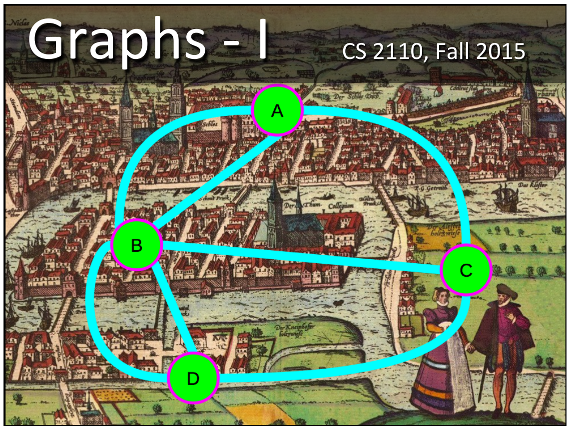
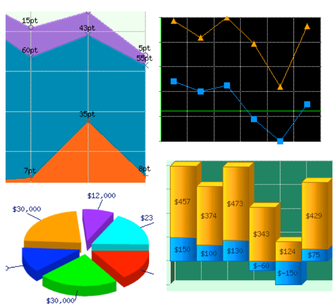


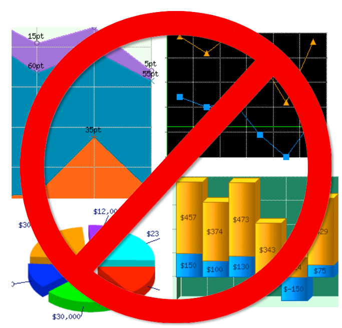
Readings

- Chapter 28: Graphs
- Chapter 29: Graph Implementations

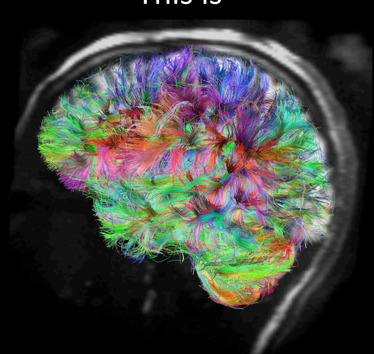
These *aren't* the graphs we're interested in



These *aren't* the graphs we're interested in

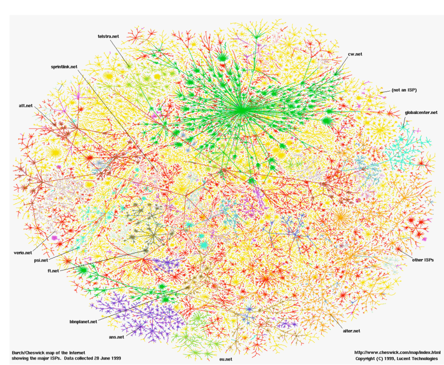


This is

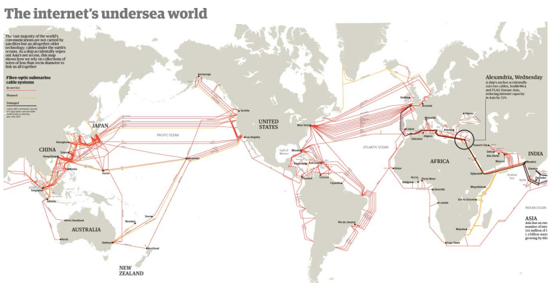


V.J. Wedeen and L.L. Wald, Martinos Center for Biomedical Imaging at MGH

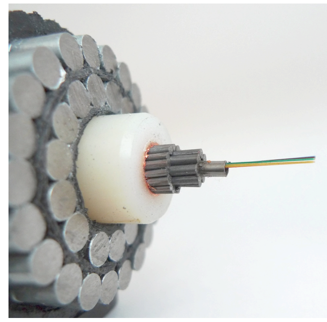
And so is this



And this

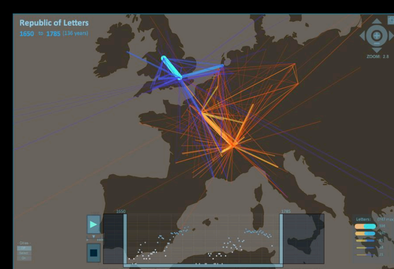


This carries Internet traffic across the oceans



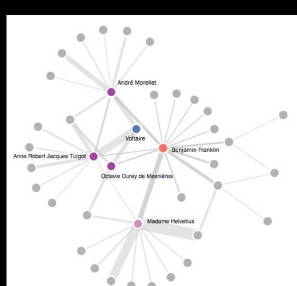
A social graph

An older social graph



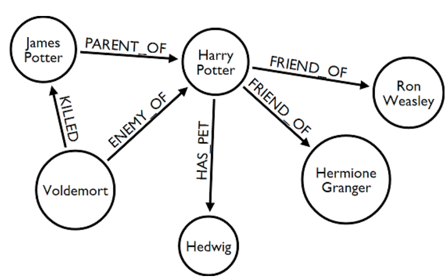
Locke's (blue) and Voltaire's (yellow) correspondence.
Only letters for which complete location information is available are shown.
Data courtesy the Electronic Enlightenment Project, University of Oxford.

An older social graph



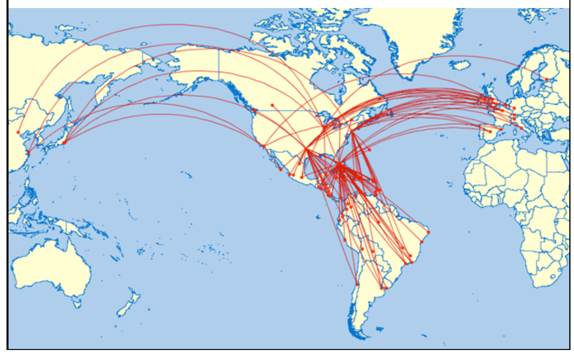
Voltaire and Benjamin Franklin

A fictional social graph

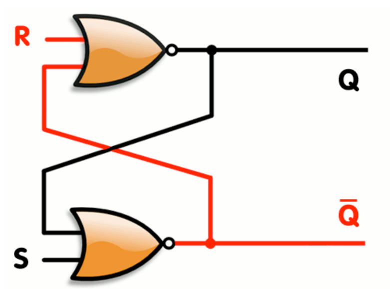


A transport graph

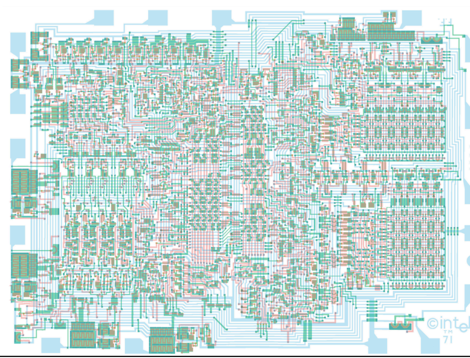
Another transport graph



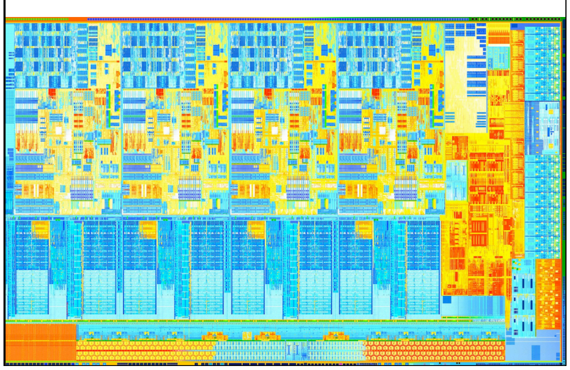
A circuit graph (flip-flop)



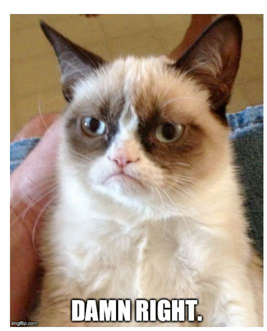
A circuit graph (Intel 4004)



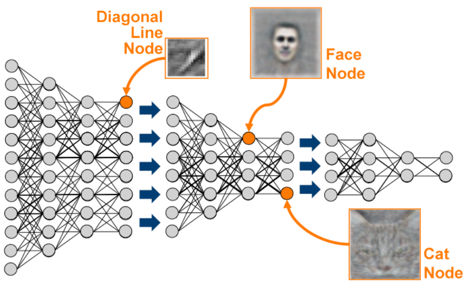
A circuit graph (Intel Haswell)



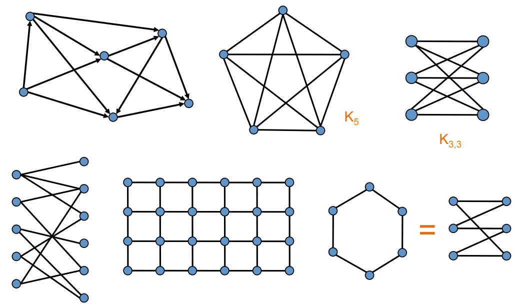
This is not a graph, this is a cat



This is a graph(ical model) that has learned to recognize cats



Some abstract graphs

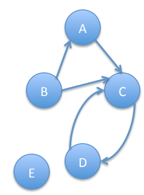


Applications of Graphs

- Communication networks
- Social networks
- Routing and shortest path problems
- Commodity distribution (network flow)
- Traffic control
- Resource allocation
- Numerical linear algebra (sparse matrices)
- Geometric modeling (meshes, topology, …)
- Image processing (e.g. graph cuts)
- Computer animation (e.g. motion graphs)
- Systems biology
- Digital humanities (e.g. Republic of Letters)
- …

Directed Graphs

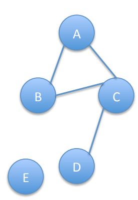
- A directed graph (digraph) is a pair (V, E) where
  - V is a set
  - E is a set of ***ordered*** pairs (u, v) where $u,v \in V$
    - Often require $u \neq v$ (i.e. no self-loops)
- An element of V is called a vertex or node
- An element of E is called an edge or arc
- $|V|$ = size of V, often denoted n
- $|E|$ = size of E, often denoted m



$V = \{A, B, C, D, E\}$
$E = \{(A,C), (B,A), (B,C), (C,D), (D,C)\}$
$|V| = 5$
$|E| = 5$

Undirected Graphs

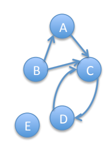
- An undirected graph is just like a directed graph!
  - … except that E is now a set of ***unordered*** pairs $\{u, v\}$ where $u,v \in V$
- Every undirected graph is easily converted to an equivalent directed graph
  - Replace every undirected edge with two directed edges in opposite directions
- … but not vice versa

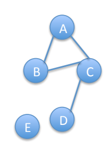


$V = \{A, B, C, D, E\}$
$E = \{\{A,C\}, \{B,A\}, \{B,C\}, \{C,D\}\}$
$|V| = 5$
$|E| = 4$

Graph Terminology

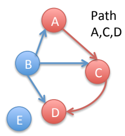
- Vertices u and v are called
  - the source and sink of the directed edge (u, v), respectively
  - the endpoints of (u, v) or $\{u, v\}$
- Two vertices are adjacent if they are connected by an edge
- The outdegree of a vertex u in a directed graph is the number of edges for which u is the source
- The indegree of a vertex v in a directed graph is the number of edges for which v is the sink
- The degree of a vertex u in an undirected graph is the number of edges of which u is an endpoint



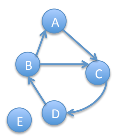


More Graph Terminology

- A path is a sequence $v_0, v_1, v_2, \dots, v_p$ of vertices such that for $0 \le i \le p-1$,
  - $(v_i, v_{i+1}) \in E$ if the graph is directed
  - $\{v_i, v_{i+1}\} \in E$ if the graph is undirected
- The length of a path is its number of edges
  - In this example, the length is 2
- A path is simple if it doesn't repeat any vertices
- A cycle is a path $v_0, v_1, v_2, \dots, v_p$ such that $v_0 = v_p$
- A cycle is simple if it does not repeat any vertices except the first and last
- A graph is acyclic if it has no cycles
- A ***d***irected ***a***cyclic ***g***raph is called a DAG

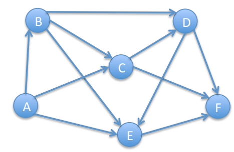


DAG



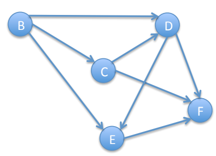
Not a DAG

Is this a DAG?



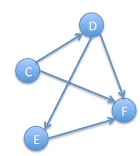
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?



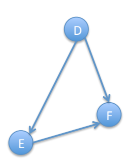
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?



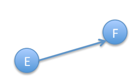
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?



- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?



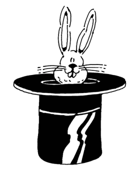
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?

- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

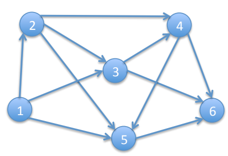
Is this a DAG?

YES!



- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

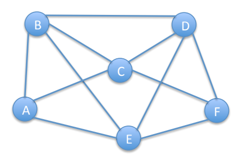
Topological Sort



- We just computed a topological sort of the DAG
  - This is a numbering of the vertices such that all edges go from lower- to higher-numbered vertices
  - Useful in job scheduling with precedence constraints

Graph Coloring

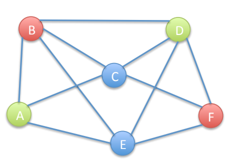
- A coloring of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color



- How many colors are needed to color this graph?

Graph Coloring

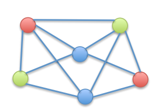
- A coloring of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color



- How many colors are needed to color this graph?

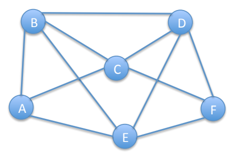
An Application of Coloring

- Vertices are **jobs**
- Edge (u, v) is present if jobs u and v each require access to the **same shared resource**, and thus cannot execute simultaneously
- Colors are **time slots** to schedule the jobs
- Minimum number of colors needed to color the graph = minimum number of time slots required



Planarity

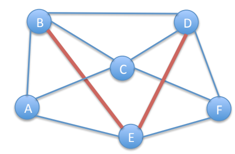
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?

Planarity

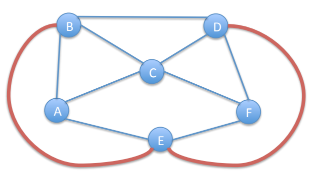
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?
  – Yes!

Planarity

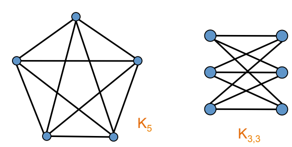
- A graph is planar if it **can** be drawn in the plane without any edges crossing



- Is this graph planar?
  – Yes!

Detecting Planarity

Kuratowski's Theorem:



- A graph is planar if and only if it does not contain a copy of K_5 or $K_{3,3}$ (possibly with other nodes along the edges shown)

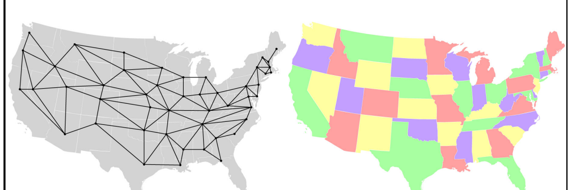
Four-Color Theorem:

Every planar graph is 4-colorable

[Appel & Haken, 1976]

(Every map defines a planar graph – countries are vertices, and two adjacent countries define an edge)

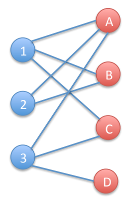
Another 4-colored planar graph



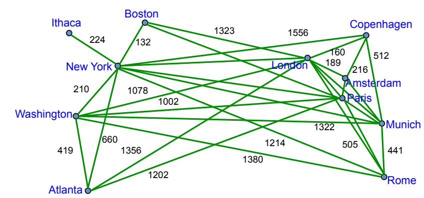
http://www.cs.cmu.edu/~bryant/boolean/maps.html

Bipartite Graphs

- A directed or undirected graph is bipartite if the vertices can be partitioned into two sets such that no edge connects two vertices in the same set
- The following are equivalent
  - G is bipartite
  - G is 2-colorable
  - G has no cycles of odd length

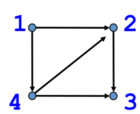


Traveling Salesperson

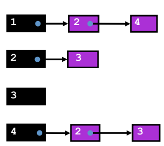


Find a path of minimum distance that visits every city

Representations of Graphs



Adjacency List



Adjacency Matrix

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	0	0	0
4	0	1	1	0

Adjacency Matrix or Adjacency List?

- Definitions:
  - n = number of vertices
  - m = number of edges
  - $d(u)$ = degree of u = number of edges leaving u
- Adjacency Matrix
  - Uses space $O(n^2)$
  - Can iterate over all edges in time $O(n^2)$
  - Can answer "Is there an edge from u to v?" in $O(1)$ time
  - Better for dense graphs (lots of edges)
- Adjacency List
  - Uses space $O(m + n)$
  - Can iterate over all edges in time $O(m + n)$
  - Can answer "Is there an edge from u to v?" in $O(d(u))$ time
  - Better for sparse graphs (fewer edges)

Graph Algorithms

- Search
  - Depth-first search
  - Breadth-first search
- Shortest paths
  - Dijkstra's algorithm
- Minimum spanning trees
  - Prim's algorithm
  - Kruskal's algorithm

Readings

- Chapter 28: Graphs
- Chapter 29: Graph Implementations