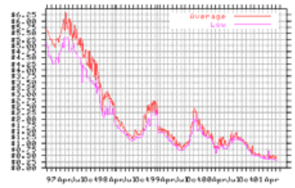
Graph Terminology

CSE 373
Data Structures

What are graphs?

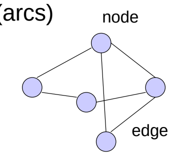
- Yes, this is a graph….



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

Graphs

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

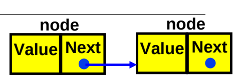


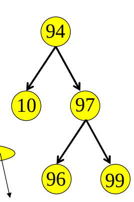
Varieties

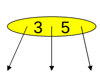
- Nodes
  - › Labeled or unlabeled
- Edges
  - › Directed or undirected
  - › Labeled or unlabeled

Motivation for Graphs

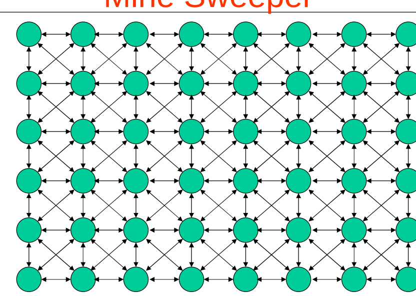
- Consider the data structures we have looked at so far…
- Linked list: nodes with 1 incoming edge + 1 outgoing edge
- Binary trees/heaps: nodes with 1 incoming edge + 2 outgoing edges
- B-trees: nodes with 1 incoming edge + multiple outgoing edges







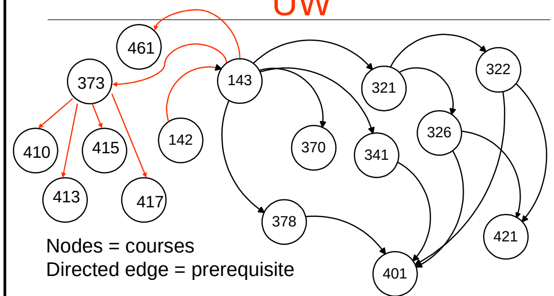
A Very Regular Graph: Mine Sweeper



Motivation for Graphs

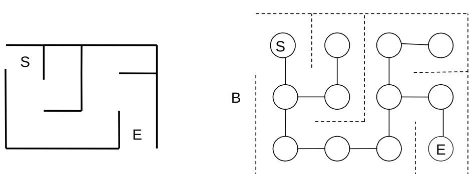
- How can you generalize these data structures?
- Consider data structures for representing the following problems…

CSE Course Prerequisites at UW



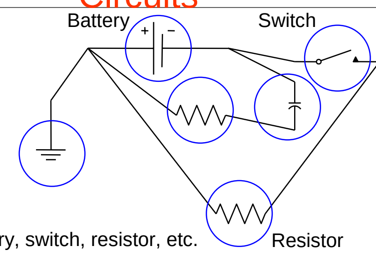
Nodes = courses
Directed edge = prerequisite

Representing a Maze



Nodes = rooms
Edge = door or passage

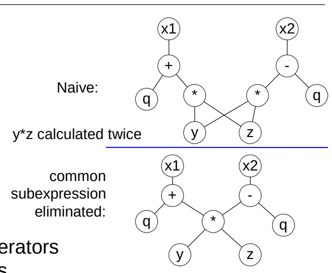
Representing Electrical Circuits



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

Program statements

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

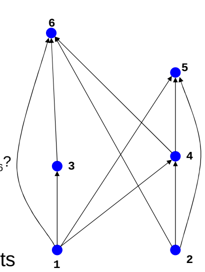


Nodes = symbols/operators
Edges = relationships

Precedence

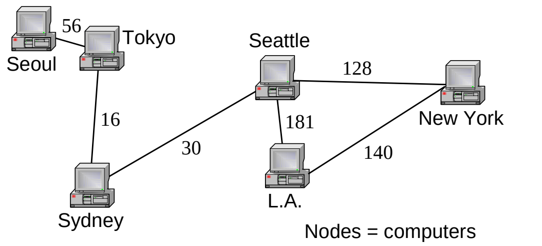
S_1	`a=0;`
S_2	`b=1;`
S_3	`c=a+1`
S_4	`d=b+a;`
S_5	`e=d+1;`
S_6	`e=c+d;`

Which statements must execute before S_6?
S_1, S_2, S_3, S_4

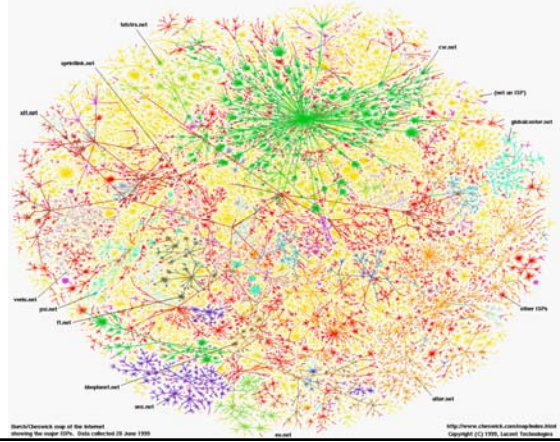


Nodes = statements
Edges = precedence requirements

Information Transmission in a Computer Network



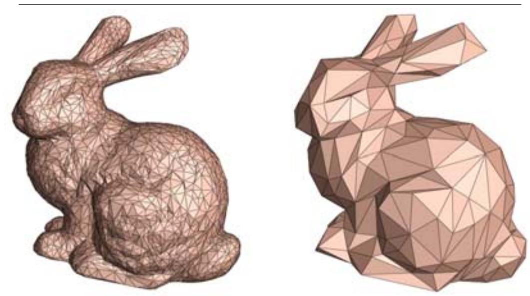
Nodes = computers
Edges = transmission rates



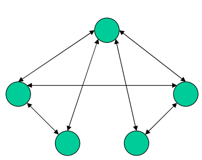
Traffic Flow on Highways

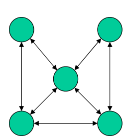
Nodes = cities
Edges = # vehicles on connecting highway

Polygonal Meshes

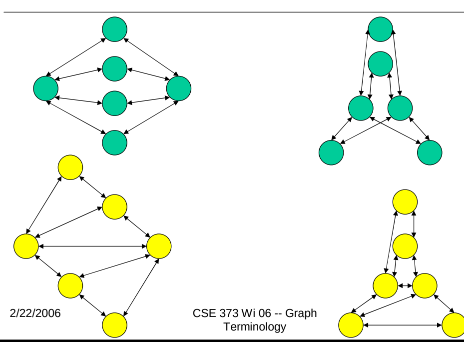


Isomorphism

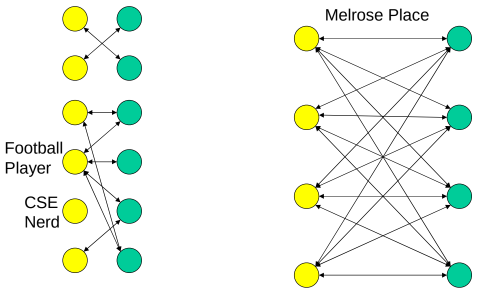




Isomorphism

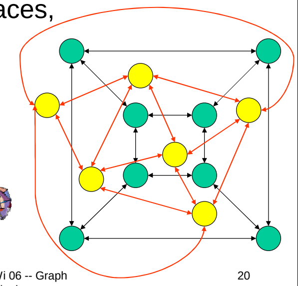


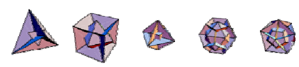
Bipartite Graphs



Duality

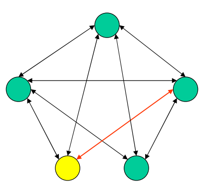
- Vertices become faces, faces vertices
- Max-flow becomes Min-cut

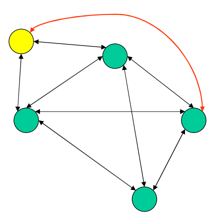




Planarity

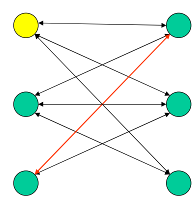
Can the circuit be put onto the chip in one layer?

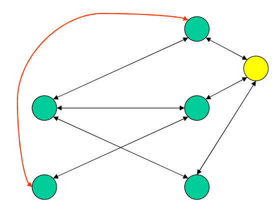




Planarity

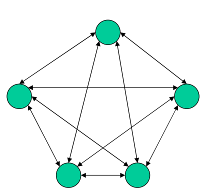
Can the circuit be put onto the chip in one layer?



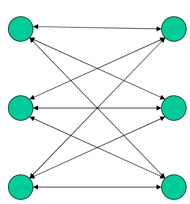


Planarity

Can the circuit be put onto the chip in one layer?



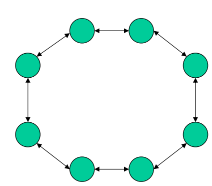
K_5



K_3,3

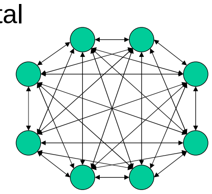
Sparsely Connected Graph

- n vertices
- worst n/2 edges between two vertices
- n edges total



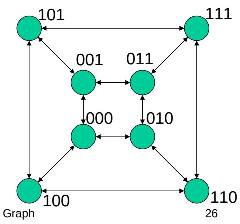
Densely Connected Graph

- n vertices total
- worst 1 edge between two vertices
- ½(n^2-n) edges total

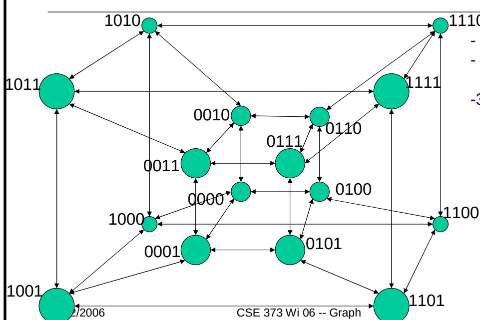


In Between (Hypercube)

- n vertices
- worst log n edges between two vertices
- ½ n log n edges total



In Between (Hypercube)



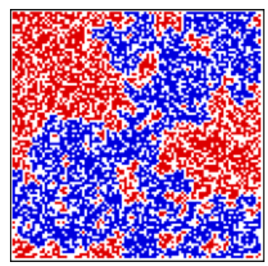
- 16 nodes
- worst 4 edges btwn two nodes
-32 total edges

S: (16,8,16)
D: (16,1,120)

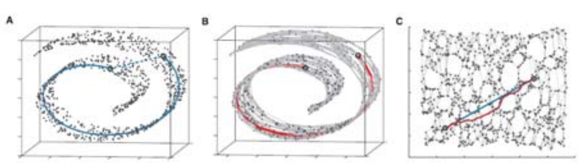
S: (32,16,32)
H: (32,5,80)
D: (32,1,496)

S: (64,32,64)
H: (64,6,192)
D: (64,1,2016)

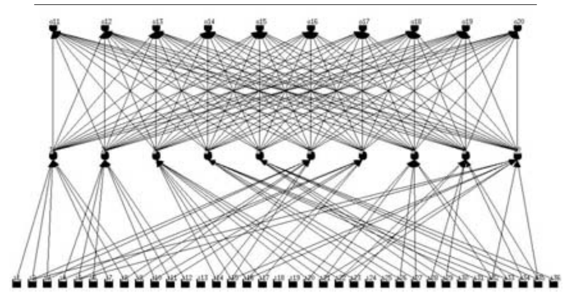
Statistical Mechanics



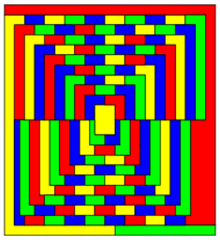
Modeling Nonlinear Data

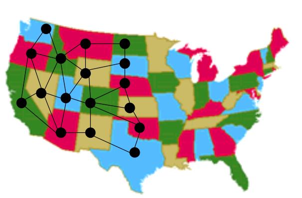


Neural Networks



Colorings





"We should mention that both our programs use only integer arithmetic, and so we need not be concerned with round-off errors and similar dangers of floating point arithmetic. However, an argument can be made that our 'proof' is not a proof in the traditional sense, because it contains steps that can never be verified by humans. In particular, we have not proved the correctness of the compiler we compiled our programs on, nor have we proved the infallibility of the hardware we ran our programs on. These have to be taken on faith, and are conceivably a source of error. However, from a practical point of view, the chance of a computer error that appears consistently in exactly the same way on all runs of our programs on all the compilers under all the operating systems that our programs run on is infinitesimally small compared to the chance of a human error during the same amount of case-checking. Apart from this hypothetical possibility of a computer consistently giving an incorrect answer, the rest of our proof can be verified in the same way as traditional mathematical proofs. We concede, however, that verifying a computer program is much more difficult than checking a mathematical proof of the same length."

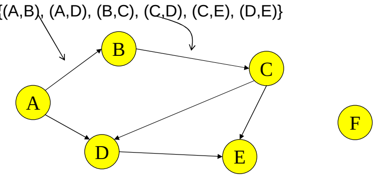
Graph Definition

- 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

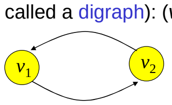
- Here is a directed graph $G = (V, E)$
  - › Each edge is a pair (v_1, v_2), where v_1, v_2 are vertices in V
  - › V = {A, B, C, D, E, F}

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

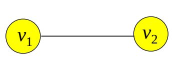


Directed vs Undirected Graphs

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



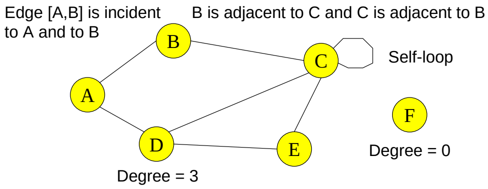
- If the order of edge pairs (v_1, v_2) does not matter, the graph is called an undirected graph: in this case, $(v_1, v_2) = (v_2, v_1)$



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
  - › Some undirected graphs allow "self loops". These will need slightly different notation, because {u,u} = {u}. Therefore, use [u,v] and [u,u] to represent the edges of such graphs.
- 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)

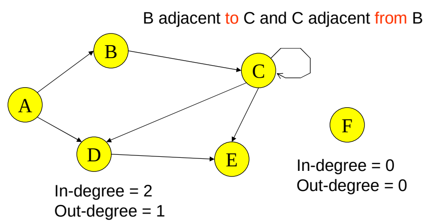
Undirected Graph Terminology



Directed Graph 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

$$2e = \sum_{v \in V} \deg(v)$$

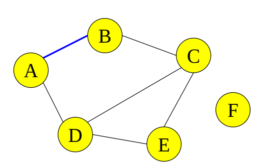
Add up the degrees of all vertices.

- 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

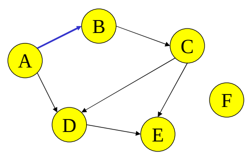


	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	1	0
D	1	0	1	0	1	0
E	0	0	1	1	0	0
F	0	0	0	0	0	0

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

Space = $|V|^2$

Adjacency Matrix for a Digraph



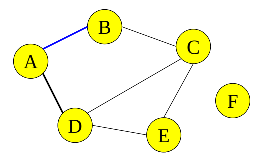
$$
\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & 0 & 1 & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 1 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 1 & 1 & 0 \\
D & 0 & 0 & 0 & 0 & 1 & 0 \\
E & 0 & 0 & 0 & 0 & 0 & 0 \\
F & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

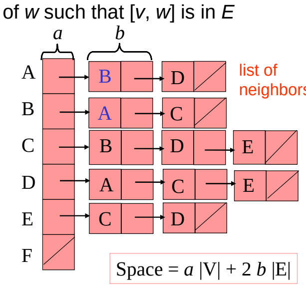
$$M(v, w) = \begin{cases} 1 \text{ if } (v, w) \text{ is in E} \\ 0 \text{ otherwise} \end{cases}$$

Space = $|V|^2$

Adjacency List

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

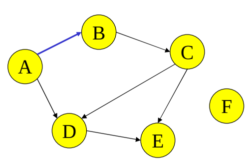


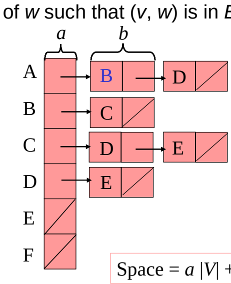


Space = $a\,|V| + 2\,b\,|E|$

Adjacency List for a Digraph

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





A is a source
E is a sink
F is disconnected from the rest

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