**Data Structures**

Depth First traversal of a Graph

Adjacency Matrix

**Depth First Traversal of a graph:**

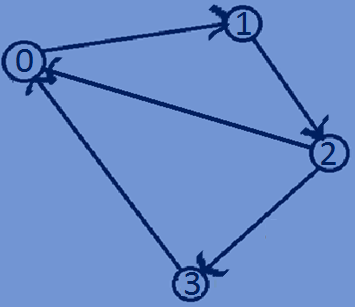
* DFS is an algorithm for traversing all the vertices of a graph.
* Unlike trees,graphs may have cycles so there may be possibility that we visit

The same vertex more than once.To avoid visiting the node more than once

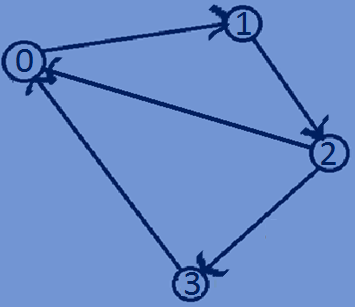
We use a visited array which keeps track of the visited vertices,if we visit a

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:**

 0 1 2 3

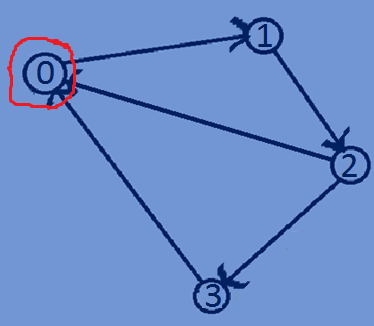
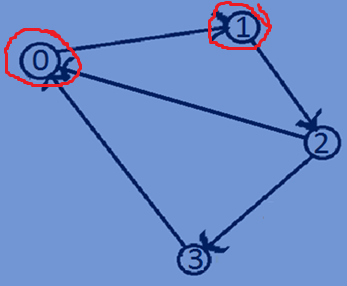
|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |

0

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

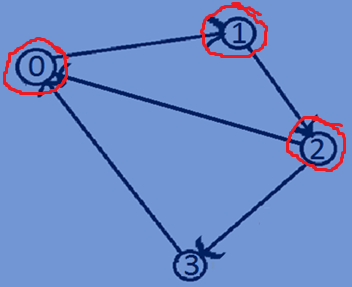
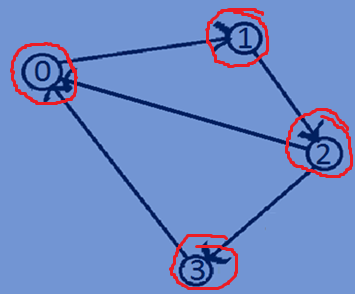
2

3



Visited[0]=1; visited[1]=1;

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



Visited[2]=1; visited[3]=1;

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

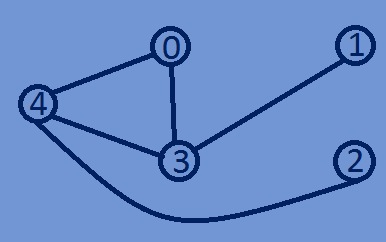
**Function for Depth First traversal of a graph:**

//DFS traversal of a Graph  
**void** DFSTraversal(graph \*array,**int** visited[],**int** i,**int** n){  
 printf("%d->",i);  
 visited[i]=1;  
 **for**(**int** j=0;j<n;j++){  
 **if**(visited[j]==0 && \*(array +i\*n+ j)==1){  
 DFSTraversal(array,visited,j,n);  
 }  
 }  
}

**Whole program:**

#include<stdio.h>  
#include<stdlib.h>  
#include<time.h>  
**typedef int** graph;  
//constructing a weighted undirectedgraph  
graph \*buildUndirectedGraph (**int** n) {  
 **int** i,j;  
 graph \*array = (graph \*) malloc(n \* n \* **sizeof**(graph));  
 srand((**unsigned**)time(NULL));  
 **for** (i = 0; i < n; i++) {  
 **for** (j = 0; j < n; j++) {  
 **if** (i == j) {  
 \*(array + i \* n + j) = 0;  
 } **else if** (i != j) {  
 **int** temp=rand()%2;  
 \*(array + i \* n + j) =temp;  
 \*(array + j \* n + i) =temp;  
 }  
 }  
 }  
 **return** array;  
}  
//constructing a weighted directed graph  
graph \*buildDirectedGraph(**int** n){  
 **int** i,j;  
 graph \*array = (graph \*) malloc(n \* n \* **sizeof**(graph));  
 srand((**unsigned**)time(NULL));  
 **for** (i = 0; i < n; i++) {  
 **for** (j = 0; j < n; j++) {  
 **if** (i == j) {  
 \*(array + i \* n + j) = 0;  
 } **else if** (i != j) {  
 **int** temp=rand()%2;  
 \*(array + i \* n + j) =temp;  
 }  
 }  
 }  
 **return** array;  
}  
//DFS traversal of a Graph  
**void** DFSTraversal(graph \*array,**int** visited[],**int** i,**int** n){  
 printf("%d->",i);  
 visited[i]=1;  
 **for**(**int** j=0;j<n;j++){  
 **if**(visited[j]==0 && \*(array +i\*n+ j)==1){  
 DFSTraversal(array,visited,j,n);  
 }  
 }  
}  
//printing adjacent matrix of a graph  
**void** printAdjacencyMatrix(graph \*array,**int** n){  
 **for** (**int** i = 0; i < n; i++) {  
 **for** (**int** j = 0; j < n; j++) {  
 printf("%d ", \*(array + i\*n + j));  
 }  
 printf("\n");  
 }  
}  
**int** main(){  
 **int** n=5;  
 **int** visited[5]={0};  
 graph \*array;  
 array=buildUndirectedGraph(n);  
 printAdjacencyMatrix(array,n);  
 DFSTraversal(array,visited,0,n);  
 **int** visited1[5]={0};  
 array=buildDirectedGraph(n);  
 printAdjacencyMatrix(array,n);  
 DFSTraversal(array,visited1,0,n);  
 **return** 0;  
}

**Output:**

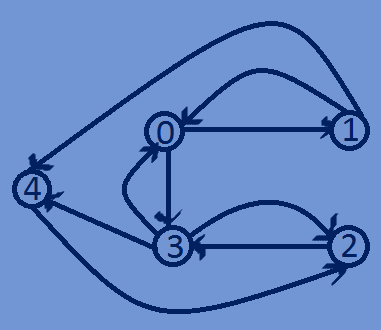
0 0 0 1 1

0 0 0 1 0

0 0 0 0 1

1 1 0 0 1

1 0 1 1 0

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

0 1 0 1 0

1 0 0 0 1

0 0 0 1 0

1 0 1 0 1

1. 0 1 0 0

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