));

p->data = data;

p->status = 0;

p->next = NULL;

p->adj = NULL ;

if(flag==0)

{

start = p;

q = p;

flag++;

}

else

{

q->link = p;

q = p;

}

}

p = start;

while(p)

{

printf("Enter links to %d : (0 to stop) :", p->data);

flag=0;

while(1)

{

scanf("%d", &data);

if(data==0)

break;

k = (struct Edge\*)malloc(sizeof(struct Edge));

k->adj = NULL;

if(flag==0)

{

p->adj = k;

l = k;

flag++;

}

else

{

l->adj = k;

l = k;

}

q = start;

while(q)

{

if(q->data == data)

{

k->next = q;

break;

}

q=q->next;

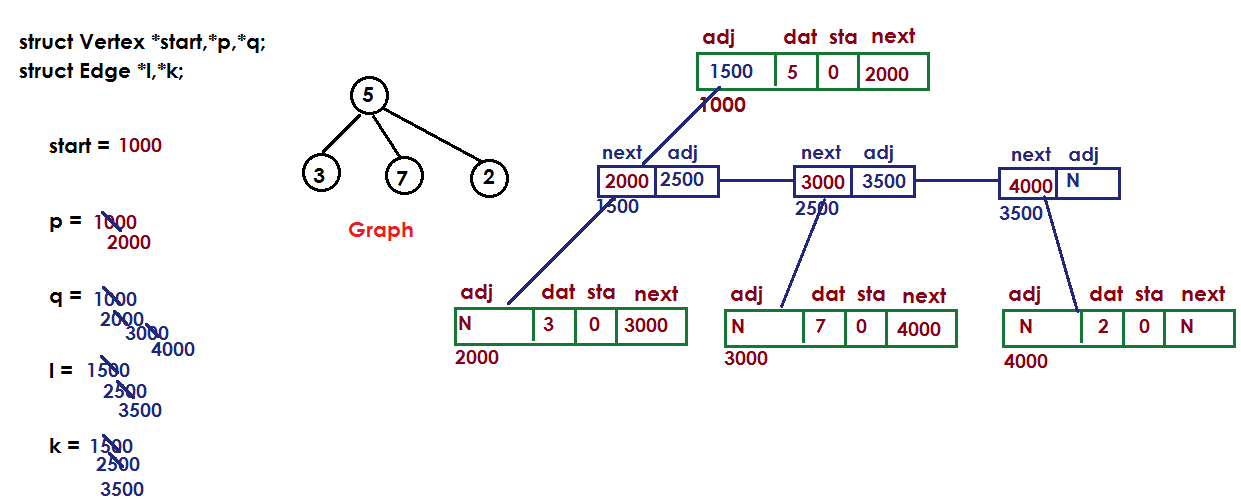
}

}

p=p->next;

}

}

****

**Graph traversals**

* Graph traversal means visiting every vertex and edge exactly once in a well-defined order.
* While using certain graph algorithms, you must ensure that each vertex of the graph is visited exactly once.
* The order in which the vertices are visited are important and may depend upon the algorithm or question that you are solving.

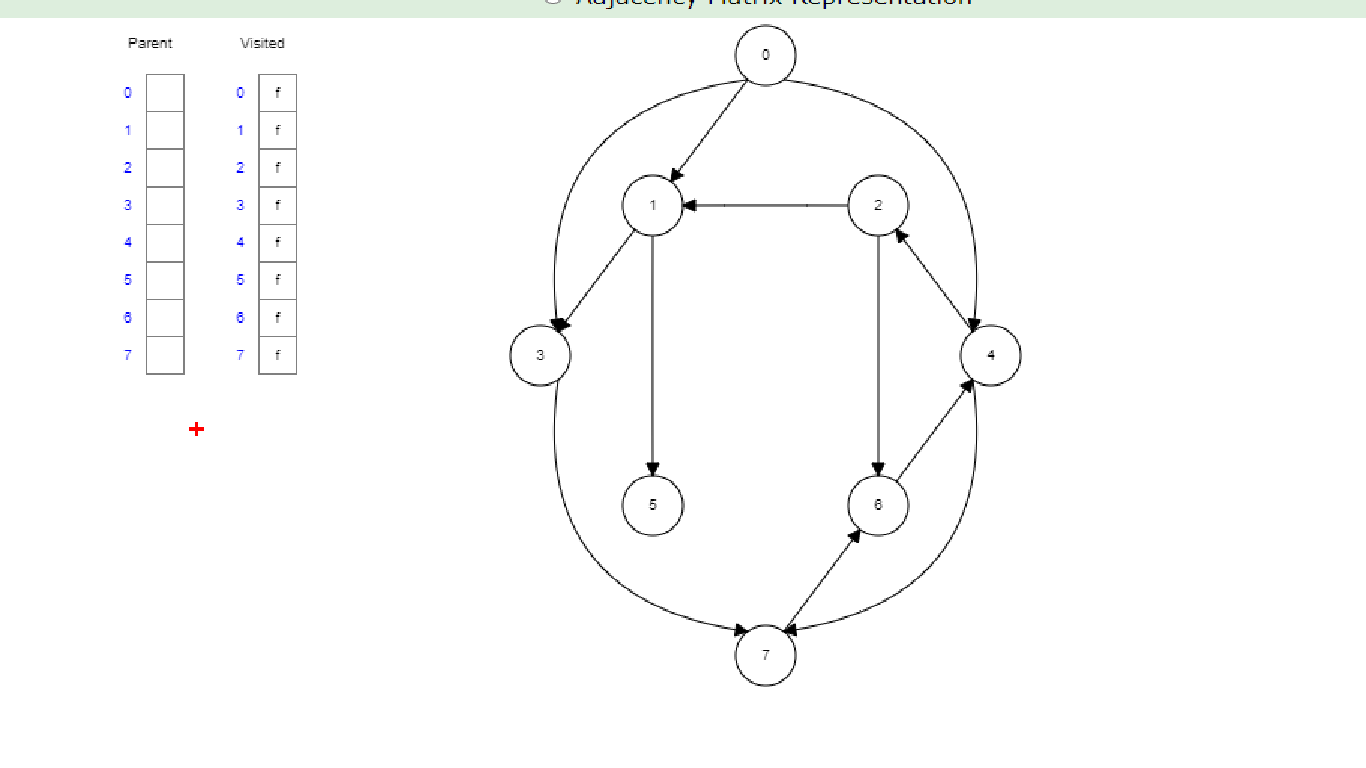
**Breadth First Search (BFS):**

* BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbor nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbor nodes.
* As the name BFS suggests, you are required to traverse the graph breadth wise as follows:

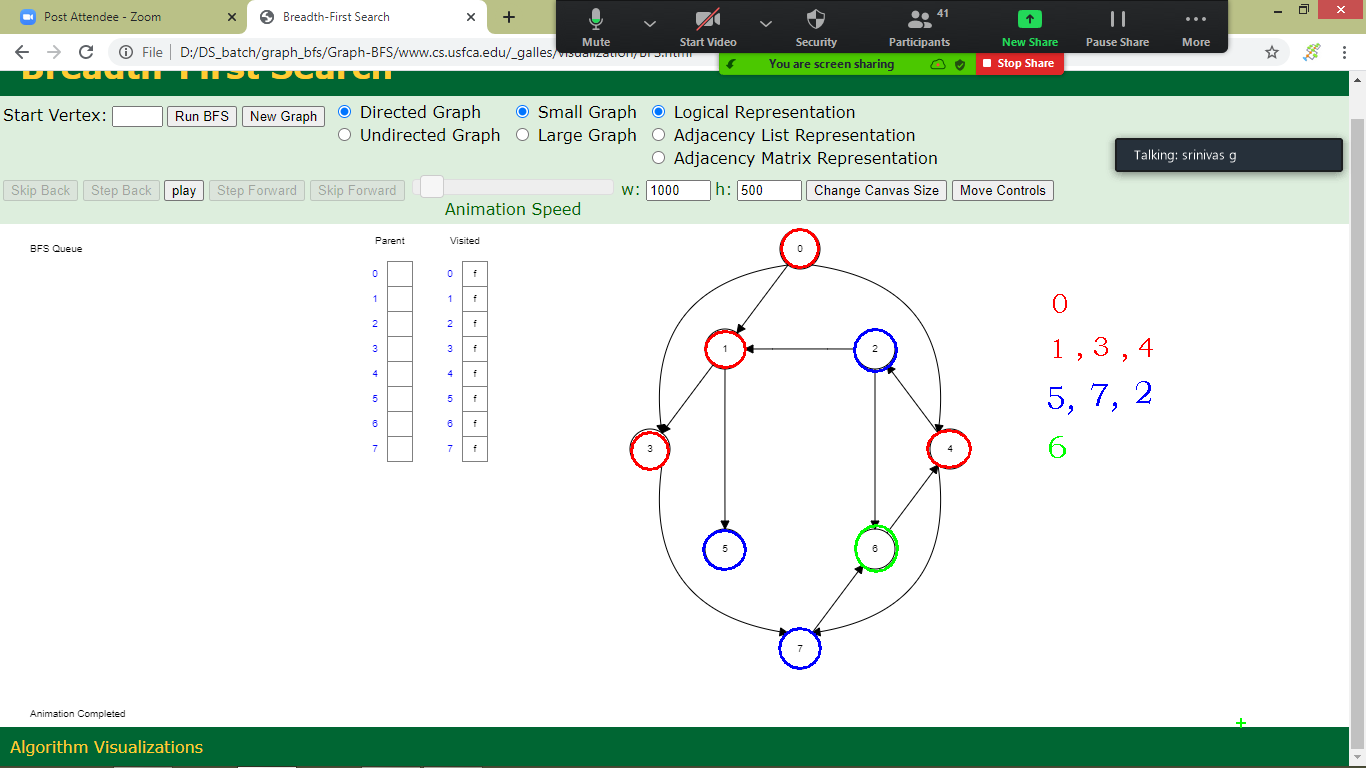
1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer



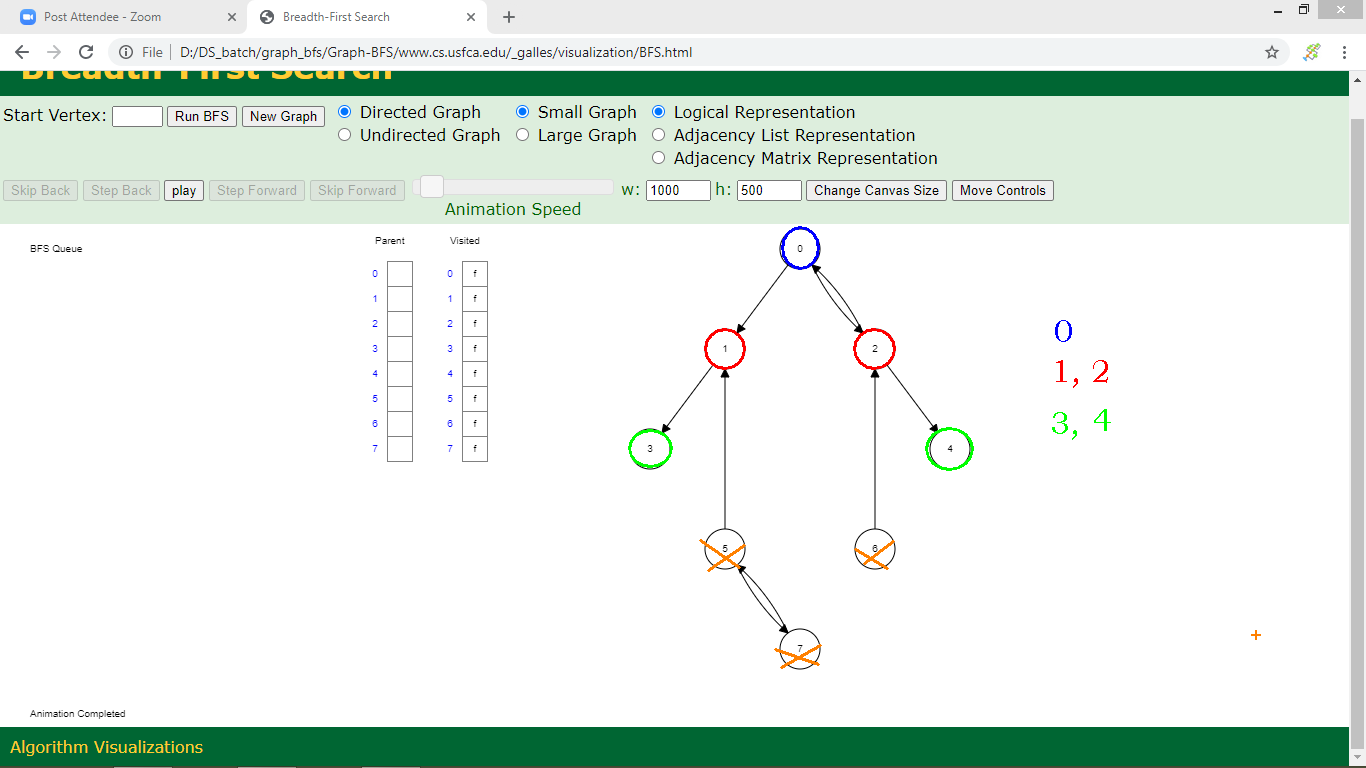
Example graph:



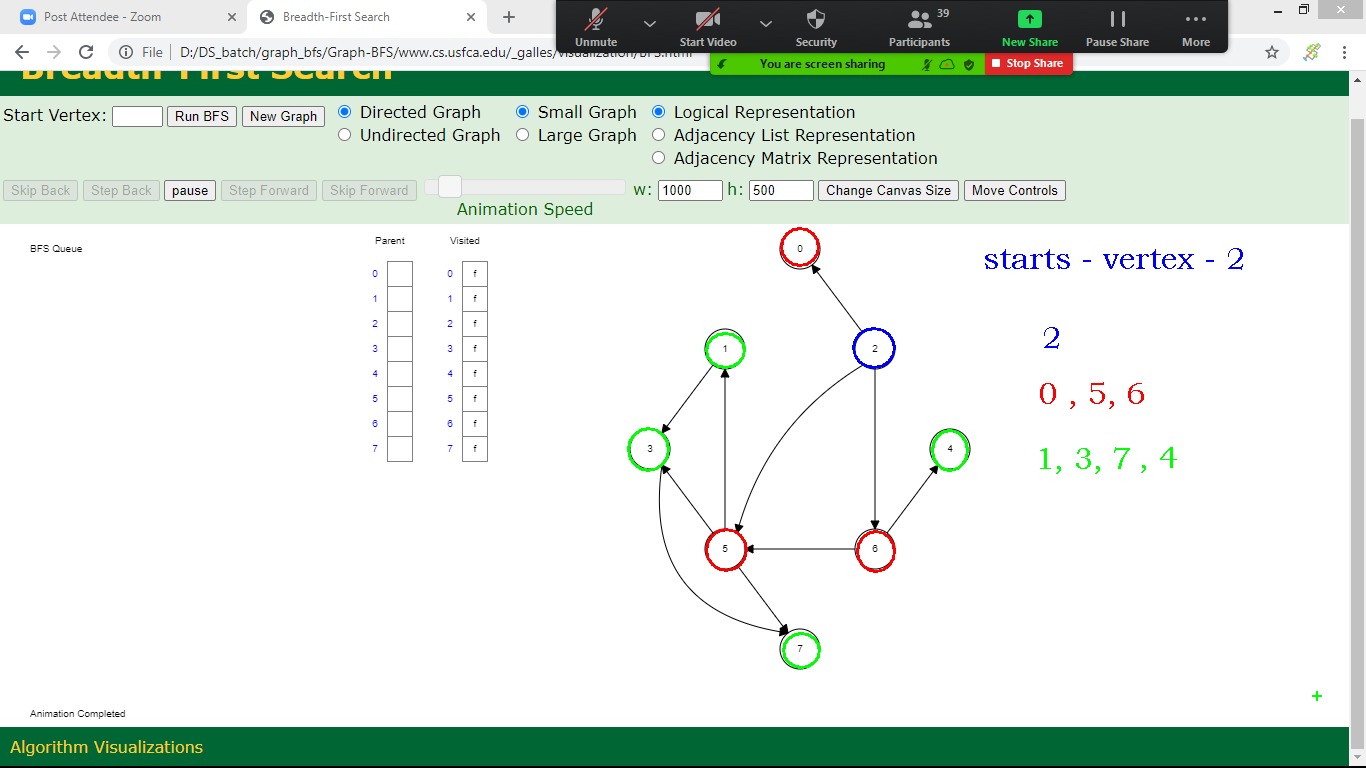
**BFS result :**



**Example graph:**



Example graph:



**Code Implementation:**

void bfs()

{

int qu[20],i=1,j=0;

p=start;

while(p!=NULL)

{

p->status=0;

p=p->next;

}

p=start;

qu[0]=p->data;

p->status=1;

while(1)

{

if(qu[j]==0)

break;

p=start;

while(p!=NULL)

{

if(p->data==qu[j])

break;

p=p->next;

}

k=p->adj;

while(k!=NULL)

{

q=k->next;

if(q->status==0)

{

qu[i]=q->data;

q->status=1;

qu[i+1]=0;

i++;

}

k=k->adj;

}

j++;

}

j=0;

printf("Breadth First Search Results");

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

while(qu[j]!=0)

{

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

j++;

}

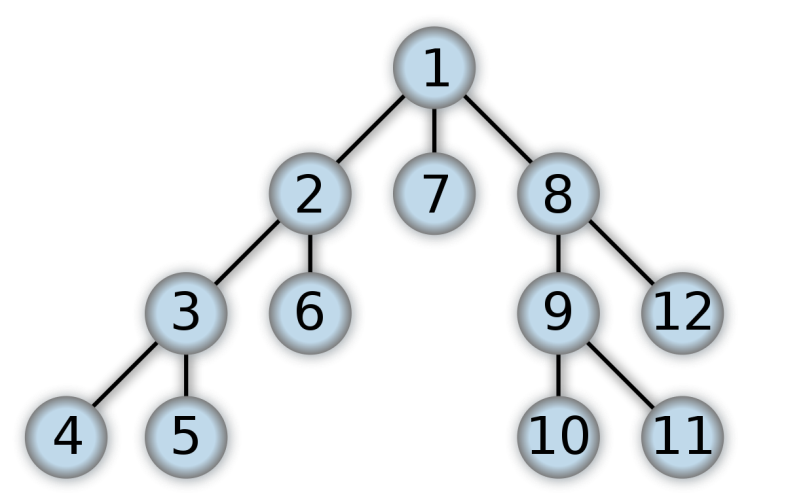
getch();

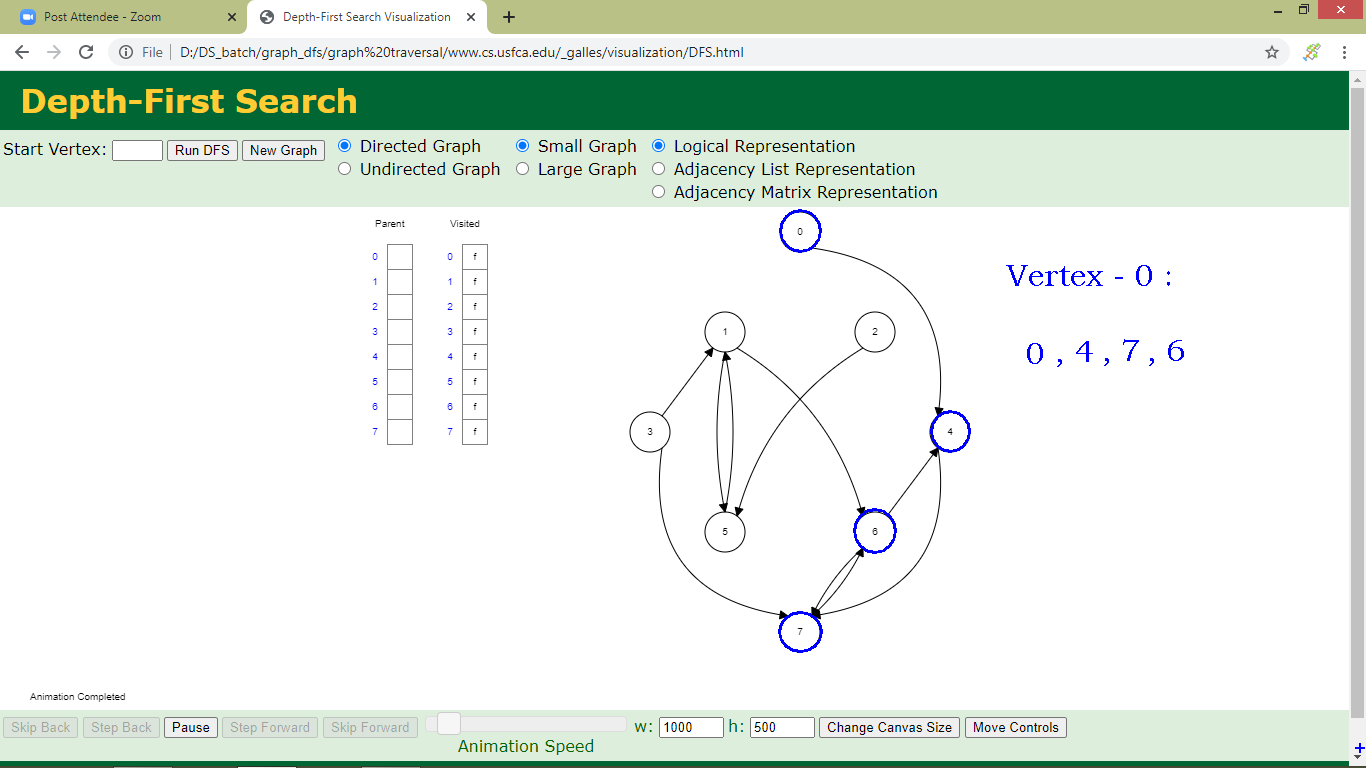
return;

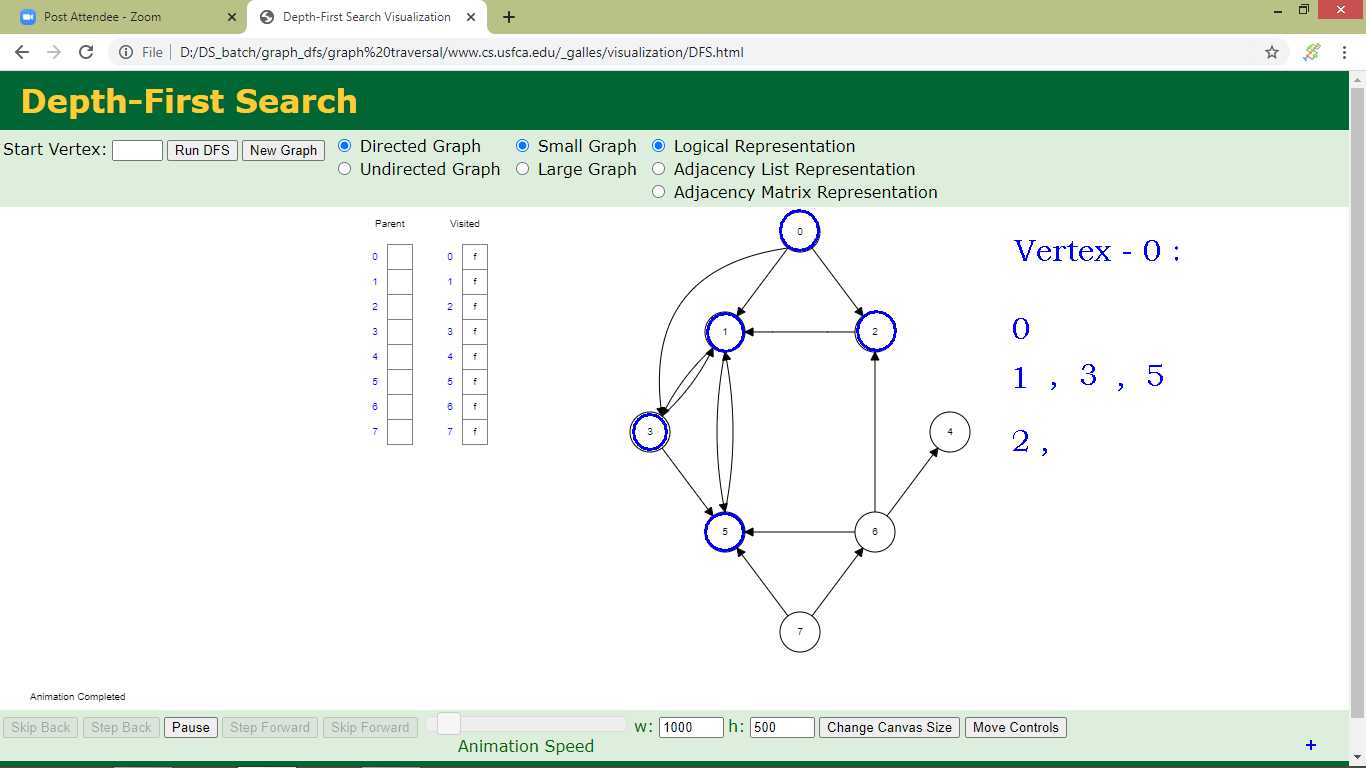
}

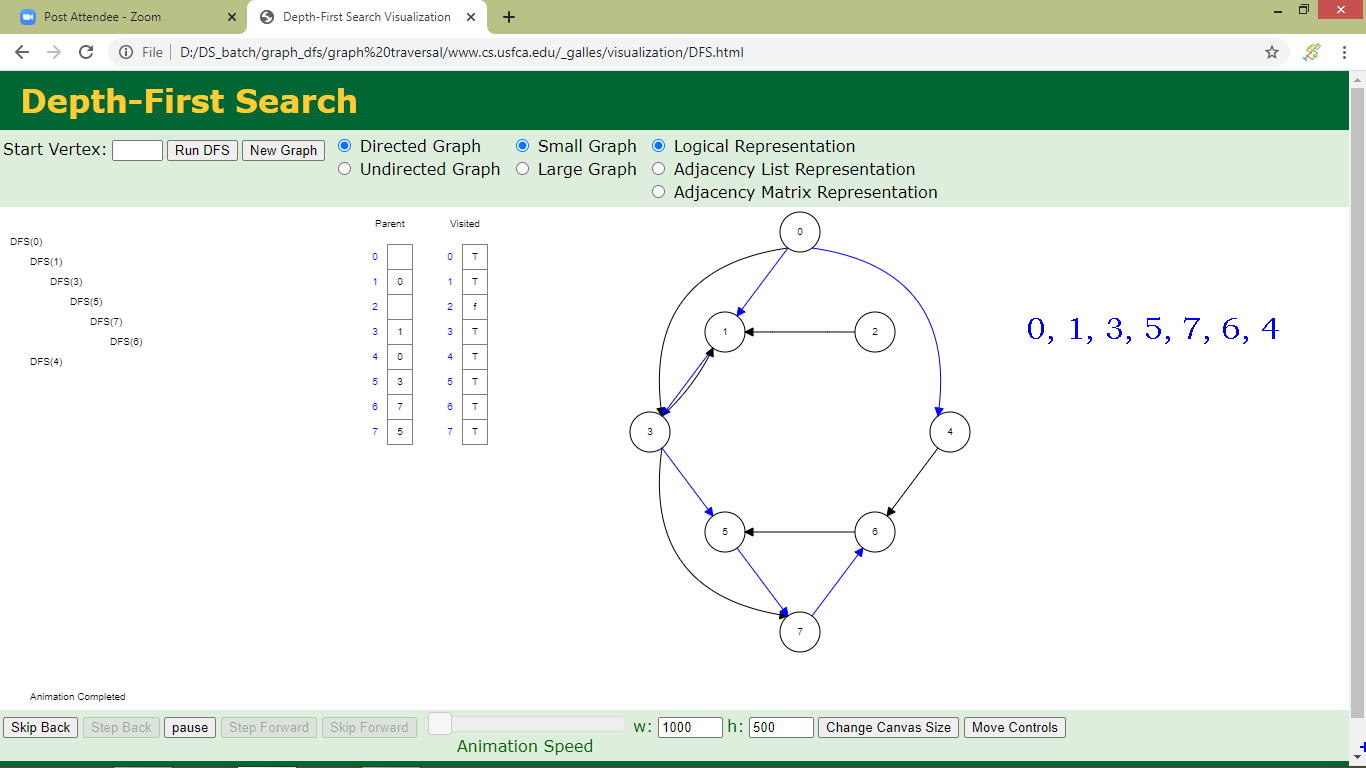
**Depth First Search (DFS):**

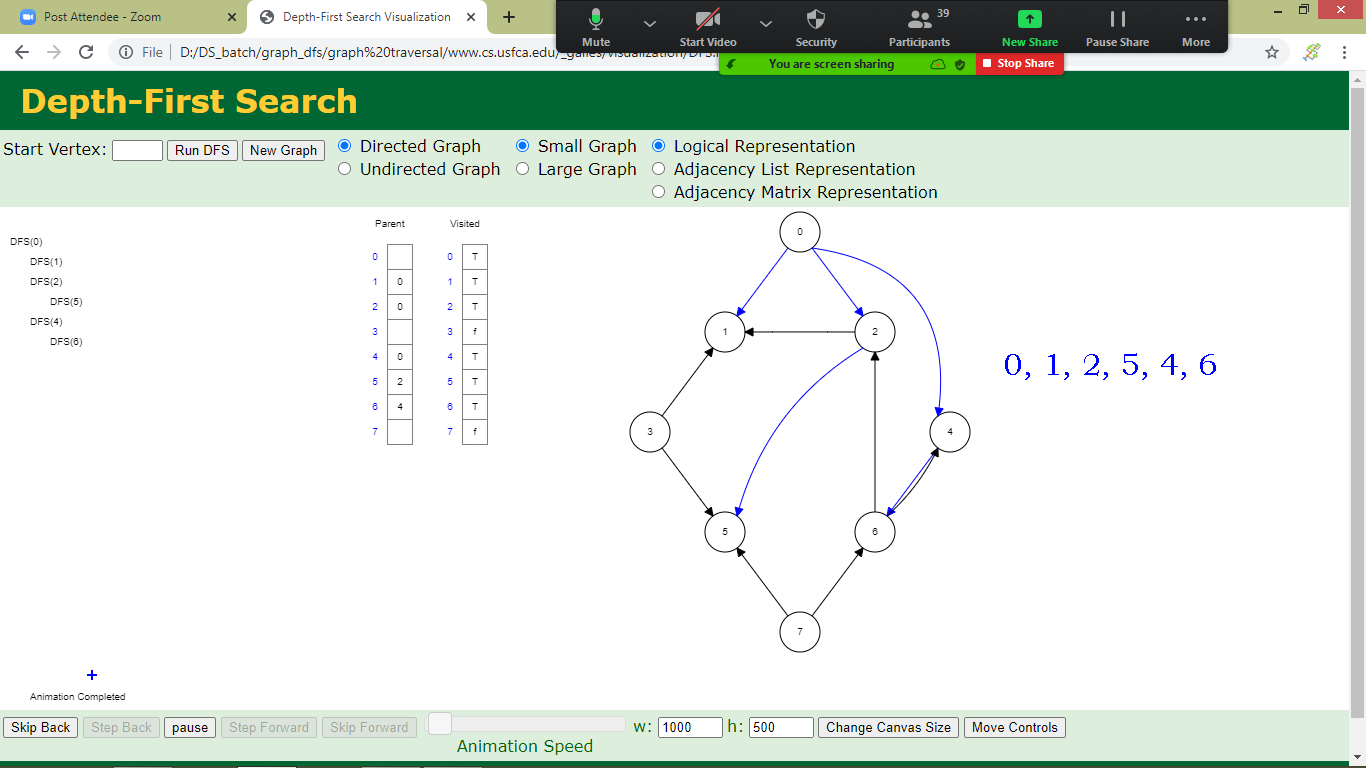
* The DFS algorithm is a recursive algorithm that uses the idea of backtracking.
* It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.
* Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse.
* All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

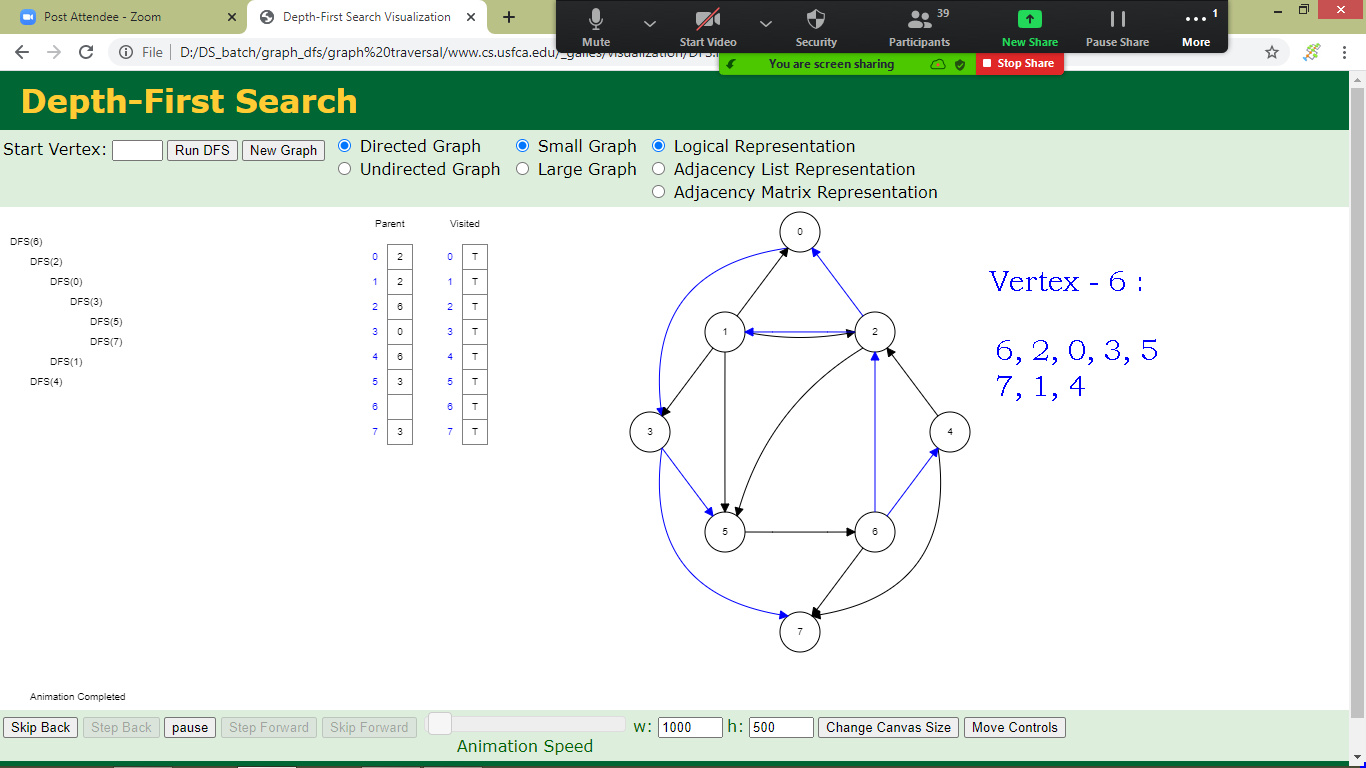












**Code Implementation:**

void dfs()

{

int stack[25],top=1;

printf("Depth First Search Results");

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

p=start;

while(p!=NULL)

{

p->status=0;

p=p->next;

}

p=start;

stack[0]=0;

stack[top]=p->data;

p->status=1;

while(1)

{

if(stack[top]==0)

break;

p=start;

while(p!=NULL)

{

if(p->data==stack[top])

break;

p=p->next;

}

printf("%d\t",stack[top]);

top--;

k=p->adj;

while(k!=NULL)

{

q=k->next;

if(q->status==0)

{

top++;

stack[top]=q->data;

q->status=1;

}

k=k->adj;

}

}

return;

}