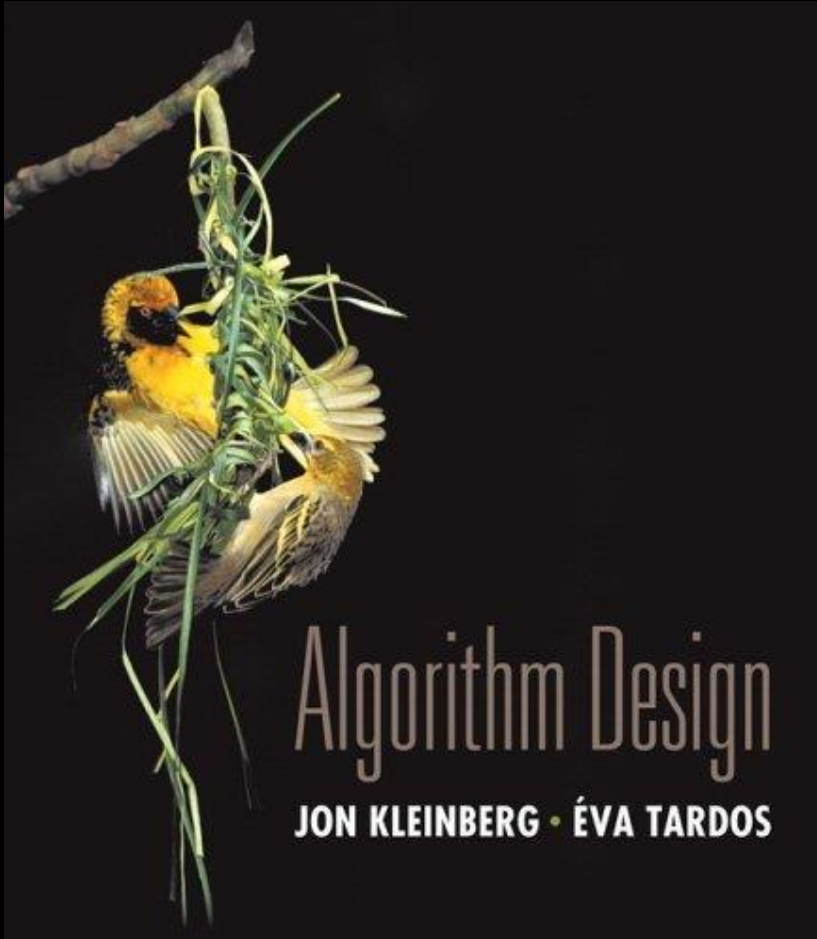


Chapter 3

Graphs



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

3.1 Basic Definitions and Applications




Used when including notation I would use instead of what the text is using.


The notation I use tends to follow West and is more common in graph theory papers/books.

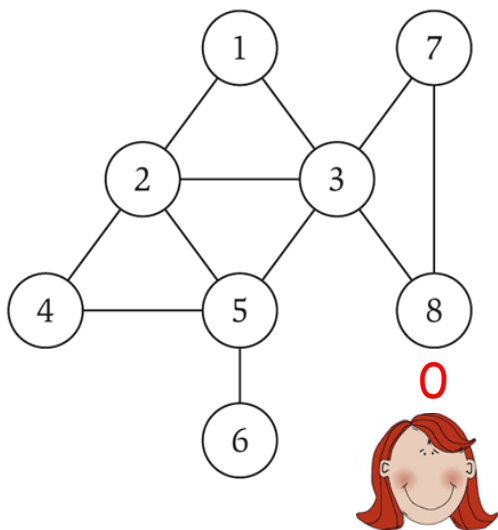
<http://all-free-download.com/free-vector/girl-face-cartoon-clip-art.html>

Undirected Graphs

Undirected graph. $G = (V, E)$

- $V =$ nodes [vertices ]
- $E =$ edges between pairs of nodes.
- Captures pairwise relationship between objects.
- Graph size parameters: $n = |V|$, $m = |E|$.

IMPORTANT! 
Singular: vertex
Plural : vertices



For programs, $V = \{0, 1, \dots, 7\}$ 

$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

$E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 \}$

$n = 8$

$m = 11$

$E = \{(1,2), (1,3), \dots, (5,6)\}$ 

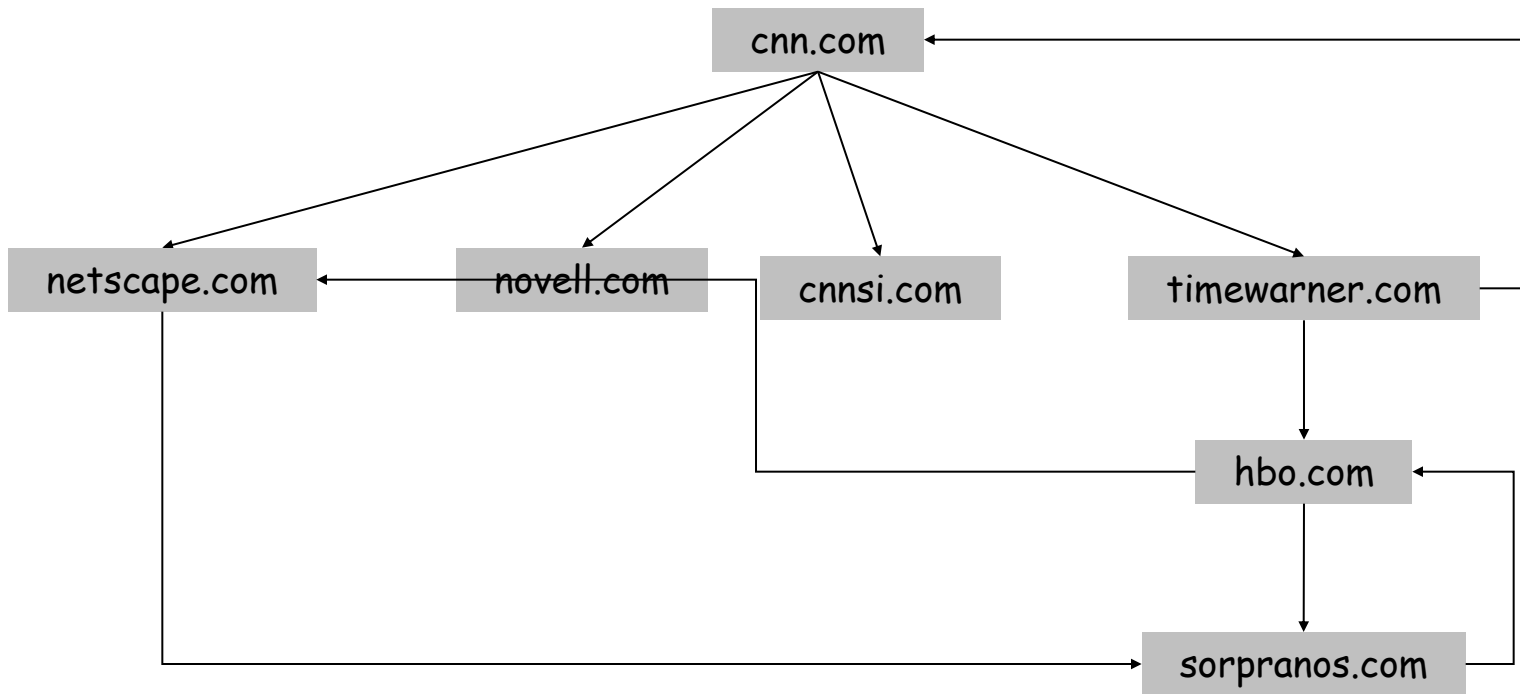
Some Graph Applications

<i>Graph</i>	<i>Nodes</i>	<i>Edges</i>
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires

World Wide Web

Web graph.

- Node: web page.
- Edge: hyperlink from one page to another.



9-11 Terrorist Network

Social network graph.

- Node: people.
- Edge: relationship between two

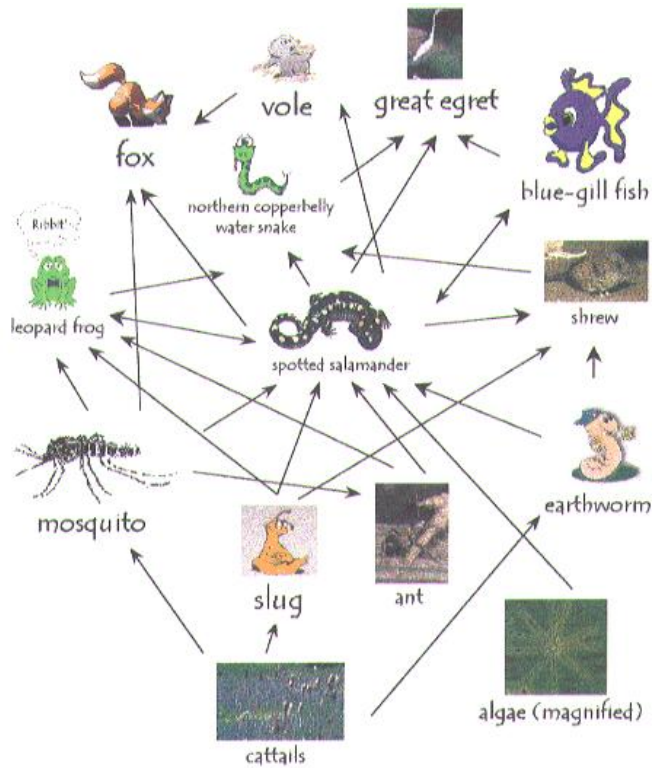


Reference: Valdis Krebs, http://www.firstmonday.org/issues/issue7_4/krebs

Ecological Food Web

Food web graph.

- Node = species.
- Edge = from prey to predator.



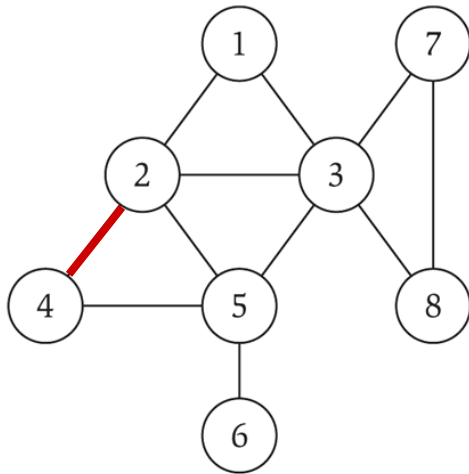
Reference: <http://www.twingroves.district96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.gif>

Graph Representation: Adjacency Matrix

Adjacency matrix. n -by- n matrix with $A_{uv} = 1$ if (u, v) is an edge.

- Two representations of each edge.
- Space proportional to n^2 .
- Checking if (u, v) is an edge takes $\Theta(1)$ time.
- Identifying all edges takes $\Theta(n^2)$ time.

Matrix: A
entry: $a_{u,v}$ 



Number starting with 0 

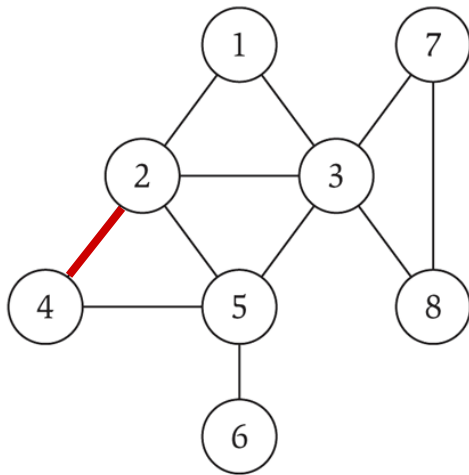
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Graph Representation: Adjacency List

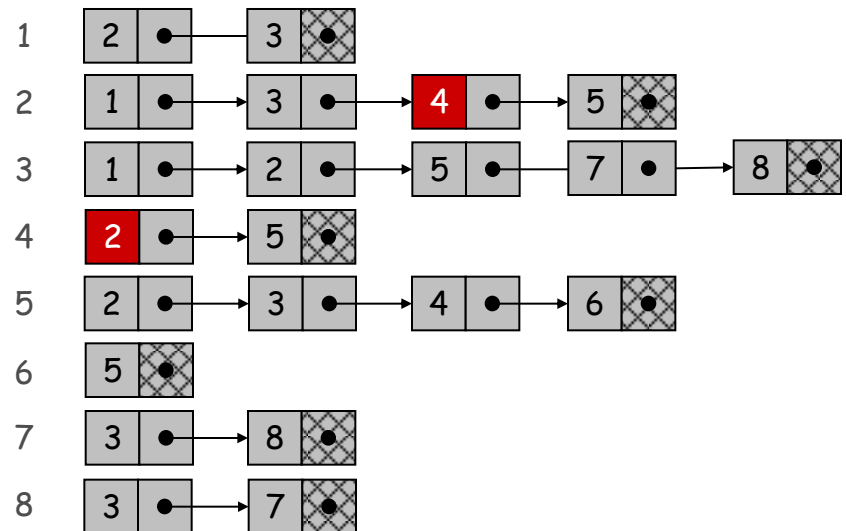
Adjacency list. Node indexed array of lists.

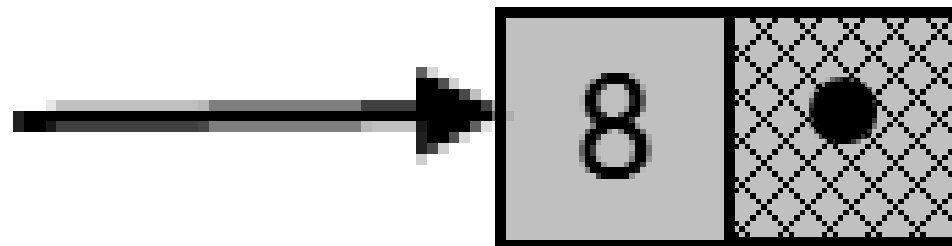
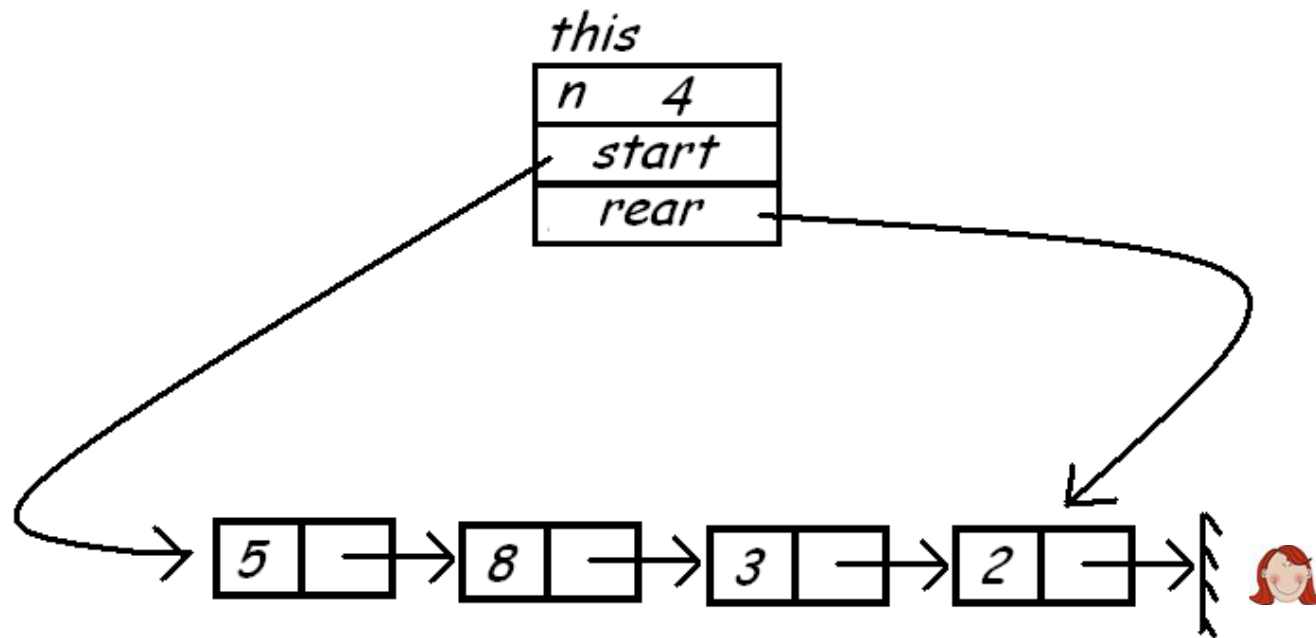
- Two representations of each edge.
- Space proportional to $m + n$.
- Checking if (u, v) is an edge takes $O(\text{deg}(u))$ time.
- Identifying all edges takes $\Theta(m + n)$ time.

degree = number of neighbors of u




Number starting with 0 





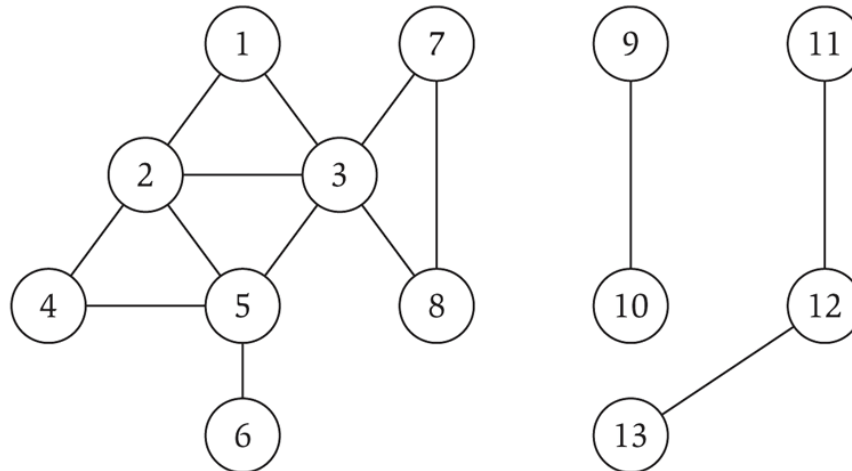
Paths and Connectivity

Def. A **path** in an undirected graph $G = (V, E)$ is a sequence P of nodes $v_1, v_2, \dots, v_{k-1}, v_k$ with the property that each consecutive pair v_i, v_{i+1} is joined by an edge in E .

Def. A path is **simple** if all nodes are distinct.

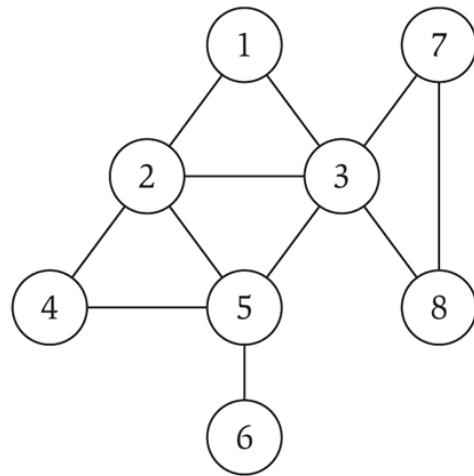
[A path is a simple walk 

Def. An undirected graph is **connected** if for every pair of nodes u and v , there is a path between u and v .



Cycles

Def. A **cycle** is a path $v_1, v_2, \dots, v_{k-1}, v_k$ in which $v_1 = v_k$, $k > 2$, and the first $k-1$ nodes are all distinct.



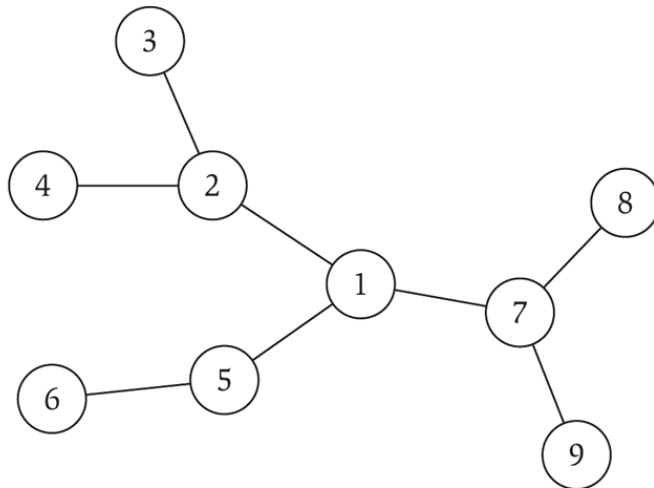
cycle $C = 1-2-4-5-3-1$

Trees

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

Theorem. Let G be an undirected graph on n nodes. Any two of the following statements imply the third.

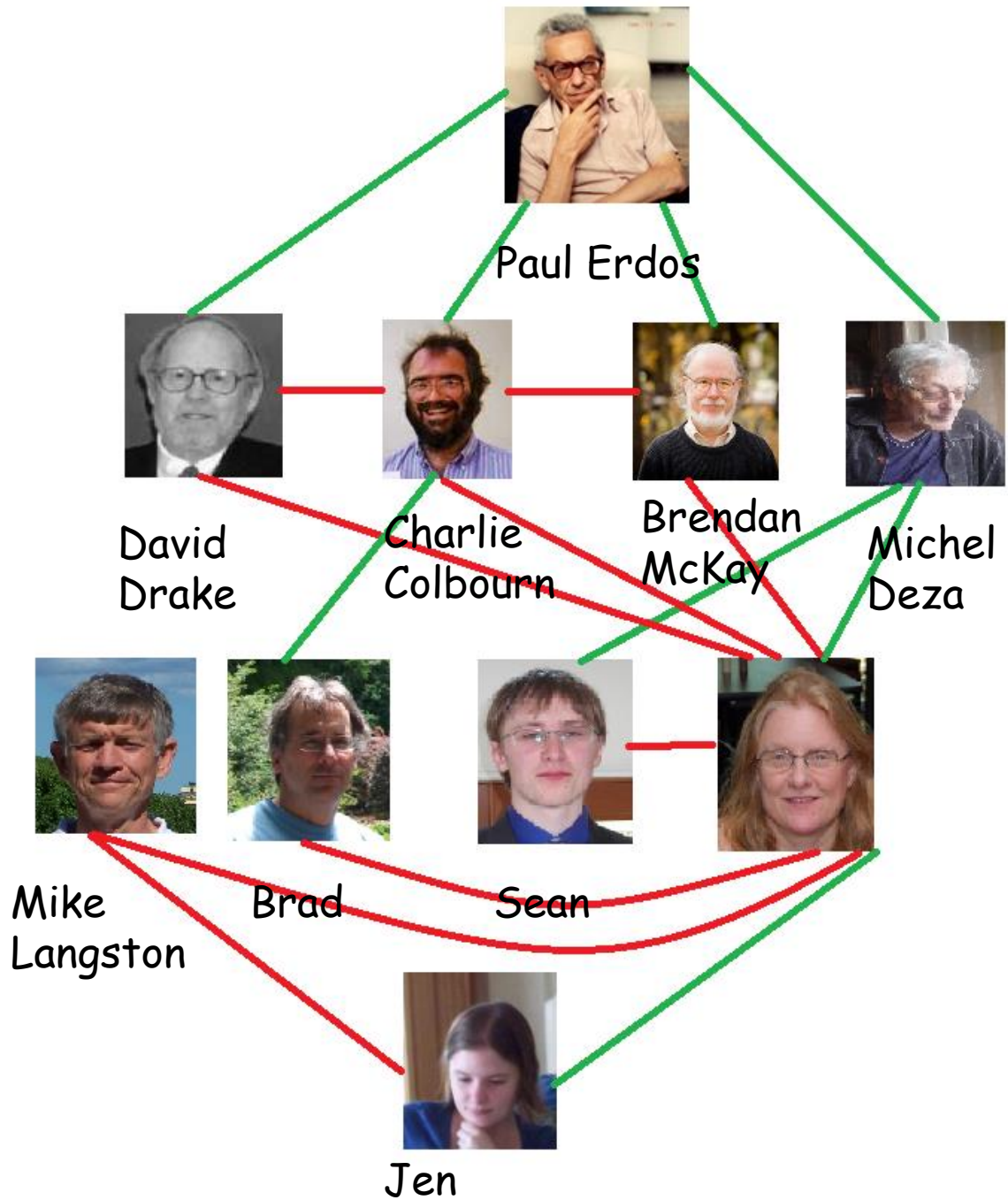
- G is connected.
- G does not contain a cycle.
- G has $n-1$ edges.



http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon

Six Degrees of Kevin Bacon is a parlor game based on the "six degrees of separation" concept, which posits that any two people on Earth are six or fewer acquaintance links apart. That idea eventually morphed into this parlor game, wherein movie buffs challenge each other to find the shortest path between an arbitrary actor and venerated Hollywood character actor Kevin Bacon. It rests on the assumption that any individual involved in the Hollywood, California, film industry can be linked through his or her film roles to Kevin Bacon within six steps. The game requires a group of players to try to connect any such individual to Kevin Bacon as quickly as possible and in as few links as possible. It can also be described as a trivia game based on the concept of the small world phenomenon.

A subgraph of
the Erdős
number graph



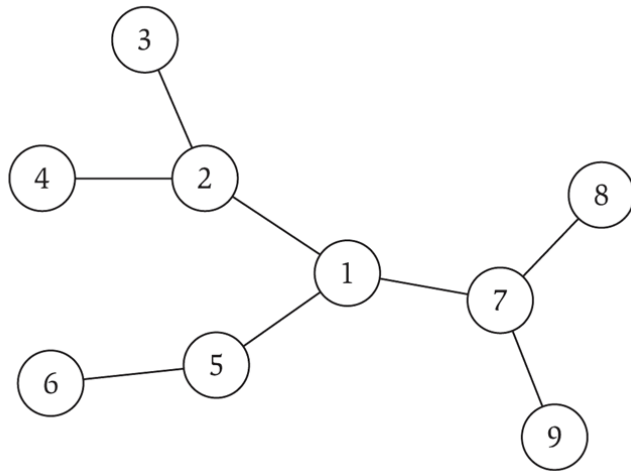
Rooted Trees

Rooted tree. Given a tree T , choose a root node r and orient each edge away from r .

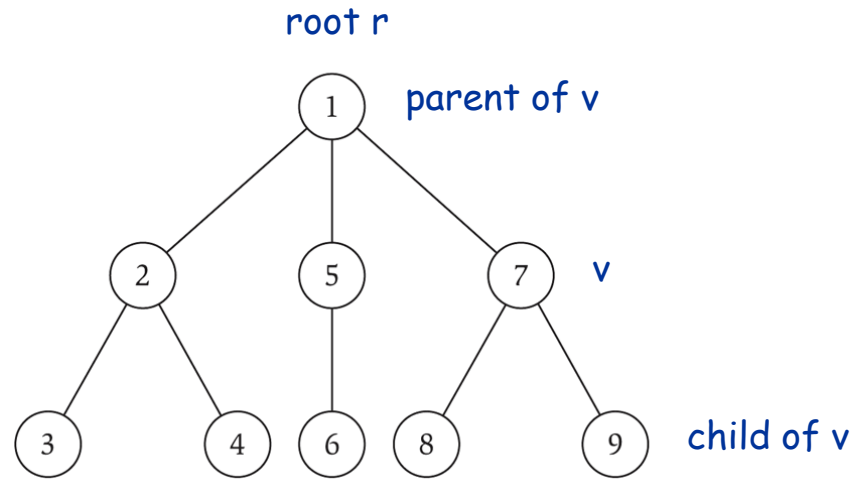
Importance. Models hierarchical structure.



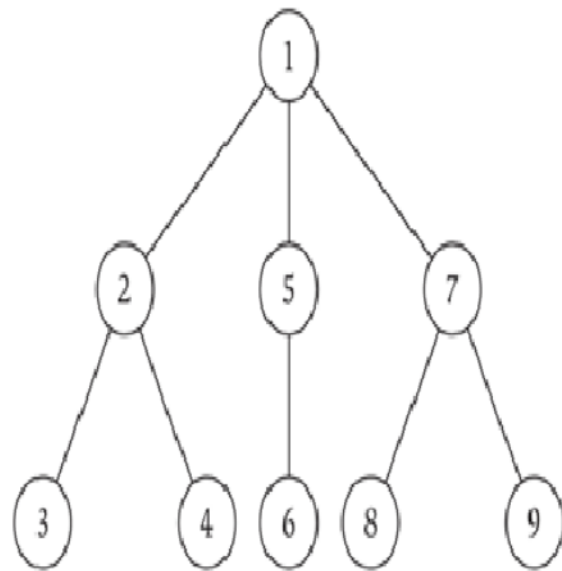
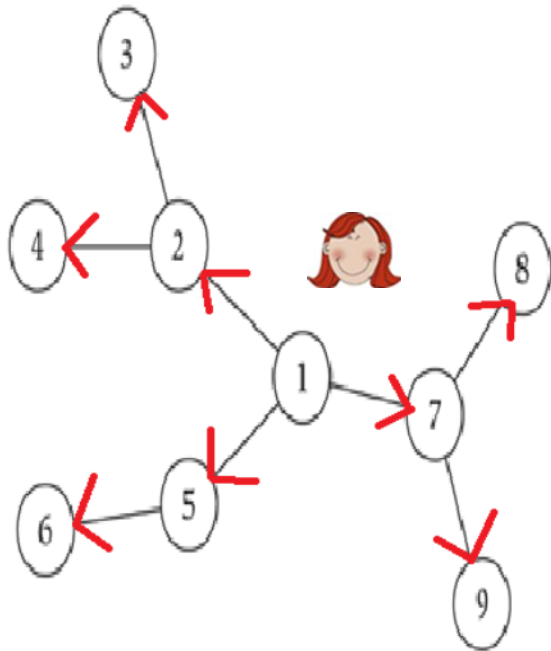
out-tree, or rooted arborescence



a tree

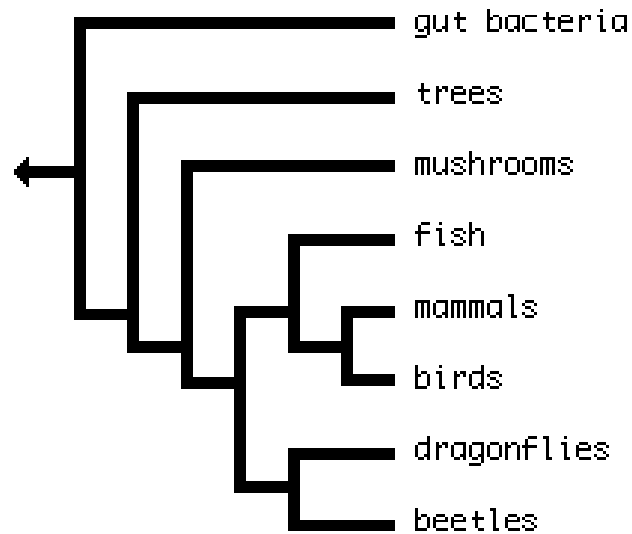


the same tree, rooted at 1



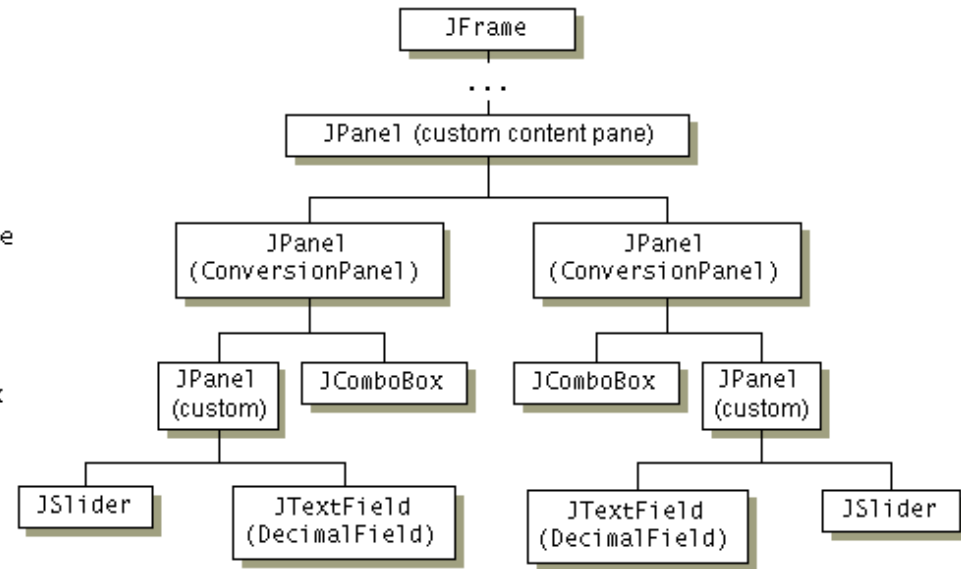
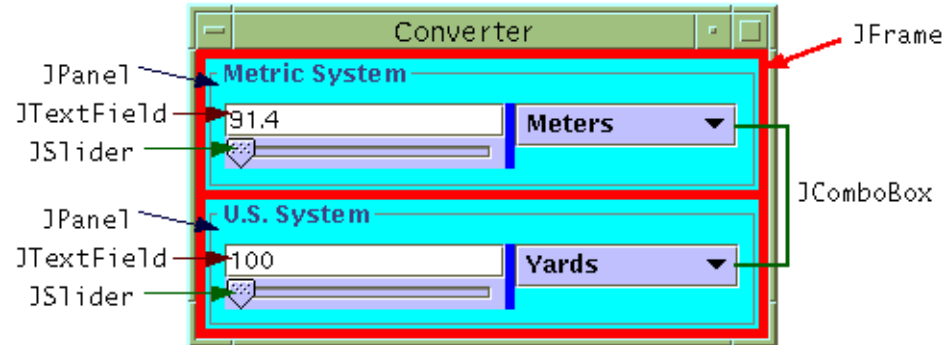
Phylogeny Trees

Phylogeny trees. Describe evolutionary history of species.



GUI Containment Hierarchy

GUI containment hierarchy. Describe organization of GUI widgets.



Reference: <http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html>

3.2 Graph Traversal

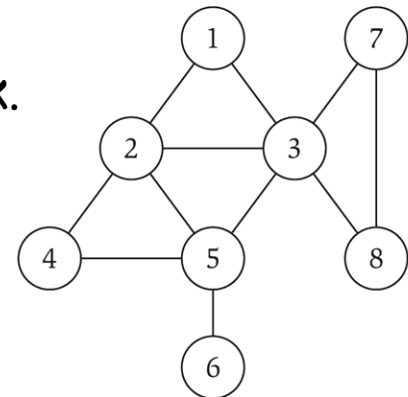
Connectivity

s-t connectivity problem. Given two nodes 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 ?

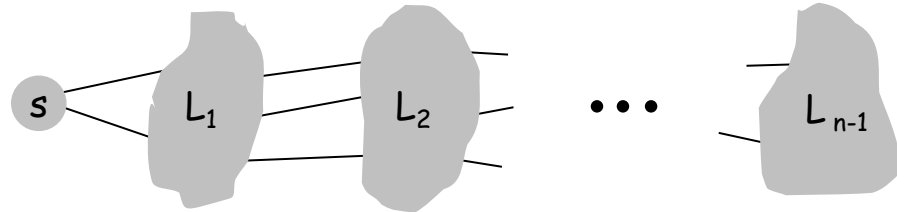
Applications.

- Friendster.
- Maze traversal.
- Kevin Bacon number.
- Erdos number.
- Fewest number of hops in a communication network.



Breadth First Search

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



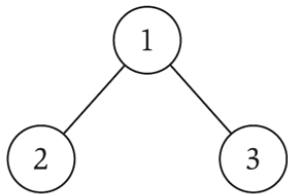
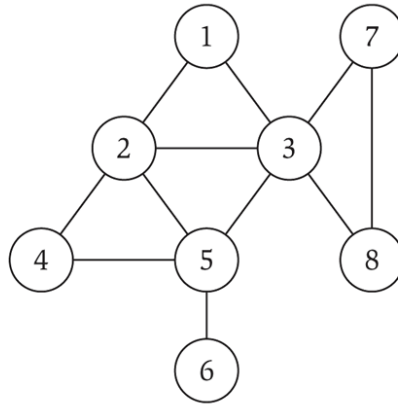
BFS algorithm.

- $L_0 = \{ s \}$.
- $L_1 =$ 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 .

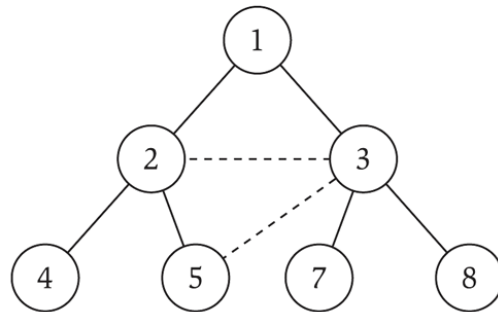
Theorem. For each i , L_i consists of all nodes at distance exactly i from s . There is a path from s to t iff t appears in some layer.

Breadth First Search

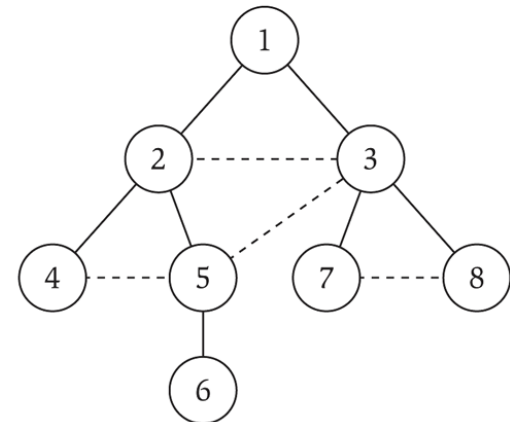
Property. Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G . Then the level of x and y differ by at most 1.



(a)



(b)



(c)

L_0

L_1

L_2

L_3

Breadth First Search: Analysis

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

Pf.

- Easy to prove $O(n^2)$ running time:
 - at most n lists $L[i]$
 - each node occurs on at most one list; for loop runs $\leq n$ times
 - when we consider node u , there are $\leq n$ incident edges (u, v) , and we spend $O(1)$ processing each edge
- Actually runs in $O(m + n)$ time:
 - when we consider node u , there are $\text{deg}(u)$ incident edges (u, v)
 - total time processing edges is $\sum_{u \in V} \text{deg}(u) = 2m$ ▪

↑
each edge (u, v) is counted exactly twice
in sum: once in $\text{deg}(u)$ and once in $\text{deg}(v)$

BFS starting at a vertex s using an array for the queue:

Data structures:

A queue $Q[0..(n-1)]$ of vertices, $qfront$, $qrear$.
 $parent[i]$ = BFS tree parent of node i .

The parent of the root s is s .

To initialize:

```
// Set parent of each node to be -1 to indicate  
// that the vertex has not yet been visited.
```

```
for (i=0; i < n; i++) parent[i]= -1;
```

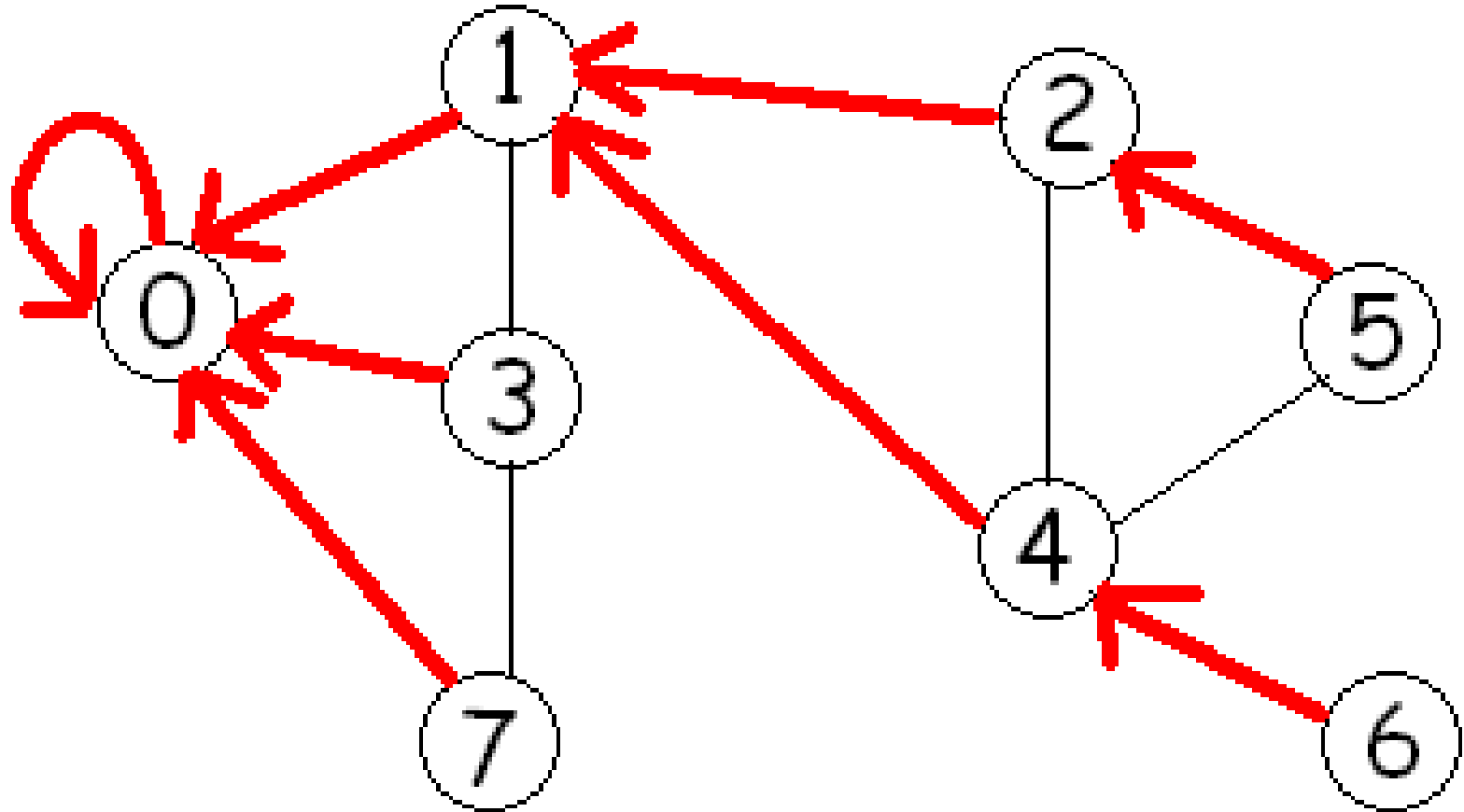
```
// Initialize the queue so that BFS starts at  $s$ 
```

```
qfront=0; qrear=1; Q[qfront]=  $s$ ;
```

```
parent[ $s$ ]= $s$ ;
```

```
while (qfront < qrear) // Q is not empty
    u= Q[qfront]; qfront++;
    for each neighbour v of u
        if (parent[v] == -1) // not visited
            parent[v]= u;
            Q[qrear]= v; qrear++;
        end if
    end for
end while
```

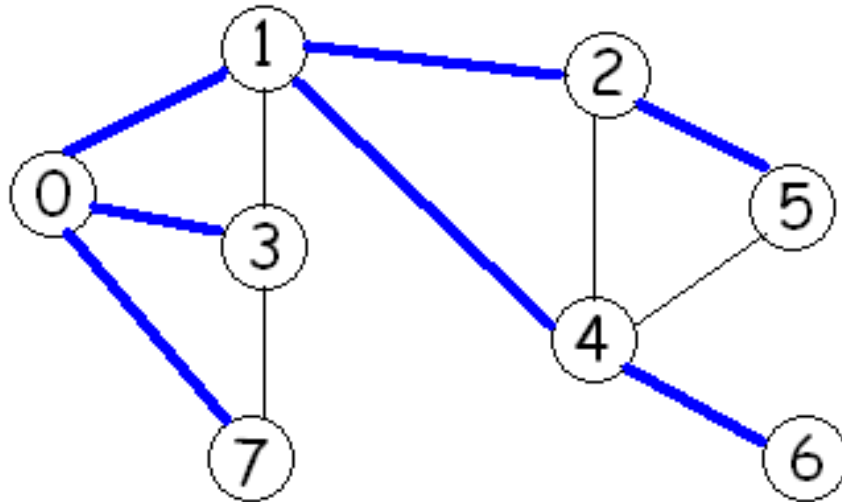
Red arcs represent parent information:



$BFI[v]$ = Breadth first index of v
= step at which v is visited.

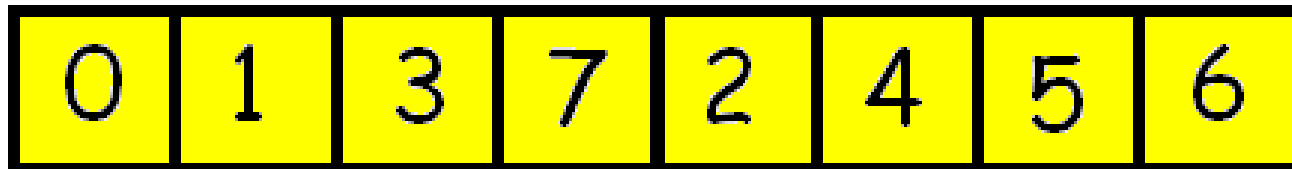
The $BFI[v]$ is equal to v 's position in the queue.

Renumbering with BFI before solving independent set can make some algorithms drastically faster. I suspect it would also help for dominating set.



0 1 2 3 4 5 6 7

Q



To initialize:

```
// Set parent of each node to be -1 to indicate  
// that the vertex has not yet been visited.  
for (i=0; i < n; i++) parent[i]= -1;
```

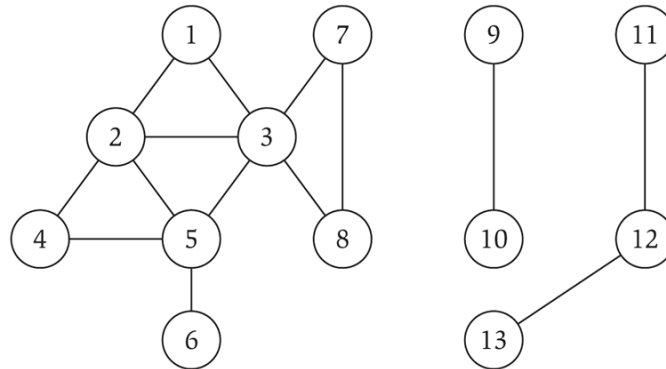
```
// Initialize the queue so that BFS starts at s  
qfront=0; qrear=1; Q[qfront]= s;  
parent[s]=s;
```

```
BFI[s]= 0;
```

```
while (qfront < qrear) // Q is not empty
    u= Q[qfront]; qfront++;
    for each neighbour v of u
        if (parent[v] == -1) // not visited
            parent[v]= u; BFI[v]= qrear;
            Q[qrear]= v; qrear++;
        end if
    end for
end while
```

Connected Component

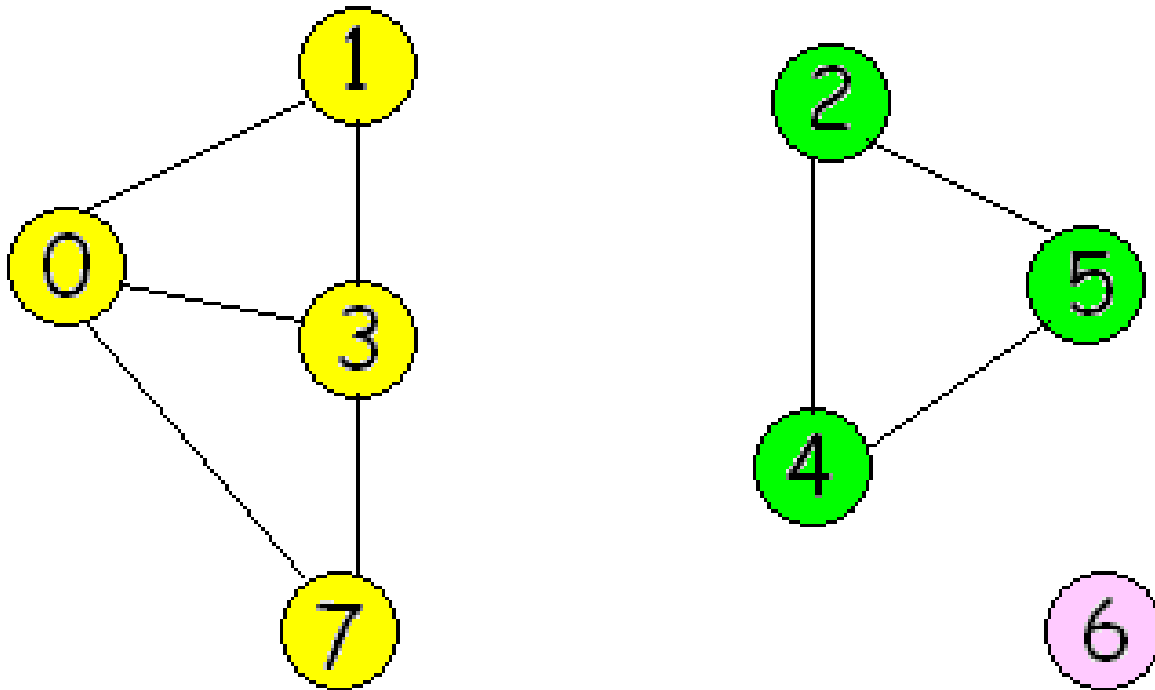
Connected component. Find all nodes reachable from s .



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

One application:

How many connected components does a graph have and which vertices are in each component?



To find the connected components:

```
for (i=0; i < n; i++)
```

```
    parent[i]= -1;
```

```
nComp= 0;
```

```
for (i=0; i < n; i++)
```

```
    if (parent[i] == -1)
```

```
        nComp++;
```

```
        BFS(i, parent, component, nComp);
```

```
BFS(s, parent, component, nComp)
```

```
// Do not initialize parent.
```

```
// Initialize the queue so that BFS starts at s
```

```
qfront=0; qrear=1; Q[qfront]= s;
```

```
parent[s]=s;
```

```
component[s]= nComp;
```

```
while (qfront < qrear) // Q is not empty
```

```
    u= Q[qfront]; qfront++;
```

```
    for each neighbour v of u
```

```
        if (parent[v] == -1) // not visited
```

```
            parent[v]= u; component[v]= nComp;
```

```
            Q[qrear]= v; qrear++;
```

```
        end if
```

```
    end for
```

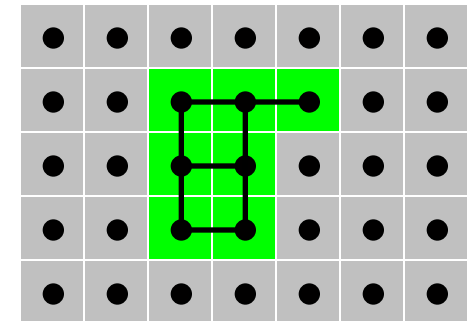
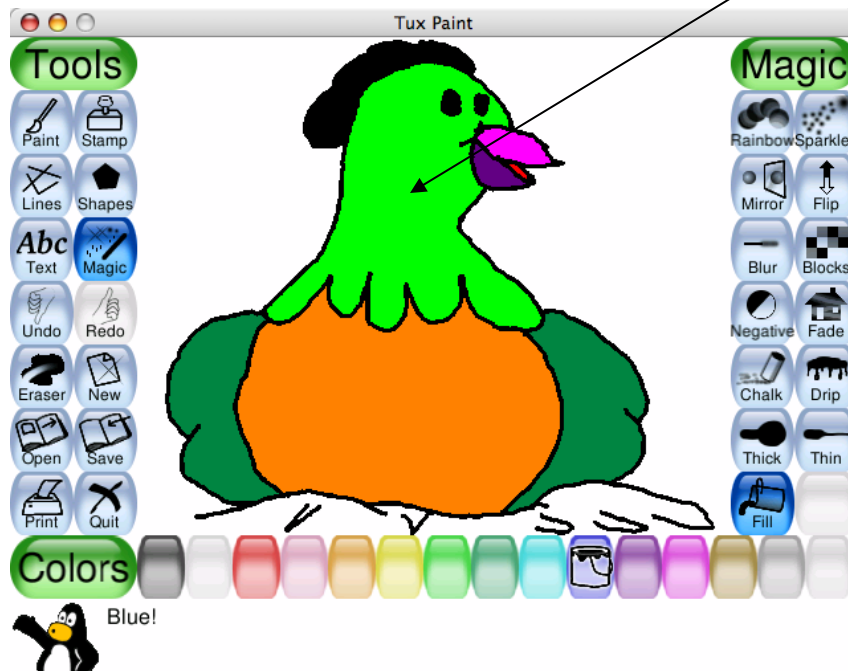
```
end while
```

Flood Fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Node: pixel.
- Edge: two neighboring lime pixels.
- Blob: connected component of lime pixels.

recolor lime green blob to blue

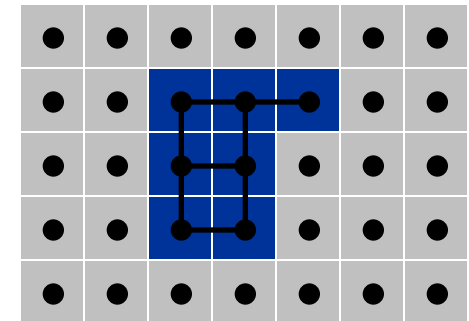


Flood Fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Node: pixel.
- Edge: two neighboring lime pixels.
- Blob: connected component of lime pixels.

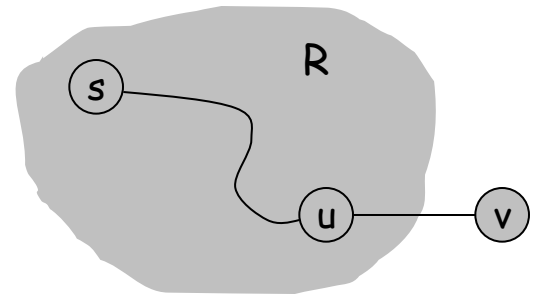
recolor lime green blob to blue



Connected Component

Connected component. Find all nodes reachable from s .

R will consist of nodes to which s has a path
Initially $R = \{s\}$
While there is an edge (u, v) where $u \in R$ and $v \notin R$
 Add v to R
Endwhile



it's safe to add v

Theorem. Upon termination, R is the connected component containing s .

- BFS = explore in order of distance from s .
- DFS = explore in a different way.

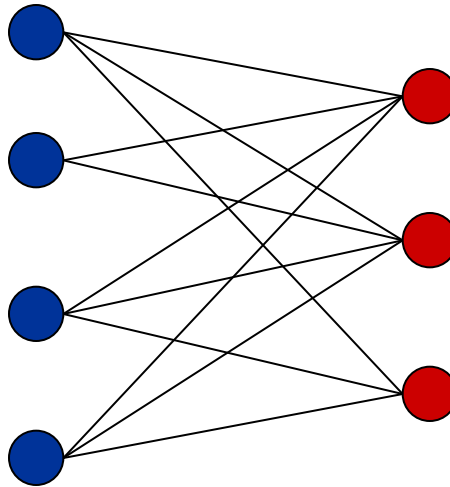
3.4 Testing Bipartiteness

Bipartite Graphs

Def. An undirected graph $G = (V, E)$ is **bipartite** if the nodes can be colored red or blue such that every edge has one red and one blue end.

Applications.

- Stable marriage: men = red, women = blue.
- Scheduling: machines = red, jobs = blue.

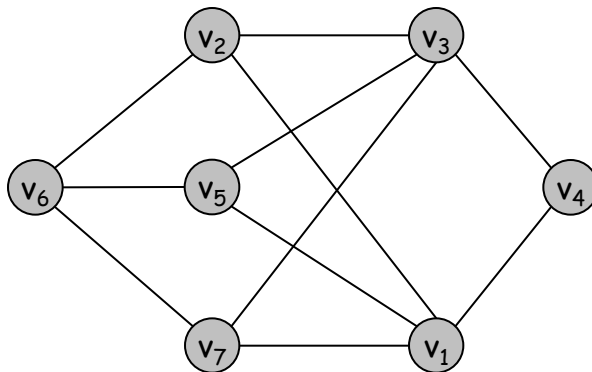


a bipartite graph

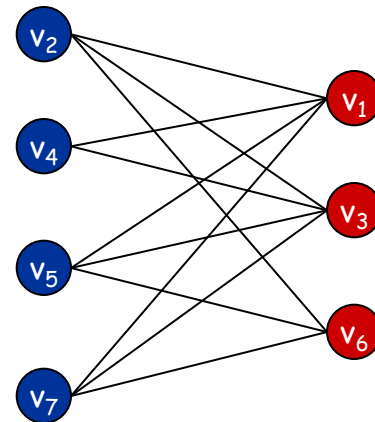
Testing Bipartiteness

Testing bipartiteness. Given a graph G , is it bipartite?

- Many graph problems become:
 - easier if the underlying graph is bipartite (matching)
 - tractable if the underlying graph is bipartite (independent set)
- Before attempting to design an algorithm, we need to understand structure of bipartite graphs.



a bipartite graph G

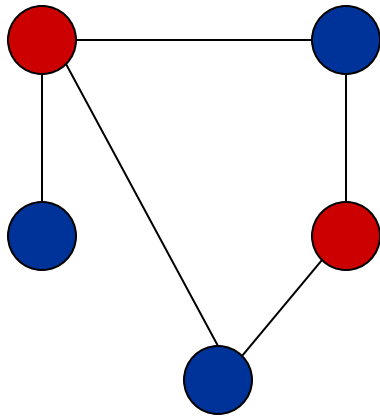


another drawing of G

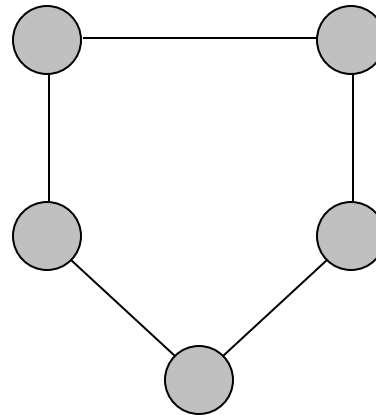
An Obstruction to Bipartiteness

Lemma. If a graph G is bipartite, it cannot contain an odd length cycle.

Pf. Not possible to 2-color the odd cycle, let alone G .



bipartite
(2-colorable)

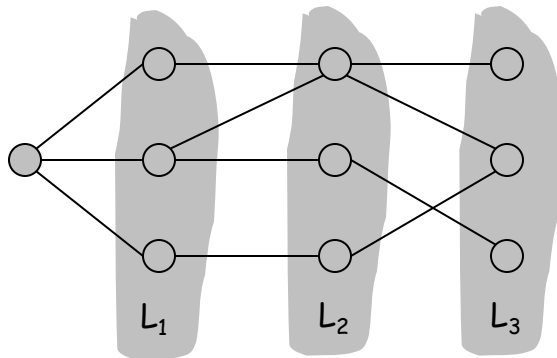


not bipartite
(not 2-colorable)

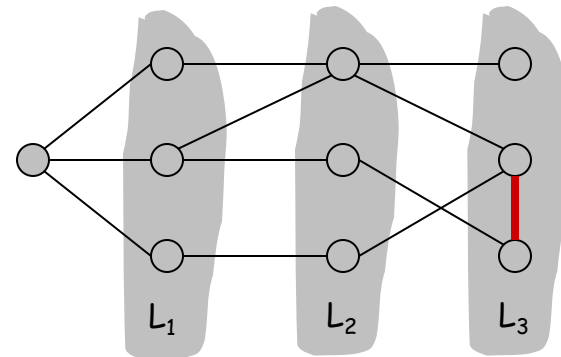
Bipartite Graphs

Lemma. Let G be a connected graph, and let L_0, \dots, L_k be the layers produced by BFS starting at node s . Exactly one of the following holds.

- (i) No edge of G joins two nodes of the same layer, and G is bipartite.
- (ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).



Case (i)



Case (ii)

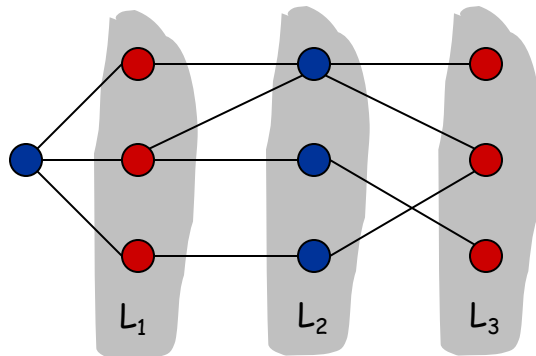
Bipartite Graphs

Lemma. Let G be a connected graph, and let L_0, \dots, L_k be the layers produced by BFS starting at node s . Exactly one of the following holds.

- (i) No edge of G joins two nodes of the same layer, and G is bipartite.
- (ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).

Pf. (i)

- Suppose no edge joins two nodes in adjacent layers.
- By previous lemma, this implies all edges join nodes on same level.
- Bipartition: red = nodes on odd levels, blue = nodes on even levels.



Case (i)

Bipartite Graphs

Lemma. Let G be a connected graph, and let L_0, \dots, L_k be the layers produced by BFS starting at node s . Exactly one of the following holds.

- (i) No edge of G joins two nodes of the same layer, and G is bipartite.
- (ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).

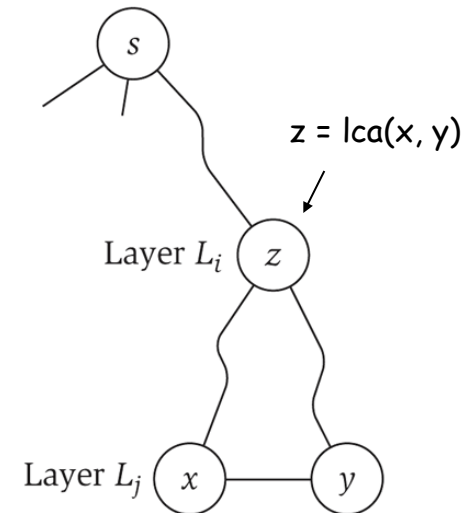
Pf. (ii)

- Suppose (x, y) is an edge with x, y in same level L_j .
- Let $z = \text{lca}(x, y) =$ lowest common ancestor.
- Let L_i be level containing z .
- Consider cycle that takes edge from x to y , then path from y to z , then path from z to x .
- Its length is $1 + \underbrace{(j-i)}_{\text{path from } y \text{ to } z} + \underbrace{(j-i)}_{\text{path from } z \text{ to } x}$, which is odd. ▪

$\underbrace{\hspace{1.5cm}}$
 (x, y)

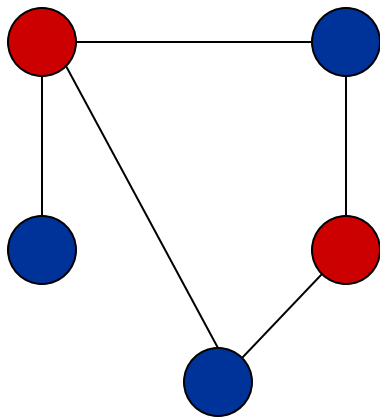
$\underbrace{\hspace{1.5cm}}$
 path from y to z

$\underbrace{\hspace{1.5cm}}$
 path from z to x

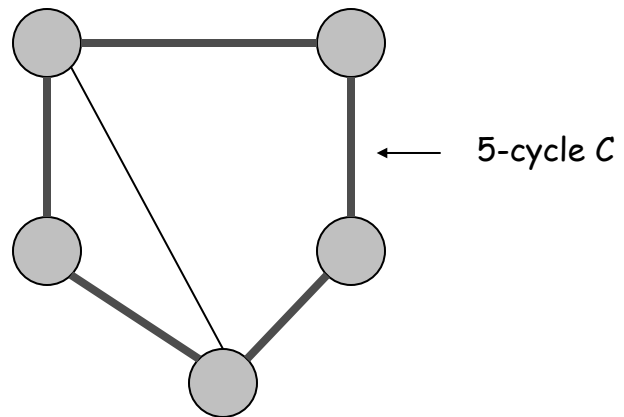


Obstruction to Bipartiteness

Corollary. A graph G is bipartite iff it contains no odd length cycle.



bipartite
(2-colorable)



not bipartite
(not 2-colorable)