**Assignment 6**

**Aim**:

Represent a given graph using adjacency list and traverse each node using Breadth-firstsearch.

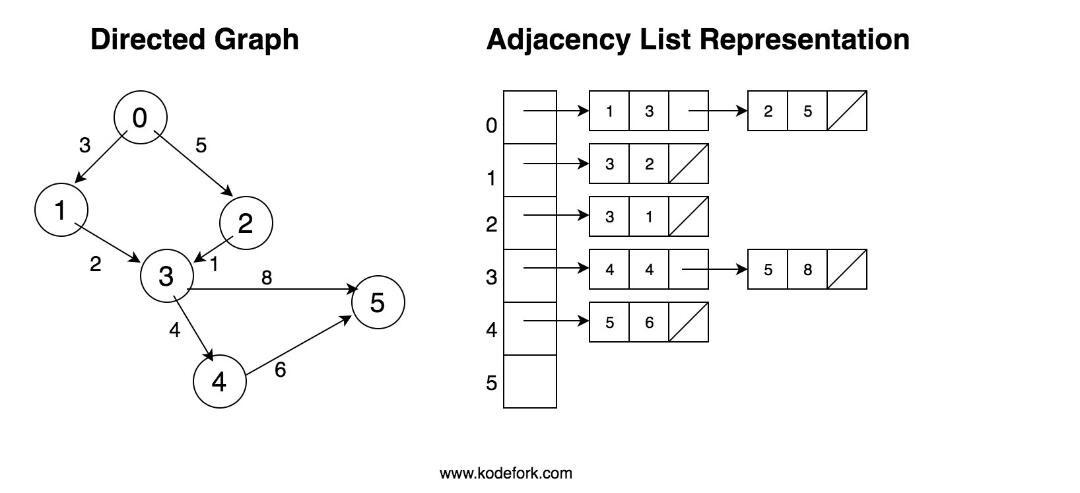
**Objective** :

To learn to implement adjacency-list representation of graph and BFS traversal of nodes.

**Theory**:

**1) Adjacency List:**

* A graph can be represented using a linked list. For each vertex, a list of adjacent vertices is maintained using a linked list. It creates a seperate linked list for each vertex V in the graph G = (V, E)
* Adjacency list of a graph with n nodes can be represented by an array of pointers. Each pointer points to a linked list of the corresponding vertex.



**2) Breadth-First Traversal:**

* It is a popular approach used for visiting the vertices of a graph.
* This method starts from a given vertex suppose A. A is marked as visited. All vertices adjacent to A are visited next. The method continues until all vertices are visited.
* The algorithm for BFS has to maintain a list of vertices that are visited.

**Algorithm:**

**For Adjacency-list creation:**

int nodes be variable to store number of nodes

node\* heads[100] be array to store head pointer of each linked list.

1. Enter number of nodes.
2. for i = 0 to i = nodes, initialize heads[i] to pointer pointing to node with data ‘i’
3. for i = 0 to i = nodes:

\* Enter number of adjacent nodes.

\* Enter the adjacent nodes and append it to end of linked list heads[i]

**Breadth-First Search Algorithm:**

1. Enqueue the starting vertex.
2. v = dequeue()
3. if v is not visited, print it and mark it as visited (visited[v] = 1)
4. Enqueue all adjacent vertices of v that are not visited.
5. Repeat steps 2 to 4 till Queue is empty.

**Program:**

Problem Statement : Represent a given graph using adjacency list and traverse each node using Breadth first search

#include<iostream>

using namespace std;

struct node

{

int data;

node \*next;

};

node \*heads[10];

void adjacencyListRepre(int);

node \*createNode(int value);

void display(int );

void bfs(int vertices);

void enqueue(int value);

int dequeue();

//QUEUE

int Front = -1;

int Rear = -1;

int queue[20];

int main()

{

int choise;

int vertices;

cout<<"Enter the number of vertices :";

cin>>vertices;

do{

cout<<"------------------------------------------------------------------------\n";

cout<<"\t\t\tBreadth First Search \n\n";

cout<<"\t\t1.Create a Graph\n";

cout<<"\t\t2.Display the Graph\n";

cout<<"\t\t3.Breadth First Search\n";

cout<<"\t\t4.Exit\n";

cout<<"\tYour choise : ";

cin>>choise;

switch(choise)

{

case 1:

{

adjacencyListRepre(vertices);

}

break;

case 2:

display(vertices);

break;

case 3:

bfs(vertices);

break;

}

}while(choise<4);

}

void adjacencyListRepre(int vertices)

{

int edges,ev;

//node \*heads[vertices];

for(int i=0;i<vertices;i++)

heads[i] =NULL;

for (int i = 0; i < vertices; i++)

{

cout<<"Enter the number of edges connected to vertex "<<i<<" : ";

cin>>edges;

for (int j = 0; j < edges; j++)

{

cout<<"Enter the "<<j+1 <<" th edge (Enter only ending vertex): ";

cin>>ev;

if(heads[i]==NULL)

heads[i] = createNode(ev);

else

{

node \*temp = heads[i];

while(temp->next!=NULL)

temp = temp->next;

temp->next = createNode(ev);

}

}

}

}

void display(int vertices)

{

for (int i = 0; i < vertices; i++)

{

node \*temp = heads[i];

while(temp!=NULL)

{

cout<<"Edge from vertex "<<i<<" to verticex "<<temp->data<<"\n";

temp = temp->next;

}

cout<<"\n";

}

}

node \*createNode(int value)

{

node \*temp =new node;

temp->data = value;

temp->next = NULL;

return temp;

}

void enqueue(int value)

{

if(Front ==-1&&Rear == -1)

{

Front = 0;

Rear = 0;

}

else

{

Rear = Rear +1;

}

queue[Rear] =value;

}

int dequeue()

{

int x = queue[Front];

if(Front==Rear)

{

Front = -1;

Rear = -1;

}

else

{

Front = Front+1;

}

return x;

}

void bfs(int vertices)

{

int visited[vertices];

for (int i = 0; i < vertices; i++)

{

visited[i] = 0;

}

enqueue(0);

visited[0] = 1;

cout<<"\tBFS is as : ";

while(Front!=(-1))

{

int x= dequeue();

cout<<"\t"<<x;

node \*temp = heads[x];

while(temp!=NULL)

{

if(!visited[temp->data])

{

enqueue(temp->data);

visited[temp->data] = 1;

}

temp = temp->next;

}

}

cout<<"\n";

}

**Conclusion :**

We can use int visited array to maintain list of visited vertices.

BFS can be implemented with the help of a queue.

Adjacency list implementation of graph is memory efficient when graph has a large number of vertices, but small number of edges.