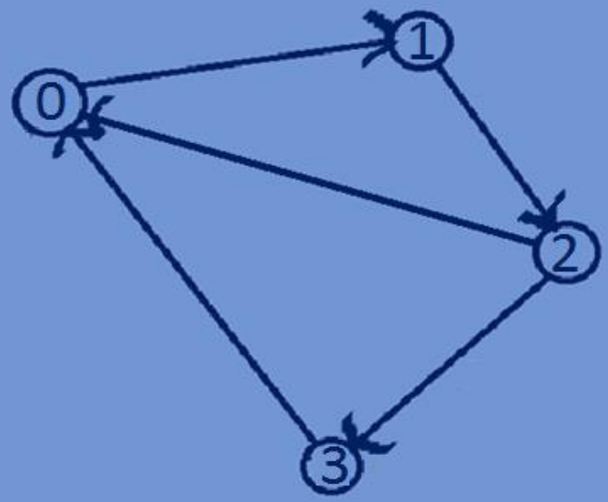
Data Structures

Depth First traversal of a Graph

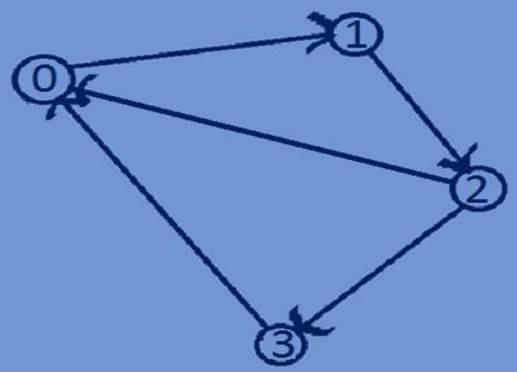
Adjacency List

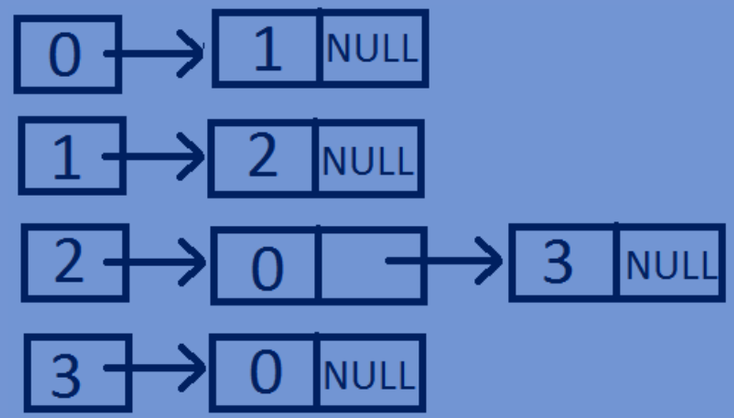
Depth First Traversal of a graph:

- DFS is an algorithm for traversing all the vertices of a graph.
- Unlike trees,graphs may have cycles so there may be possibility that we visit The same vertex more than once.To avoid visiting the node more than once We use a visited array which keeps track of the visited vertices,if we visit a vertex then we mark it as visited.A vertex that has already been marked will not be selected for traversal.

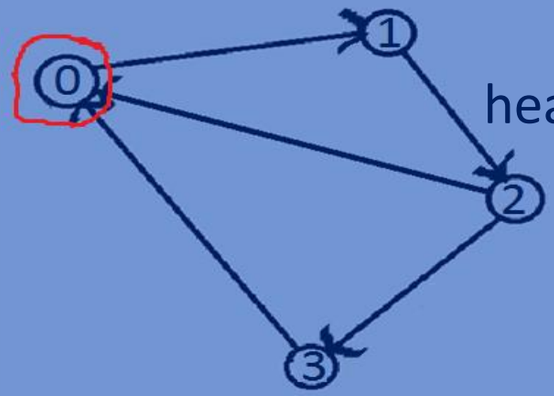


Depth First Traversal of a graph:





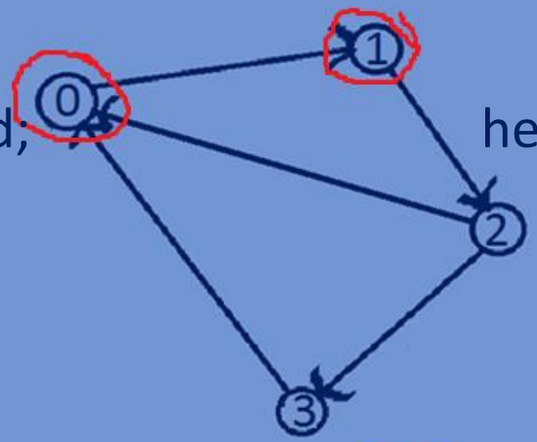
visited={0,0,0,0}



head=graph->array[0].Head;

Visited[0]=1;

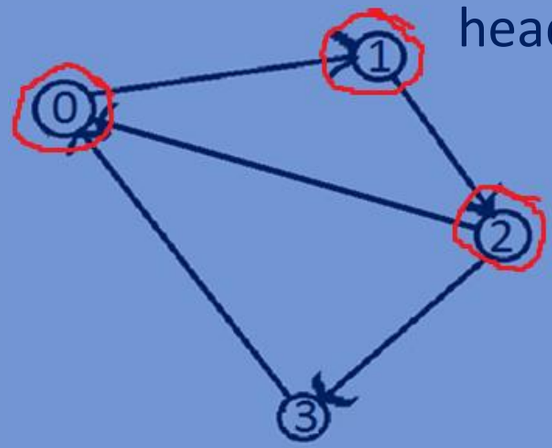
Visited=>{1,0,0,0}



head=graph->array[1].Head

visited[1]=1;

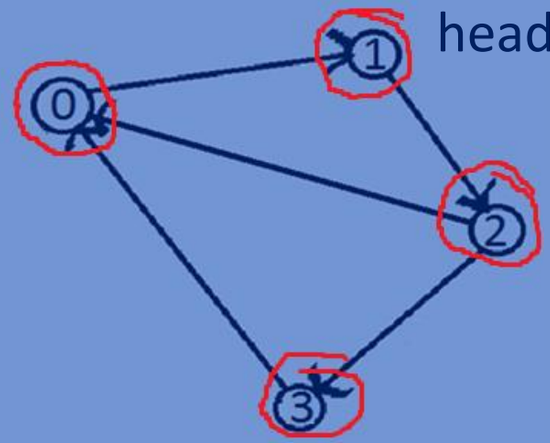
visited=>{1,1,0,0}



head=graph->array[2].Head;

Visited[2]=1;

Visited=>{1,1,1,0 }



head=graph->array[3].Head;

visited[3]=1;

visited=>{1,1,1,1}

Function for Depth First traversal of a graph:

```
//DFS traversal of a Graph
void DFSTraversal(Graph *graph,int visited[],int i){
    Node *head=graph->array[i].Head;
    printf("%d->",i);
    visited[i]=1;
    while(head){
        i=head->dest;
        if(visited[i]==0){
            DFSTraversal(graph,visited,head->dest);
        }
        head=head->next;
    }
}
```

Whole program:

```c
#include<stdio.h>
#include<stdlib.h>
//Structure for representing a NODE in the Adjacency List
typedef struct Node{
    int dest;
    int weight;
    struct Node *next;
}Node;
//structure for representing an adjacency liat
typedef struct List{
    Node *Head;
}List;
// A structure to represent a graph - here graph is an array of
Adjacency lists
// size of the array will be  equal to the number of vertices in
graph
typedef struct Graph{
```

```
    int totVertices;
    List *array;
}Graph;
//function To create a new node in the adjacency list
Node *createNewNode(int dest,int weight){
    Node *newnode=(Node*)malloc(sizeof(Node));
    newnode->dest=dest;
    newnode->weight=weight;
    newnode->next=NULL;
    return newnode;
}
//Function To creates a graph of n vertices
Graph *createGraph(int n){
    Graph *graph=(Graph*)malloc(sizeof(Graph));
    graph->totVertices=n;
    graph->array=(List*)malloc(n*sizeof(List));
    //Initialise each adjacency list as empty by making head as NULL
    for(int i=0;i<n;i++){
        graph->array[i].Head=NULL;
    }
    return graph;
}
```

```c
//function for Adding an edge to a directed graph
void addedge(Graph *graph,int src,int dest,int weight){
    Node *newnode=createNewNode(dest,weight);
    newnode->next=graph->array[src].Head;
    graph->array[src].Head=newnode;
}
//Function for printing Adjacency list corresponding to each vertex
void printGraph(Graph *graph){
    for(int i=0;i<graph->totVertices;i++){
        Node *Headnode=graph->array[i].Head;
        printf("connected vertices of vertex %d are:head",i);
        while(Headnode){
            printf("->%d",Headnode->dest);
            Headnode=Headnode->next;
        }
        printf("\n");
    }
}
//DFS traversal of a Graph
void DFSTraversal(Graph *graph,int visited[],int i){
    Node *head=graph->array[i].Head;
    printf("%d->",i);
```

```
    visited[i]=1;
    while(head){
        i=head->dest;
        if(visited[i]==0){
            DFSTraversal(graph,visited,head->dest);
        }
        head=head->next;
    }
}
//main function
int main(){
    int n=5;
    int visited[5]={0};
    Graph *graph=createGraph(n);
    addedge(graph,0,1,2);
    addedge(graph,0,2,1);
    addedge(graph,1,2,3);
    addedge(graph,2,3,1);
    addedge(graph,2,4,7);
    addedge(graph,2,1,1);
    addedge(graph,3,4,5);
    addedge(graph,4,0,4);
```

```
    printGraph(graph);
    DFSTraversal(graph,visited,0);
    return 0;
}
```

Output:

connected vertices of vertex 0 are:head->2->1

connected vertices of vertex 1 are:head->2

connected vertices of vertex 2 are:head->1->4->3

connected vertices of vertex 3 are:head->4

connected vertices of vertex 4 are:head->0

0->2->1->4->3->