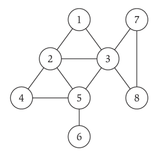
Objectives

- Data structures: Graphs
- Graph Traversal

Undirected Graphs G = (V, E)

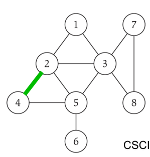
- V = nodes (vertices)
- E = edges between pairs of nodes
- Captures pairwise relationship between objects
- Graph size parameters: n = |V|, m = |E|



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

Graph Representation: Adjacency Matrix

- n×n matrix with A_{uv} = 1 if (u, v) is an edge
  - Two representations of each edge (symmetric matrix)
  - Space: $\Theta(n^2)$
  - Checking if (u, v) is an edge: $\Theta(1)$ time
  - Identifying all edges: $\Theta(n^2)$ time

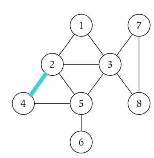


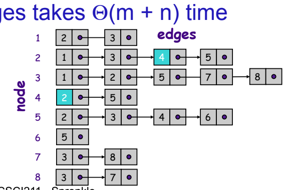
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Graph Representation: Adjacency List

- Node indexed array of lists
  - Two representations of each edge
  - Space = 2m + n = O(m + n)
  - Checking if (u, v) is an edge takes O(deg(u)) time
  - Identifying all edges takes Θ(m + n) time

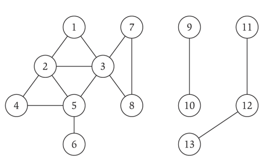
degree = number of neighbors of u





Paths and Connectivity

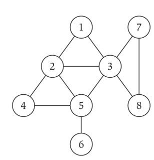
- Def. A ***path*** in an undirected graph G = (V, E) is a sequence P of nodes $v_1, v_2, \ldots, v_{k-1}, v_k$
  - each consecutive pair v_i, v_{i+1} is joined by an edge in E
- Def. A path is ***simple*** if all nodes are *distinct*
- Def. An undirected graph is ***connected*** if ∀ pair of nodes u and v, there is a path between u and v



- *Short path*
- *Distance*

Cycles

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

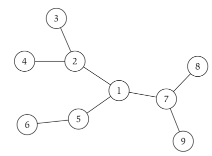


cycle C = 1-2-4-5-3-1

TREES

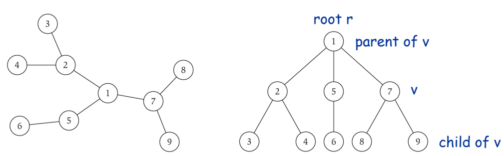
Trees

- Def. An undirected graph is a ***tree*** if it is connected and does not contain a cycle
- Simplest connected graph
  - Deleting any edge from a tree will disconnect it



Rooted Trees

- Given a tree T, choose a root node *r* and orient each edge away from *r*
  - Has n-1 edges
- Models hierarchical structure



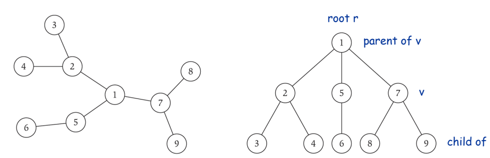
a tree

the same tree, rooted at 1

Why *n-1* edges?

Rooted Trees

- Why *n-1* edges?
  - Each node except for root has an edge to its parent

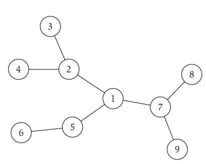


a tree

the same tree, rooted at 1

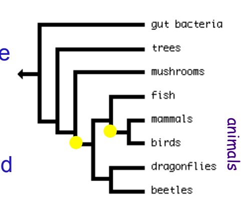
Trees

- 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



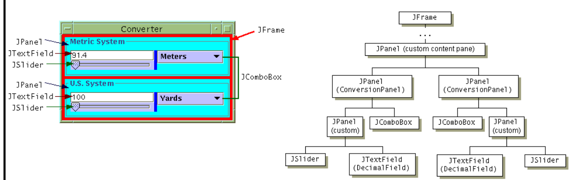
Phylogeny Trees

- Describe evolutionary history of species
  - mammals and birds share a common ancestor that they do not share with other species
  - all animals are descended from an ancestor not shared with mushrooms, trees, and bacteria



GUI Containment Hierarchy

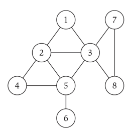
- Describe organization of GUI widgets



GRAPH CONNECTIVITY & TRAVERSAL

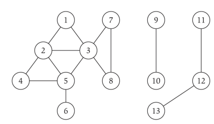
Connectivity

- **s-t connectivity problem.** Given nodes *s* and *t*, is there a path between *s* and *t*?
- **s-t shortest path problem.** Given nodes *s* and *t*, what is the length of the shortest path between *s* and *t*?
- Applications
  - Facebook
  - Maze traversal
  - Kevin Bacon number
  - Fewest number of hops in a communication network



Application: Connected Component

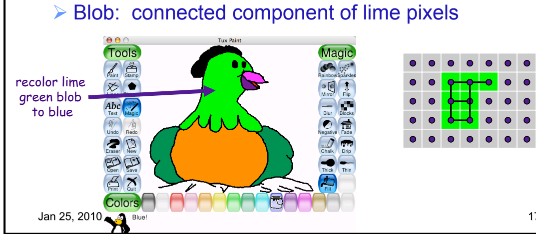
- Find all nodes ***reachable*** from *s*



- Connected component containing node 1 is { 1, 2, 3, 4, 5, 6, 7, 8 }

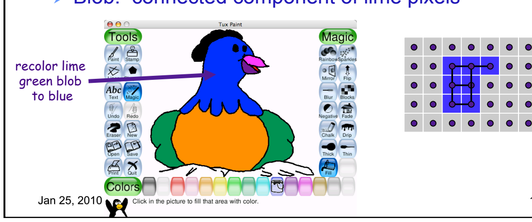
Application: Flood Fill

- 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



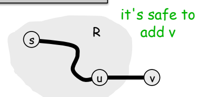
Application: Flood Fill

- 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



A General Algorithm

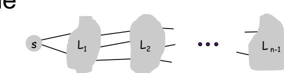
```
R will consist of nodes to which s has a path
R = {s}
While there is an edge (u,v) where u∈R and v∉R
      add v to R
```



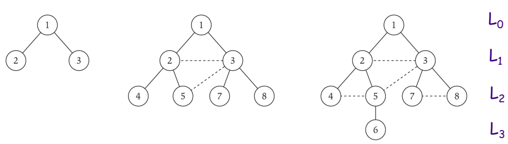
- *R* will be the **connected component** containing s
- Algorithm is underspecified
  - In what order should we consider the edges?

Breadth-First Search

- **Intuition**. Explore outward from *s* in all possible directions (edges), adding nodes one "layer" at a time
- **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

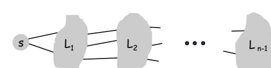


Breadth-First Search



Breadth-First Search

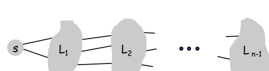
- **Theorem.** For each *i*, L_i consists of all nodes at distance exactly *i* from *s*. *There is a path from s to t iff t appears in some layer.*



- *What does this mean?*
- *Can we determine the distance between s and t?*

Breadth-First Search

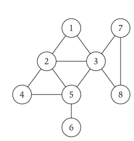
- **Theorem.** For each *i*, L_i consists of all nodes at distance exactly *i* from *s*. There is a path from *s* to *t* iff *t* appears in some layer.
  - Shortest path to *t* from *s*, is the *i* from L_i
  - All nodes ***reachable*** from *s* are in L_1, L_2, …, L_{n-1}



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.

G:



If x is in L_i, then y must be in L_{i+1} or earlier

Connected Component

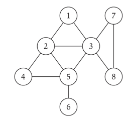
- Find all nodes ***reachable*** from *s*

In general….

```
R will consist of nodes to which s has a path
R = {s}
While there is an edge (u,v) where u∈R and v∉R
      add v to R
```

- Theorem. Upon termination, R is the connected component containing s
  - BFS = explore in order of distance from s
  - DFS = explore until hit "deadend"

Depth-First Search

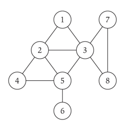


- Need to keep track of where you've been
- When reach a "dead-end" (already explored all neighbors), backtrack to node with unexplored neighbor
- Algorithm:

```
DFS(u):
   Mark u as "Explored" and add u to R
   For each edge (u, v) incident to u
      If v is not marked "Explored" then
         DFS(v)
```

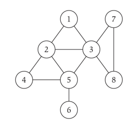

Depth-First Search

- How does DFS work on this graph?
  - Starting from node 1



DFS vs BFS

- Compare the resulting trees

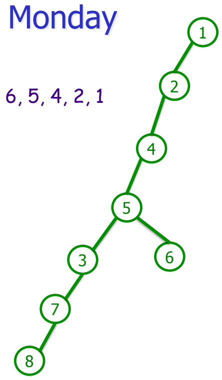


DFS Analysis

- Let T be a depth-first search tree, let *x* and *y* by nodes in T, and let (*x*, *y*) be an edge of G that is not an edge of T. Then one of *x* or *y* is an ancestor of the other.

Where We Were on Monday

Explored: 1, 2, 4, 5, 3, 7, 8, 6
Now: 1, 2, 4, 5, 3, 7, 8, 7, 3, 5, 6, 5, 4, 2, 1
R: 1, 2, 4, 5, 7, 8, 6



Analysis of Connected Components

- For any two nodes *s* and *t* in a graph, their connected components are either identical or disjoint
- Proof?