Reading

• Reading Chapter 13
  › Sections 13.1 and 13.2
What are graphs?

• Yes, this is a graph….

• But we are interested in a different kind of “graph”
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
- **Up-trees**: nodes with multiple incoming edges + 1 outgoing edge
Motivation for Graphs

• How can you generalize these data structures?
• Consider data structures for representing the following problems…
Representing a Maze

Nodes = cells
Edges = door or passage
CSE Course Prerequisites at UW

Nodes = courses
Directed edge = prerequisite

Graph Terminology
Representing Electrical Circuits

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

Nodes = symbols/operators
Edges = relationships

\[ x_1 = q + y \cdot z \]
\[ x_2 = y \cdot z - q \]

Naive:

\[ + \quad \ast \quad - \]
\[ q \quad y \quad z \quad y \quad z \quad q \]
\[ x_1 \quad x_2 \]

Common subexpression eliminated:

\[ + \quad \ast \quad - \]
\[ q \quad y \quad z \quad y \quad z \quad q \]
\[ x_1 \quad x_2 \]
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
Isomorphism

Same number of vertices connected in the same way

Time complexity to test if 2 graphs are isomorphic?
Bipartite Graphs

Two disjoint sets of vertices. Edges link a vertex from one set to a vertex in the other set.

Football Player

CSE Nerd

Melrose Place

Graph Terminology
Planarity

Can the circuit be put onto the chip in one layer?
Related Problems: Puzzles

Two problems:

1) Can you draw these without lifting your pen, drawing each line only once
2) Can you start and end at the same point.
Sparsely Connected Graph

- \(n\) vertices
- \(n\) edges total
- Ring
Densely Connected Graph

- n vertices total
- \( \frac{n(n-1)}{2} \) edges total (w/o self loops)
In Between (Hypercube)

- $n$ vertices
- $\log n$ edges between two vertices
- $\frac{1}{2} n \log n$ edges total
In Between (Hypercube)

- 16 nodes
- 4 edges
  between 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)
Neural Networks
Colorings
Four Color Conjecture

- is it true that any map can be colored using four colors in such a way that adjacent regions (i.e. those sharing a common boundary segment, not just a point) receive different colors (1852)?
- Many attempts at proof
- Finally “solved” by computer program (1974)
  › Still extremely complex....
“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 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 graph $G = (V, E)$
  - Each **edge** is a pair $(v_1, v_2)$, where $v_1, v_2$ are vertices in $V$
  - $V = \{A, B, C, D, E, F\}$
  - $E = \{(A,B), (A,D), (B,C), (C,D), (C,E), (D,E)\}$

![Graph Diagram]

- $A$ - B - C - D - E - F

*Graph Terminology*
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 $\text{deg}(v)$
Undirected Terminology

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

B is adjacent to C and C is adjacent to B

Self-loop

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

B adjacent to C and C adjacent from B

In-degree = 2
Out-degree = 1

In-degree = 0
Out-degree = 0
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
Handshaking Theorem II

• For a directed graph:

$$\sum_{v \in G} ind(v) = \sum_{v \in g} outd(v) = e$$
Graph ADT

- Nothing unexpected
  - Build the graph (vertices, edges)
  - Return the edges incident in (or out) of a vertex
  - Find if two vertices are adjacent etc..
  - Replace ..., Insert...Remove ...

- What is interesting
  - How to represent graphs in memory
  - What representation to use for what algorithms
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

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

Space = \(|V|^2\)
Adjacency Matrix for a Digraph

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

Space = \(|V|^2\)
Adjacency List

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

Space $= a |V| + 2 b |E|$
Adjacency List for a Digraph

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

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