

# CS61B Lecture #34

## Administrivia:

- Project due Tuesday night.
- Autograder running with preliminary test sets.

**Today's Readings:** Graph Structures: *DSIJ*, Chapter 12

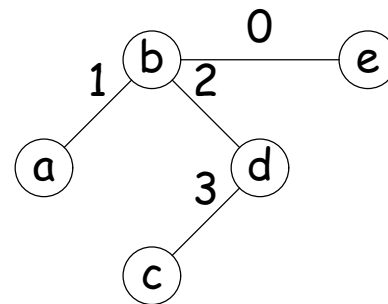
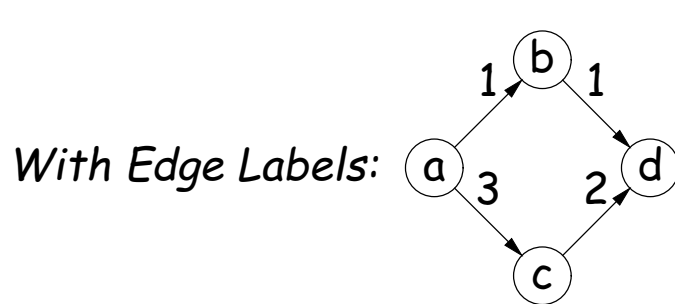
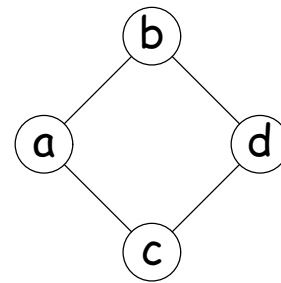
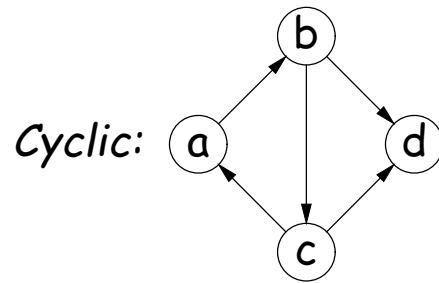
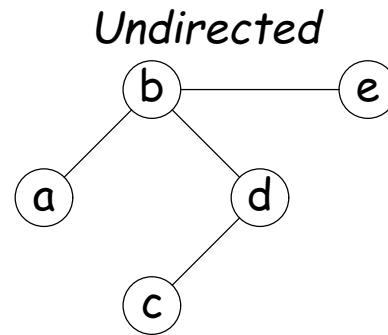
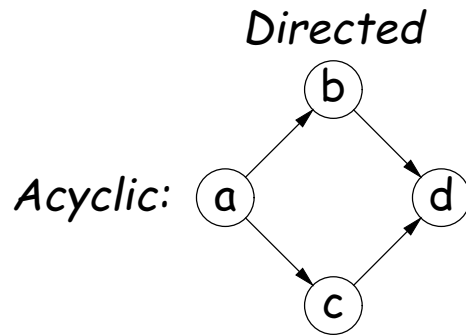
# Why Graphs?

- For expressing non-hierarchically related items
- Examples:
  - Networks: pipelines, roads, assignment problems
  - Representing processes: flow charts, Markov models
  - Representing partial orderings: PERT charts, makefiles

# Some Terminology

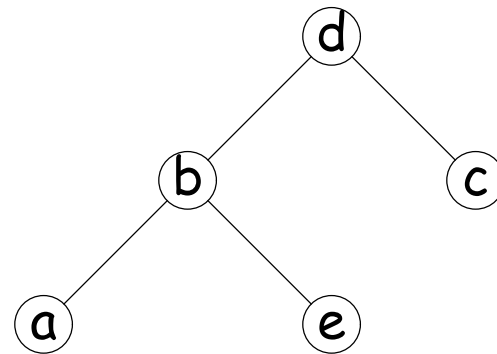
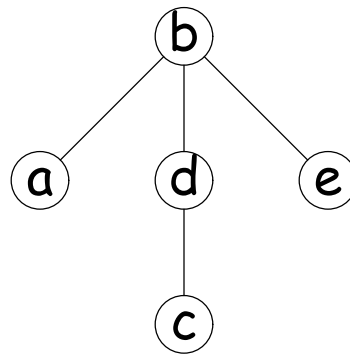
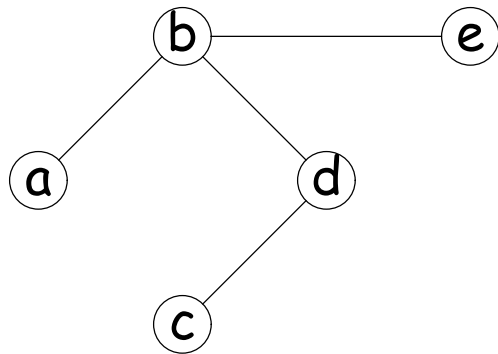
- A *graph* consists of
  - A set of *nodes* (aka *vertices*)
  - A set of *edges*: pairs of nodes.
  - Nodes with an edge between are *adjacent*.
  - Depending on problem, nodes or edges may have *labels* (or *weights*)
- Typically call node set  $V = \{v_0, \dots\}$ , and edge set  $E$ .
- If the edges have an order (first, second), they are *directed edges*, and we have a *directed graph* (*digraph*), otherwise an *undirected graph*.
- Edges are *incident* to their nodes.
- Directed edges *exit* one node and *enter* the next.
- A *cycle* is a path without repeated edges leading from a node back to itself (following arrows if directed).
- A graph is *cyclic* if it has a cycle, else *acyclic*. Abbreviation: Directed Acyclic Graph—*DAG*.

# Some Pictures



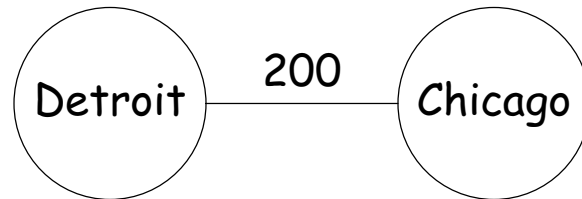
# Trees are Graphs

- A graph is *connected* if there is a (possibly directed) path between every pair of nodes.
- That is, if one node of the pair is *reachable* from the other.
- A DAG is a (rooted) tree iff connected, and every node but the root has exactly one parent.
- A connected, acyclic, undirected graph is also called a *free tree*. Free: we're free to pick the root; e.g.,

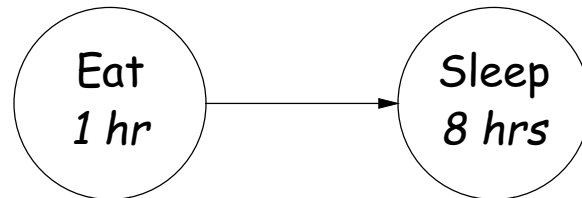


# Examples of Use

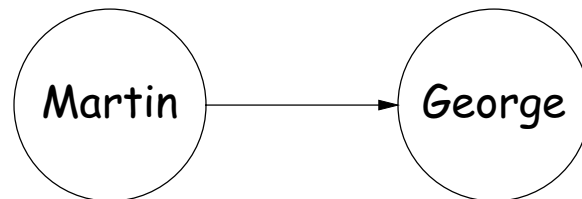
- Edge = Connecting road, with length.



- Edge = Must be completed before; Node label = time to complete.

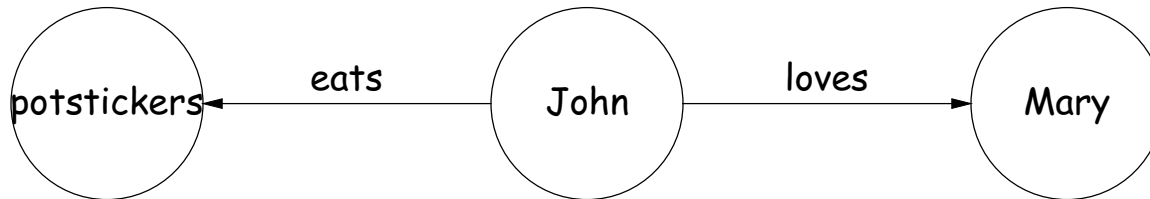


- Edge = Begat

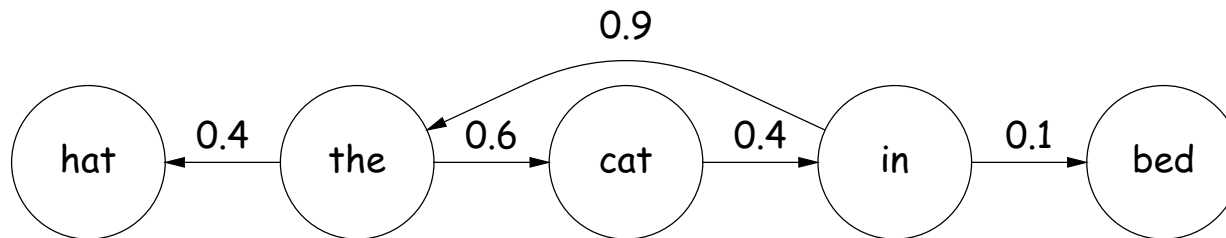


# More Examples

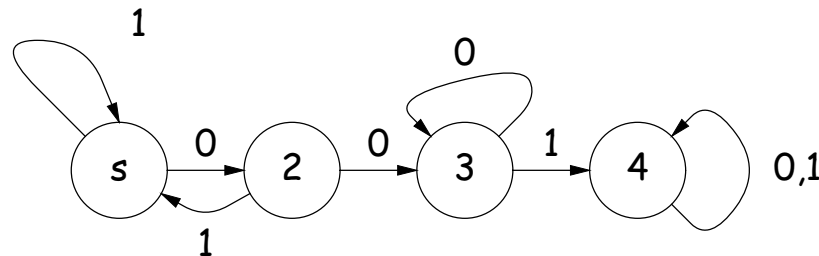
- Edge = some relationship



- Edge = next state might be (with probability)

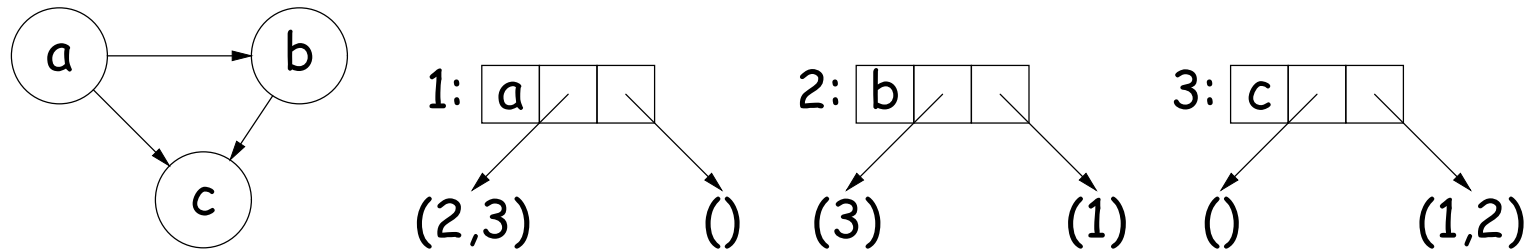


- Edge = next state in state machine, label is triggering input. (Start at s. Being in state 4 means "there is a substring '001' somewhere in the input".)



# Representation

- Often useful to number the nodes, and use the numbers in edges.
- *Edge list representation*: each node contains some kind of list (e.g., linked list or array) of its successors (and possibly predecessors).



- *Edge sets*: Collection of all edges. For graph above:

$$\{(1, 2), (1, 3), (2, 3)\}$$

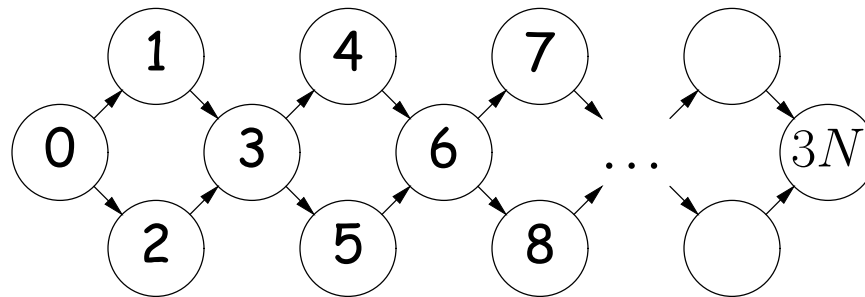
- *Adjacency matrix*: Represent connection with matrix entry:

$$\begin{array}{c}
 1 \quad 2 \quad 3 \\
 1 \quad \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\
 2 \\
 3
 \end{array}$$



# Traversing a Graph

- Many algorithms on graphs depend on traversing all or some nodes.
- Can't quite use recursion because of cycles.
- Even in acyclic graphs, can get combinatorial explosions:



Treat 0 as the root and do recursive traversal down the two edges out of each node:  $\Theta(2^N)$  operations!

- So typically try to visit each node constant # of times (e.g., once).