**Data Structures**

Breadth First traversal of a Graph

Adjacency List

**Breadth First Traversal of a graph:**

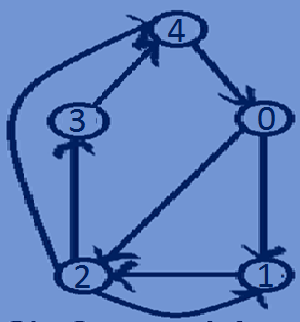
* BFS is an algorithm for traversing all the vertices of a graph in level order fashion.
* Unlike trees,graphs may have cycles so there may be possibility that we visit

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

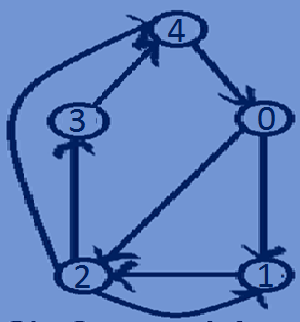
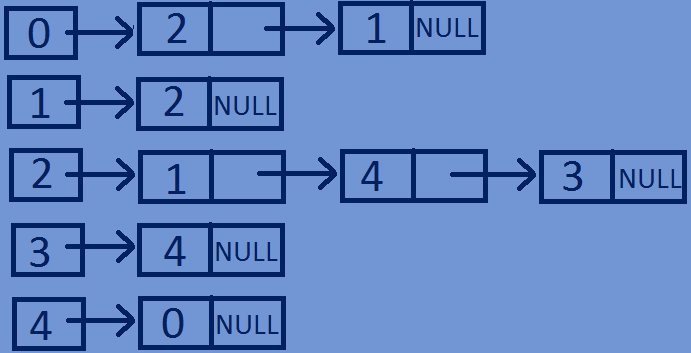
We use a visited array which keeps track of the visited vertices,if we visit a

vertex then we mark it as visited.A vertex that has already been marked

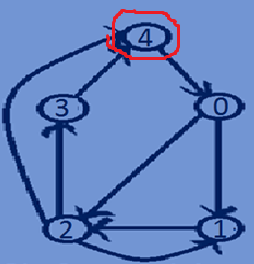
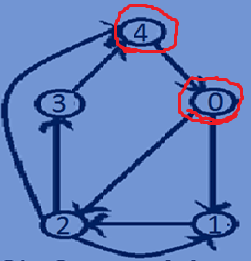
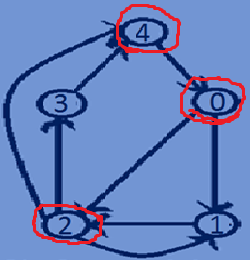
will not be selected for traversal.



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

**Breadth First Traversal of a graph:**

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

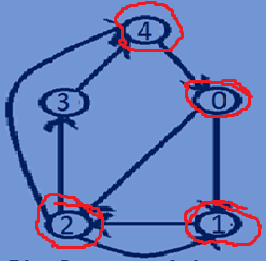
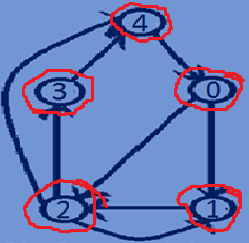


print(4) print(0)

visited[4]=1 dequeue(4) dequeue(0)

 visited[0]=1 visited[2]=1





print(2) print(1)

dequeue(2) dequeue(1)

 visited[1]=1 visited[3]=1 print(3)

 dequeue(3)

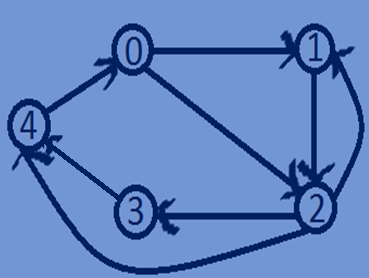
**Function for Depth First traversal of a graph:**

//BFS traversal of a graph  
**void** BFSTraversal(Graph \*graph,**int** visited[],**int** startvertex){  
 lin\_list \*queue=NULL;//creating a linked list(queue)  
 //mark the current node as visited and enqueue it.  
 visited[startvertex]=1;  
 queue=enqueue(queue,startvertex);  
 **while**(queue){  
 **int** vertex=queue->data;  
 printf("->%d",vertex);  
 queue=dequeue(queue);  
 Node \*head=graph->array[vertex].Head;  
 //visit all the adjacent vertices of the current vertex and mark them visited and enqueue them.  
 **while**(head){  
 **if**(visited[head->dest]==0){  
 visited[head->dest]=1;  
 queue=enqueue(queue,head->dest);  
 }  
 head=head->next;  
 }  
 }  
}

**Whole program:**

#include<stdio.h>  
#include<stdlib.h>  
//creating a node.  
**typedef struct** lin\_list{  
 **int** data;  
 **struct** lin\_list \*next;  
}lin\_list;  
//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;  
}  
//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");  
 }  
}  
//adding a newnode at the end of a linked list  
lin\_list \*enqueue(lin\_list \*head,**int** data){  
 lin\_list \*newnode=(lin\_list\*)malloc(**sizeof**(lin\_list));  
 newnode->data=data;  
 newnode->next=NULL;  
 lin\_list \*temp=head;  
 **if**(head==NULL){  
 head=newnode;  
 }  
 **else** {  
 **while** (temp->next != NULL) {  
 temp = temp->next;  
 }  
 temp->next = newnode;  
 }  
 **return** head;  
}  
//popping of first node from linked list  
lin\_list \*dequeue(lin\_list \*head){  
 lin\_list \*temp=head;  
 head=head->next;  
 free(temp);  
 **return** head;  
}  
//BFS traversal of a graph  
**void** BFSTraversal(Graph \*graph,**int** visited[],**int** startvertex){  
 lin\_list \*queue=NULL;//creating a linked list(queue)  
 //mark the current node as visited and enqueue it.  
 visited[startvertex]=1;  
 queue=enqueue(queue,startvertex);  
 **while**(queue){  
 **int** vertex=queue->data;  
 printf("->%d",vertex);  
 queue=dequeue(queue);  
 Node \*head=graph->array[vertex].Head;  
 //visit all the adjacent vertices of the current vertex and mark them visited and enqueue them.  
 **while**(head){  
 **if**(visited[head->dest]==0){  
 visited[head->dest]=1;  
 queue=enqueue(queue,head->dest);  
 }  
 head=head->next;  
 }  
 }  
}  
//main function  
**int** main(){  
 **int** n=5,visited[5]={0};//making all the vertices as not visited  
 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);printf("\n");  
 printGraph(graph);  
 BFSTraversal(graph,visited,4);  
 **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

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