CS 361
Data Structures & Algs
Lecture 12

Prof. Tom Hayes
University of New Mexico
09-30-2010

Last Time

Terminology for graphs:

- vertex, edge, path, cycle, connected, component, tree, forest, empty graph
- Also: equivalence relation, equivalence class, rooting trees
- Adjacency lists vs adjacency matrix
- # of edges, # of components

returned Quiz #2

Today

Spanning Trees

BFS

DFS

Testing Bipartiteness

How many edges?

Suppose G is a graph, with N vertices. What is the fewest edges G can have?

Zero.

Suppose G is a connected graph with N vertices. What is the fewest edges G can have?

N-1. Proof? Induction: Start with empty graph. Then there are N connected components. Each edge we add can reduce the number of components by 0 or by 1. So it takes at least N-1 edges to make G connected.

Testing Connectivity

Input: A graph G, and vertices s,t.

Output: A path from s to t, if one exists, and otherwise output “Disconnected”

How do we proceed?

- Start at s, and “search outward”

Testing Connectivity

Input: A graph G, and vertices s,t.

Output: A path from s to t, if one exists, and

otherwise output “Disconnected”

How do we proceed?

Start at s, and “search outward”

Build up a tree, rooted at s, as we go.

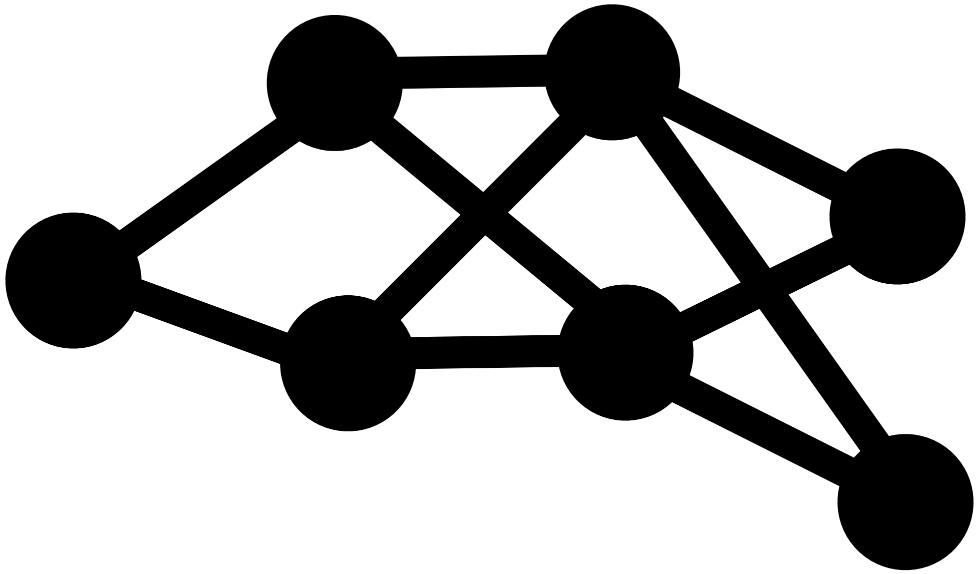
Eventually, we will find all nodes in the component of s. If t is there, the path from t to s is t, parent(t), parent(parent(t)), ..., s

Spanning Trees

Given any connected graph G=(V,E), there exists a subset E' of E, such that T = (V,E') is a tree. Such a tree is called a spanning tree of G.

> "spanning" because it "spans" across all the nodes of G, not just a subset.

Ex:

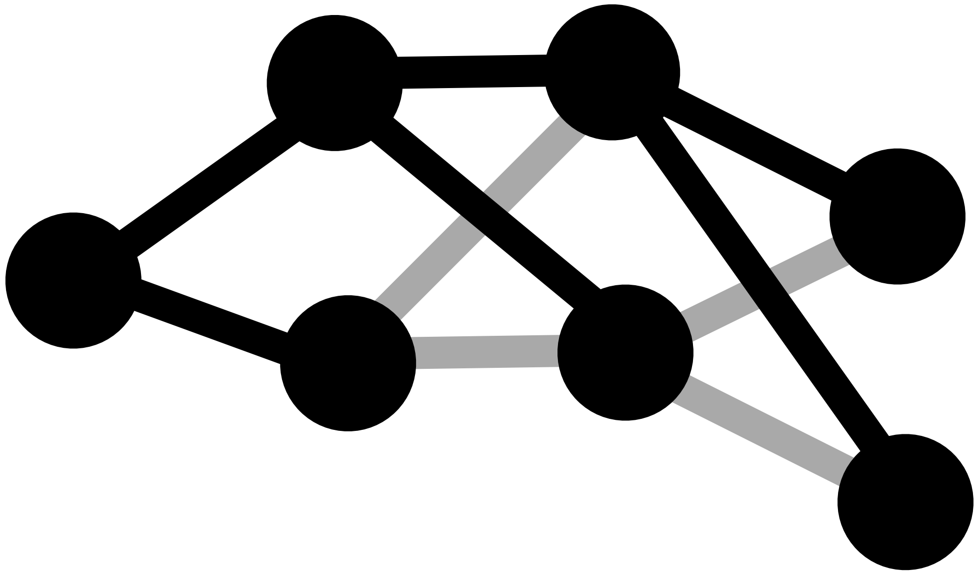


Spanning Trees

Given any connected graph G=(V,E), there exists a subset E' of E, such that T = (V,E') is a tree. Such a tree is called a spanning tree of G.

"spanning" because it "spans" all the nodes of G, not just a subset.

Ex:

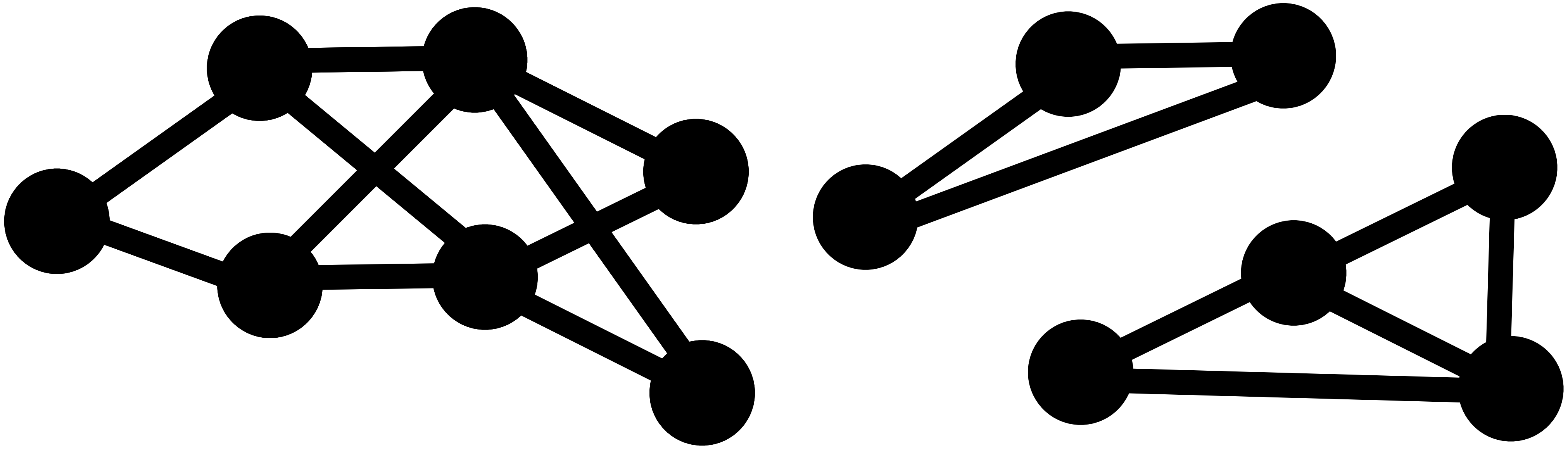


Spanning Forests

More generally, for any graph, G=(V,E), there exists a subset E’ of E, such that F = (V,E’) is a forest. Such a tree is called a spanning forest of G.

“spanning” because it “spans” all the nodes of G, not just a subset.

Ex:

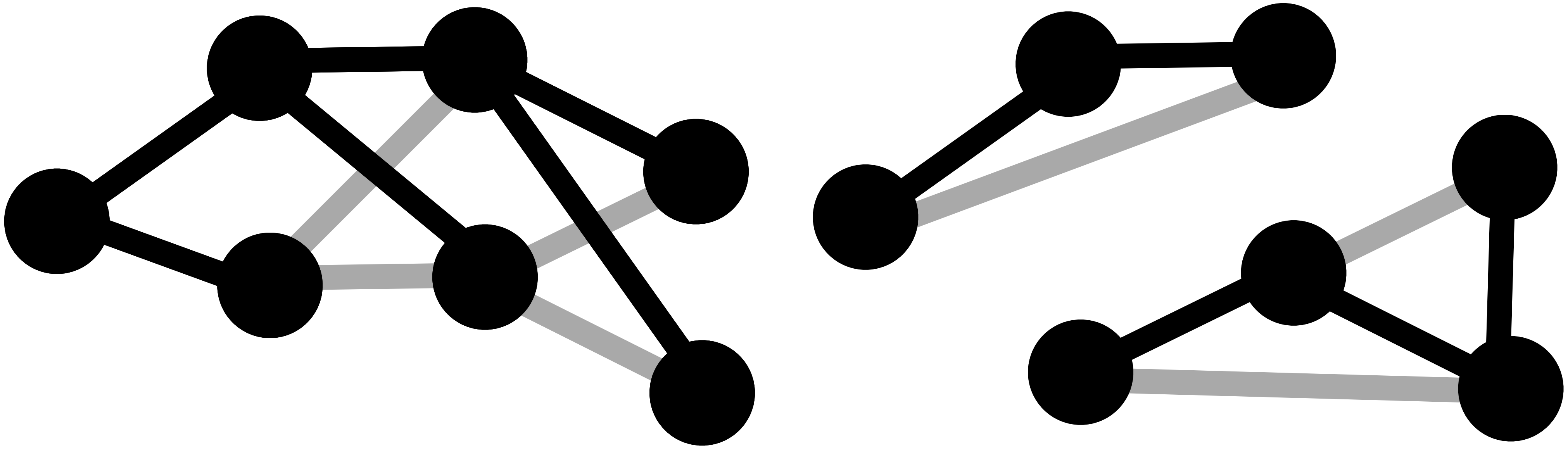


Spanning Forests

More generally, for any graph, G=(V,E), there exists a subset E' of E, such that F = (V,E') is a forest. Such a tree is called a spanning forest of G.

"spanning" because it "spans" all the nodes of G, not just a subset.

Ex:



Finding a Component

INPUT: Graph G, vertex s:

OUTPUT: The set of vertices reachable from s   (i.e. the connected component of G containing s)

Idea: Start with s. While there is an active node v, add its neighbors to the set. Then v becomes inactive. (Adding a node already in the set has no effect.)

Finding All Components

INPUT: Graph G.

Find: All components of G.

Idea: Initially, no vertices are found. All are active. While there is a found active vertex, add its neighbors to the current component, then make it inactive.

If there are no found active vertices, use any unfound vertex to start a new component.

Search Trees

Each time we “find” a new vertex, it is because it is a neighbor of a particular previously found vertex (or we are starting on a new component).

By saving this as the “parent” of the newly found vertex, we build up a rooted forest (tree if the graph is connected).

This is a spanning tree. We often call it a “search tree” because of the way it arose

Breadth-First Search

BFS finds the vertices in the connected component of the start vertex s one “level” at a time.

Level L_i = {vertices whose “distance” to s equals i}

distance(v,w) = number of edges in the shortest path from v to w.

Question: how to implement?

Edges in BFS

Theorem: Each time BFS looks at an edge, it is either:

(a) joining a found node (level L_i) to an unfound node (level L_{i+1}). Becomes an edge in the tree, or

(b) joining a found node (level L_i) to an already found node (level L_i or L_{i+1}).

Why only these?

Rephrased:

Theorem: If T is a BFS tree for a graph G, then every edge in G either:

(a) is in T, and joins adjacent levels, or

(b) is not in T, and joins nodes in the same or adjacent levels.

Level: equal distance from the root.

Depth-First Search

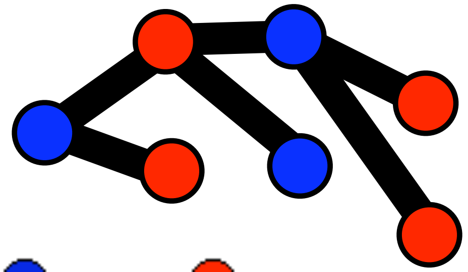
DFS: explores fully from each vertex, before backing up to try another one.

```
DFS(s): Mark s as found.
For each unfound neighbor v of s:
    add edge (v,s) to T.
    DFS(v)
```

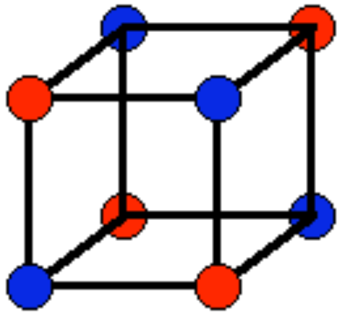
Bipartite Graphs

In a bipartite graph, the nodes can be colored RED and BLUE so that every edge joins 1 red and 1 blue node.

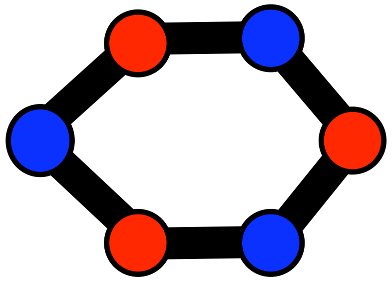
Examples: Every tree is bipartite.



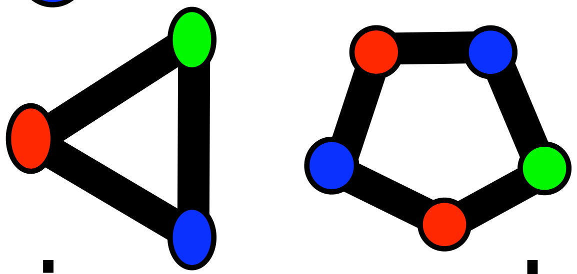
The boolean cube is bipartite.



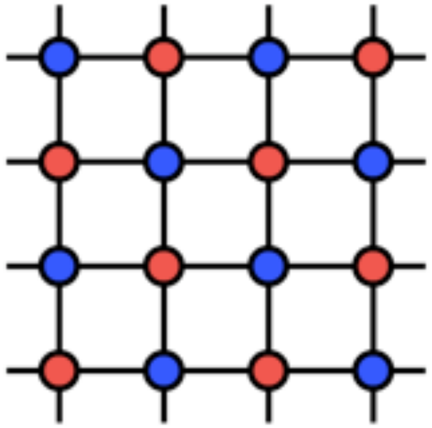
Even cycles.



NOT: odd cycles



any graph containing an odd cycle

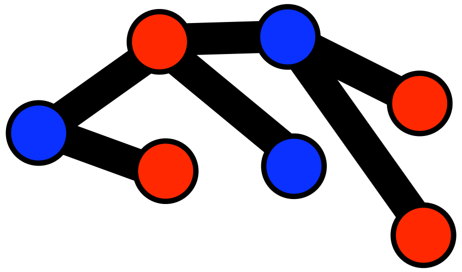


Checking Bipartiteness

Input: A connected graph (for simplicity)

Output: True if bipartite. False if not.

Algorithm:



(1) Build a search tree (spanning).
BFS or DFS are both ok for this.

(2) Root: blue. Make each new node the opposite color from its parent.

(3) For "back edges" joining 2 old nodes, check that both are opposite colors.

Why does it work?

(1) Obviously, if it returns true, then there is a 2-coloring.

(2) Suppose it returns false. Then it found an edge between 2 same-colored nodes {v,w} in the graph. This means there is a loop of odd length in the graph:

v to root, root to w, then edge {v,w}.

A bipartite graph cannot have such a loop. After any odd number of steps, must be at a different color than start.