Graph Introduction

CSE 373

Data Structures

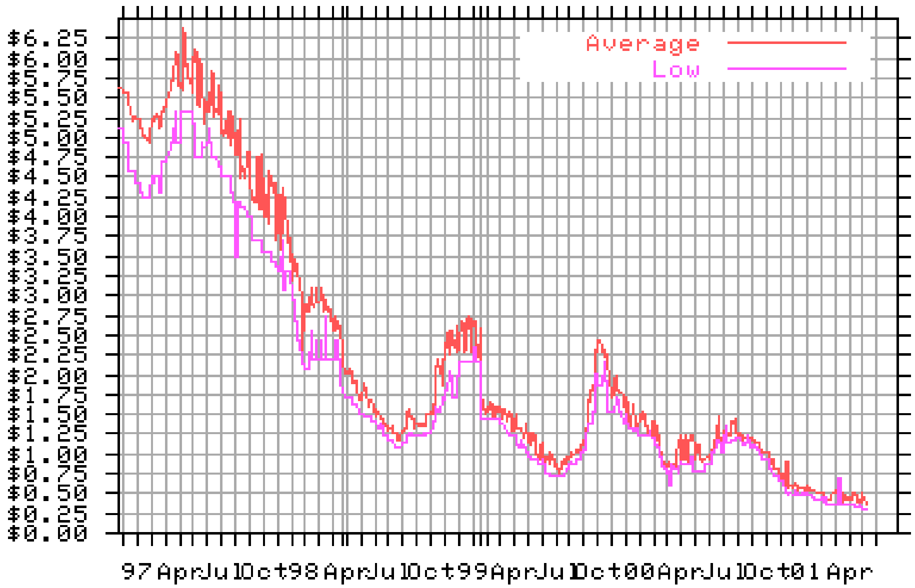
Lecture 18

Reading

- Reading
  - Section 9.1

What are graphs?

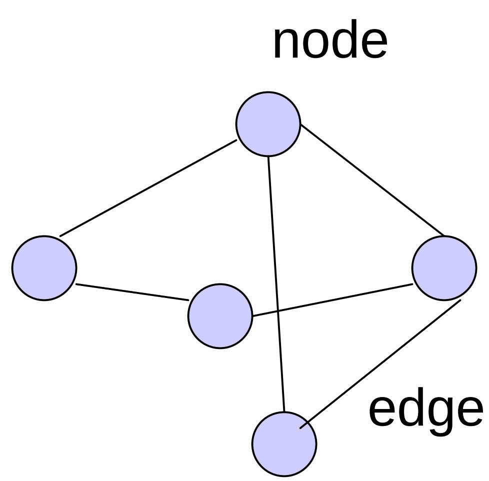
- Yes, this is a graph….



- But we are interested in a different kind of "graph"

Graphs

- Graphs are composed of
  - Nodes (vertices)
  - Edges

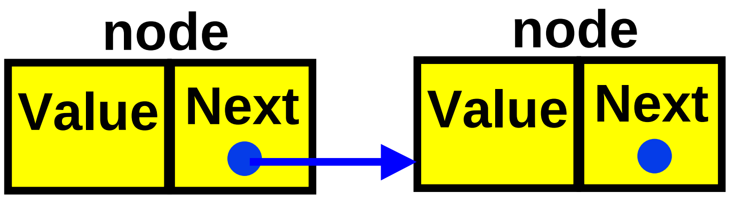


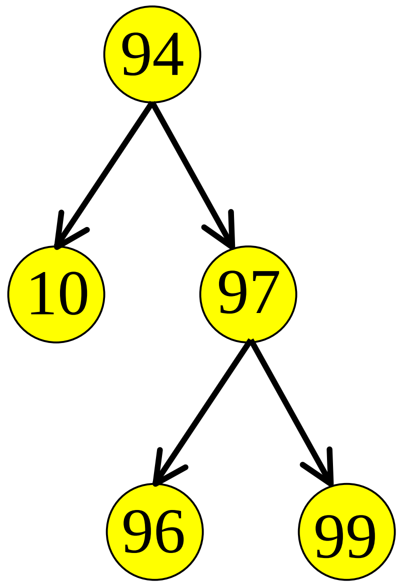
Varieties

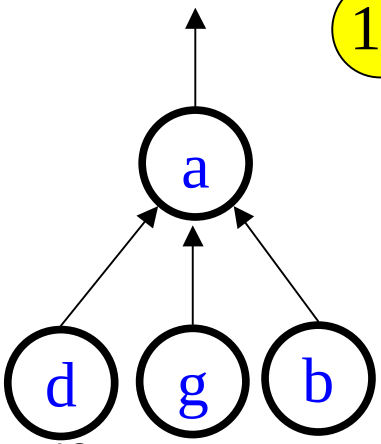
- Nodes
  - Labeled or unlabeled
- Edges
  - Directed or undirected
  - Labeled or unlabeled

Motivation for Graphs

- Consider the data structures we have looked at so far…
- Linked list: nodes with 1 incoming edge + 1 outgoing edge
- Binary trees/heaps: nodes with 1 incoming edge + 2 outgoing edges
- Binomial trees/B-trees: nodes with 1 incoming edge + multiple outgoing edges
- Up-trees: nodes with multiple incoming edges + 1 outgoing edge



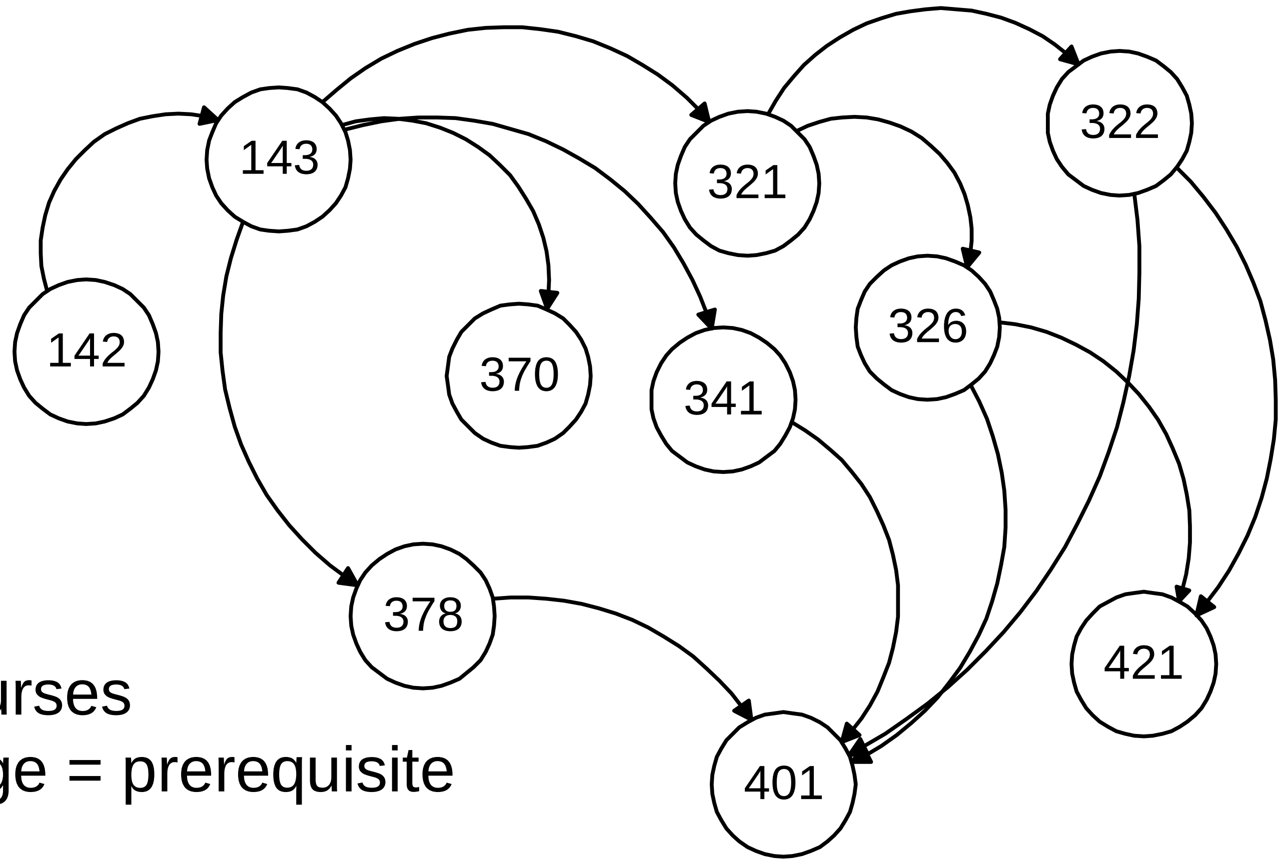




Motivation for Graphs

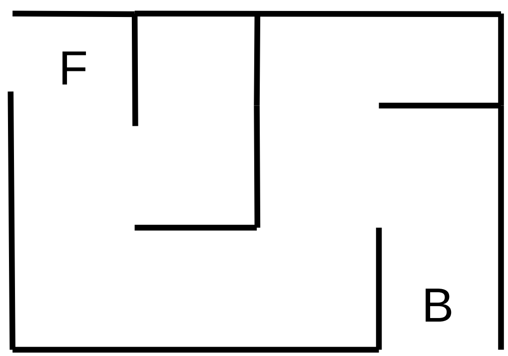
- What is common among these data structures?
- How can you generalize them?
- Consider data structures for representing the following problems…

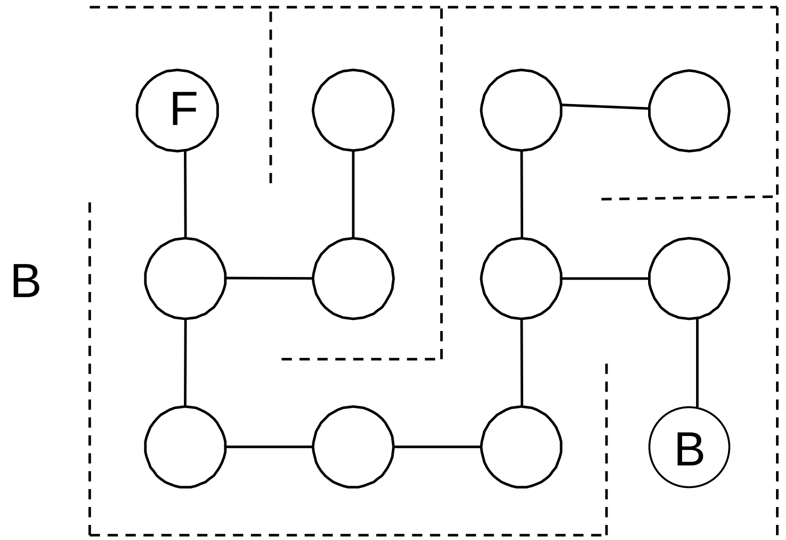
CSE Course Prerequisites at UW



Nodes = courses
Directed edge = prerequisite

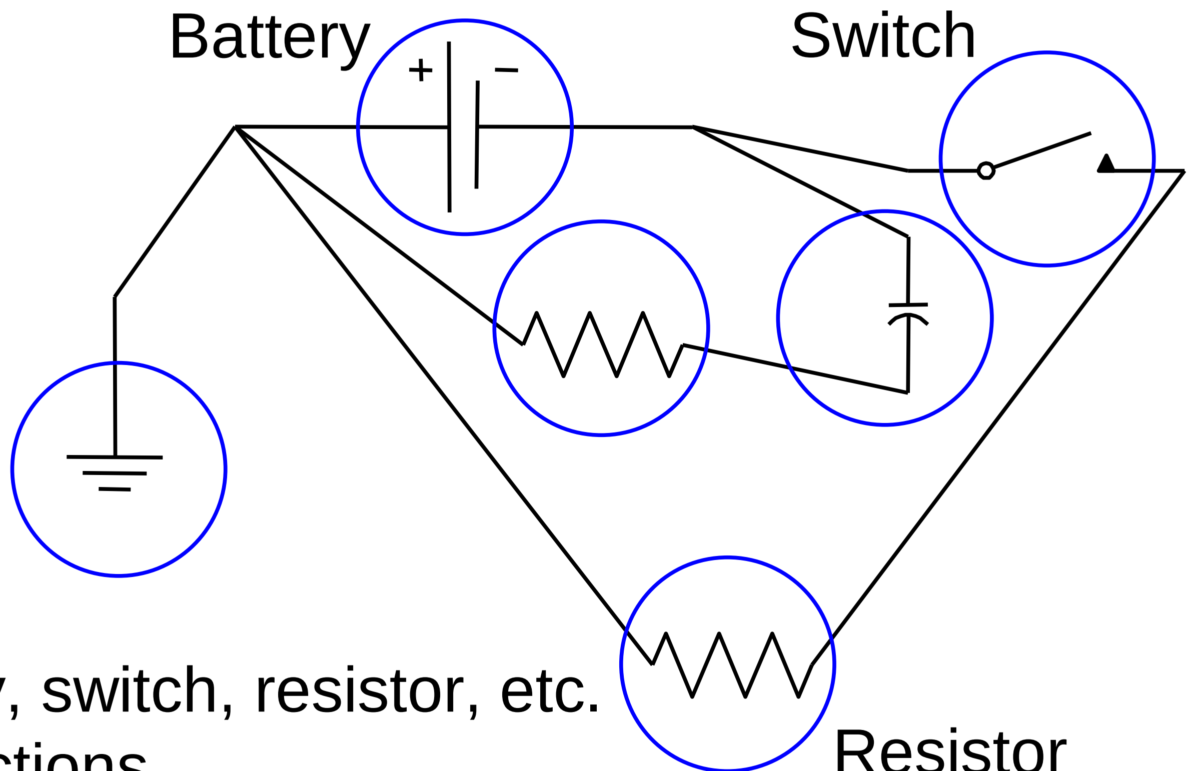
Representing a Maze





Nodes = rooms
Edge = door or passage

Representing Electrical Circuits

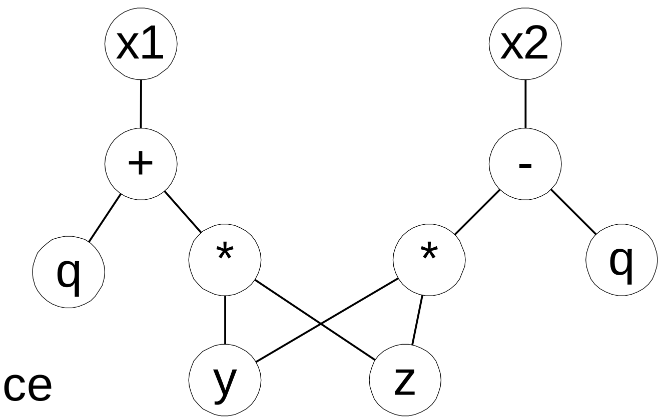


Nodes = battery, switch, resistor, etc.
Edges = connections

Program statements

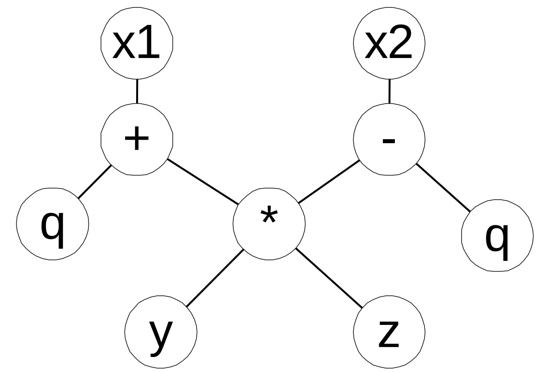
```
x1=q+y*z
x2=y*z-q
```

Naive:



y*z calculated twice

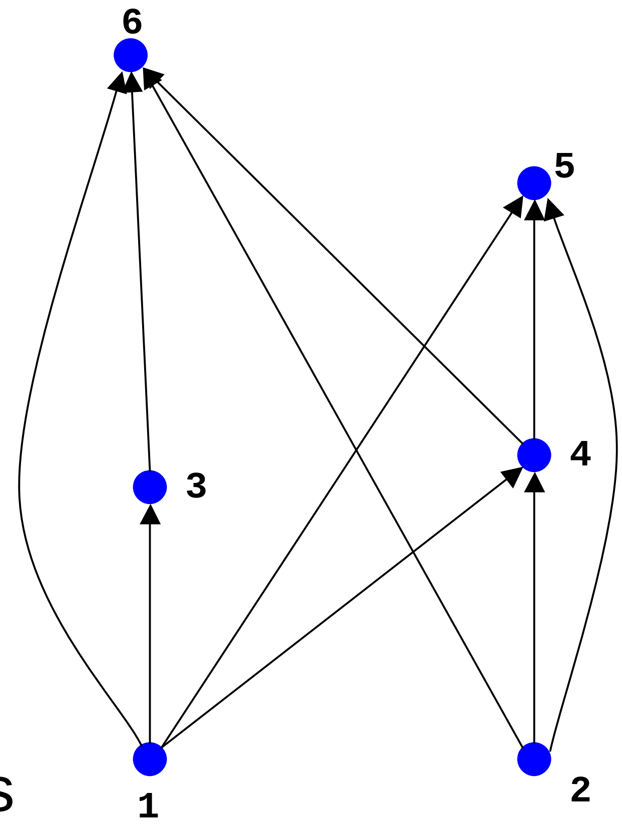
common
subexpression
eliminated:



Nodes = symbols/operators
Edges = relationships

Precedence

S_1	`a=0;`
S_2	`b=1;`
S_3	`c=a+1`
S_4	`d=b+a;`
S_5	`e=d+1;`
S_6	`e=c+d;`



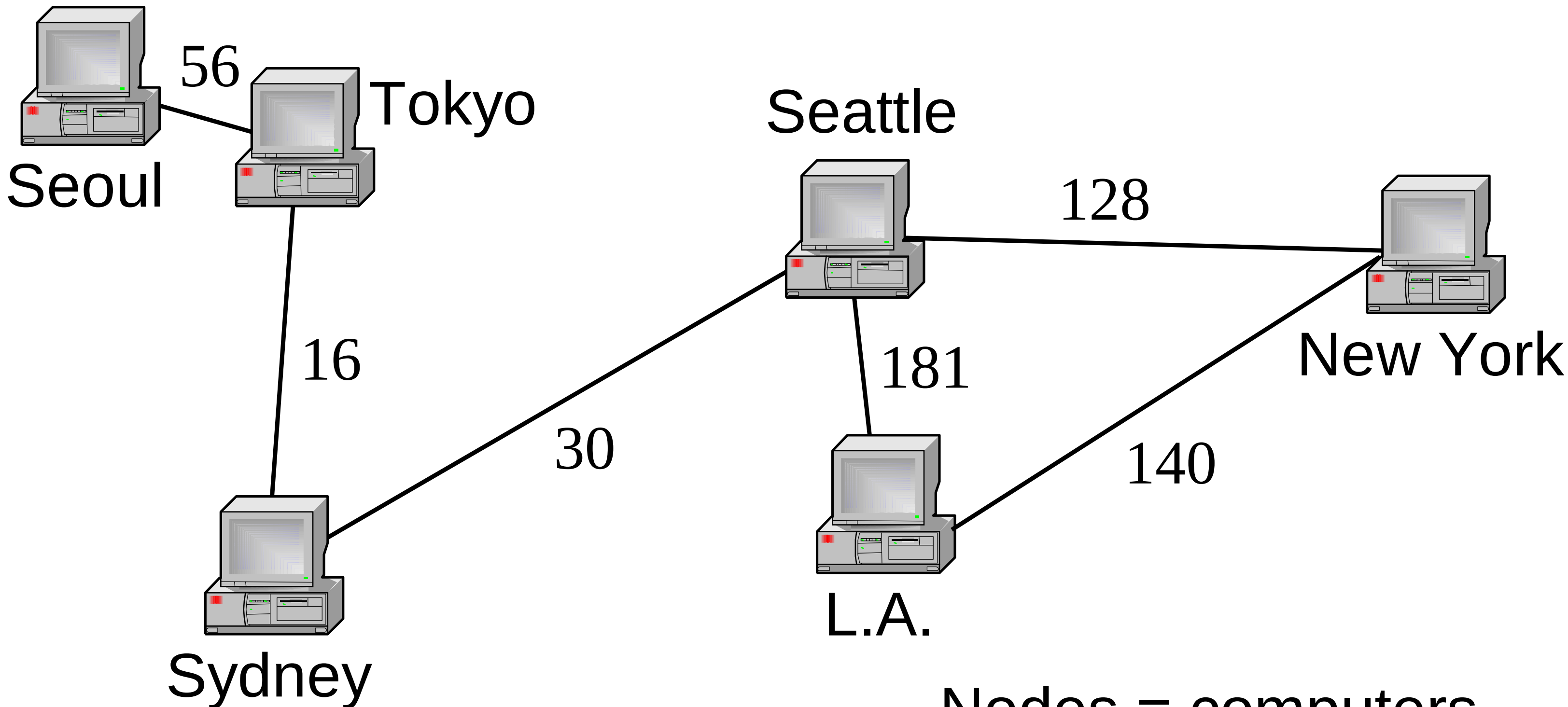
Which statements must execute before S_6?

S_1, S_2, S_3, S_4

Nodes = statements

Edges = precedence requirements

Information Transmission in a Computer Network

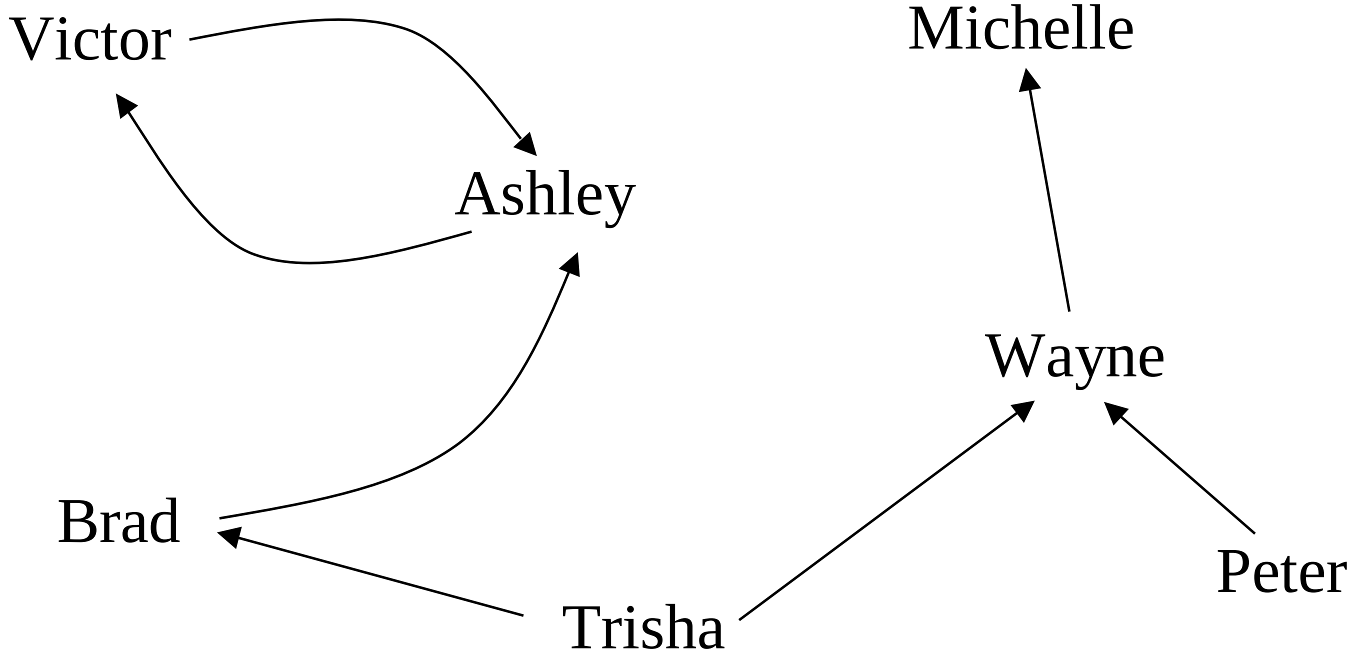


Nodes = computers
Edges = transmission rates

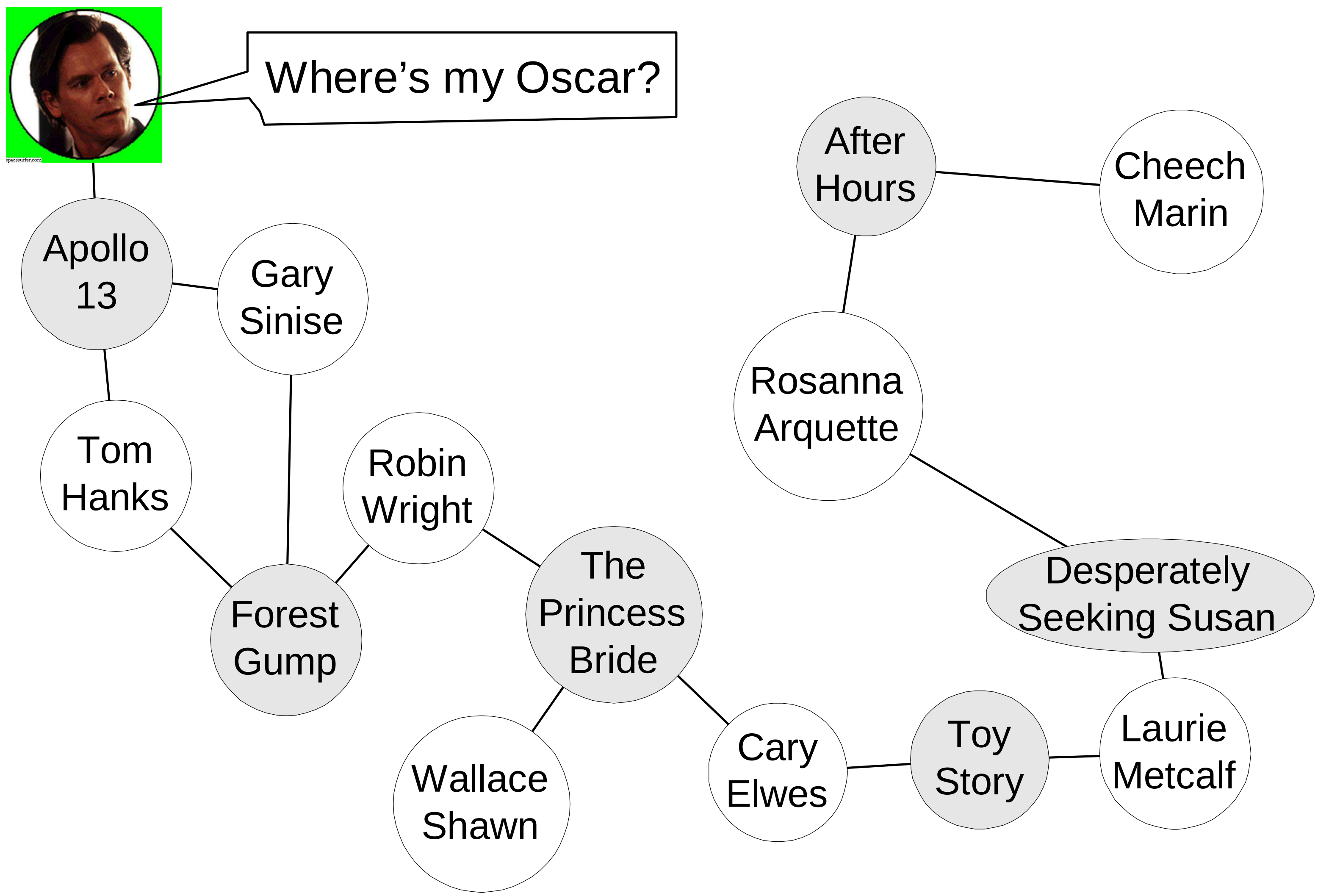
Traffic Flow on Highways

Nodes = cities
Edges = # vehicles on connecting highway

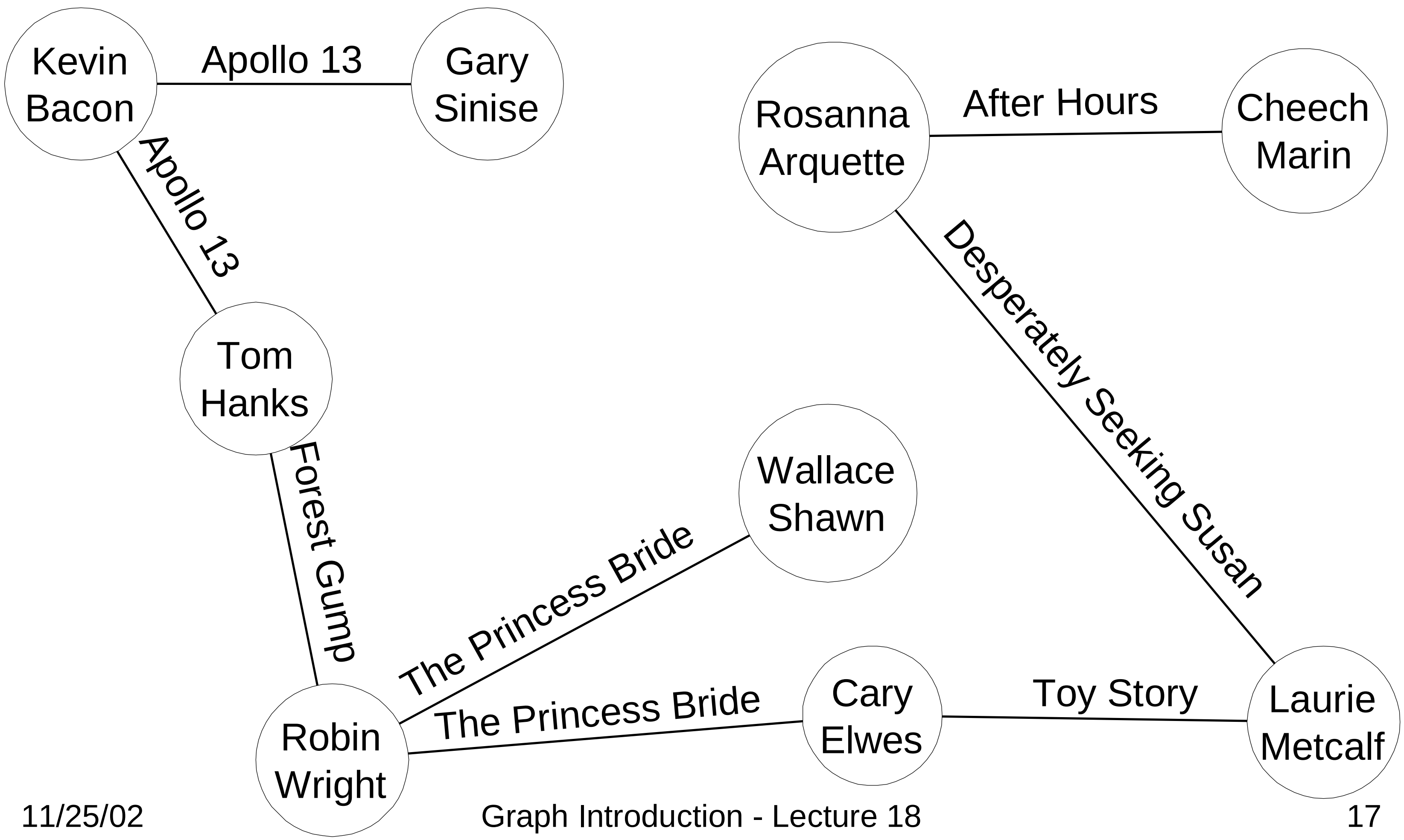
Soap Opera Relationships



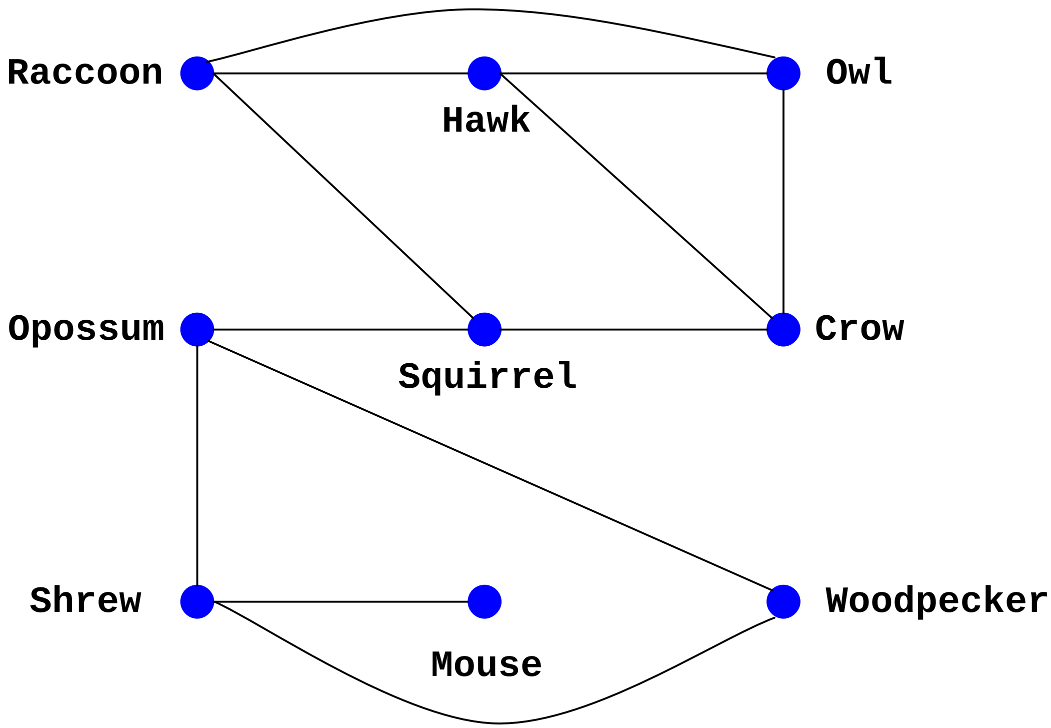
Six Degrees of Separation from Kevin Bacon



Six Degrees of Separation from Kevin Bacon



Niche overlaps



Graph Definition

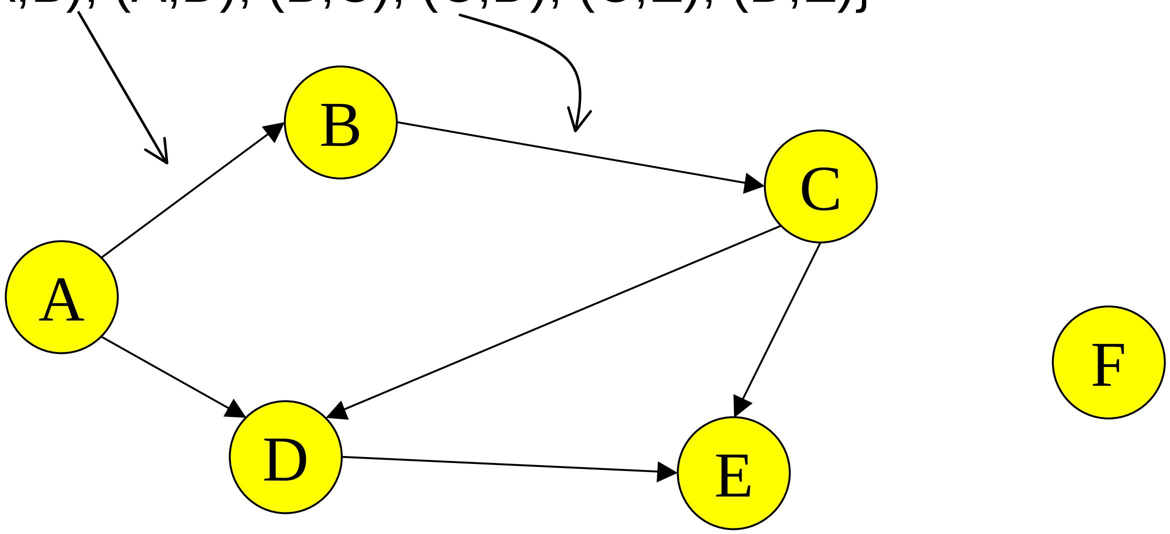
- A graph is simply a collection of nodes plus edges
  - › Linked lists, trees, and heaps are all special cases of graphs
- The nodes are known as vertices (node = “vertex”)
- Formal Definition: A graph G is a pair (V, E) where
  - › V is a set of vertices or nodes
  - › E is a set of edges that connect vertices

Graph Example

- Here is a directed graph $G = (V, E)$
  - Each edge is a pair (v_1, v_2), where v_1, v_2 are vertices in V

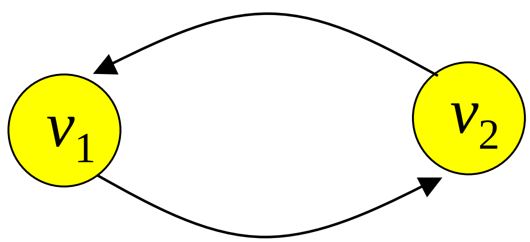
V = {A, B, C, D, E, F}

E = {(A,B), (A,D), (B,C), (C,D), (C,E), (D,E)}

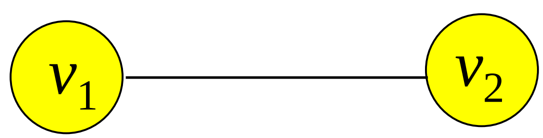


Directed vs Undirected Graphs

- If the order of edge pairs (v_1, v_2) matters, the graph is directed (also called a digraph): $(v_1, v_2) \neq (v_2, v_1)$



- If the order of edge pairs (v_1, v_2) does not matter, the graph is called an undirected graph: in this case, $(v_1, v_2) = (v_2, v_1)$



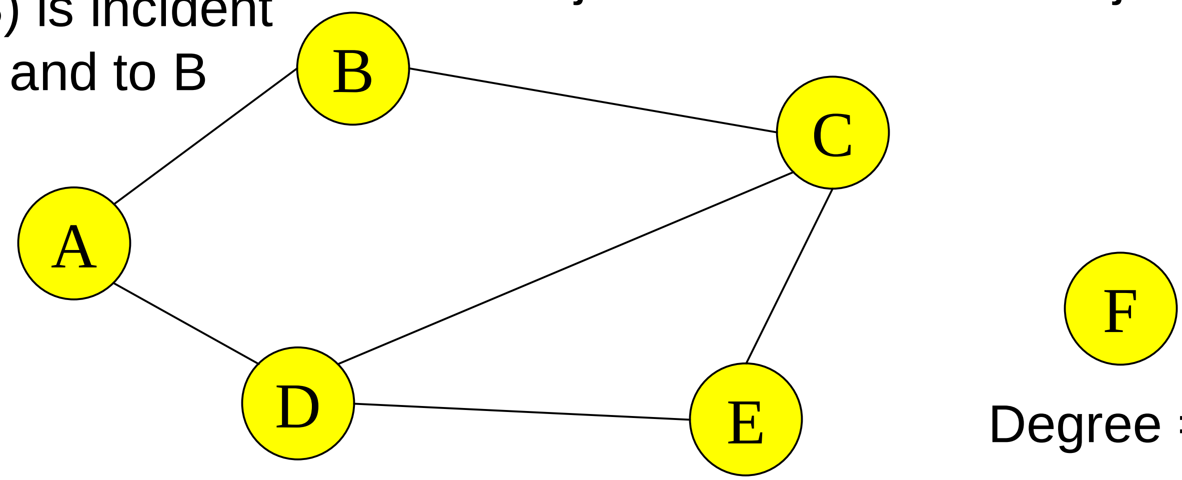
Undirected Terminology

- Two vertices u and v are adjacent in an undirected graph G if {u,v} is an edge in G
  - edge e = {u,v} is incident with vertex u and vertex v
- The degree of a vertex in an undirected graph is the number of edges incident with it
  - a self-loop counts twice (both ends count)
  - denoted with deg(v)

Undirected Terminology

B is adjacent to C and C is adjacent to B

(A,B) is incident
to A and to B



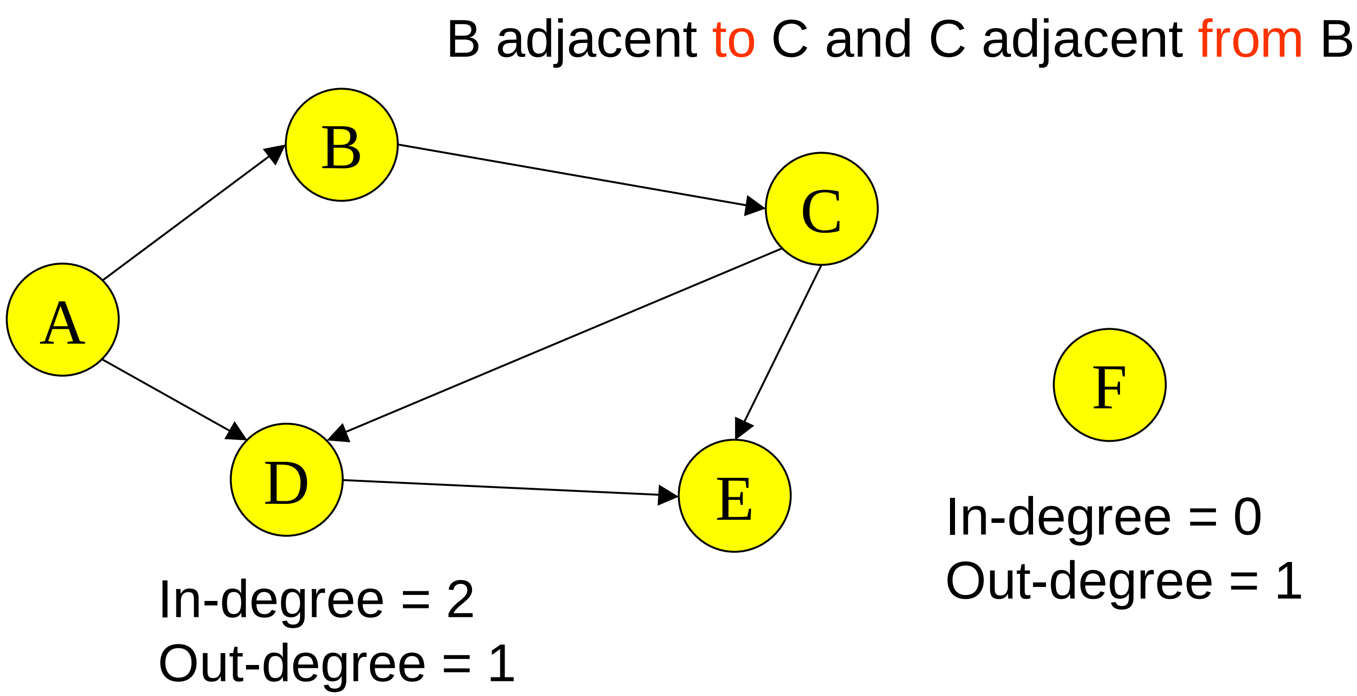
Degree = 3

Degree = 0

Directed Terminology

- Vertex u is adjacent to vertex v in a directed graph G if (u,v) is an edge in G
  - vertex u is the initial vertex of (u,v)
- Vertex v is adjacent from vertex u
  - vertex v is the terminal (or end) vertex of (u,v)
- Degree
  - in-degree is the number of edges with the vertex as the terminal vertex
  - out-degree is the number of edges with the vertex as the initial vertex

Directed Terminology



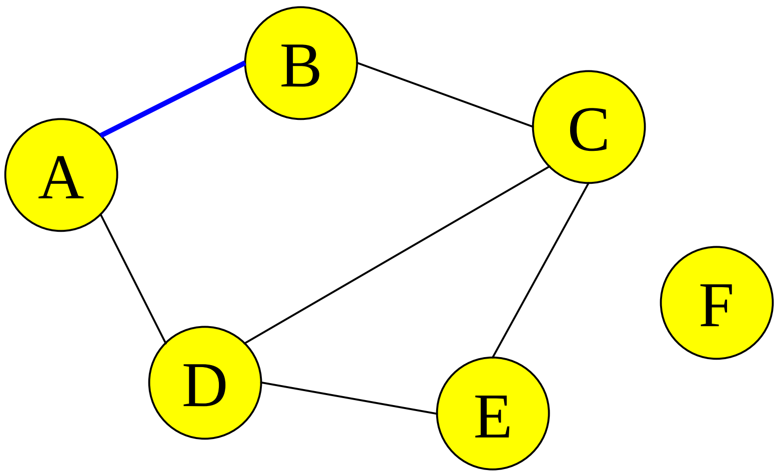
Handshaking Theorem

- Let G=(V,E) be an undirected graph with |E|=e edges
- Then $2e = \sum_{v \in V} \deg(v)$
- Every edge contributes +1 to the degree of each of the two vertices it is incident with
  - › number of edges is exactly half the sum of deg(v)
  - › the sum of the deg(v) values must be even

Graph Representations

- Space and time are analyzed in terms of:
  - Number of vertices = $|V|$ and
  - Number of edges = $|E|$
- There are at least two ways of representing graphs:
  - The *adjacency matrix* representation
  - The *adjacency list* representation

Adjacency Matrix

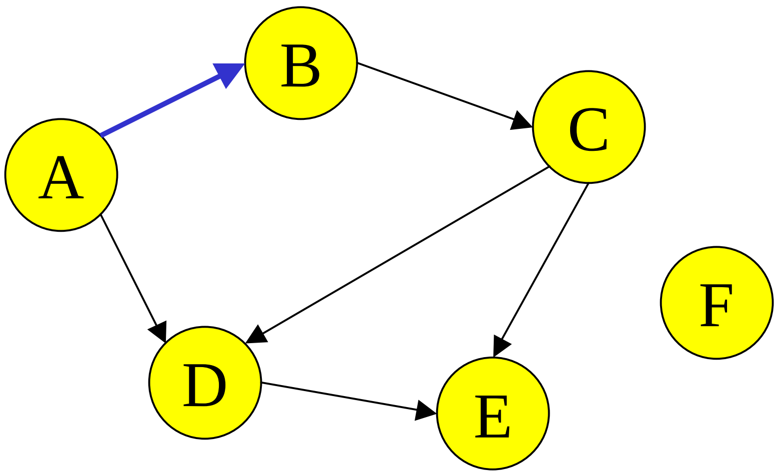


$$\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 0 & 1 & 0 & 0 \\ B & 1 & 0 & 1 & 0 & 0 & 0 \\ C & 0 & 1 & 0 & 1 & 1 & 0 \\ D & 1 & 0 & 1 & 0 & 1 & 0 \\ E & 0 & 0 & 1 & 1 & 0 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$M(v, w) = \begin{cases} 1 \text{ if } (v, w) \text{ is in E} \\ 0 \text{ otherwise} \end{cases}$$

Space = $|V|^2$

Adjacency Matrix for a Digraph



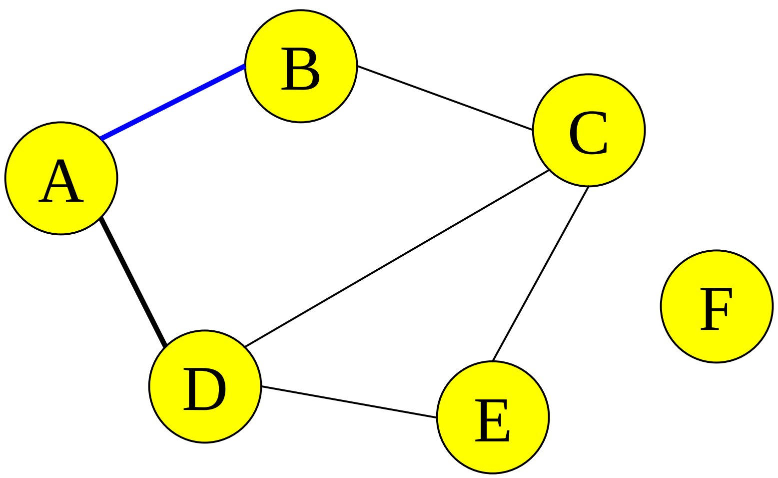
$$\begin{array}{c c}
 & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{array}$$

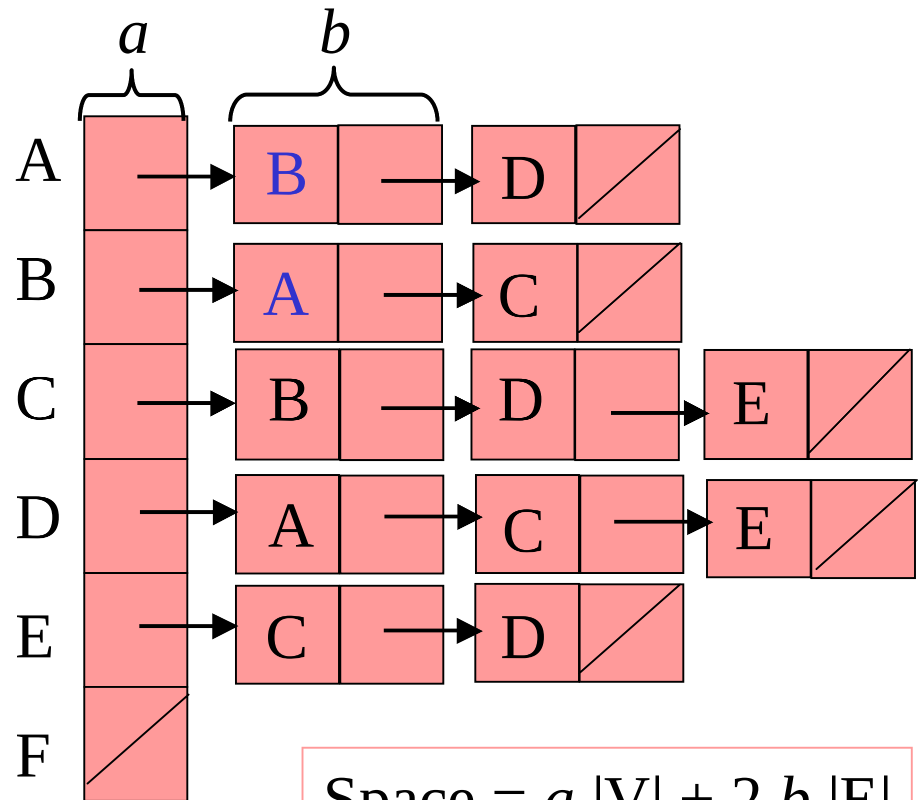
$$M(v, w) = \begin{cases} 1 \text{ if } (v, w) \text{ is in E} \\ 0 \text{ otherwise} \end{cases}$$

Space = $|V|^2$

Adjacency List

For each *v* in *V*, *L*(*v*) = list of *w* such that (*v*, *w*) is in *E*

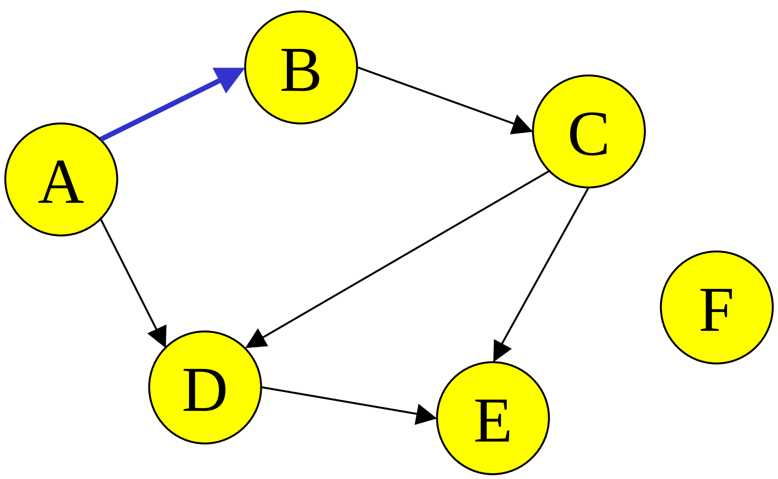


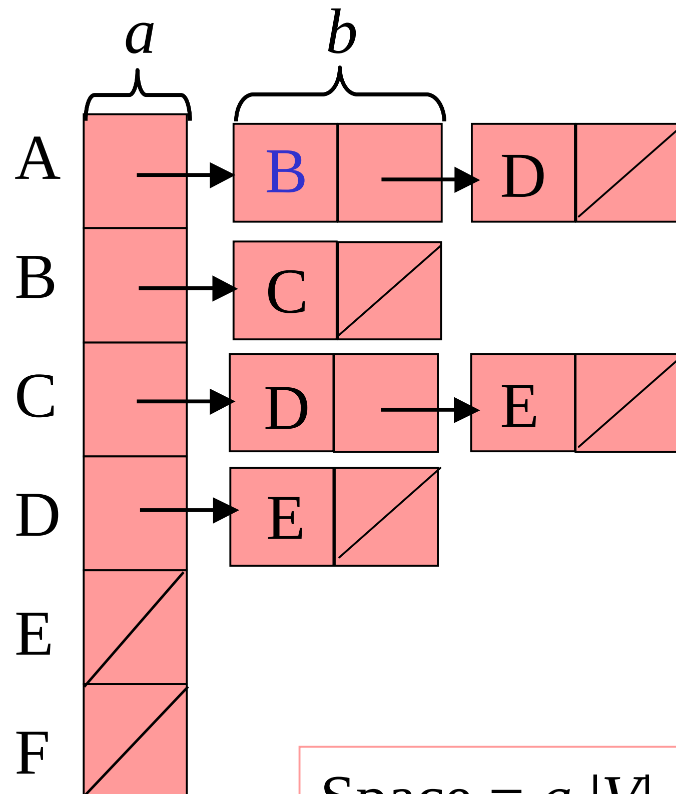


Space = $a\,|V| + 2\,b\,|E|$

Adjacency List for a Digraph

For each v in V, $L(v)$ = list of w such that (v, w) is in E



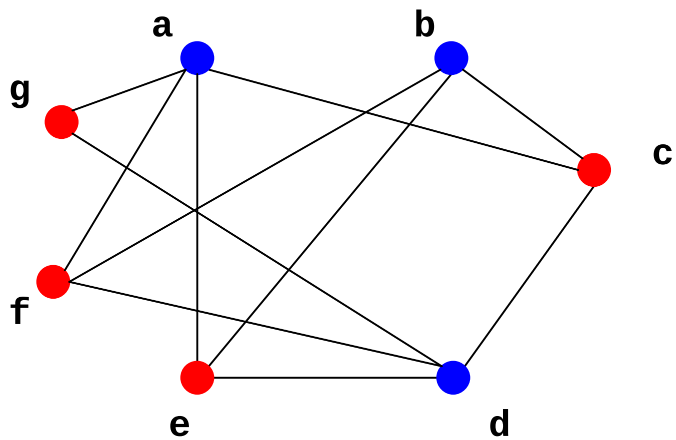


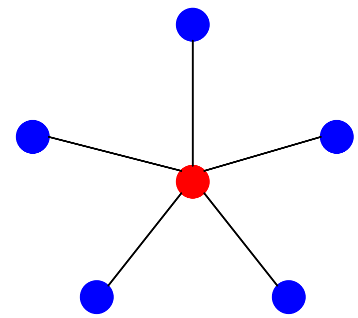
Space = $a\,|V| + b\,|E|$

Bipartite

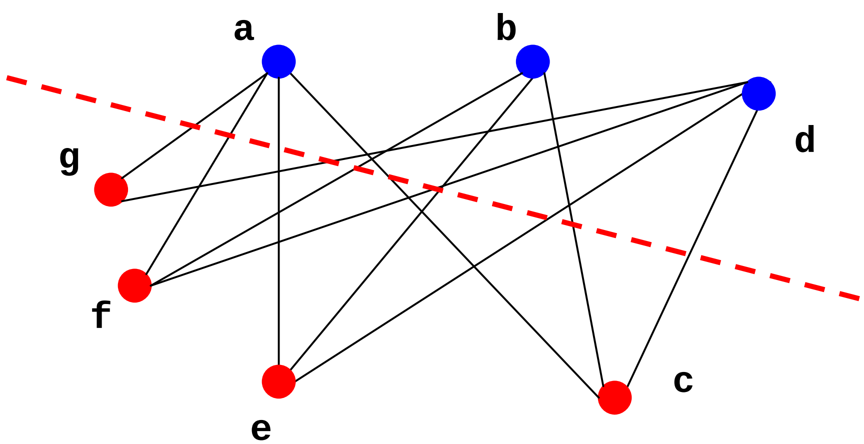
- A simple graph is bipartite if:
  - › its vertex set V can be partitioned into two disjoint non-empty sets such that
    - every edge in the graph connects a vertex in one set to a vertex in the other set
    - which also means that no edge connects a vertex in one set to another vertex in the same set
  - › no triangular or other odd length cycles

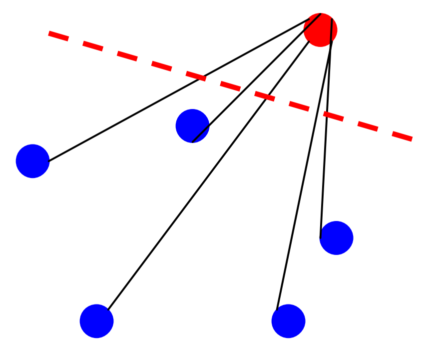
Bipartite examples



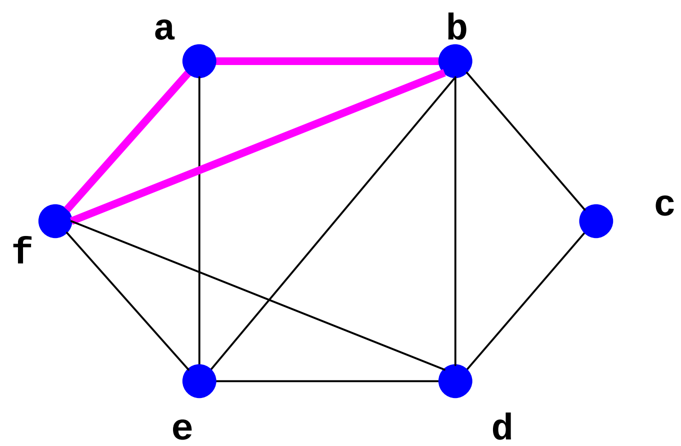


{a b d}
{c e f g}





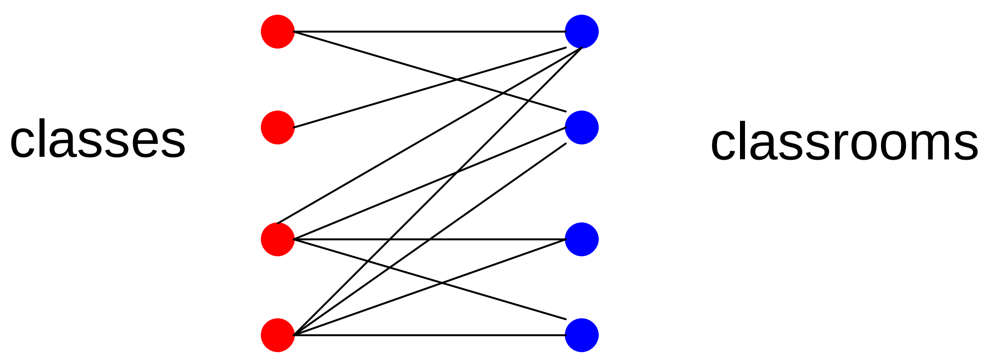
Bipartite example - not



a says that *b* and *f* should be in S_2,
but *b* says *a* and *f* should be in S_1.
TILT!

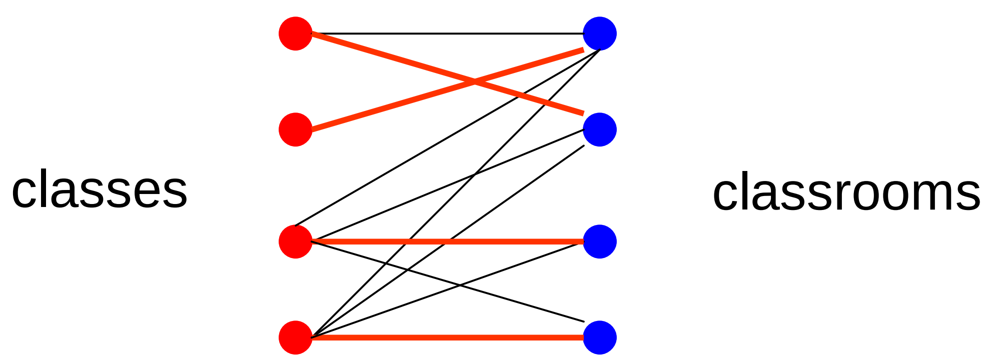
Bipartite Graph Application

- Classroom scheduling
  - Nodes are Classrooms and Classes
  - Edge between a classroom and class if the class will fit in the classroom and has the right technology.



Matching Problem

- Find an assignment of classes to classrooms so that every class fits and has the right technology.



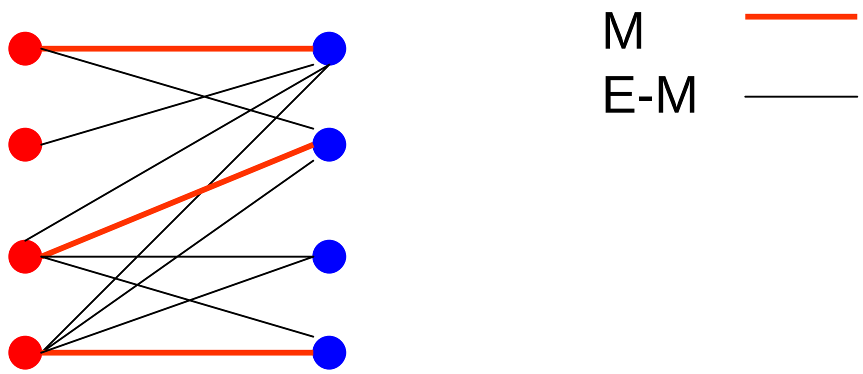
Steps in Solving the Problem

- Abstract the problem as a graph problem.
- Find an algorithm for solving the graph problem.
- Design data structures and algorithms to implement the graph solution.
- Write code

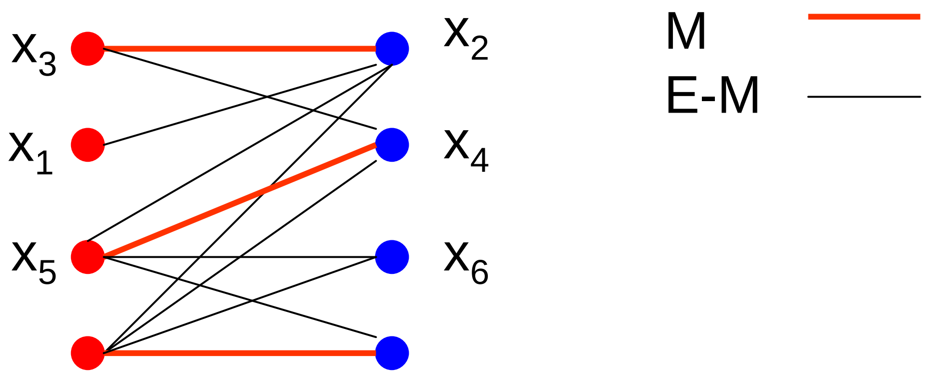
Alternating Path

- Let G = (U,V,E) be a bipartite graph where (u,v) in E only if u in U and v in V.
- A partial matching M is subset of E such that if (u,v) and (u',v') in M then either (u = u' and v = v') or (u $\neq$ u' or v $\neq$ v')
- An alternating path is $x_1,x_2,\ldots,x_{2n}$ such that
  - (x_i,x_{i+1}) in E – M if i is odd
  - (x_i,x_{i+1}) in M if i is even
  - x_1 and x_{2n} are not matched in the partial matching

Partial Matching



Alternating Path

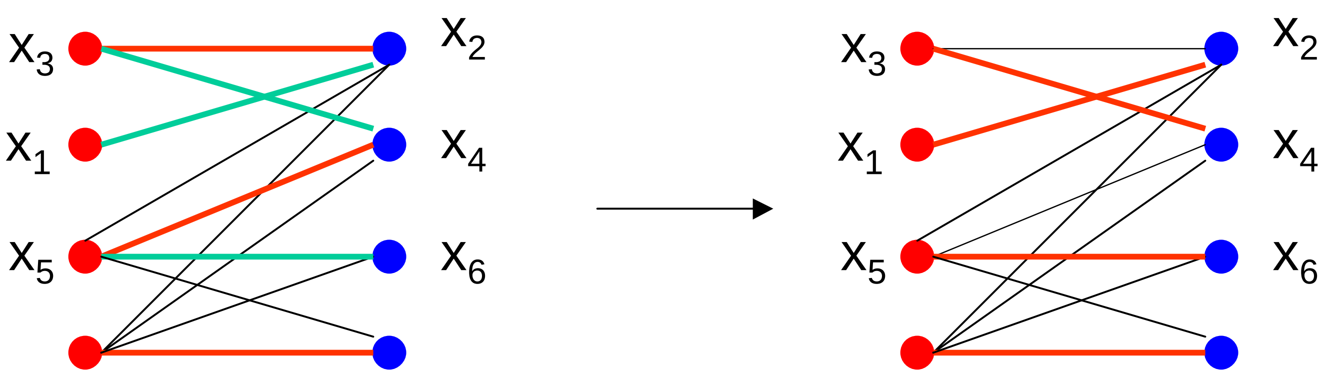


Matching Algorithm

set M to be the empty set initially
repeat
find an alternating path $x_1,x_2,\ldots,x_{2n}$
// (x_i,x_{i+1}) in E – M if i is odd and (x_i,x_{i+1}) in M if i is even
neither x_1 nor x_{2n} matched //
delete (x_i,x_{i+1}) from M if i is even
add (x_i,x_{i+1}) to M if i is odd
until no alternating path can be found

if M has every vertex of U then M is a matching
if M does not have some vertex then there is complete matching, but M is a maximum size matching

One step in the Algorithm



Maximum Matching

- Prove that M is maximum size if and only if there is no alternating path.
- Design data structures algorithms to find alternating paths or determine they don’t exist.
  - › Goal: fast data structures and algorithms