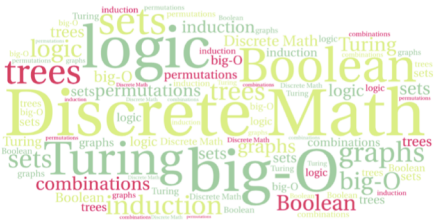
CS 220: Discrete Structures and their Applications

graphs

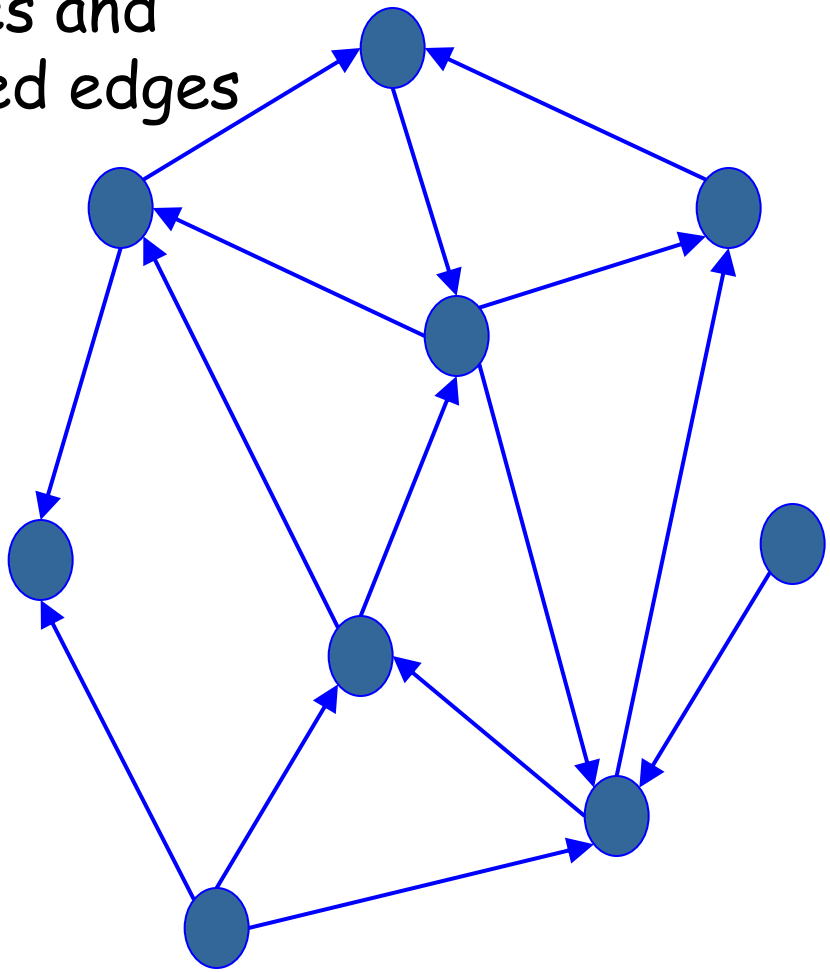
zybooks chapter 10



Colorado State University

directed graphs

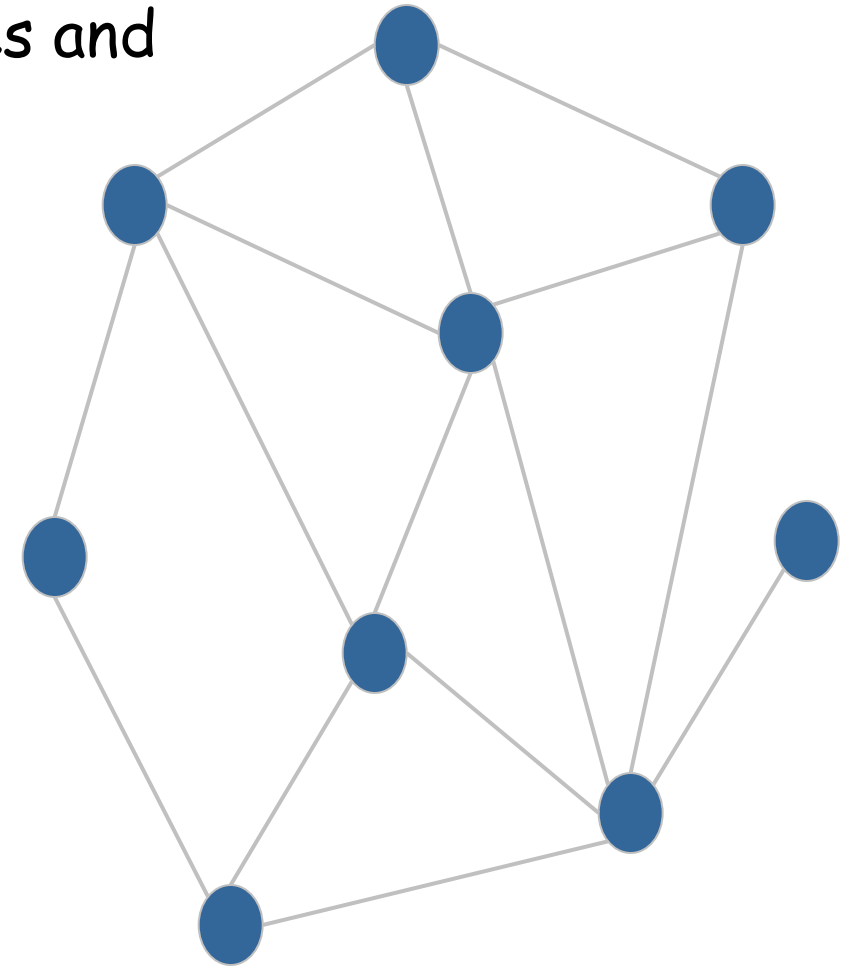
A collection of vertices and directed edges



What can this represent?

undirected graphs

A collection of vertices and edges



What can this represent?

Graph definitions

Graph **G = (V, E)** , V: set of **nodes** or vertices,

E: set of **edges** (pairs of nodes).

In an **undirected** graph, edges are unordered pairs (sets) of nodes. In a **directed** graph edges are ordered pairs of nodes.

Path: sequence of nodes $(v_0..v_n)$ s.t. $\forall i$: (v_i, v_{i+1}) is an edge. **Path length**: number of edges in the path, or sum of weights. **Simple path:** all nodes distinct.

Cycle: path with first and last node equal. **Acyclic graph**: graph without cycles. **DAG**: directed acyclic graph.

Two nodes are **adjacent** if there is an edge between them. In a **complete graph** all nodes in the graph are adjacent.

more definitions

An undirected graph is **connected** if for all nodes v_i and v_j there is a path from v_i to v_j . An undirected graph can be partitioned in **connected components**: maximal connected sub-graphs.

A directed graph can be partitioned in **strongly connected components**: maximal sub-graphs C where for every u and v in C there is a path from u to v and there is a path from v to u.

G'(V', E') is a **sub-graph** of G(V,E) if $V' \subseteq V$ and $E' \subseteq E$

The sub-graph of G **induced** by V' has all the edges

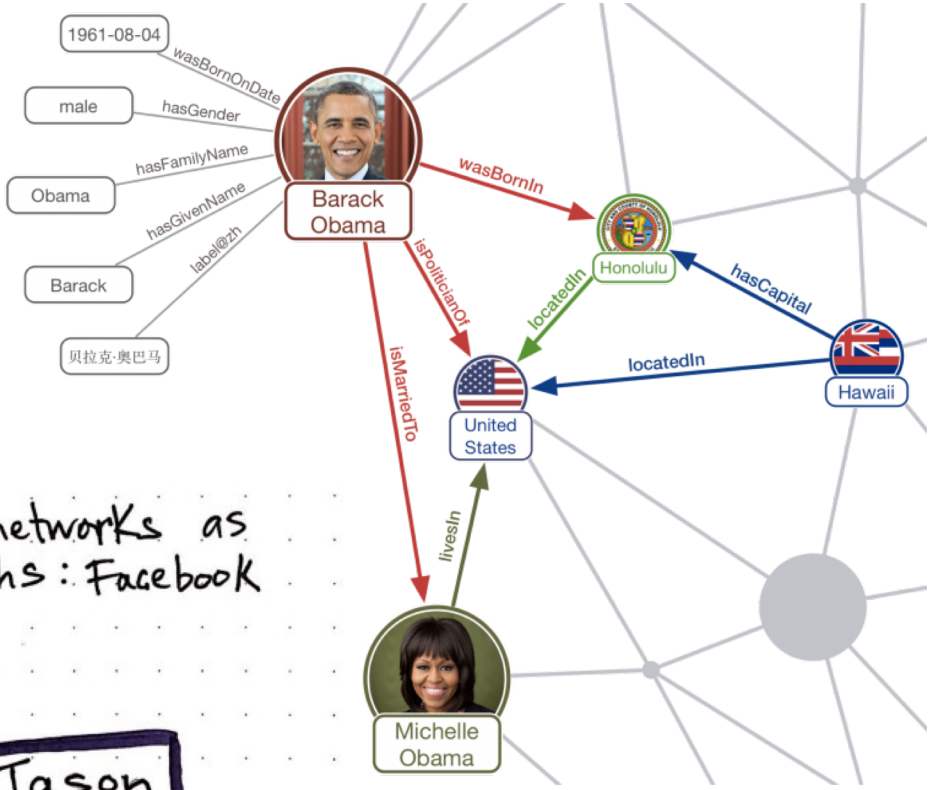
$(u,v) \in E$ such that $u \in V'$ and $v \in V'$.

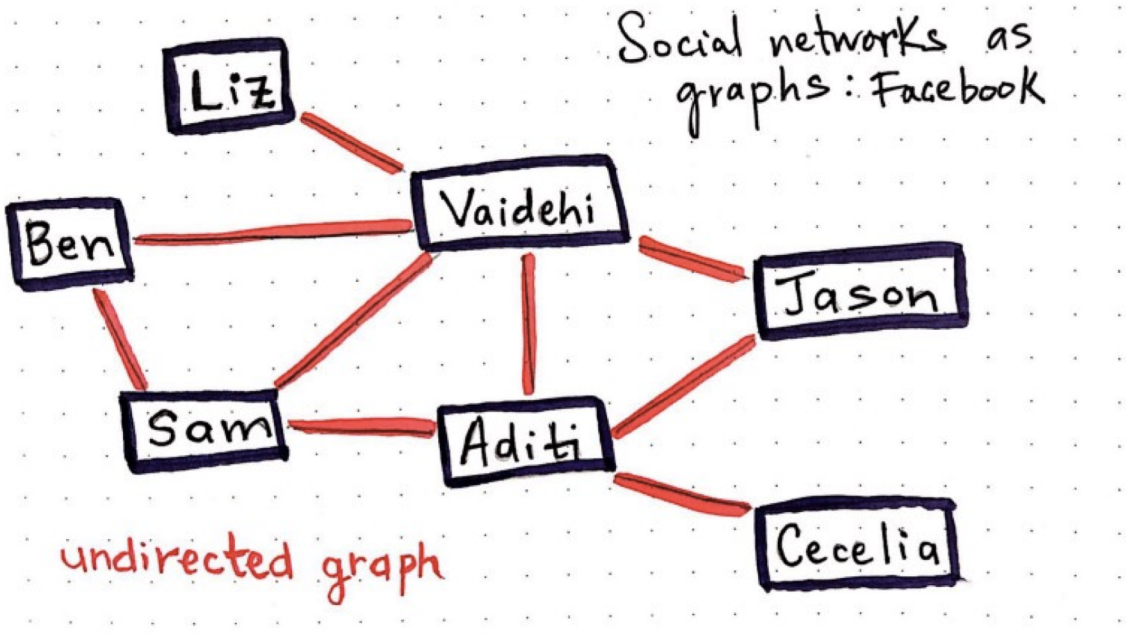
In a **weighted graph** the edges have a weight (cost, length,..) associated with them.

directed / undirected graphs in applications

Directed or undirected graph?

- ✓ Facebook friend graph
- ✓ The "follow" graph
- ✓ The "like" graph
- ✓ Knowledge graph





https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8

https://www.ambiverse.com/knowledge-graphs-encyclopaedias-for-machines/

constraint graphs

Consider the problem of classroom scheduling: given a set of classes and their times, assign them to classrooms without conflicts.

Example:

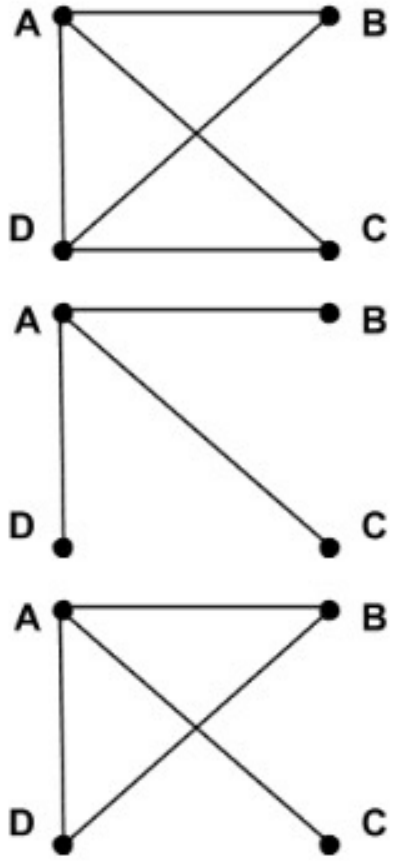
Class A: MWF, 3:00PM - 4:00PM

Class B: W, 2:00PM - 4:00PM

Class C: F, 3:30PM - 5:00PM

Class D: MWF, 2:30 - 3:30PM

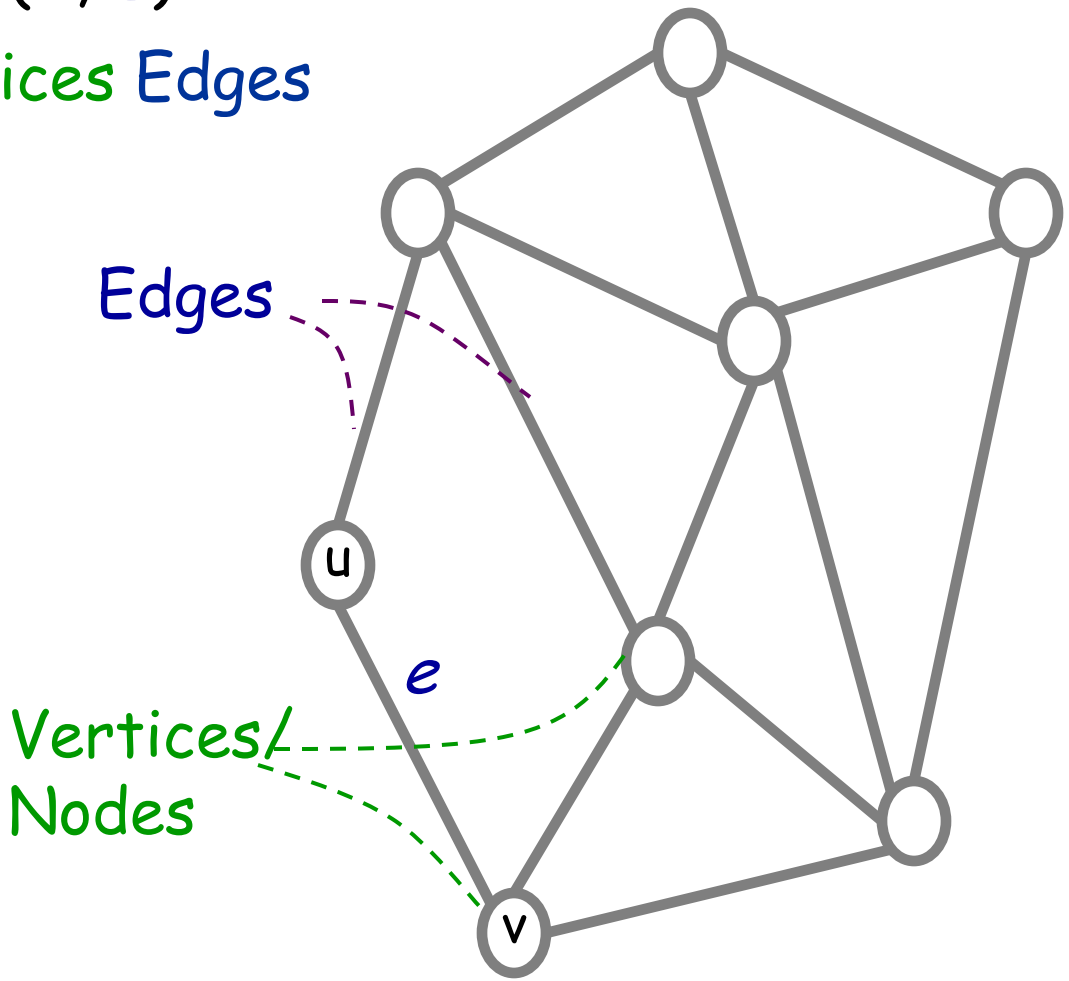
Which is the constraint graph for this scheduling problem?



terminology

$G=(V, E)$

Vertices Edges



Two vertices are adjacent if they are connected by an edge.

The vertices are the endpoints of an edge

An edge is incident on two vertices.

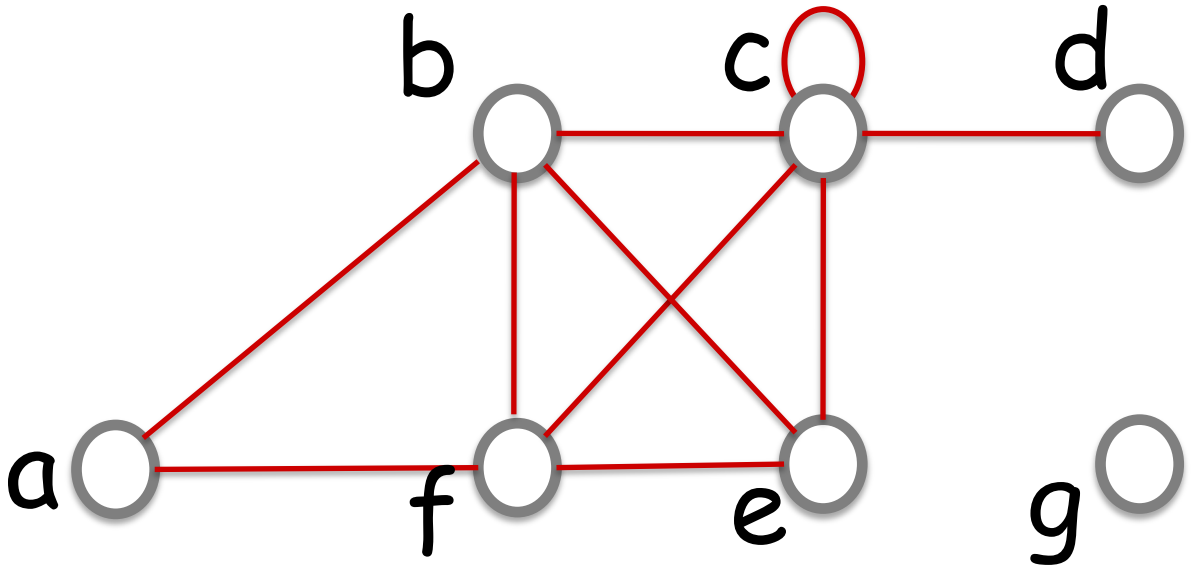
Two vertices are neighbors if they are connected by an edge

The number of neighbors of a vertex is its degree.

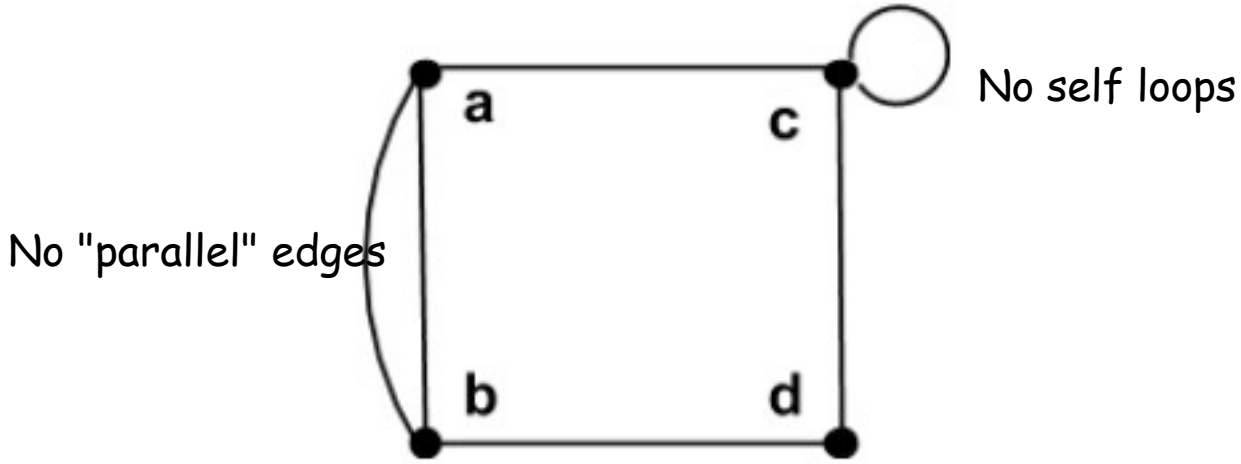
question

What is the degree of c?

A. 4
B. 5
C. 6



undirected graphs



self loop: an edge that connects a vertex to itself

simple graph: no self loops and no two edges that connect the same vertices.

We will focus on simple graphs.

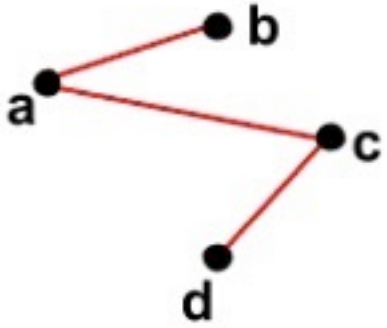
the handshake theorem

Theorem: Let G=(V,E) be an undirected graph. Then

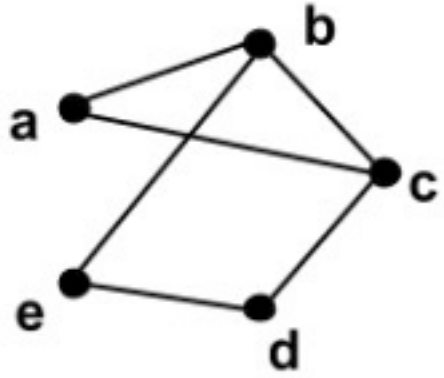
$$\sum_{v \in V} \deg(v) = 2 \mid E \mid$$

subgraphs

A graph $H = (V_H, E_H)$ is a subgraph of a graph $G = (V_G, E_G)$ if $V_H \subseteq V_G$ and $E_H \subseteq E_G$.



H



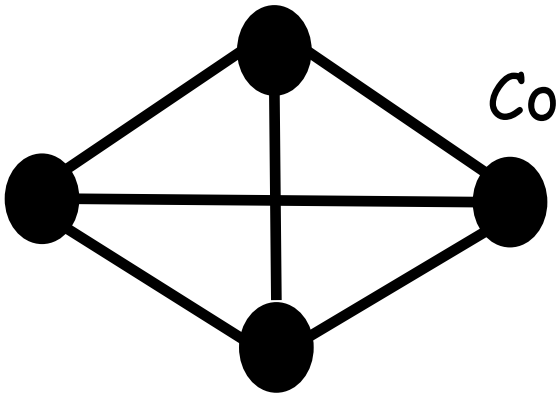
G

complete graphs

A complete graph is a simple graph that has an edge between every pair of vertices.

The complete graph with n vertices is denoted by K_n

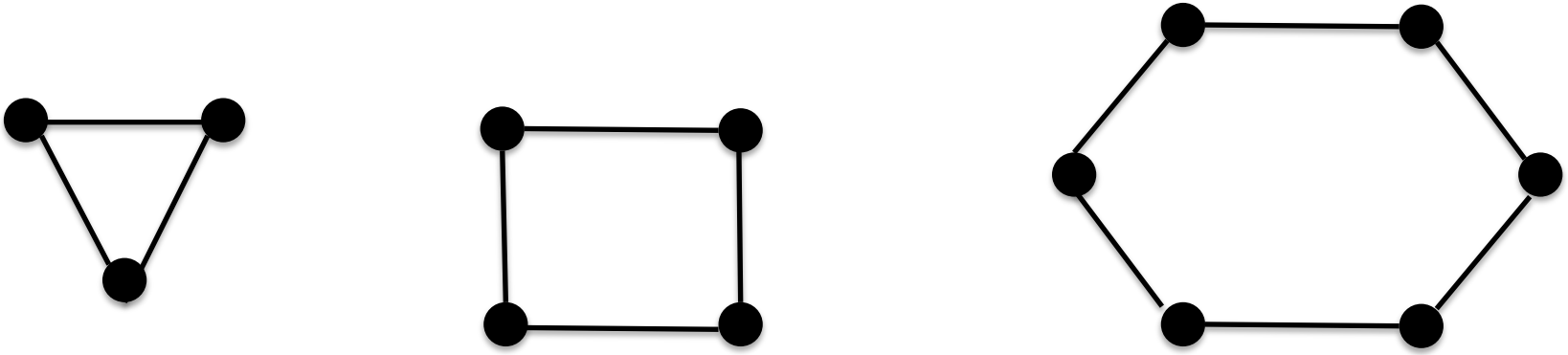
K_4:



Complete Graph

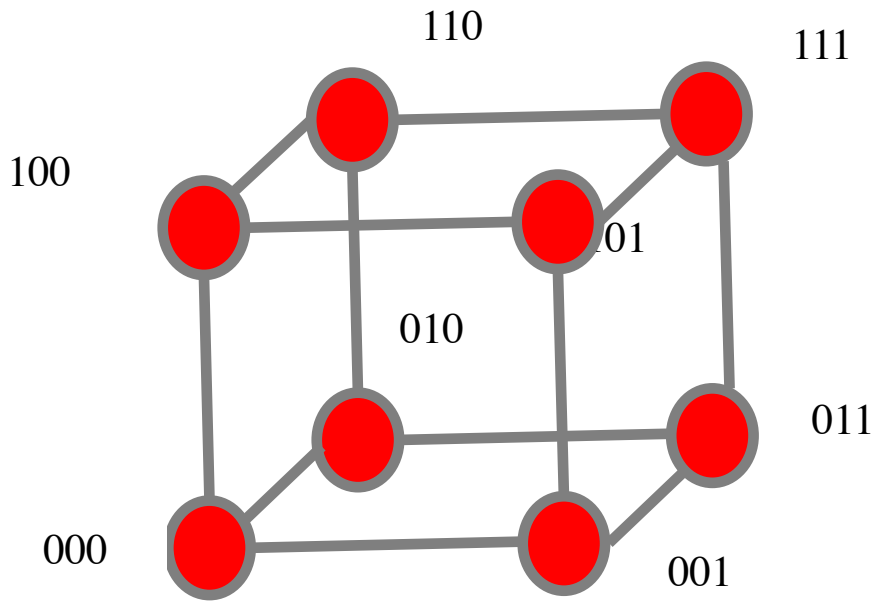
cycles

The cycle C_n, $n \geq 3$, consists of n vertices $v_1, v_2, \ldots, v_n$ and n edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$.



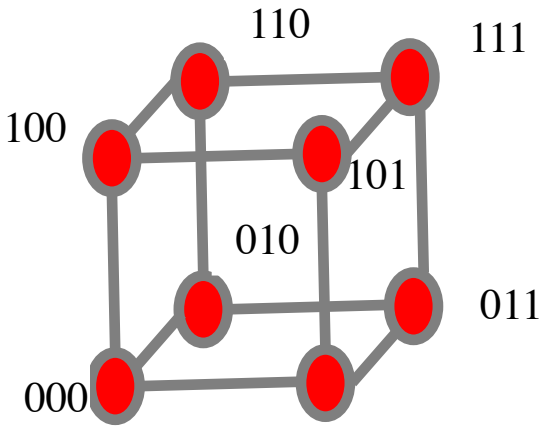
the n-dimensional hypercube

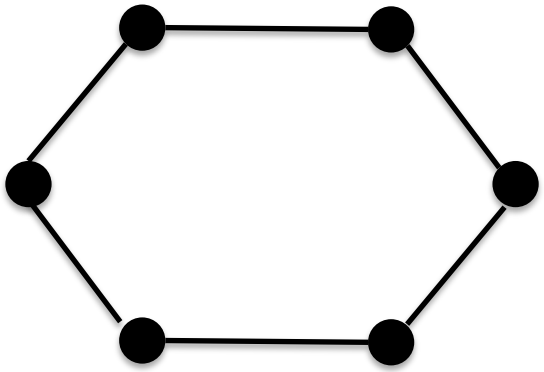
The Hypercube Q_3

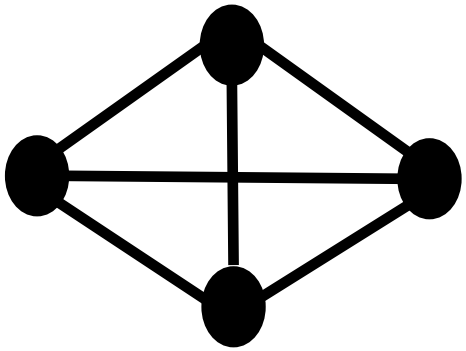


regular graphs

A regular is a graph in which all vertices have the same degree.

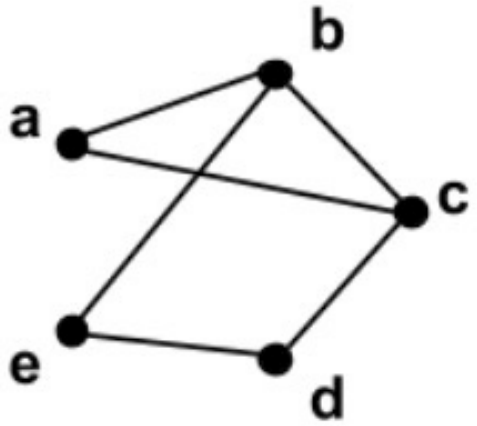


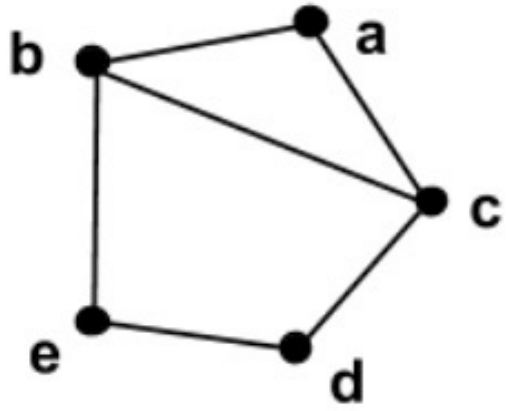




looks can be misleading

Consider the following two graphs:



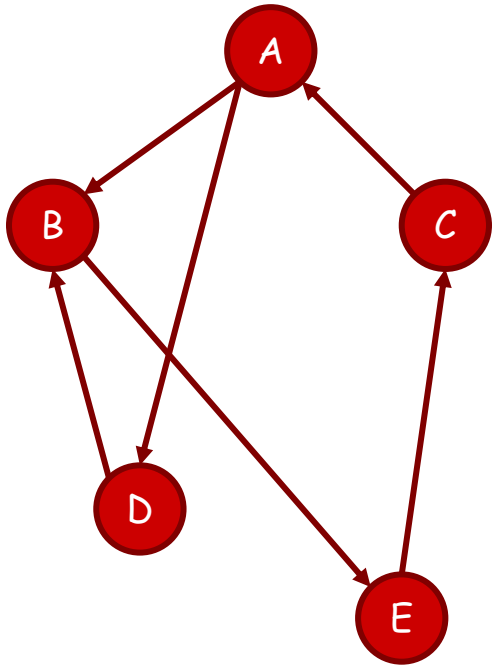


Are they the same?

adjacency matrix of a graph

mapping of vertex labels to array indices

Label	Index
A	0
B	1
C	2
D	3
E	4



	0	1	2	3	4
0	0	1	0	1	0
1	0	0	0	0	1
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0

Adjacency matrix: n x n matrix with entries that indicate if an edge between two vertices is present

For an undirected graph, what would the adjacency matrix look like?

question

Is this the adjacency matrix of an undirected graph?

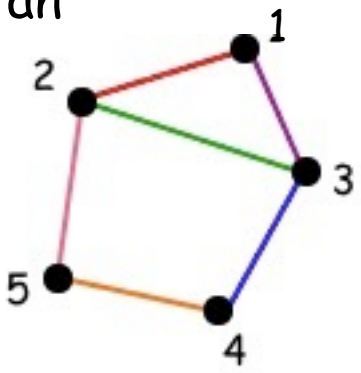
A. Yes
B. No

Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	1
2	1	0	0	0	1
3	0	1	0	1	0
4	0	1	1	0	0

adjacency matrix

Adjacency matrix for an undirected graph



	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	0	1
3	1	1	0	1	0
4	0	0	1	0	1
5	0	1	0	1	0

question

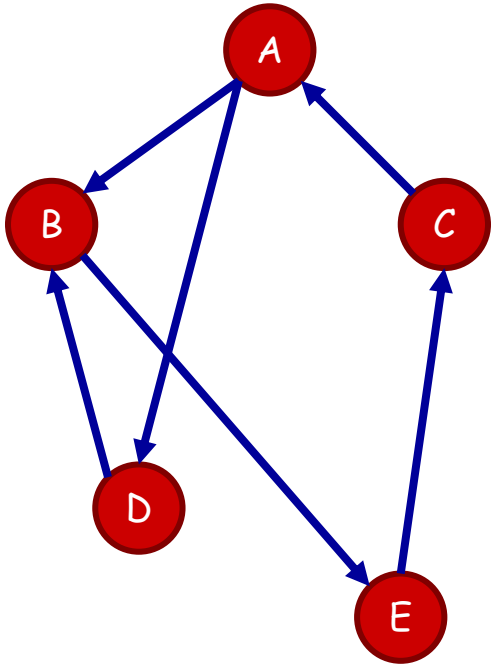
Does this graph have self loops?

A. Yes
B. No

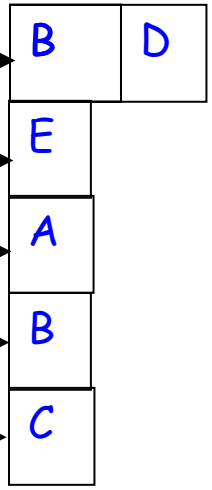
Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	1
2	1	0	0	0	1
3	0	1	0	1	0
4	0	1	1	0	0

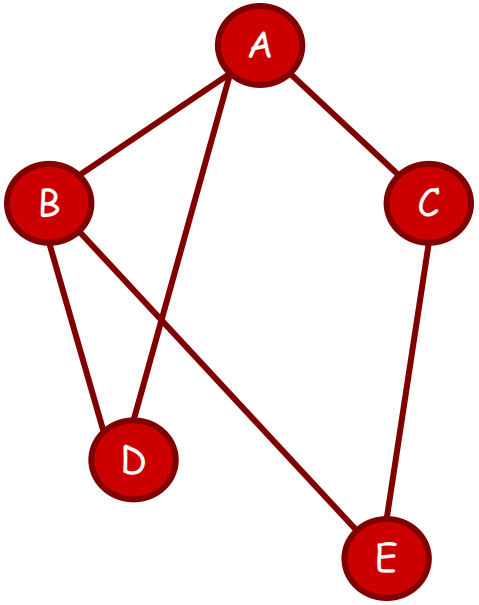
adjacency list for a directed graph



Index	Label
0	A
1	B
2	C
3	D
4	E



adjacency list for an undirected graph



Index	Label			
0	A	B	C	D
1	B	A	D	E
2	C	A	E	
3	D	A	B	
4	E	B	C	

mapping of vertex labels to list of edges

which implementation

Adjacency matrix

- Edges are entries in a square matrix
  - size: $|V|^2$
- values:
  - 1/0 to indicate presence/absence of edge in (un)directed graph

useful for dense graphs

Adjacency list

- linked-list of out-going edges per vertex

useful for sparse graphs

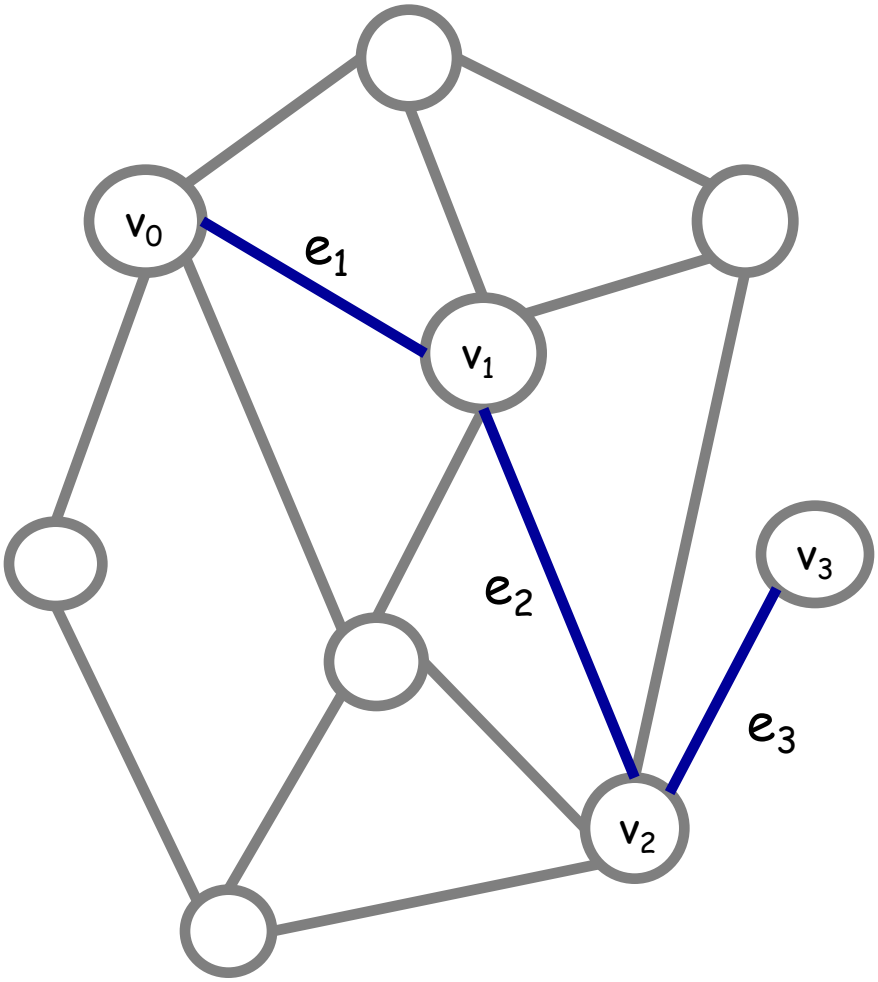
which implementation

Which implementation best supports common graph operations:

- Is there an edge between vertex i and vertex j?
- Find all vertices adjacent to vertex j
- What's the big O for each of these operations?

Which best uses space?

walks



A walk from v_0 to v_l in an undirected graph G is a sequence of alternating vertices and edges that starts and ends with a vertex:

$\langle v_0,\{v_0,v_1\},v_1,\{v_1,v_2\},v_2,\ldots,v_{l-1},\{v_{l-1},v_l\},v_l\rangle$

A walk can also be denoted by the sequence of vertices:

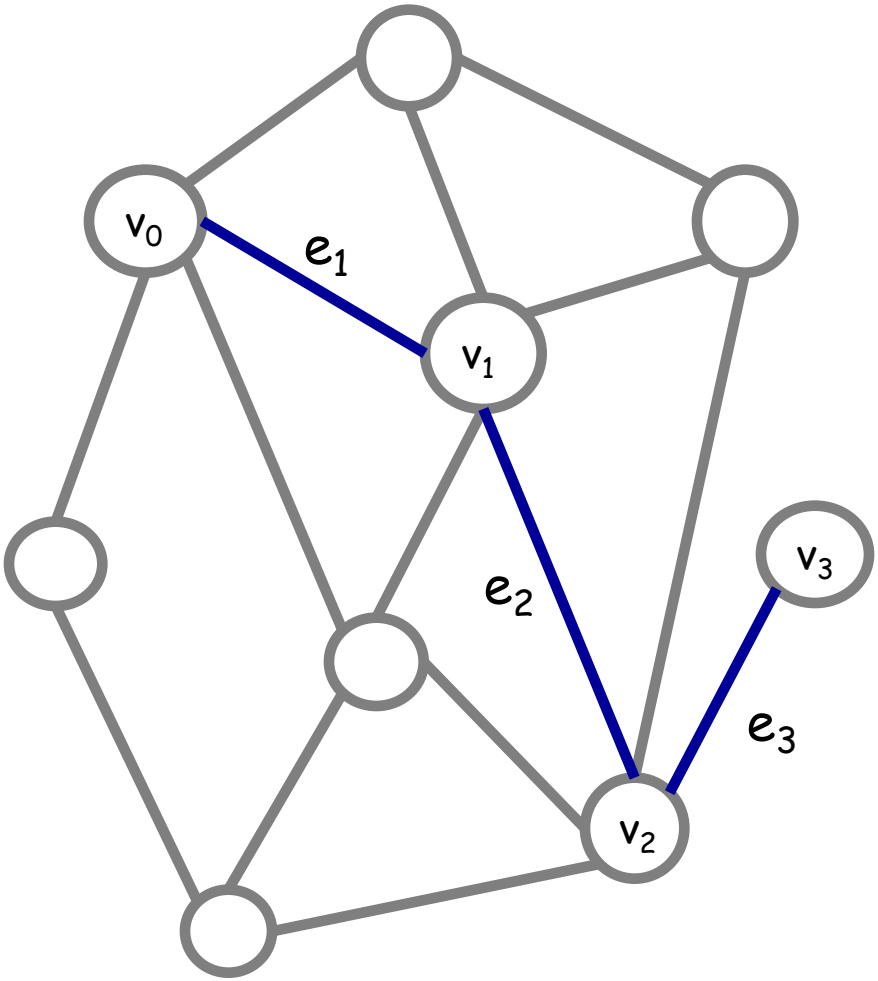
$\langle v_0,v_1,\ldots,v_l\rangle$.

The sequence of vertices is a walk only if $\{v_{i-1}, v_i\} \in E$

for $i = 1, 2,\ldots,l$.

The length of a walk is l, the number of edges in the walk.

walks, circuits, paths, cycles



A circuit is a walk in which the first vertex is the same as the last vertex.

A sequence of one vertex, denoted <a>, is a circuit of length 0.

A walk is a path if no vertex is repeated in the walk.

A circuit is a cycle if there are no other repeated vertices, except the first and the last.

Same as in directed graphs.

walks, circuits, paths, cycles

A circuit is a walk in which the first vertex is the same as the last vertex.

A walk is a path if no vertex is repeated in the walk.

A circuit is a cycle if there are no other repeated vertices, except the first and the last.

- What is the length of the longest possible walk in a graph with n vertices?
- What is the length of the longest possible path in a graph with n vertices?
- What is the length of the longest possible circuit in a graph with n vertices?
- What is the length of the longest possible cycle in a graph with n vertices?

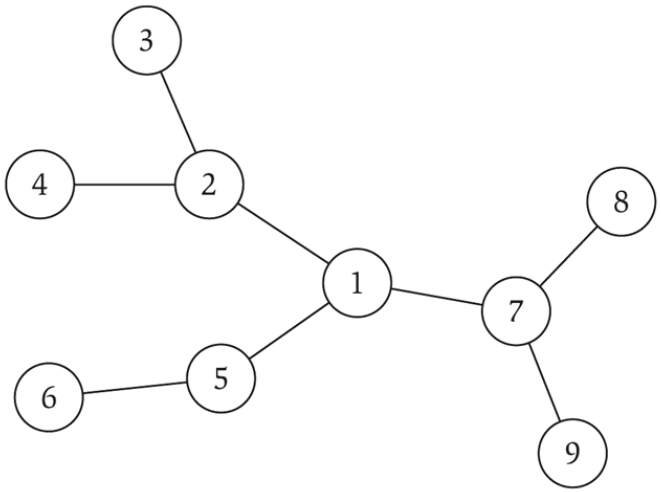
Trees

Def. An undirected graph is a tree if it is connected and does not contain a cycle.

How many edges does a tree have?

Given a set of nodes, build a tree step wise

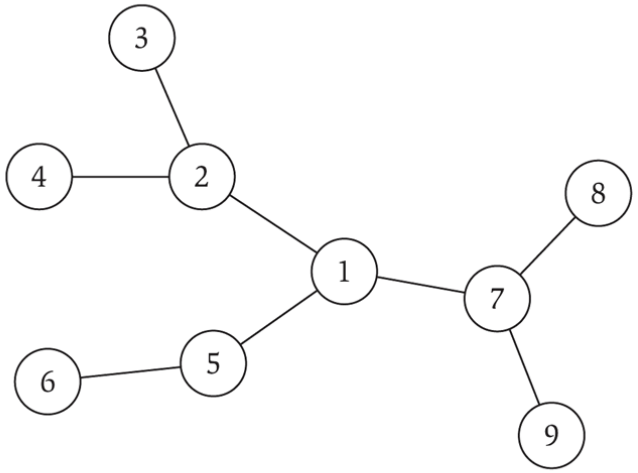
- every time you add an edge, you must add a new node to the growing tree. WHY?
- how many edges to connect n nodes?



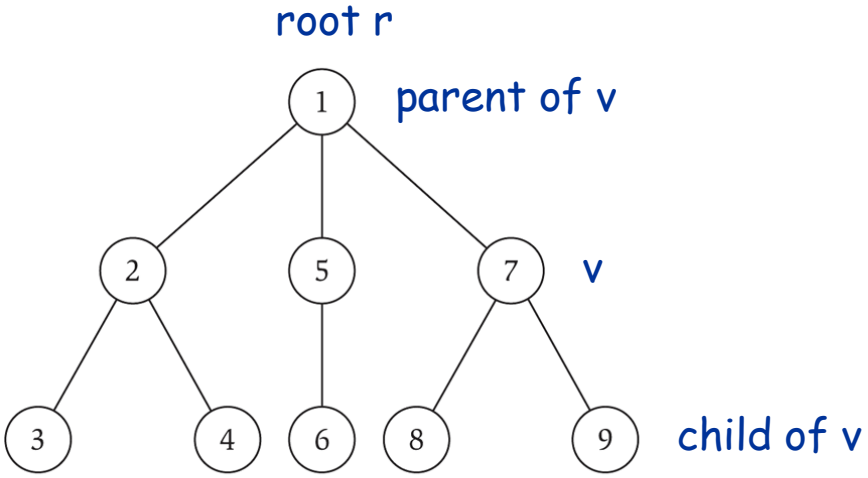
Rooted Trees

Rooted tree. Given a tree T, choose a root node r and orient each edge below r; do same for sub-trees.

Models hierarchical structure. By rooting the tree it is easy to see that it has n-1 edges.



a tree



the same tree, rooted at 1

Traversing a Binary Tree

Pre order

- visit the node
- go left
- go right

In order

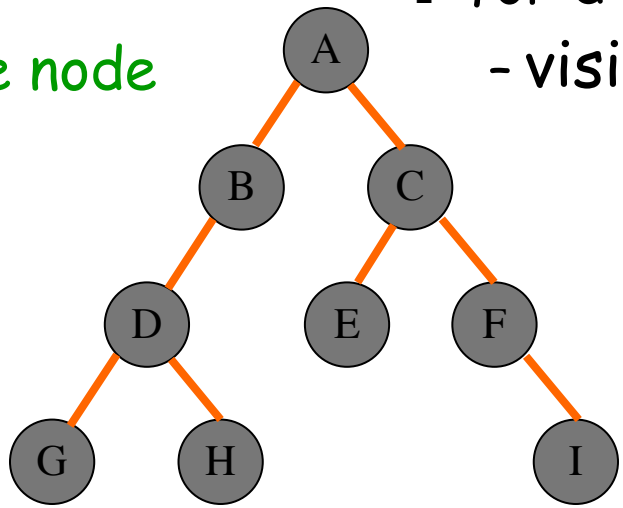
- go left
- visit the node
- go right

Post order

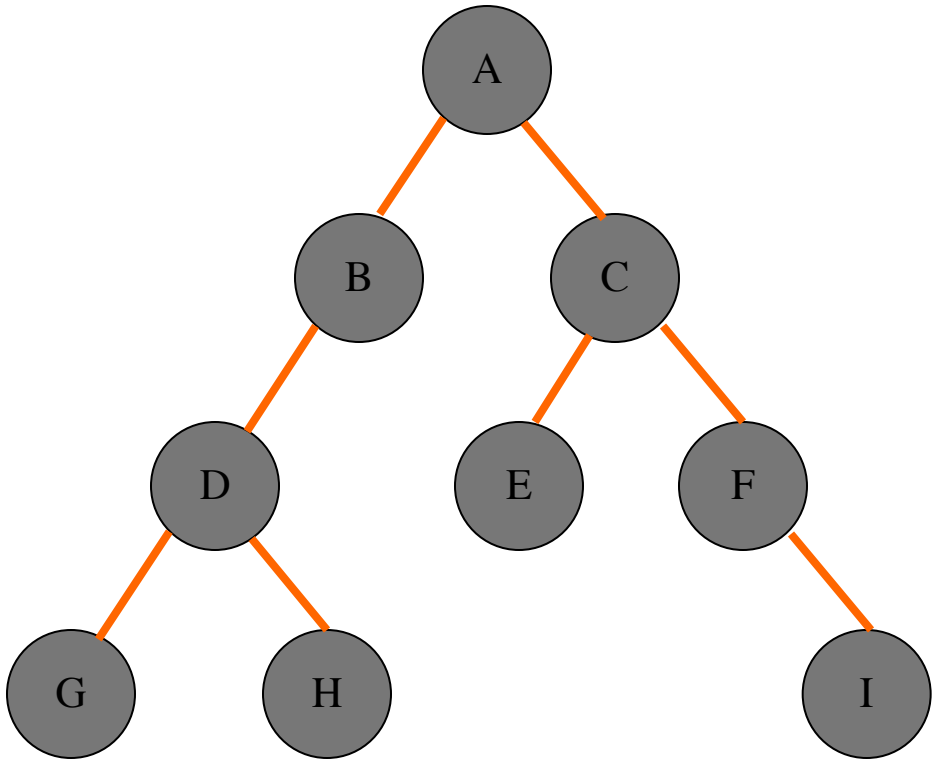
- go left
- go right
- visit the node

Level order / breadth first

- for d = 0 to height
  - visit nodes at level d



Traversal Examples



Pre order

A B D G H C E F I

In order

G D H B A E C F I

Post order

G H D B E I F C A

Level order

A B C D E F G H I

IMPLEMENTATION of these traversals??

Tree traversal Implementation

recursive implementation of preorder

- The steps:
  - visit node
  - preorder(left child)
  - preorder(right child)
- What changes need to be made for in-order, post-order?

How would you implement level order?

Graph Traversal

What makes it different from tree traversals?

Graph Traversal

What makes it different from tree traversals:

- you can visit the same node more than once
- you can get in a cycle

What to do about it?

Graph Traversal

What makes it different from tree traversals:

- you can visit the same node more than once
- you can get in a cycle

What to do about it:

- **mark** the nodes
  - -White: unvisited
  - -Grey: (still being considered) on the frontier: not all adjacent nodes have been visited yet
  - -Black: off the frontier: all adjacent nodes visited (not considered anymore)

BFS: Breadth First Search

Like **level** traversal in trees, **BFS(G,s)** explores the edges of G and locates every node v reachable from s in a level order using a queue.

BFS: Breadth First Search

Like level traversal in trees, **BFS(G,s)** explores the edges of G and locates every node v reachable from s in a level order using a queue.

BFS also computes the **distance**: number of edges from s to all these nodes, and the **shortest path** (minimal #edges) from s to v.

BFS: Breadth First Search

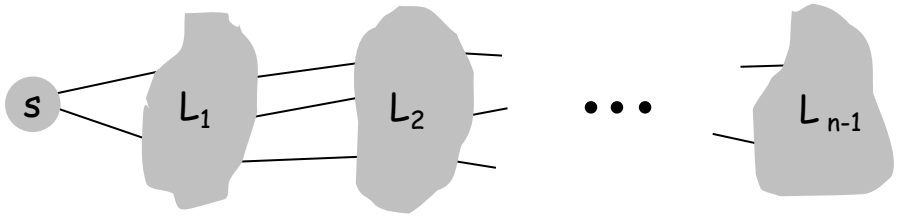
Like level traversal in trees, **BFS(G,s)** explores the edges of G and locates every node v reachable from s in a level order using a queue.

BFS also computes the **distance:** number of edges from s to all these nodes, and the **shortest path** (minimal #edges) from s to v.

BFS expands a **frontier** of **discovered** but not yet visited nodes. Nodes are colored white, grey or black. They start out undiscovered or white.

Breadth First Search

BFS intuition. Explore outward from s, adding nodes one "layer" at a time.



BFS algorithm.

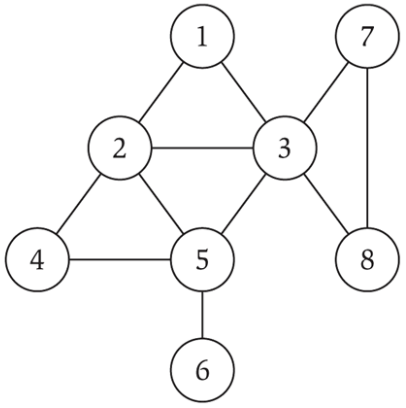
- $L_0 = \{ s \}$.
- L_1 = all neighbors of L_0.
- L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
- L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

For each i, L_i consists of all nodes at distance exactly i from s. There is a path from s to t iff t appears in some layer.

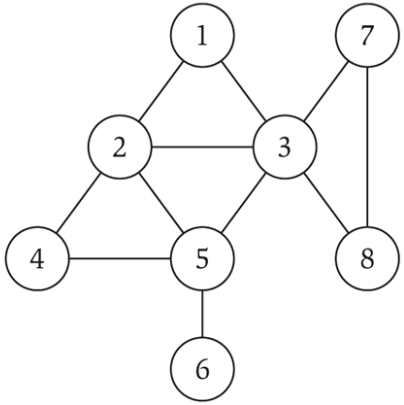
Breadth First Tree

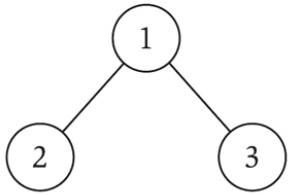
BFS produces a **Breadth First (spanning) Tree** rooted at s: when a node v in L_{i+1} is discovered as a neighbor of node u in L_i we add edge (u,v) to the BF tree

Property. Let T be a BFS tree of G, and let (x, y) be an edge of G. Then the level of x and y differ by at most 1. **WHY?**

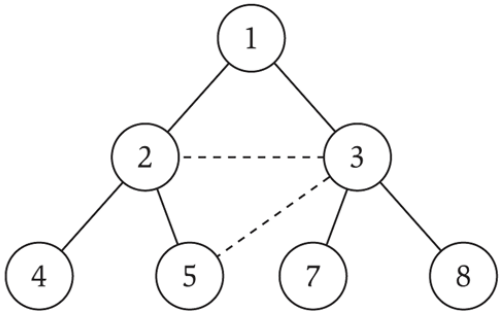


Breadth First Search

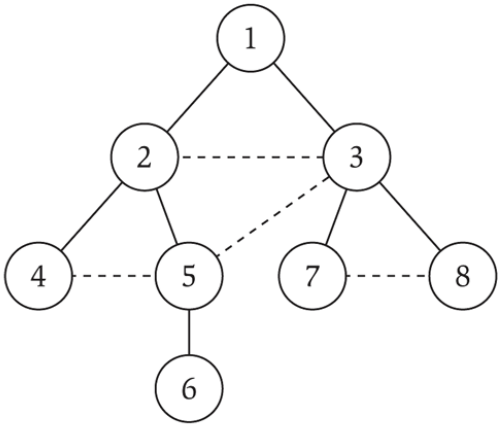




(a)



(b)



(c)

L_0

L_1

L_2

L_3

```
BFS(G,s)
  #d: distance,  c: color,  p: parent in BFS tree
  forall v in V-s {c[v]=white; d[v]=∞,p[v]=nil}
  c[s]=grey; d[s]=0; p[s]=nil;
  Q=empty;
  enque(Q,s);
  while (Q != empty)
     u = deque(Q);
      forall  v in adj(u)
         if (c[v]==white)
            c[v]=grey; d[v]=d[u]+1; p[v]=u;
            enque(Q,v)
     c[u]=black;
   # don't really need grey here, why?
```

Complexity BFS

Each node is painted white once, and is enqueued and dequeued at most once.

Complexity BFS

Each node is painted white once, and is enqueued and dequeued at most once.

Why?

Complexity BFS

Each node is painted white once, and is enqueued and dequeued at most once.

Enque and deque take constant time. The adjacency list of each node is scanned only once: when it is dequeued.

Complexity BFS

Each node is painted white once, and is enqueued and dequeued at most once.

Enque and deque take constant time. The adjacency list of each node is scanned only once, when it is dequeued.

Therefore time complexity for BFS is

$O(|V|+|E|)$ or $O(n+m)$

DFS: Depth First Search

Explores edges from the most recently discovered node; backtracks when reaching a dead-end. The algorithm below does not use white, grey, black, but uses explored (and implicitly unexplored). Recursive code:

```
DFS(u):
  mark u as Explored and add u to R
  for each edge (u,v) :
    if v is not marked Explored :
       DFS(v)
```

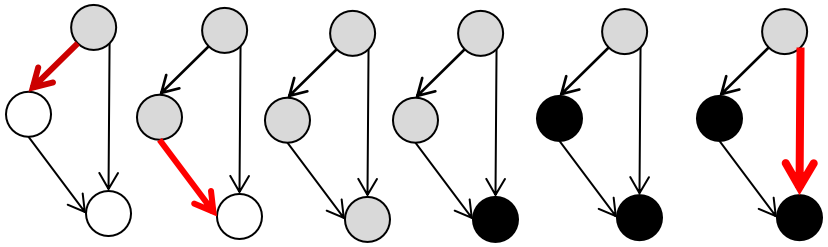
BUT, how do we find cycles in a graph?

DFS and cyclic graphs

There are two ways DFS can **revisit** a node:

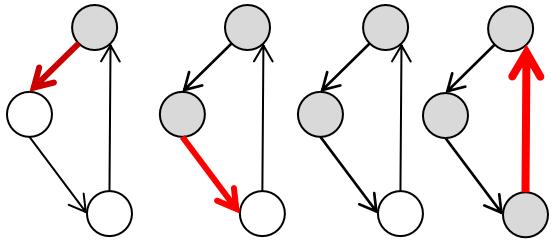
1. DFS has already fully explored the node. **What color does it have then?**

Is there a cycle then?



2. DFS is still exploring this node. **What color does it have in this case?**

Is there a cycle then?

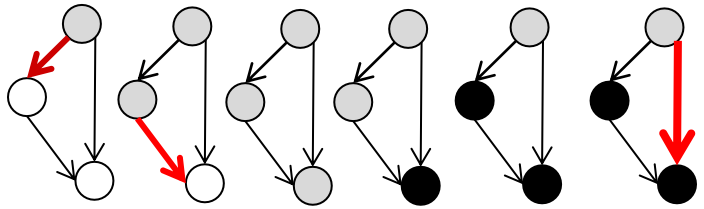


DFS and cyclic graphs

There are two ways DFS can **revisit** a node:

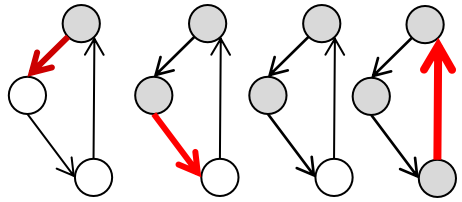
1. DFS has already fully explored the node. **What color does it have then? Is there a cycle then?**

No, the node is revisited from outside.



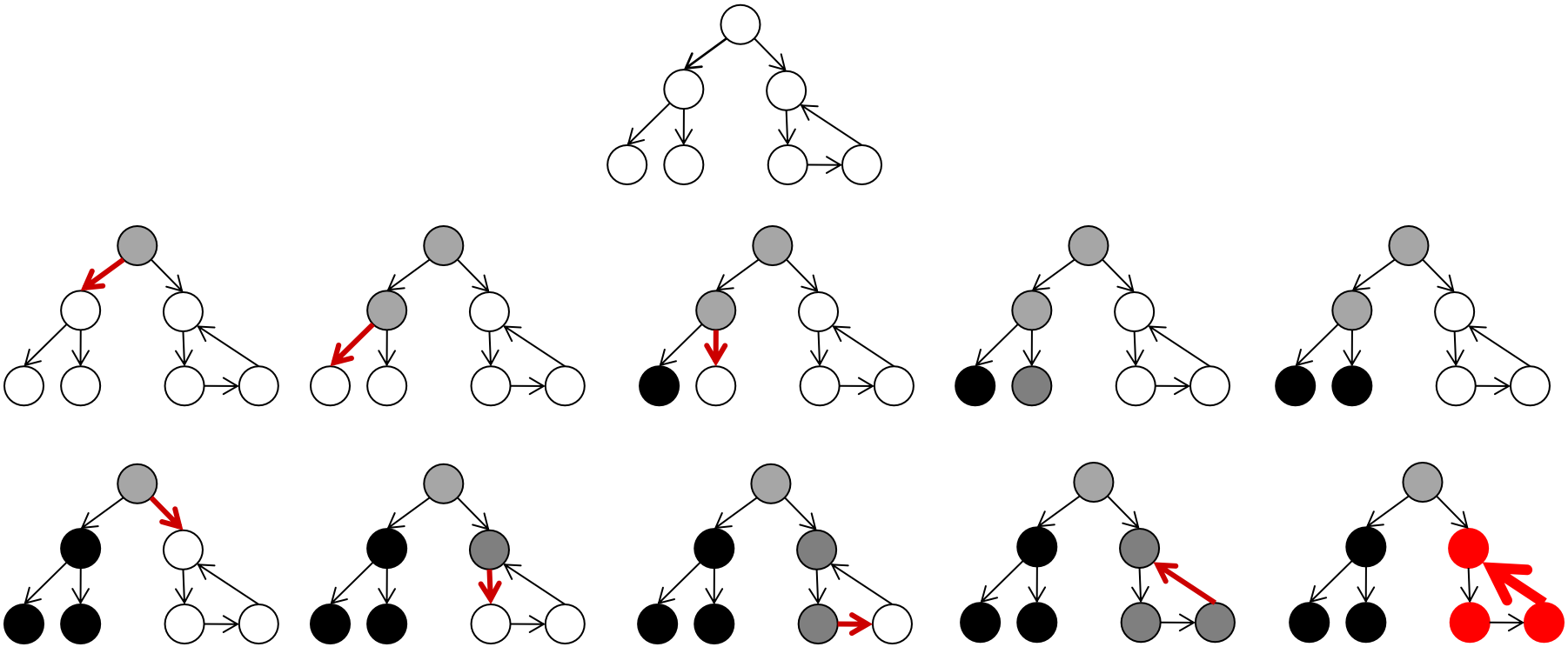
2. DFS is still exploring this node. **What color does it have in this case? Is there a cycle then?**

Yes, the node is revisited on a path containing the node itself.



So DFS with the white, grey, black coloring scheme detects a cycle when a GREY node is visited.

Cycle detection: DFS + coloring



When a grey (frontier) node is visited, a cycle is detected.

Recursive / node coloring version

```
DFS(u):
    #c: color,  p: parent
    c[u]=grey
    forall  v in Adj(u):
      if c[v]==white:
        p[v]=u
        DFS(v)
    c[u]=black
```

The above implementation of DFS runs in $O(m + n)$ time if the graph is given by its adjacency list representation.

Proof:

Same as in BFS ▪

DFS and cyclic graphs

When DFS visits a node for the first time it is white. There are two ways DFS can **revisit** a node:

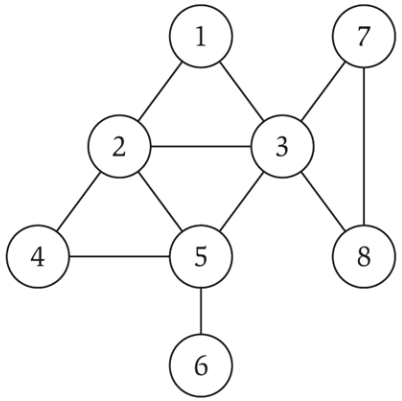
1. DFS has already fully explored the node. **What color does it have then? Is there a cycle then?**

2. DFS is still exploring this node. **What color does it have in this case? Is there a cycle then?**

Connectivity

s-t connectivity problem. Given two node s and t, is there a path between s and t?

s-t shortest path problem. Given two nodes s and t, what is the length of the shortest path between s and t? Length: either in terms of number of edges, or in terms of sum of weights.

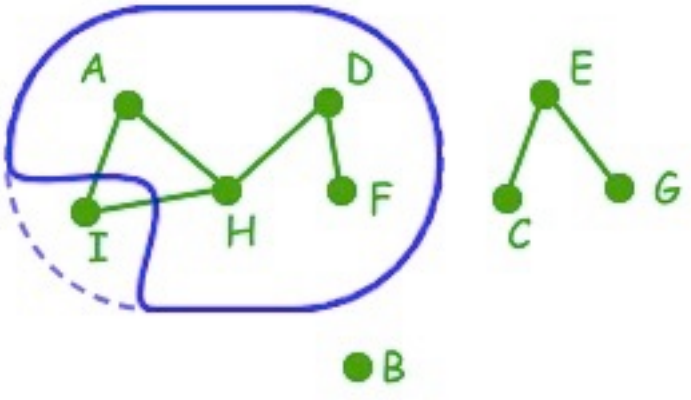


connected components

An undirected graph is called connected if there is a path between every pair of vertices.

A connected component is a maximal set of vertices that is connected.

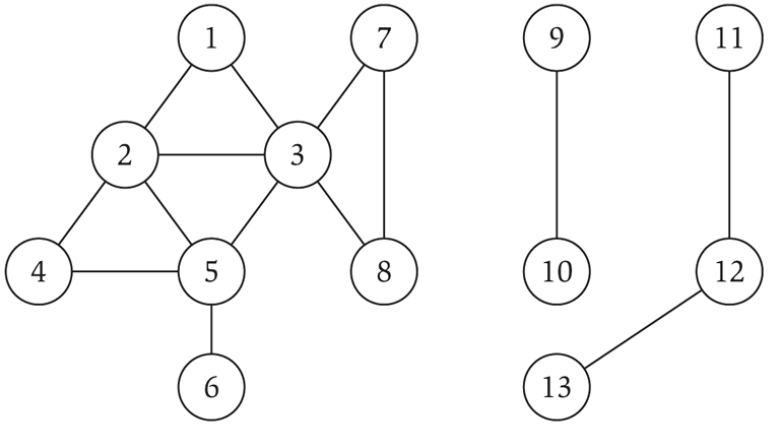
The word "maximal" means that if any vertex is added to a connected component, then the set of vertices will no longer be connected.



Connected Components

Connected graph. There is a path between any pair of nodes.

Connected component of a node s. The set of all nodes reachable from s.

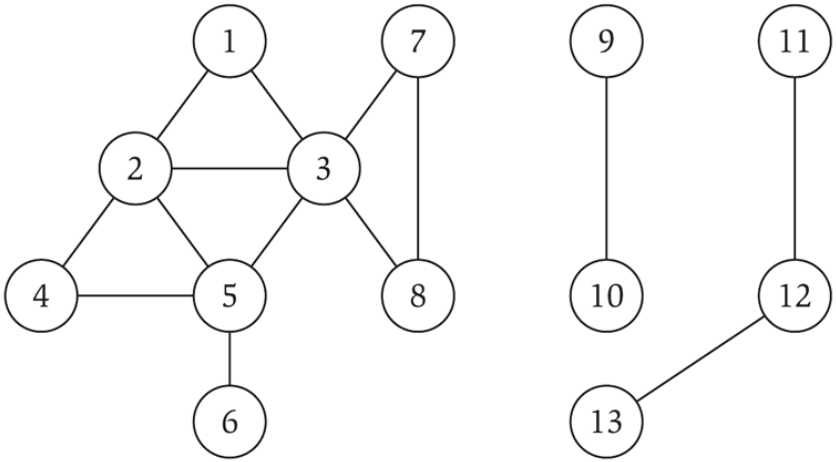


Connected component containing node 1 = { 1, 2, 3, 4, 5, 6, 7, 8 }.

Connected Components

Connected component of a node s. The set of all nodes reachable from s.

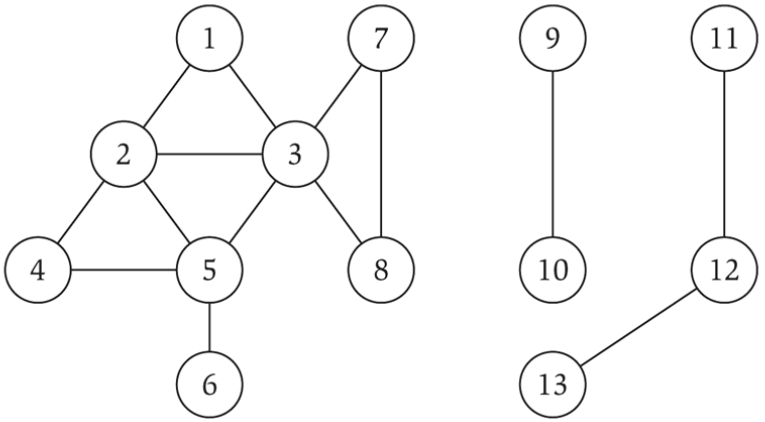
Given two nodes s, and t, their connected components are either identical or disjoint. **WHY?**



Connected Components

Connected component of a node s. The set of all nodes reachable from s.

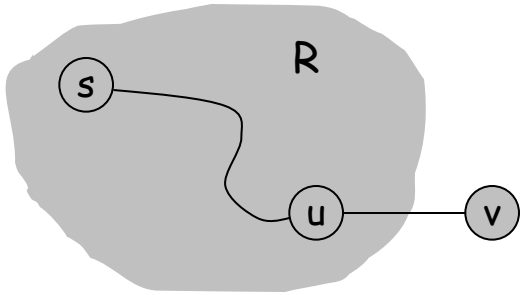
Given two nodes **s**, and **t**, their connected components are either identical or disjoint.



Two cases – either there is a path between the two nodes, or there isn't.
If there is a path: take a node u in the connected component of s, and construct a path from t to u: t to s, then s to u, so $CC_s = CC_t$
If there is no path: assume that the intersection contains a node u. Use it to construct a path between s and t: s to u, then u to t – **contradiction**.

Connected Components

A generic algorithm for finding connected components:




```
R = {s}  # the connected component of s is initially s.
while there is an edge (u,v) where u is in R and v is not in R:
    add v to R
```

Upon termination, R is the connected component containing s.

- BFS: explore in order of distance from s.
- DFS: explores edges from the most recently discovered node; backtracks when reaching a dead-end.

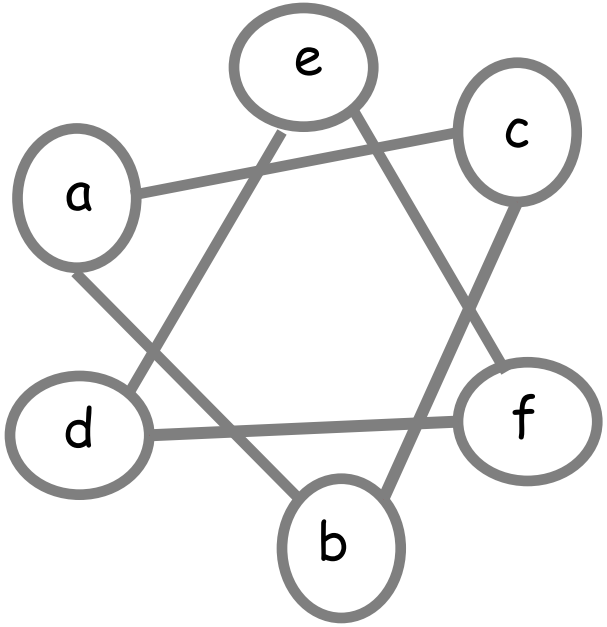
example

How many connected components does this graph have?

A. 0

B. 1

C. 2



The facebook graph

- 721 million active accounts
- 68.7 billion friendship edges (median number of friends = 99)
- The largest connected component of facebook users contains 99.9% of the users
- Average distance between any pair of users: 4.7

source: http://arxiv.org/pdf/1111.4503v1.pdf