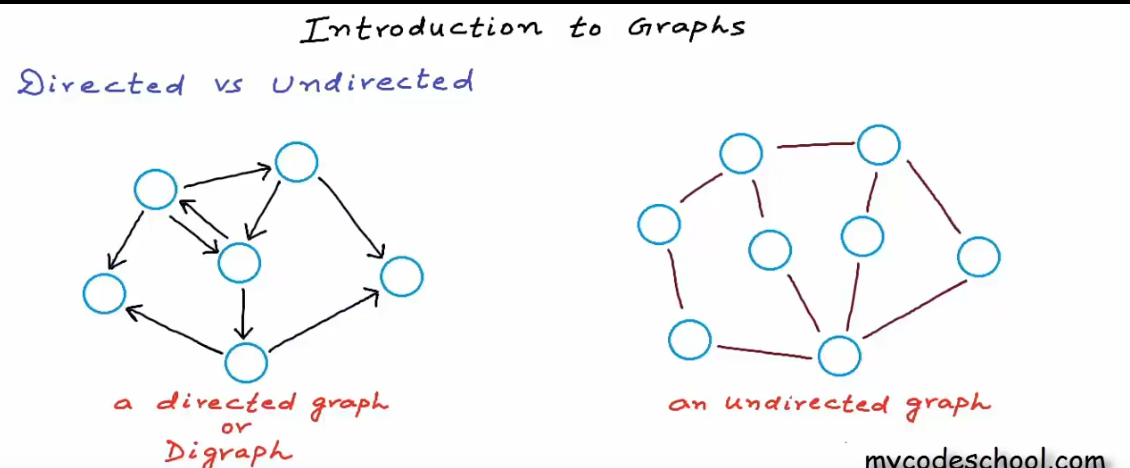


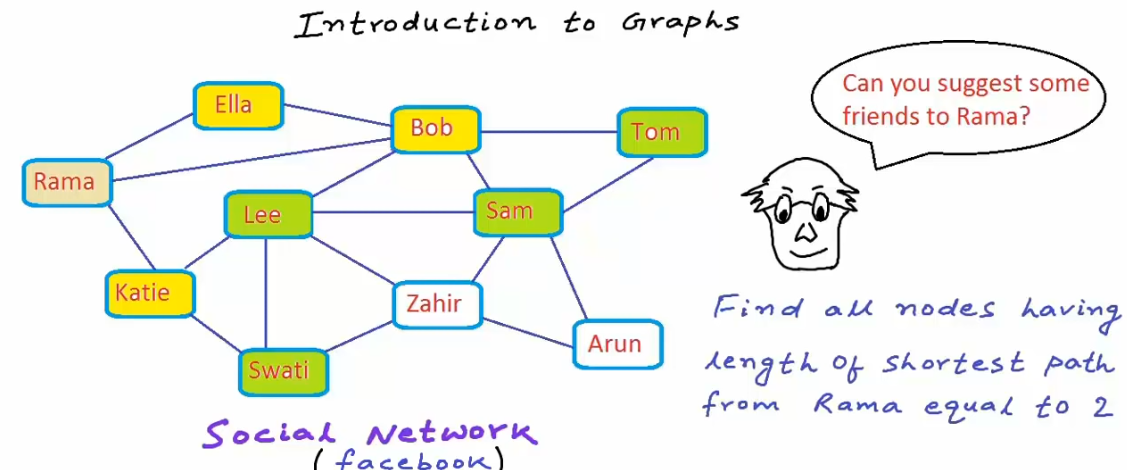
Vertices : {v1, v2,…v8}

Edges : directed and undirected.

Edges E = {{v1,v2}, {v2,v3}… ,{v7,v8}}

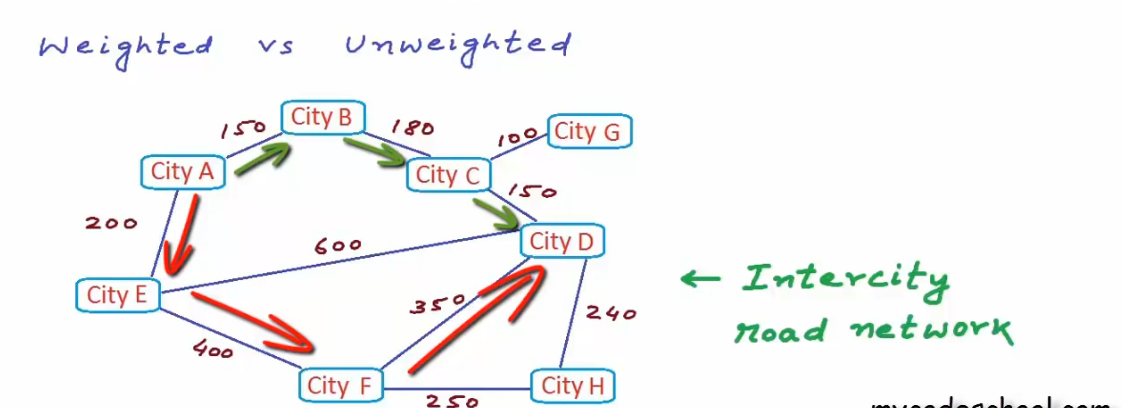


Graphs are used in social network (facebook).



Real word usage of graph :

1. Graphs are used in social network (facebook). Suggesting mutual friends.
2. In World wide web (WWW). Page links
3. Google search engine. Web-crawling(is used to give quick result).

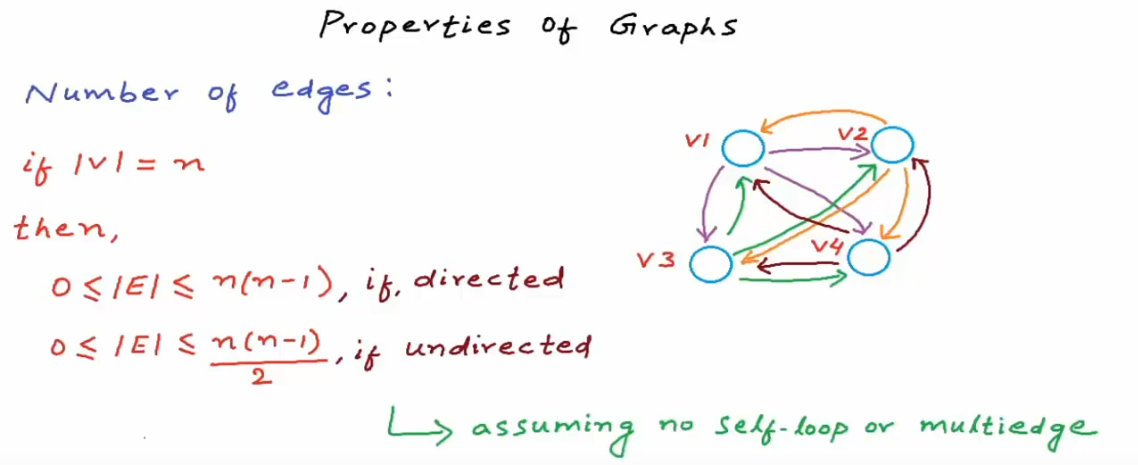


Graph Properties:

Edge: self-loop

Multi edge

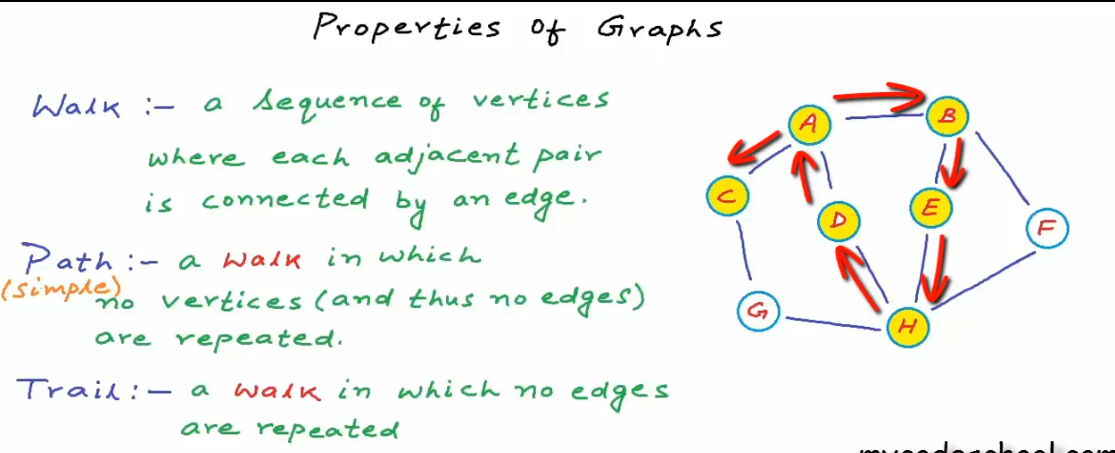
Simple Graph without multiple edges.



Dense -> too many edges.

Sparse -> too few edges.

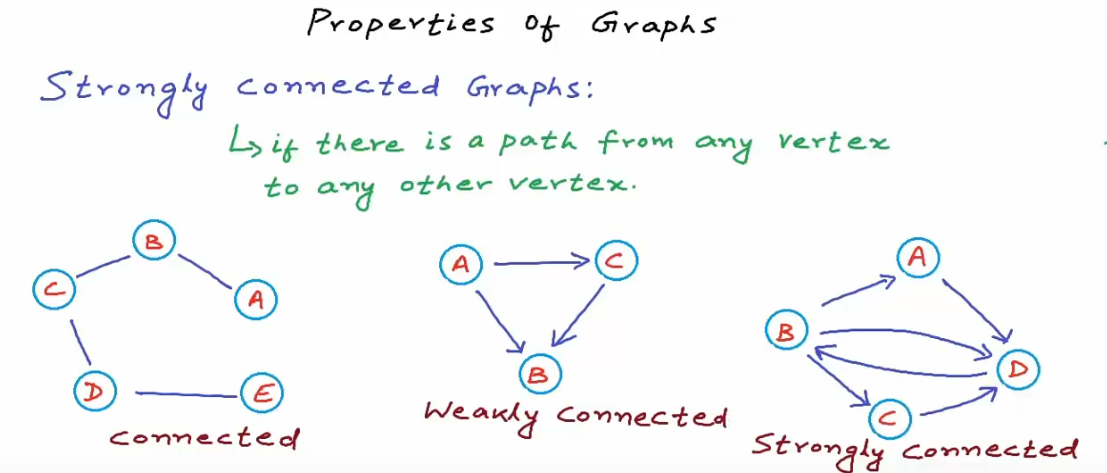
If |v|=10 , |E| <90



Closed walk: starts and ends at same vertex.

Simple Cyscle: no repetition other than start and end

Acyclic Graph: a graph with no cycle.



Implementation of Graph:

1. Adjacency Matrix: Represents graph with ‘V’ nodes into an VxV matrix, Where Aij=1 means that vertex I and j are connected.

Pros:

Easy to implement.

Removing an edge takes O(1) time.

Queries like whether there is an edge from vertex u ato vertex v are efficient and can be done O(1).

Cons:

Consumes more space O(v^2).

Adding a vertex is O(V^2) time.

1. Adjacency List: An arrays of linked lists is used. Size of array is equal to number of vertices and each entry of array corresponding to as linked list of vertices adjacent to the this index.

Pros:

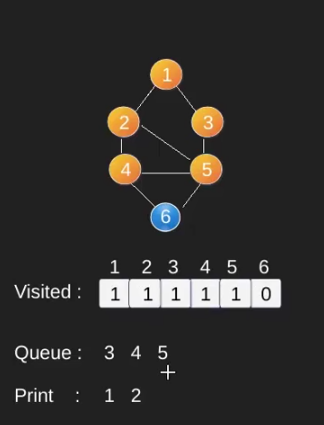
Saves space O(|V|+|E|)

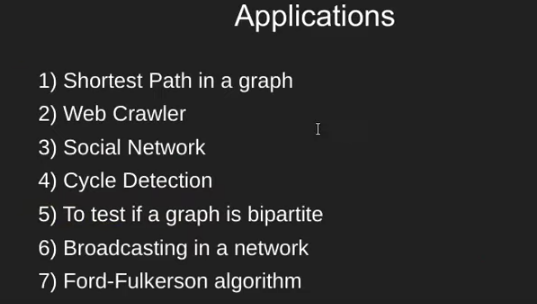
Adding a vertex is easier.

Cons:

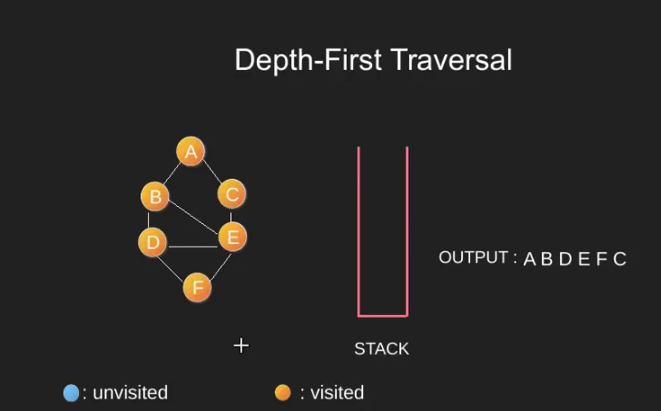
Queries like whether there is an edge from vertex u to vertex v are efficient and can be done O(V).

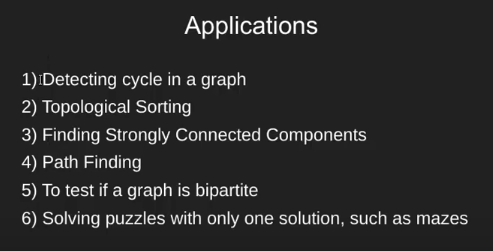
Breath First Traversal: BFT(Level search)





Depth-First Traversal:





package com.sachin.graph;

import java.util.LinkedList;

import java.util.Queue;

//Java Program to demonstrate adjacency list

//representation of graphs

public class GraphImp {

static class Graph{

int v;

LinkedList<Integer> adjList[];

Graph(int v){

this.v = v;

adjList = new LinkedList[v];

for(int i =0;i<v;i++) {

adjList[i] = new LinkedList<>();

}

}

}

public static void main(String args[]) {

int v=5;

Graph graph = new Graph(v);

addEdge(graph,0,1);

addEdge(graph,0,4);

addEdge(graph,1,2);

addEdge(graph,1,3);

addEdge(graph,1,4);

addEdge(graph,2,3);

addEdge(graph,3,4);

displayGraph(graph);

BreadthFS(graph,0);

depthFS(graph,0);

shortestPath(graph,1,3);

}

private static void depthFS(Graph graph, int value) {

boolean visited[] = new boolean[graph.v];

System.out.println("Depth first Traversal : ");

depthFSUtil(value,visited,graph);

}

private static void depthFSUtil(int value, boolean[] visited, Graph graph) {

visited[value] = true;

System.out.print(value + "-->");

for(Integer val: graph.adjList[value]) {

if(!visited[val]) {

depthFSUtil(val,visited,graph);

}

}

}

private static void shortestPath(Graph graph, int src, int des) {

boolean visited[] = new boolean[graph.v];

Queue<Integer> queue = new LinkedList<>();

visited[src]=true;

queue.add(src);

int path =1;

while(queue.size()!=0) {

if(src==des)

break;

src=queue.poll();

path++;

for(Integer value: graph.adjList[src]) {

if(!visited[src]) {

queue.add(value);

visited[value]=true;

}

}

}

System.out.println("shorted Path :" + path);

}

private static void BreadthFS(Graph graph, int first) {

boolean visited[] = new boolean[graph.v];

Queue<Integer> queue = new LinkedList<>();

queue.add(first);

visited[first] = true;

while(queue.size() !=0) {

first = queue.poll();

System.out.print(first + "->");

for(Integer value: graph.adjList[first]) {

if(!visited[value]) {

queue.add(value);

visited[value] = true;

}

}

}

}

//Display the graph

private static void displayGraph(Graph graph) {

// TODO Auto-generated method stub

for(int i=0;i<graph.v;i++) {

System.out.println("Adjacency list of vertex " + i);

System.out.println("head");

for(Integer value : graph.adjList[i]) {

System.out.print("-->" + value);

}

System.out.println();

}

}

//Add the edge

private static void addEdge(Graph graph, int src, int desc) {

// TODO Auto-generated method stub

graph.adjList[src].add(desc);

graph.adjList[desc].add(src);

}

}