CMPSCI 575/MATH 513

Combinatorics and Graph Theory

Lecture #2: Graph Theory Overview
(Tucker Section 1.1)
David Mix Barrington
9 September 2016

Graph Theory Overview

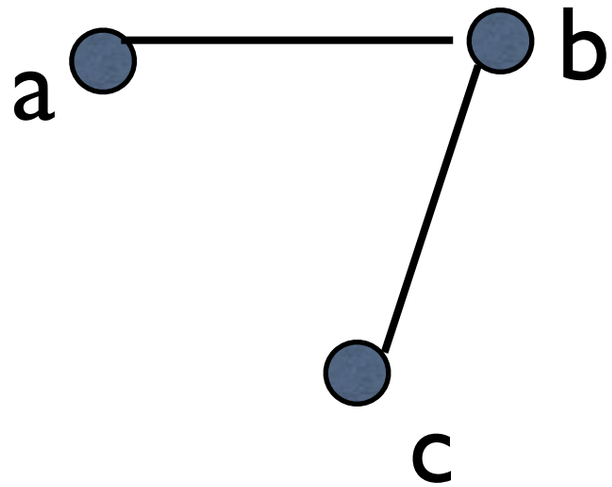
- Graph Vocabulary
- Representing Graphs
- Paths, Circuits, Connectivity
- Examples: Matching and Searching
- Examples: Modeling Networks
- Examples: Independent Sets, Vertex Bases
- Coming Attractions: Graph Algorithms

Graph Vocabulary

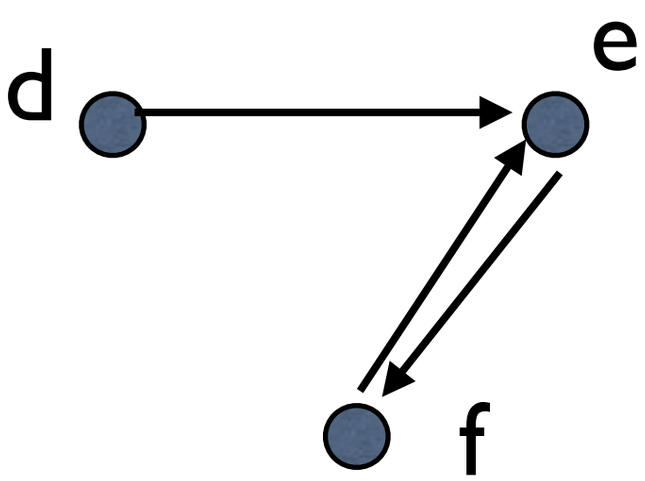
- A **graph** is a set of **vertices** V and a set of **edges** E, where each edge is an unordered pair of vertices. We draw a graph with a dot for each vertex and a line for each edge.
- A **directed graph** is a set of vertices V and a set of **directed edges** E. A directed edge is an ordered pair of vertices. We draw the directed edge (u, v) as an arrow from dot u to dot v.

Graph Pictures

- Graph with vertex set {a, b, c} and edges {a, b} and {b, c}



- Directed graph with vertex set {d, e, f} and edges (d, e), (f, e), and (e, f)



Representing Graphs

- Two graphs are considered **identical** if they have the same vertex set and same edge set. So there are many different ways to draw the same graph.

- To store a graph in a computer, we can use an **adjacency matrix** or an **adjacency list.**

- The matrix is an n by n boolean array, with a 1 in entry (i, j) if and only if there is an edge from vertex i to vertex j. In an ordinary graph, the edge {i, j} makes entries (i, j) and (j, i).

Representing Graphs

- In an adjacency list, we have an array of list data structures, one list for each vertex. The list for vertex i contains an entry for each node j such that there is an edge from i to j.
- Graphs in applied situations are often **sparse**, meaning that the average node has small **degree** (few nodes **adjacent** to it). In a sparse graph, the list representation uses far less space. In COMPSCI 311, for example, we usually analyze algorithms that view graphs as lists.

Paths, Circuits, Connectivity

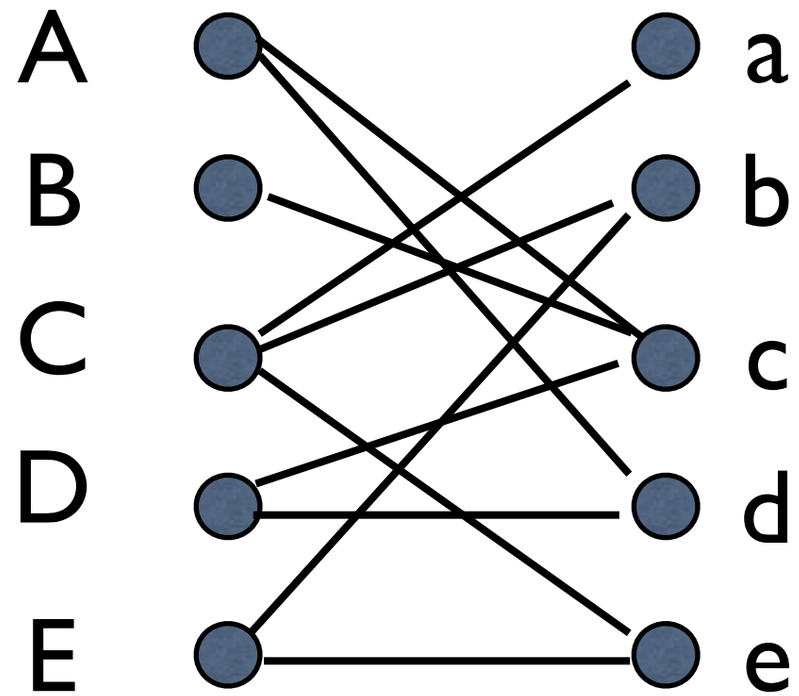
- A **path** in a graph is a sequence of edges where the endpoint of each edge is the start point of the next.
- A **circuit** is a path that starts and ends at the same vertex. The 0-edge path is not a circuit.
- The **length** of a path is the number of edges it contains.
- A graph is **connected** if there is a path from each vertex to each other vertex.

Example: Matching

- Suppose we have n people and n jobs, and each person is qualified for some jobs but not others. We'd like to assign each person to a job for which they are qualified.
- We can make a graph with a vertex for each person, a vertex for each job, and an edge (p, j) if and only if person p is qualified for job j.
- This graph is **bipartite** because no edge goes from person to person or job to job.

Example: Matching

- To assign jobs to people, we need a matching in this graph, which is a set of five edges that don't share any endpoints.
- There is no such matching because A, B and D can only do jobs c and d.

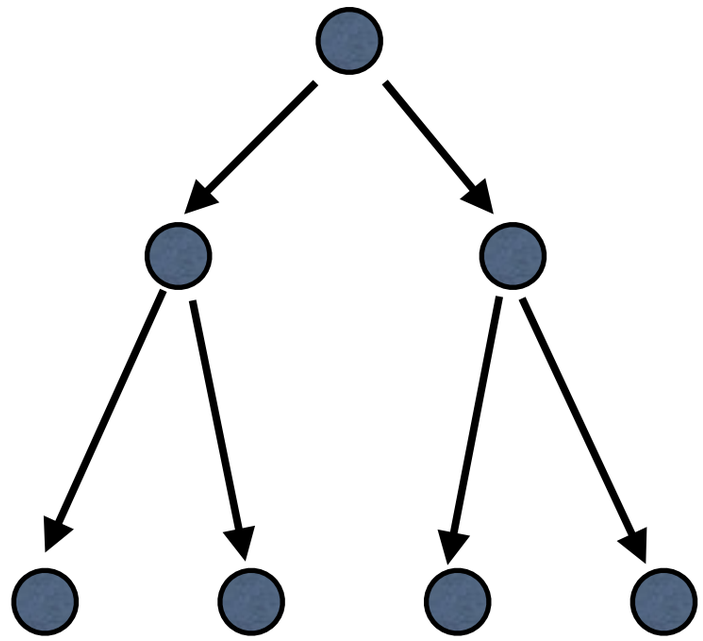


Example: Searching

- To find an element in an ordered list, we often use **binary search**.
- We test our goal element against the middle element of the list, then recursively search either the first half or the second half, depending on the result of the test.
- We search a list of size n in at most about log n tests. (In computer science all logs use base two.)

Example: Searching

- This directed graph is called a **binary tree**.
- Each node represents a test, and there is a directed edge from each test to one of the tests that could follow it.
- The **leaf** nodes are where there's no further searching to be done.

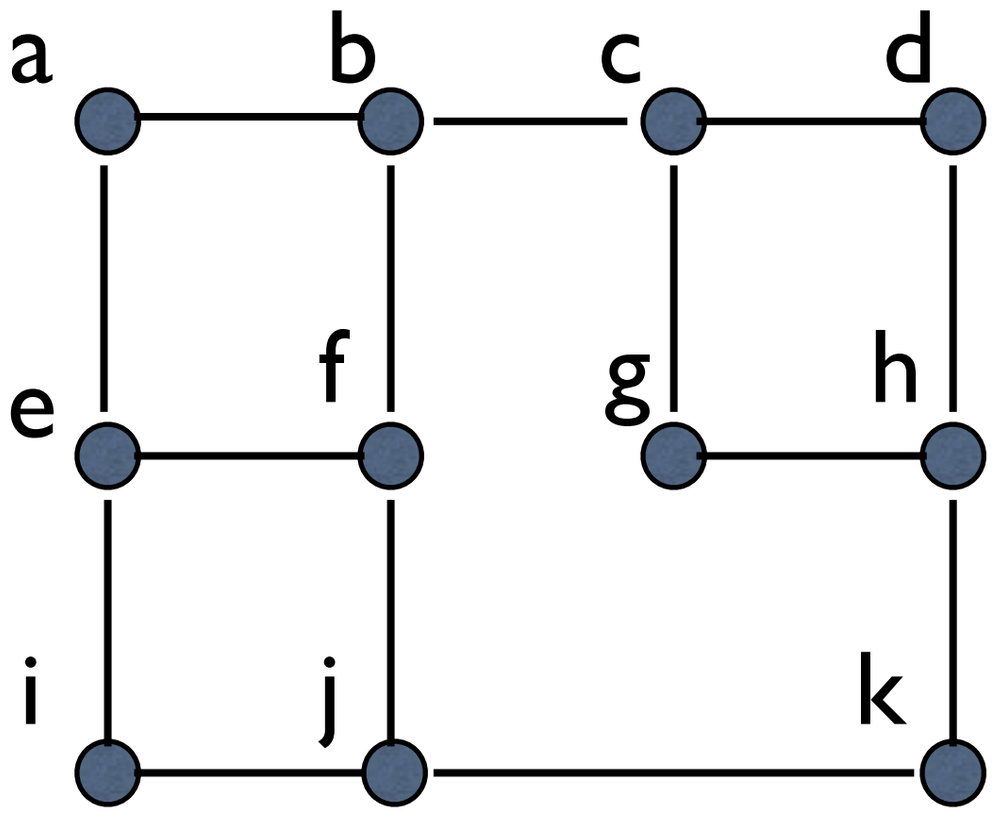


Example: Network Reliability

- Communication networks have nodes and links, which we can model as vertices and edges in an undirected graph.
- We want our network to represent a connected graph, so that we can get a message from any node to any other node.
- Could failure of a single node break its connectivity? What about two nodes? One edge? Two edges?

A Network Graph

- This network is safe against the loss of one node or one edge, but not two nodes or two edges.
- There's a subset of 10 of the 14 edges that keeps the network connected.

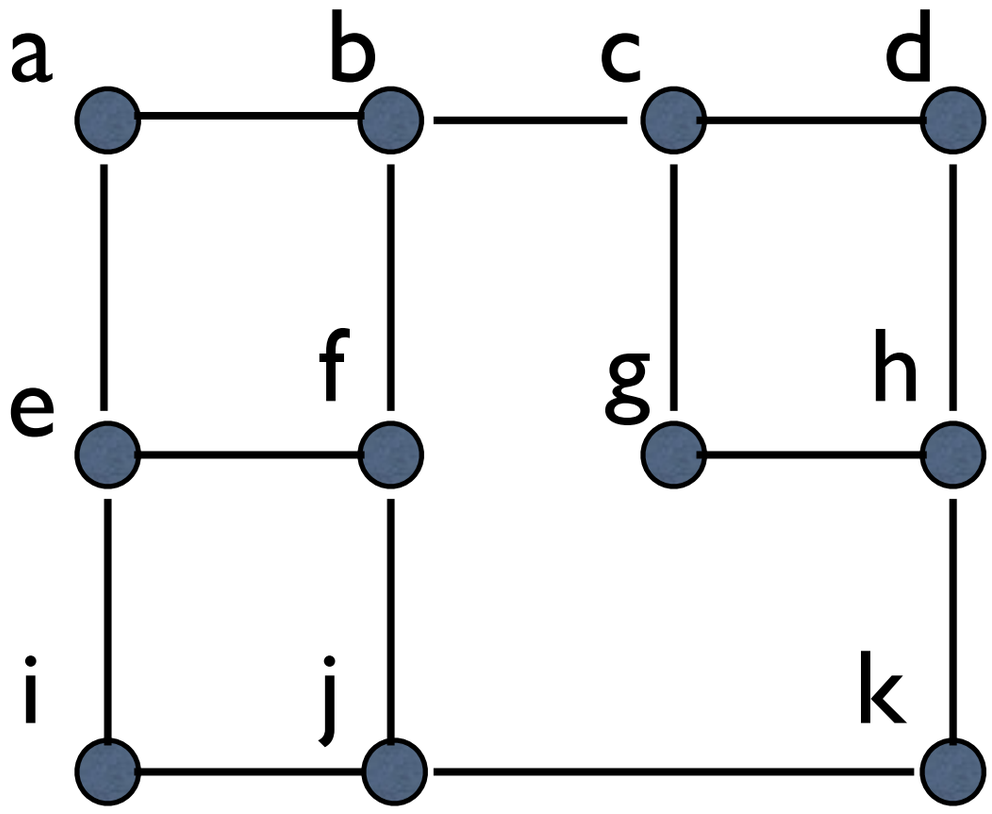


Example: Street Surveillance

- Now suppose the vertices in our (undirected) graph represent street corners and the edges represent streets.
- We'd like to station police at certain street corners so that every street has someone at one or both of its ends.
- This is usually called a **vertex cover** of the graph, though Tucker calls it an **edge cover**. Given a graph, we can ask for a vertex cover of minimum size.

Finding a Vertex Cover

- The first cover I see is {b, c, e, h, j}, of size 5. ({b, e, h, j} gets 12 of the 14 edges.)
- We can't do better because no vertex can cover more than three edges and 4*3 < 14.

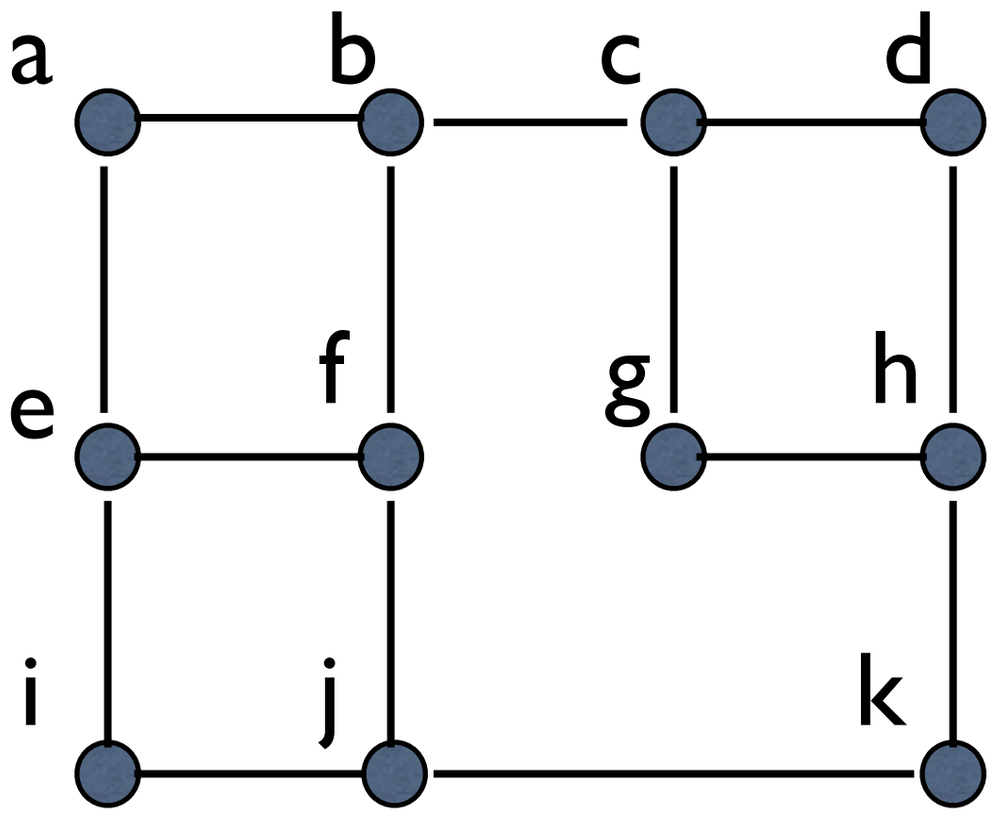


Example: Independent Sets

- Now suppose we have a state legislature with lots of committees, with overlapping membership. We want to have as many committees as possible meet at the same time.
- If nodes are committees and edges represent conflicts, we're looking for a set of nodes such that no edge connects two nodes in the set. This is called an **independent set**, and we're looking for one of maximum size.

The Same Graph Again

- An independent set of size 6 exists, {a, d, f, g, i, k}.
- Could we have size 7? No, because the nodes have degree 2 and 3, and there aren't enough degree-2's to fit 14 edges into 7 nodes.

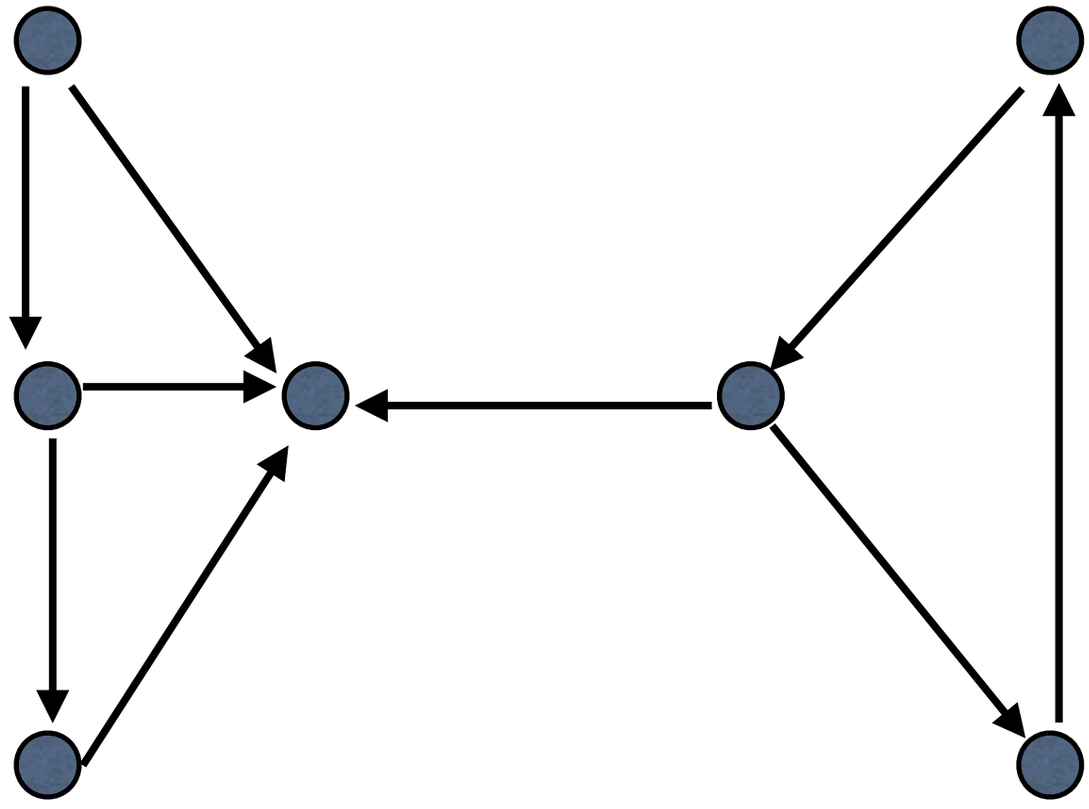


Example: Modeling Influence

- Now suppose I have a set of people and I know all the pairs (i, j) where person i can influence person j. I can model this with a directed graph, with an edge for each such pair.
- If I want to spread an idea through this whole set of people, what is the minimal number of people I need to start with?
- A **vertex basis** is a set B such that every node in the graph has a path to it starting from a node in B.

Vertex Basis Example

- My vertex basis has to include the top left vertex, which is a source.
- Any of the three vertices in the right half will do to complete a vertex basis of size 2.



More Graph Problems

- In the rest of our graph theory section we'll look at planarity, Euler and Hamilton circuits, and coloring, graph properties you may have seen before.
- Then we'll talk about a number of graph algorithms, for searching, finding minimum spanning trees, and analyzing network flow.
- We'll see examples of both polynomial-time and NP-complete problems.