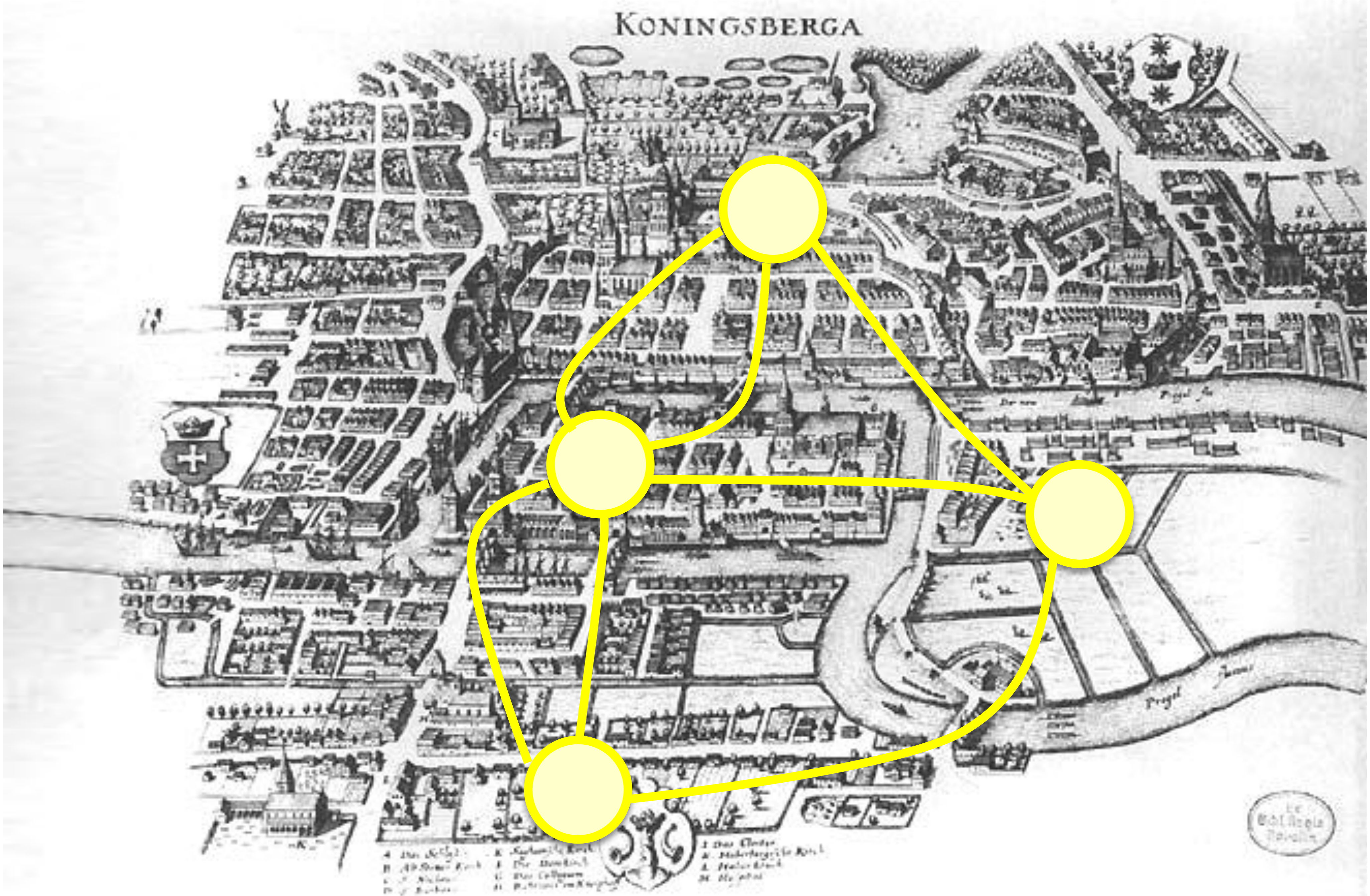
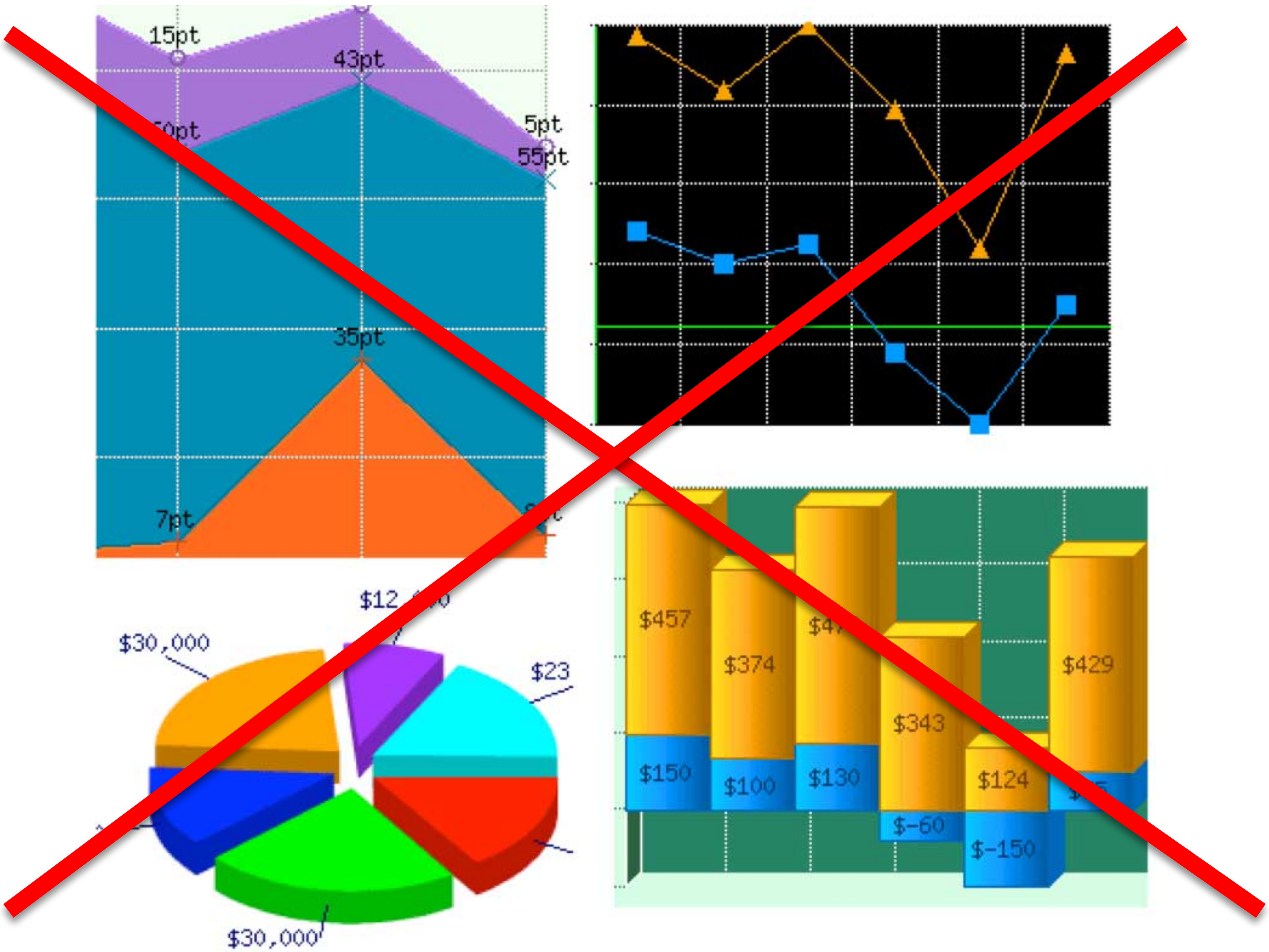


GRAPHS

Lecture 16

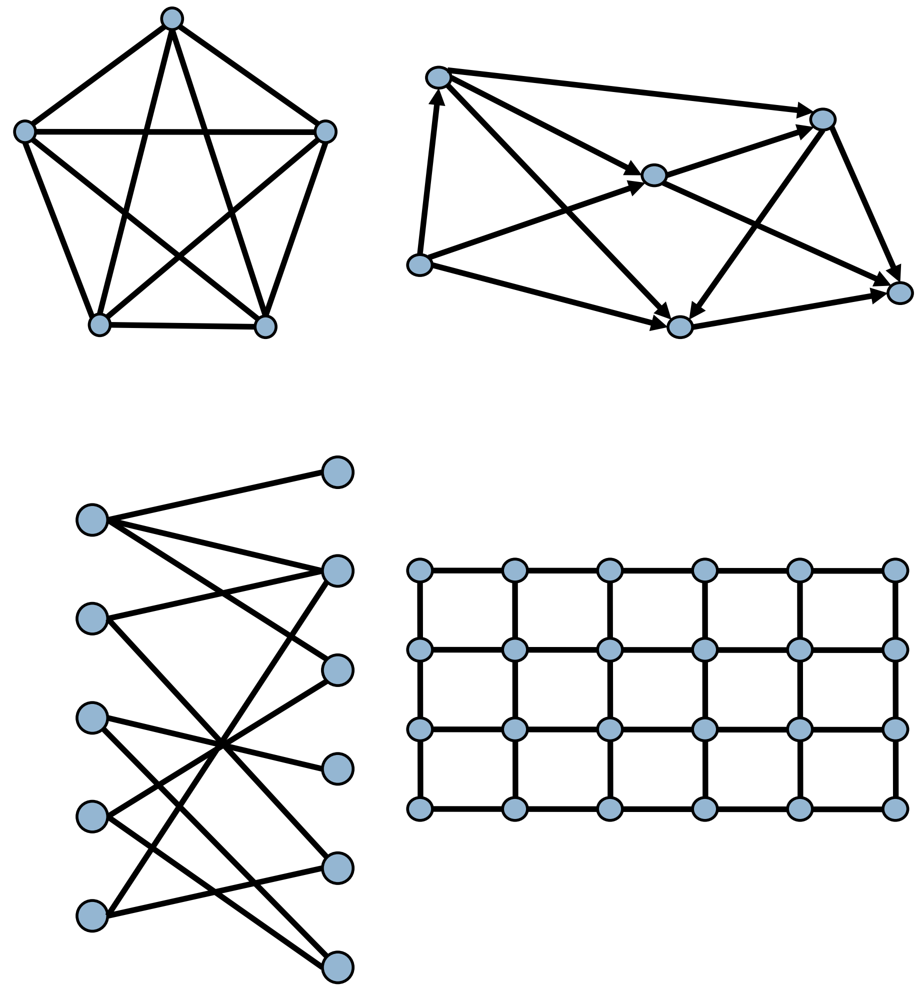
CS2110 Fall 2017

These aren't the graphs we're looking for



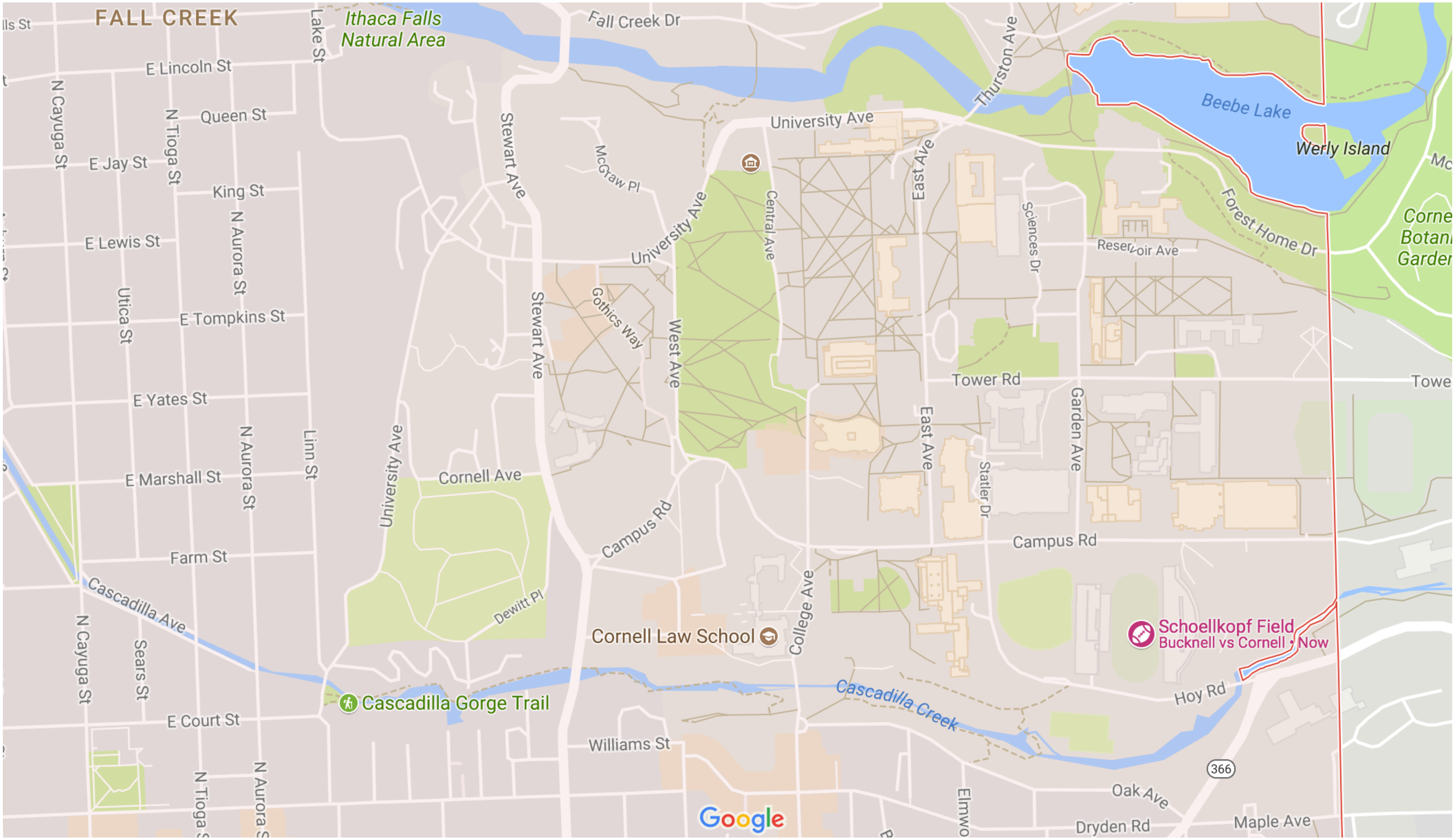
Graphs

- A graph is a data structure
- A graph has
  - a set of vertices
  - a set of edges between vertices
- Graphs are a generalization of trees

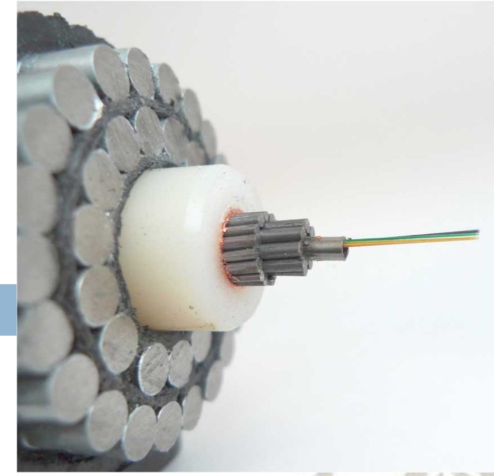


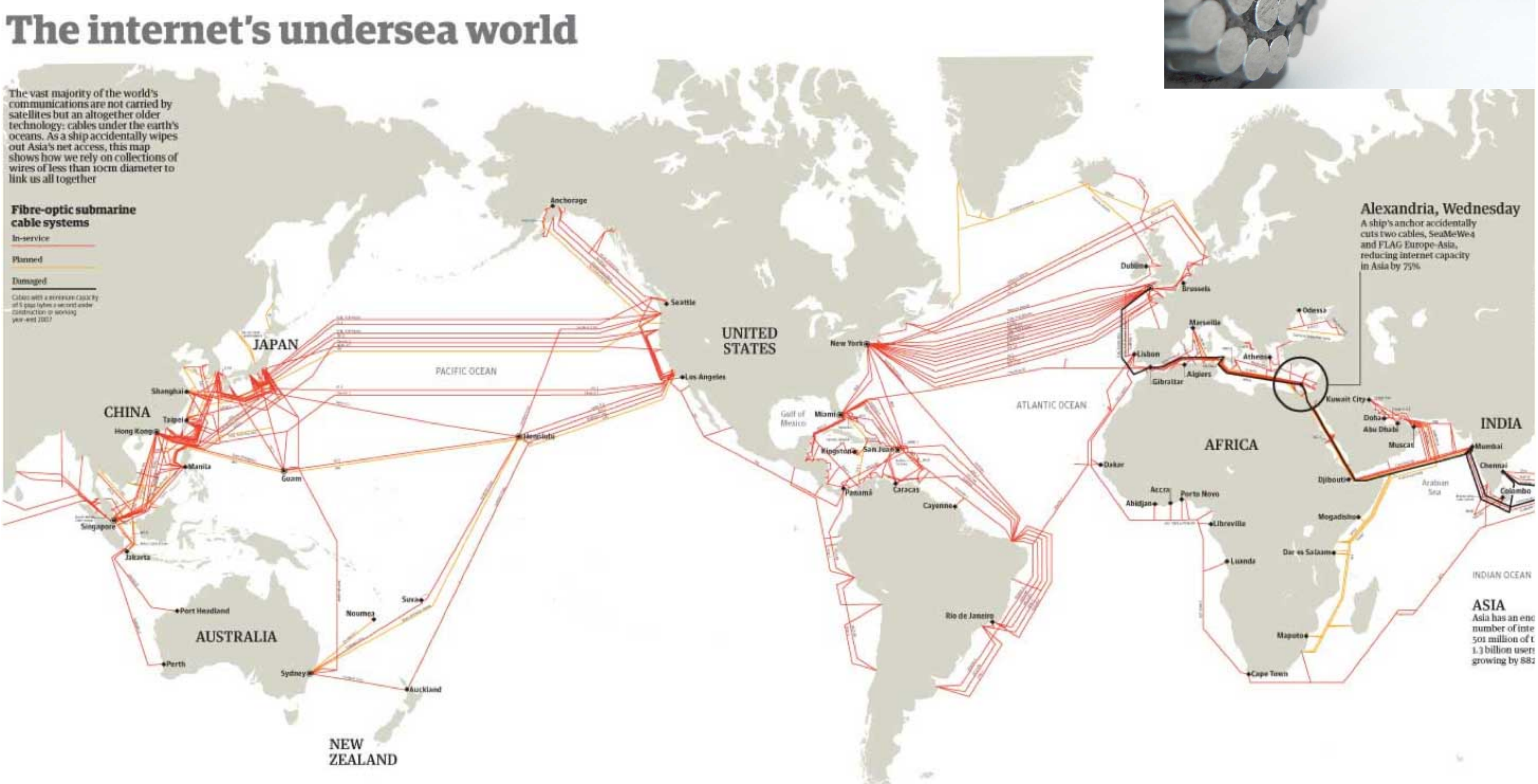
This is a graph

Another transport graph

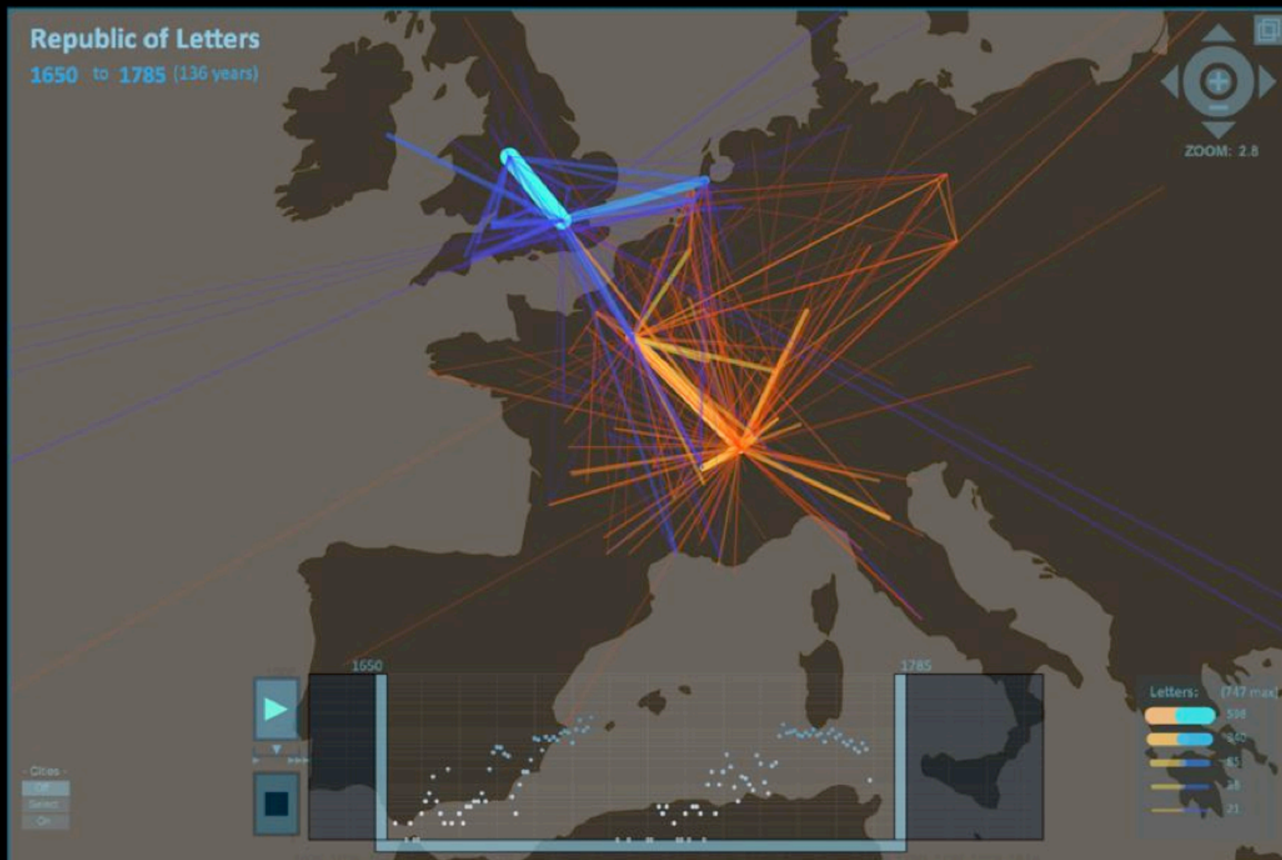


This is a graph



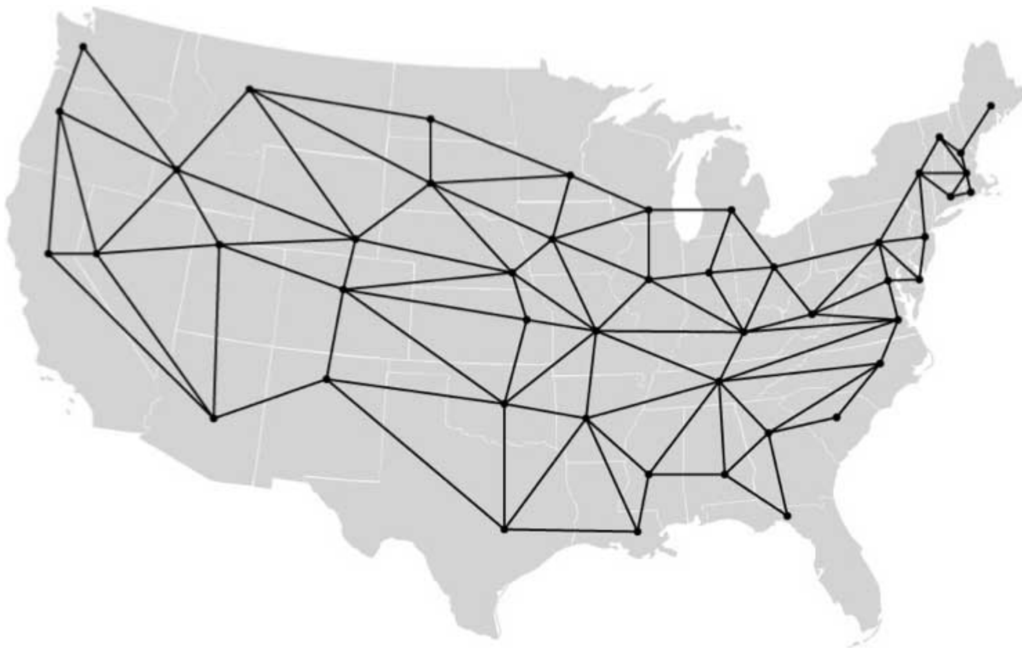


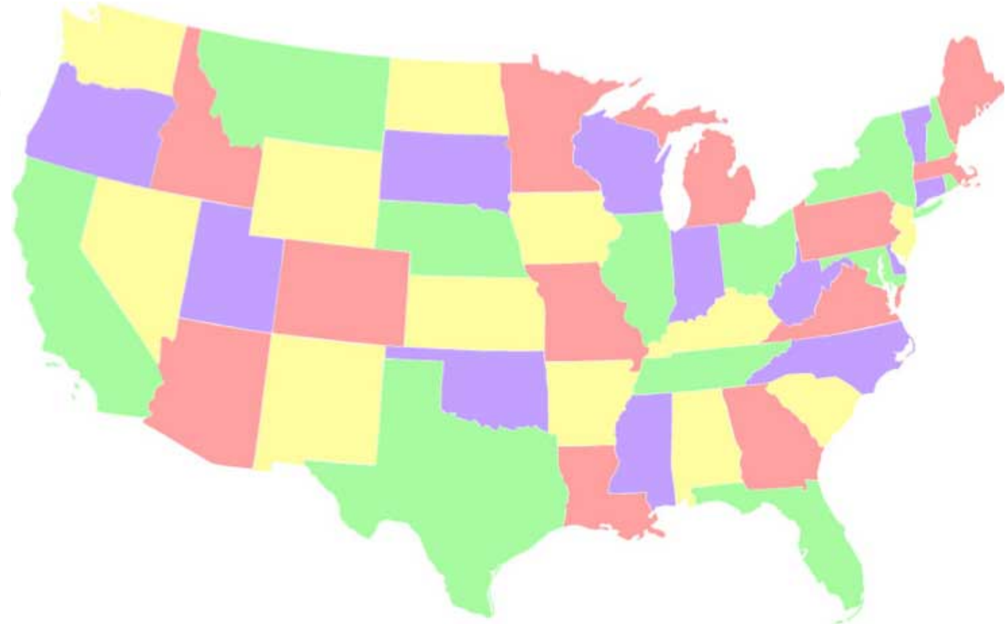
A Social Network Graph



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.

Viewing the map of states as a graph



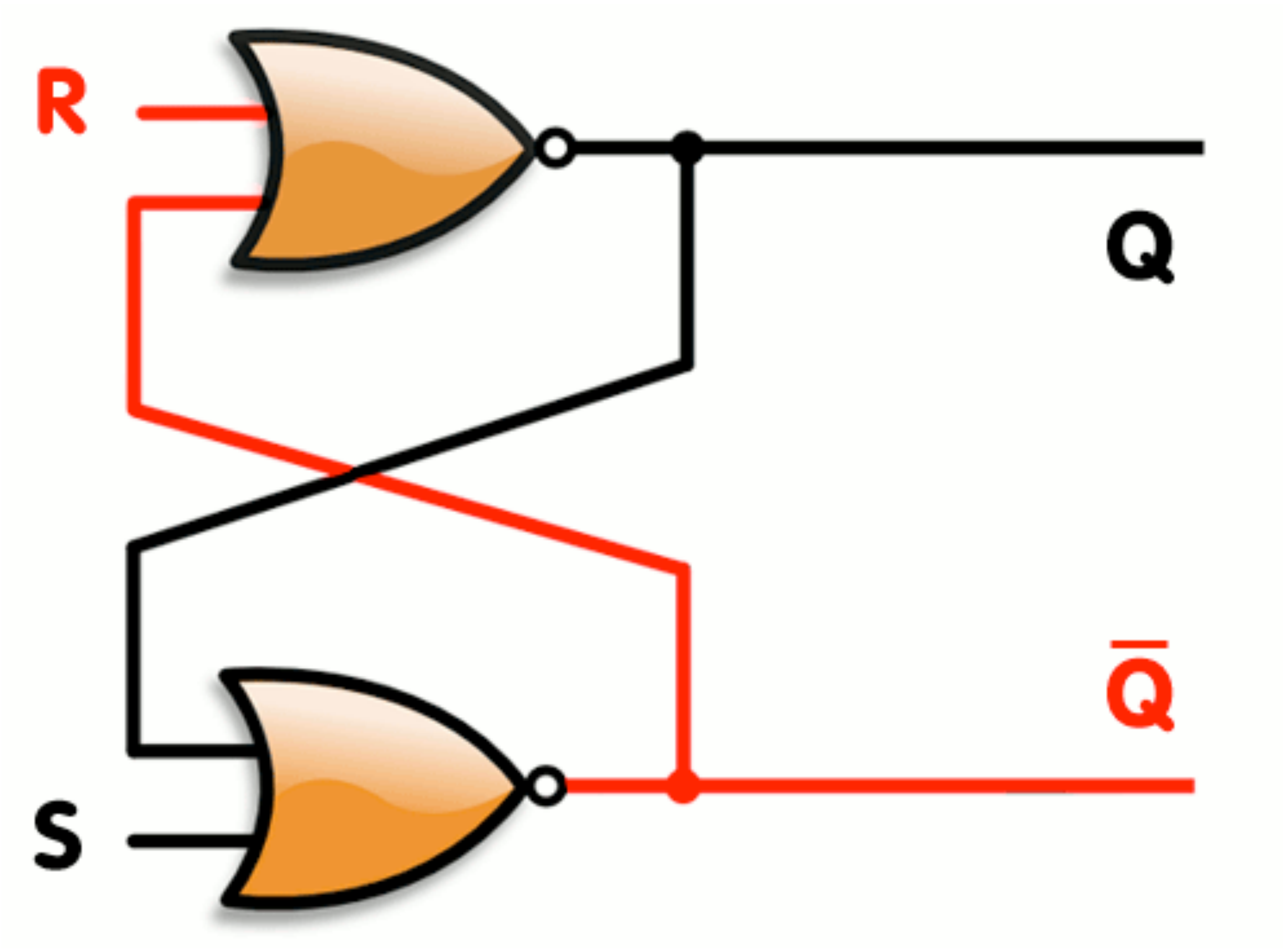


http://www.cs.cmu.edu/~bryant/boolean/maps.html

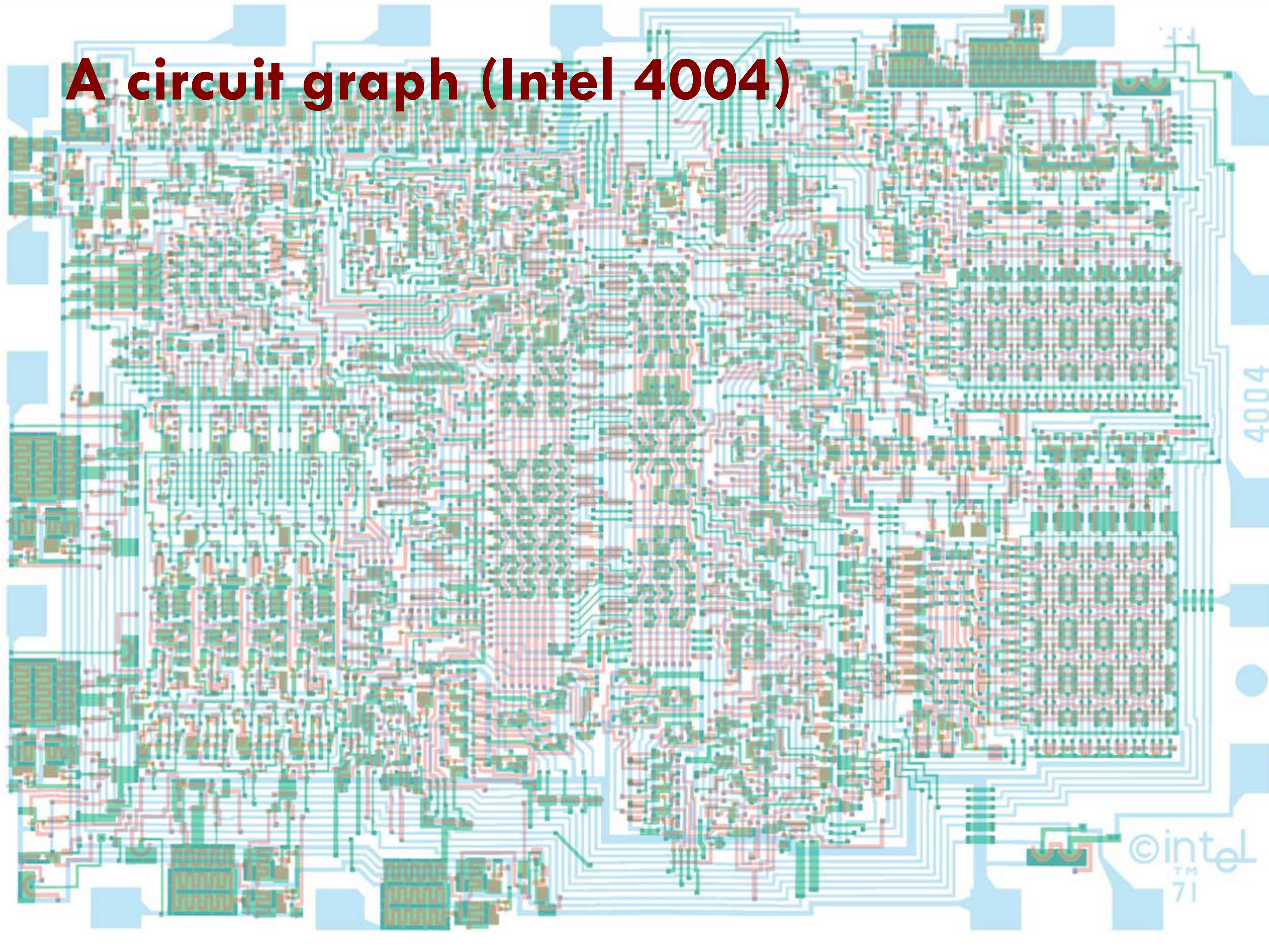
Each state is a point on the graph, and neighboring states are connected by an edge.

Do the same thing for a map of the world showing countries

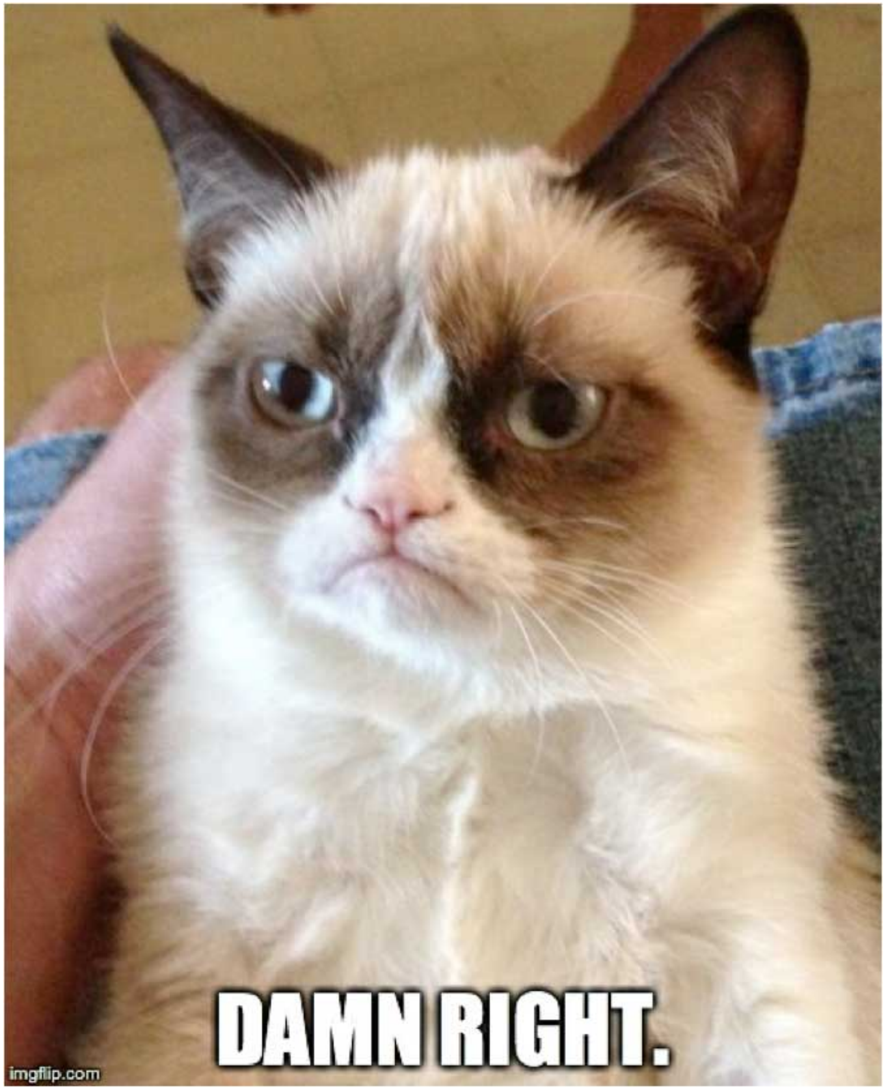
A circuit graph (flip-flop)



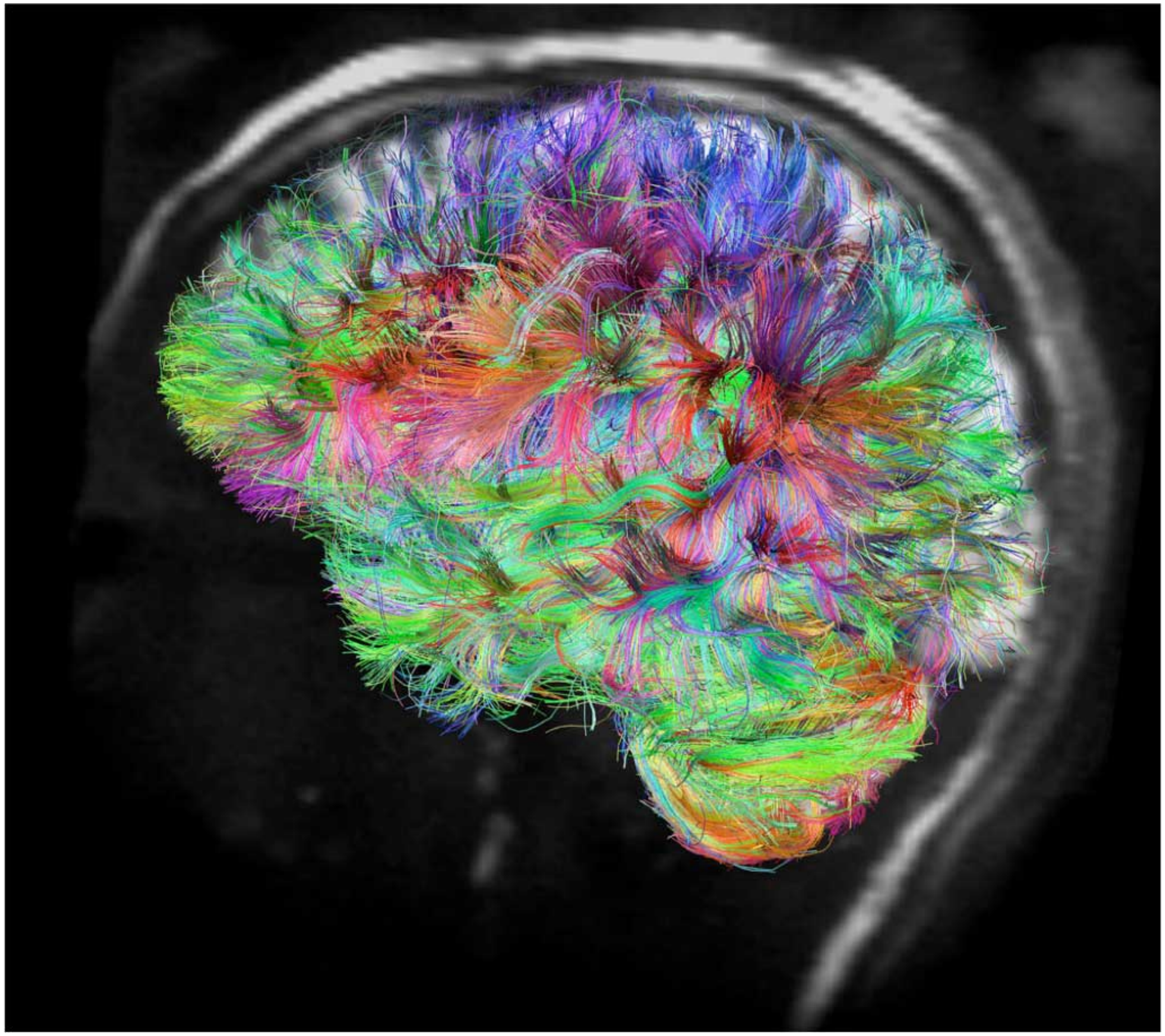
A circuit graph (Intel 4004)



This is not a graph, this is a cat

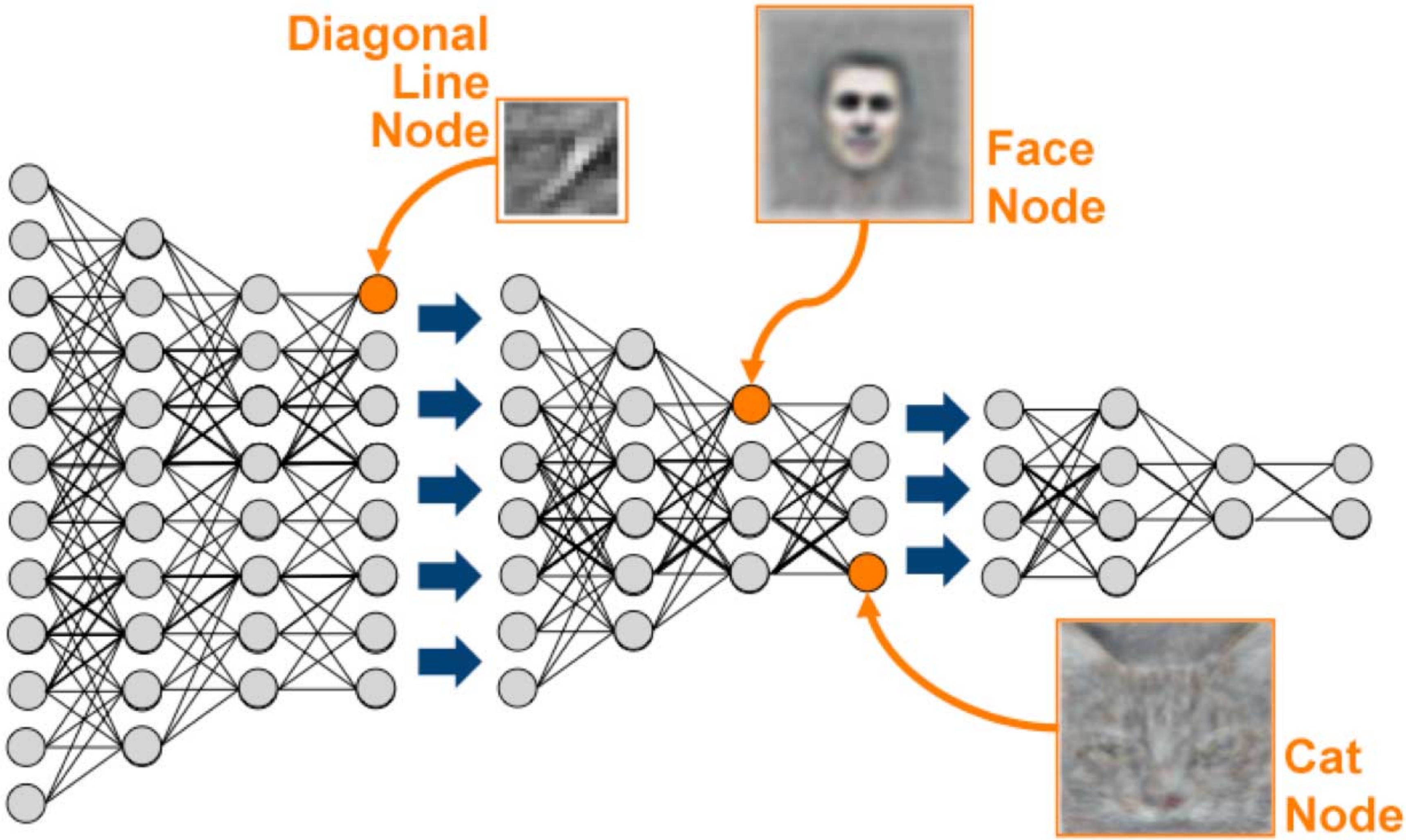


This is a graph

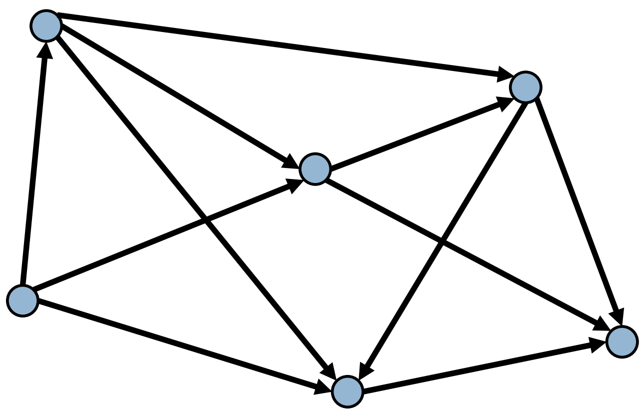


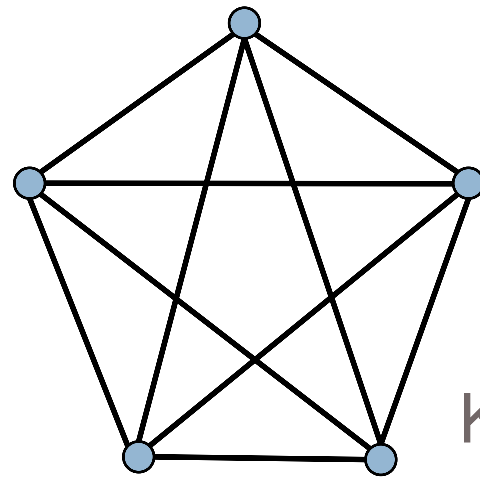
V.J. Wedeen and L.L. Wald, Martinos Center for Biomedical Imaging at MGH

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

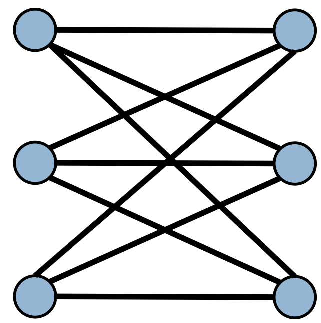


Graphs

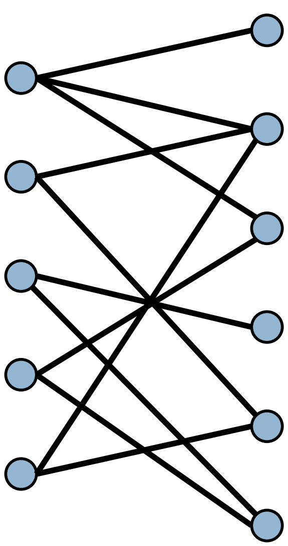


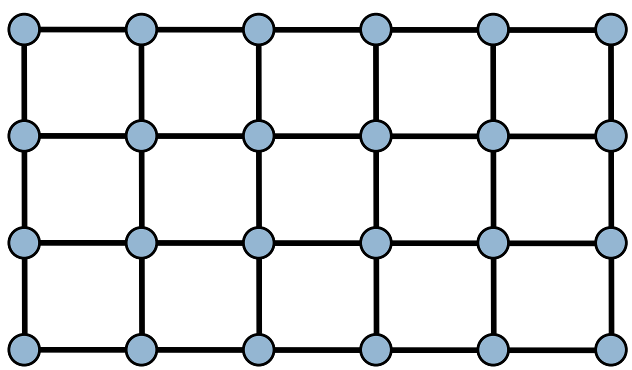


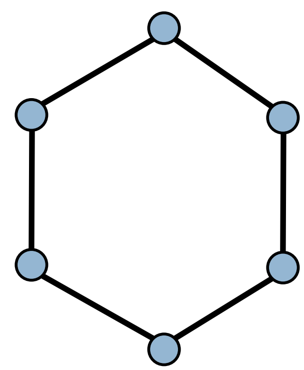
K_5



$K_{3,3}$

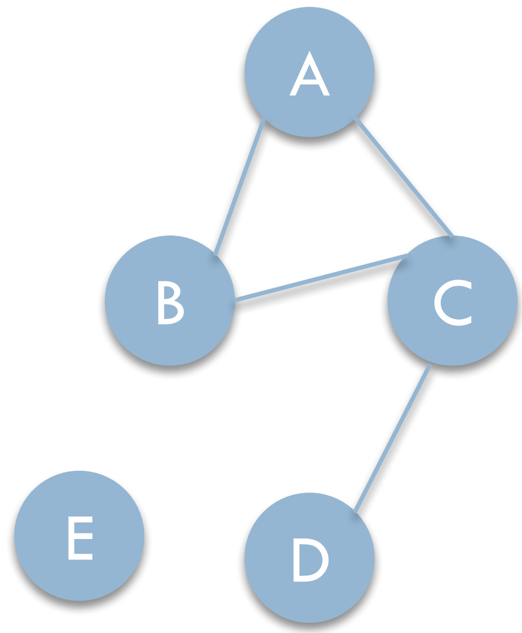






Undirect graphs

- A undirected graph is a pair (V, E) where
  - V is a (finite) set
  - E is a set of pairs (u, v) where $u, v \in V$
    - Often require $u \neq v$ (i.e. no self-loops)



- Element of V is called a vertex or node
- Element of E is called an edge or arc

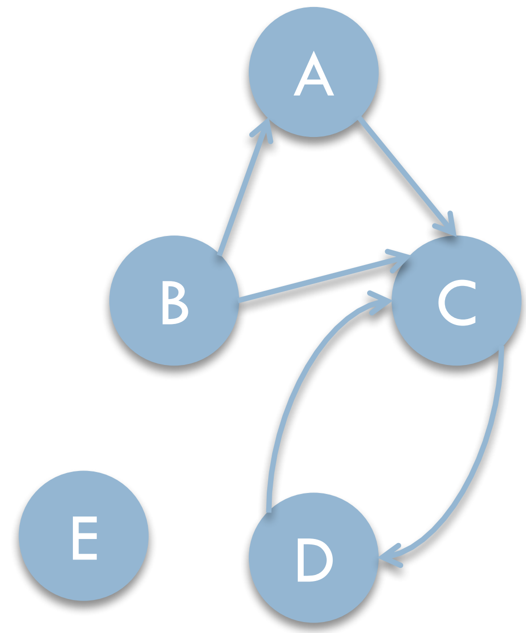
$V = \{A, B, C, D, E\}$
$E = \{(A, B), (A, C), (B, C), (C, D)\}$

$|V| = 5$
$|E| = 4$

- $|V|$ = size of V, often denoted by n
- $|E|$ = size of E, often denoted by m

Directed graphs

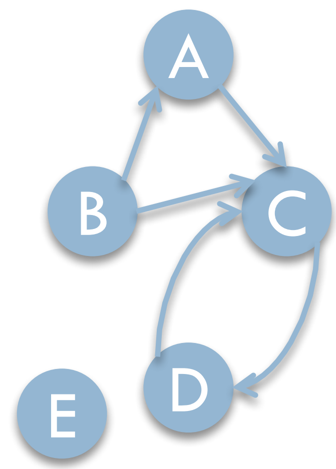
- A directed graph (digraph) is a lot like an undirected graph
  - V is a (finite) set
  - E is a set of ***ordered*** pairs (u, v) where $u,v \in V$
- Every undirected graph can be easily converted to an equivalent directed graph via a simple transformation:
  - Replace every undirected edge with two directed edges in opposite directions
- ... but not vice versa

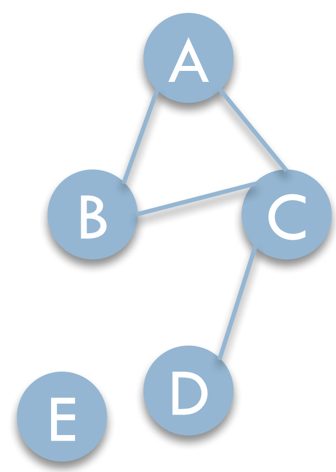


$\boldsymbol{V} = \{A, B, C, D, E\}$
$\boldsymbol{E} = \{(A, C), (B, A), (B, C), (C, D), (D, C)\}$
$|\boldsymbol{V}| = 5$
$|\boldsymbol{E}| = 5$

Graph terminology

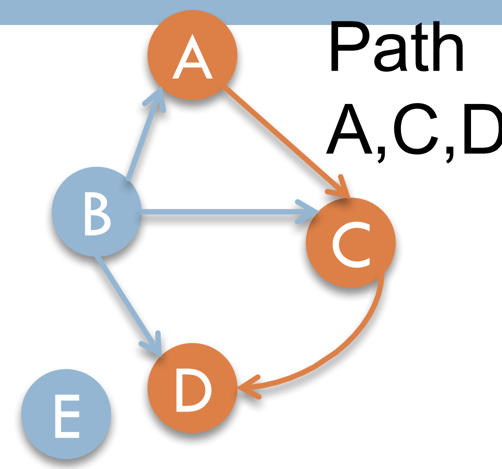
- Vertices u and v are called
  - the source and sink of the directed edge (u, v), respectively
  - the endpoints of (u, v) or $\{u, v\}$
- Two vertices are adjacent if they are connected by an edge
- The outdegree of a vertex u in a directed graph is the number of edges for which u is the source
- The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



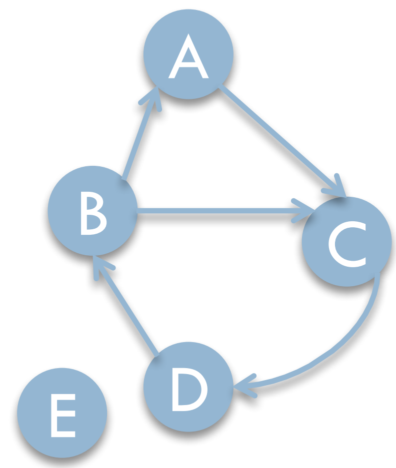


More graph terminology

- A path is a sequence $v_0, v_1, v_2, \ldots, v_p$ of vertices such that for $0 \le i < p$,
  - $(v_i, v_{i+1}) \in E$ 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
- A path is simple if it doesn't repeat any vertices
- A cycle is a path $v_0, v_1, v_2, \ldots, v_p$ 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

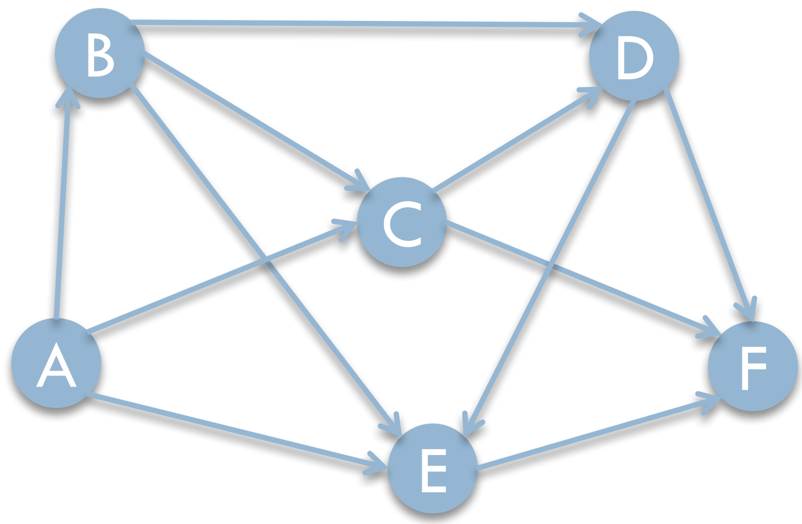


DAG



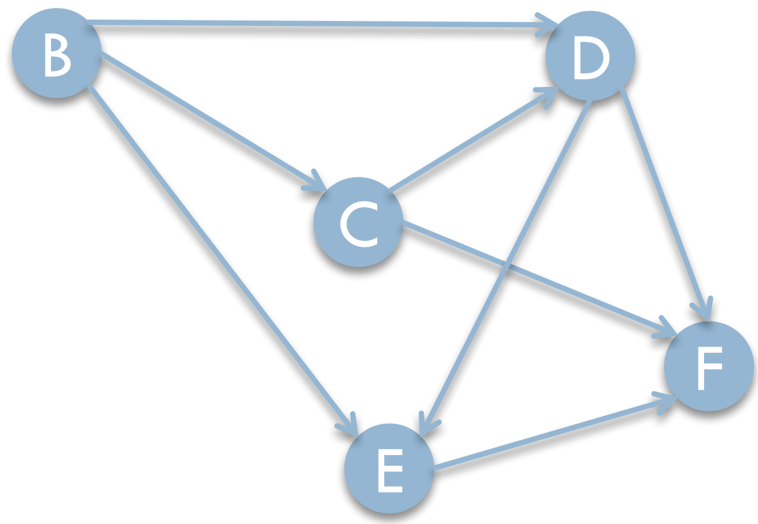
Not a DAG

Is this a DAG?



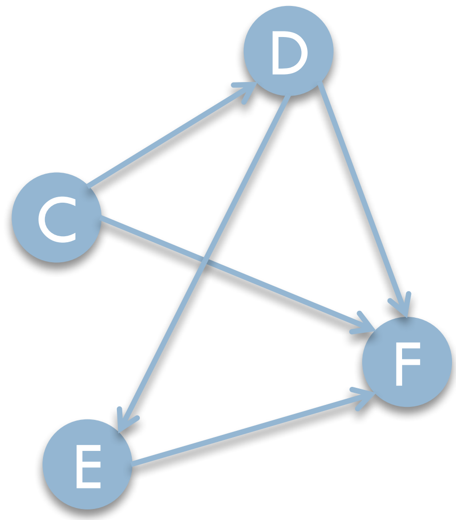
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?



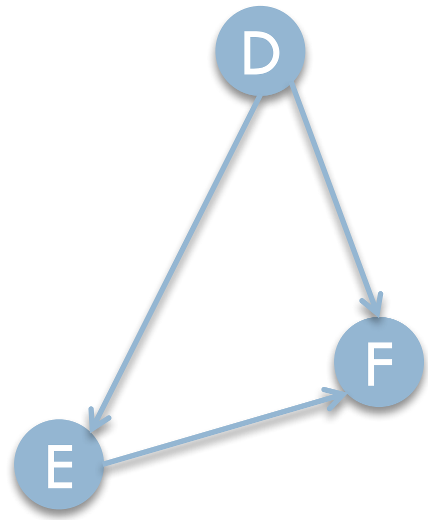
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?



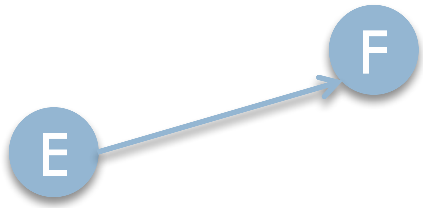
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

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?



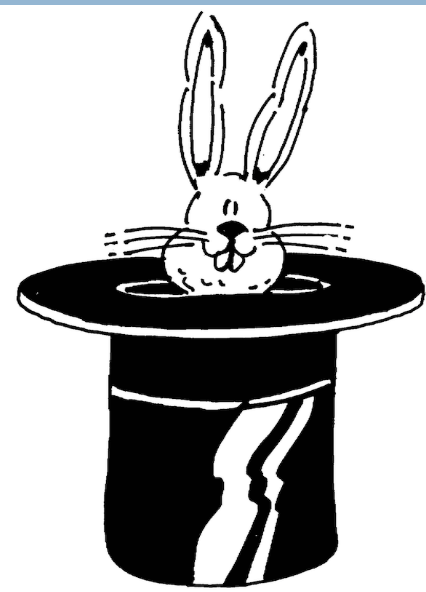
- Intuition:
  - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an *algorithm*
  - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

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

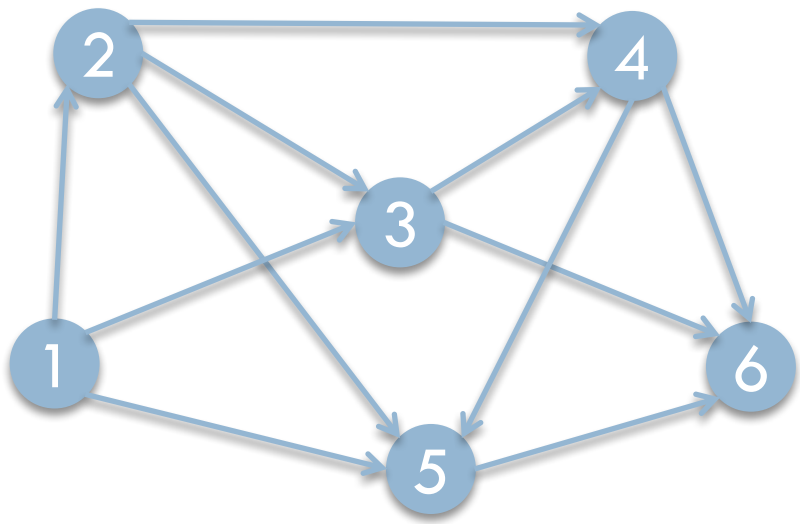
Is this a DAG?

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

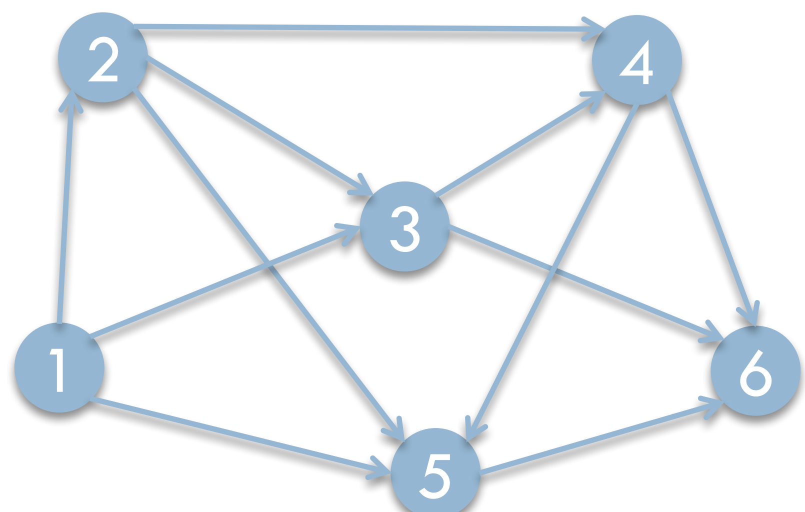


- We just computed a topological sort of the DAG
  - This is a numbering of the vertices such that all edges go from lower- to higher-numbered vertices
  - Useful in job scheduling with precedence constraints

Topological sort

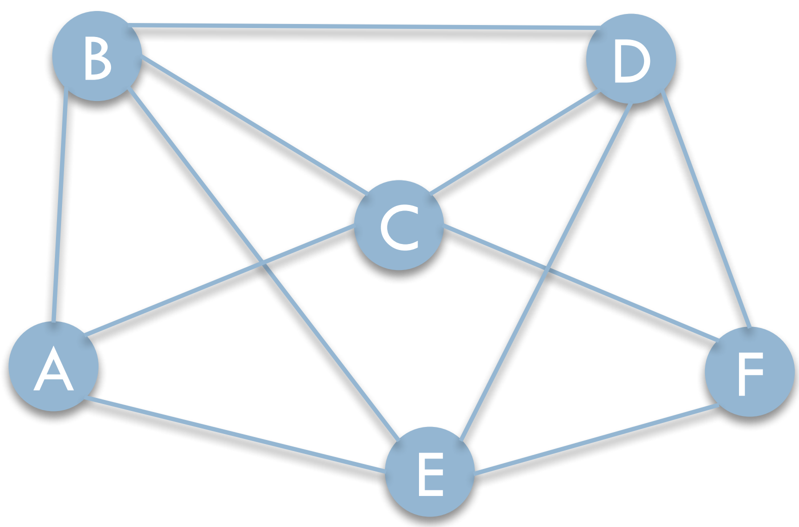
```
k= 0;
// inv: k nodes have been given numbers in 1..k in such a way that
        if n1 <= n2, there is no edge from n2 to n1.
while (there is a node of in-degree 0) {
    Let n be a node of in-degree 0;
    Give it number k;
    Delete n and all edges leaving it from the graph.
    k= k+1;
}
```

1. Abstract algorithm
2. Don’t really want to change the graph.
3. Will have to invent data structures to make it efficient.



Graph coloring

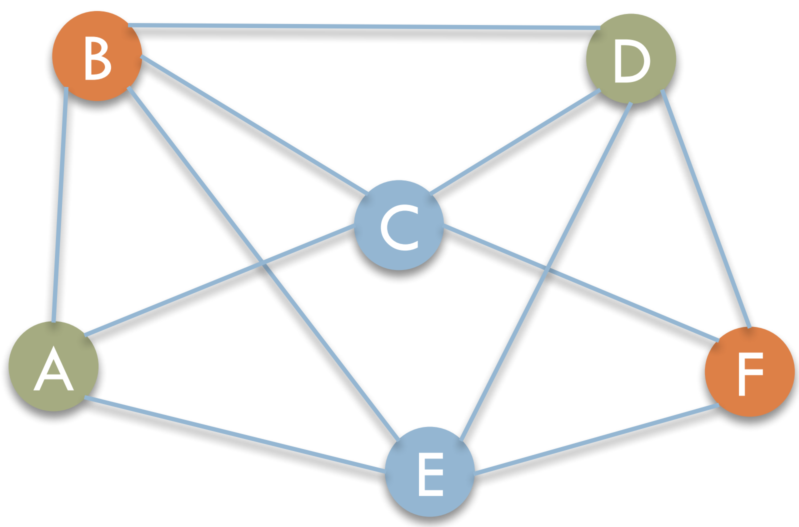
- A coloring of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color



- How many colors are needed to color this graph?

Graph coloring

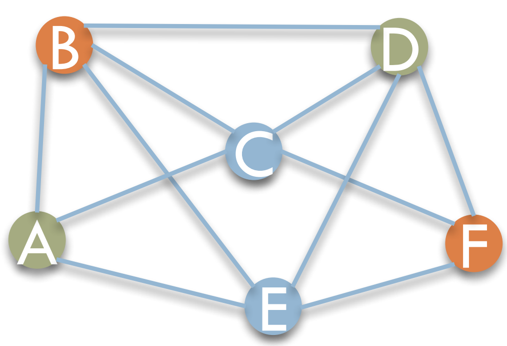
- A coloring of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color



- How many colors are needed to color this graph?

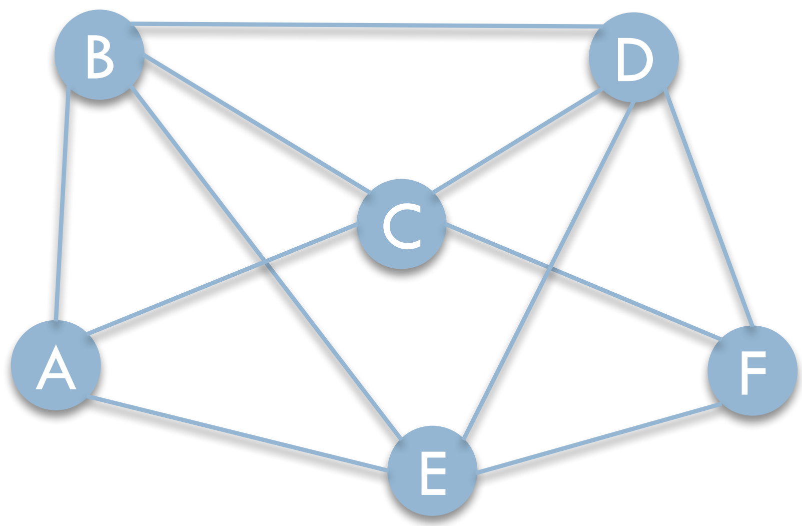
An application of coloring

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



Planarity

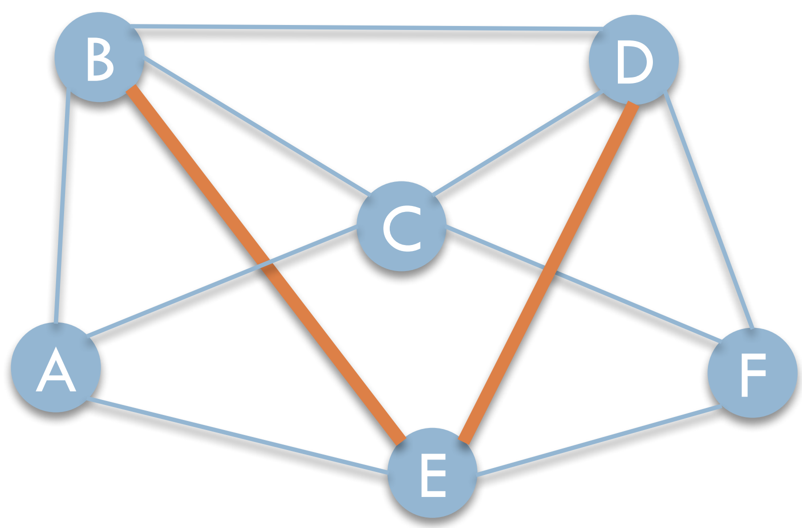
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?

Planarity

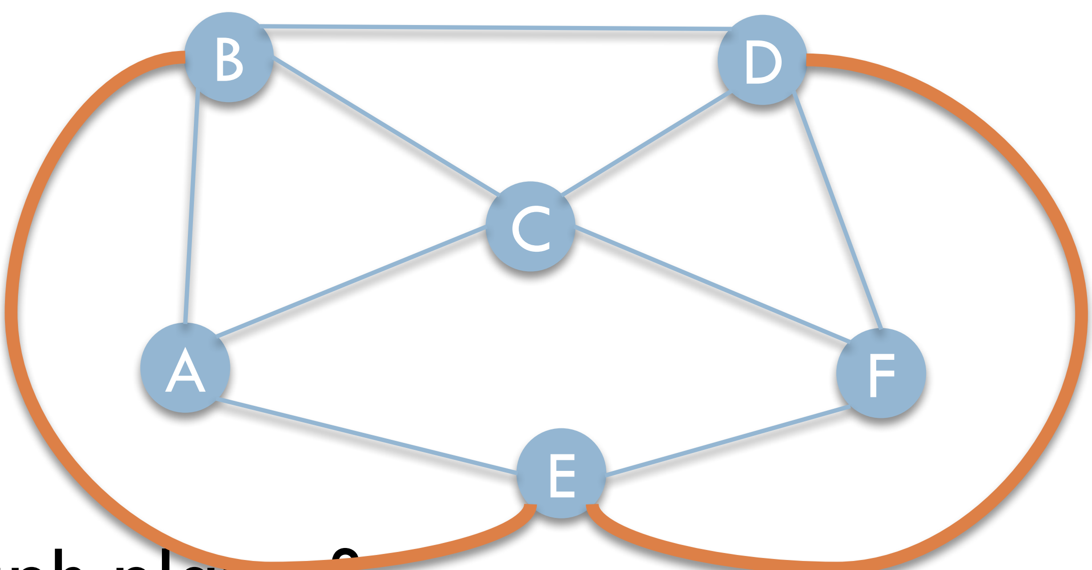
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?
  - Yes!

Planarity

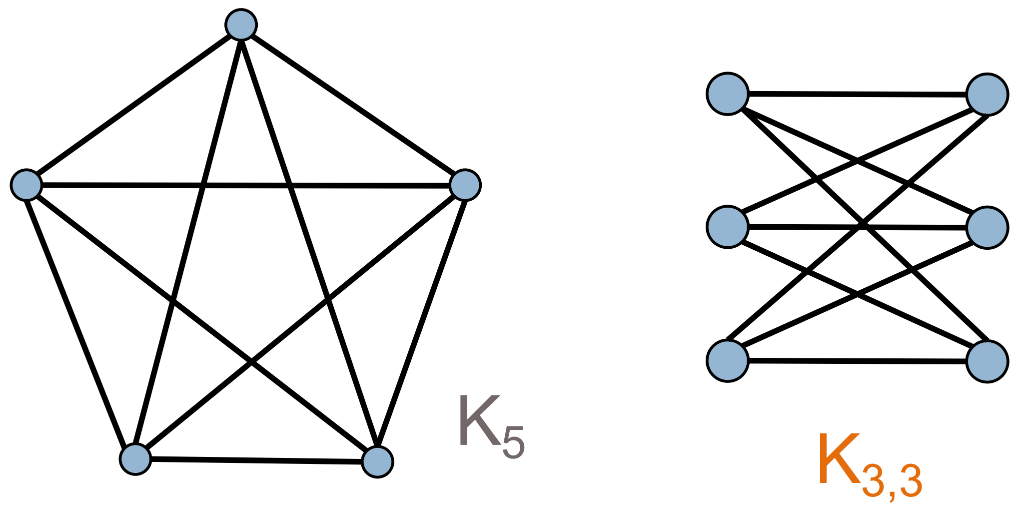
- A graph is planar if it **can** be drawn in the plane without any edges crossing



- 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)

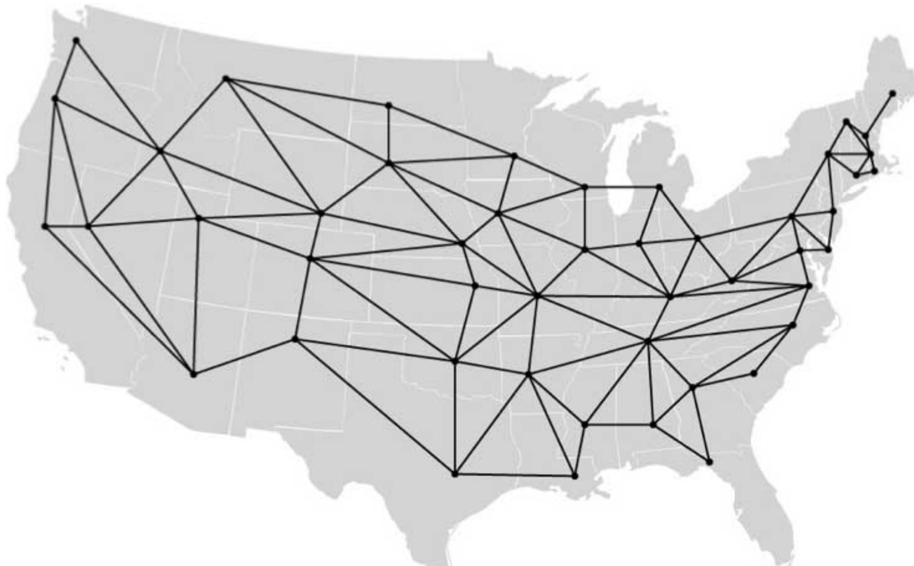
Detecting Planarity

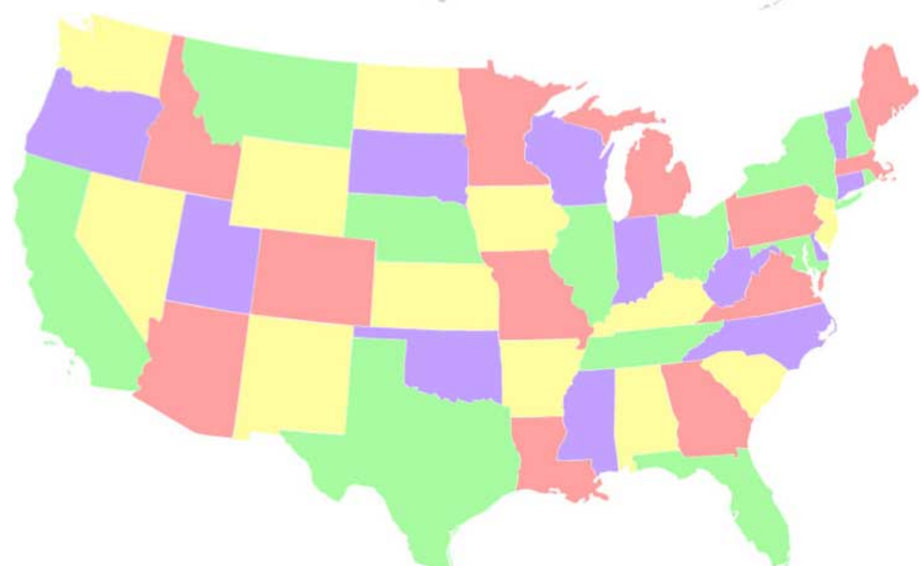
In the early 1970's, Cornell Prof John Hopcroft spent a sabbatical at Stanford and worked with PhD student Bob Tarjan. They developed the first linear-time algorithm for testing whether a graph was planar. They later received the ACM Turing Award for their work on algorithms.

Tarjan was hired at one point in the 1970's into our department, but the Ithaca weather was too depressing for him and he left for Princeton.

Coloring a graph

- How many colors are needed to color the states so that no two adjacent states have the same color?
- Asked since 1852
- 1879: Kemp publishes a proof that only 4 colors are needed!
- 1880: Julius Peterson finds a flaw in Kemp's proof…





Four Color Theorem

Every planar graph is 4-colorable [Appel & Haken, 1976]

The proof rested on checking that 1,936 special graphs had a certain property.

They used a computer to check that those 1, 936 graphs had that property!

Basically the first time a computer was needed to check something. Caused a lot of controversy.

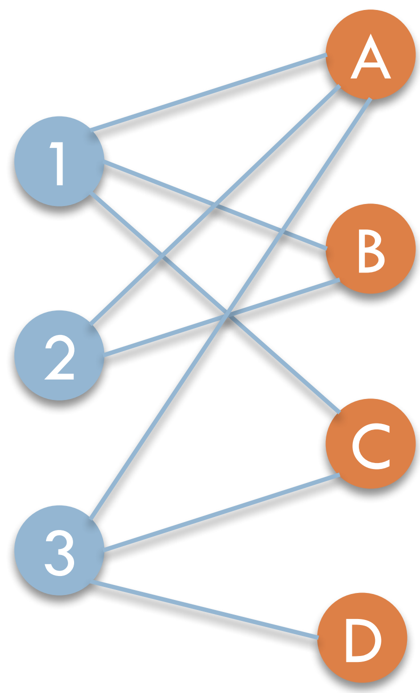
Gries looked at their computer program, a recursive program written in the assembly language of the IBM 7090 computer, and found an error, which was safe (it said something didn't have the property when it did) and could be fixed. Others did the same.

Since then, there have been improvements. And a formal proof has even been done in the Coq proof system

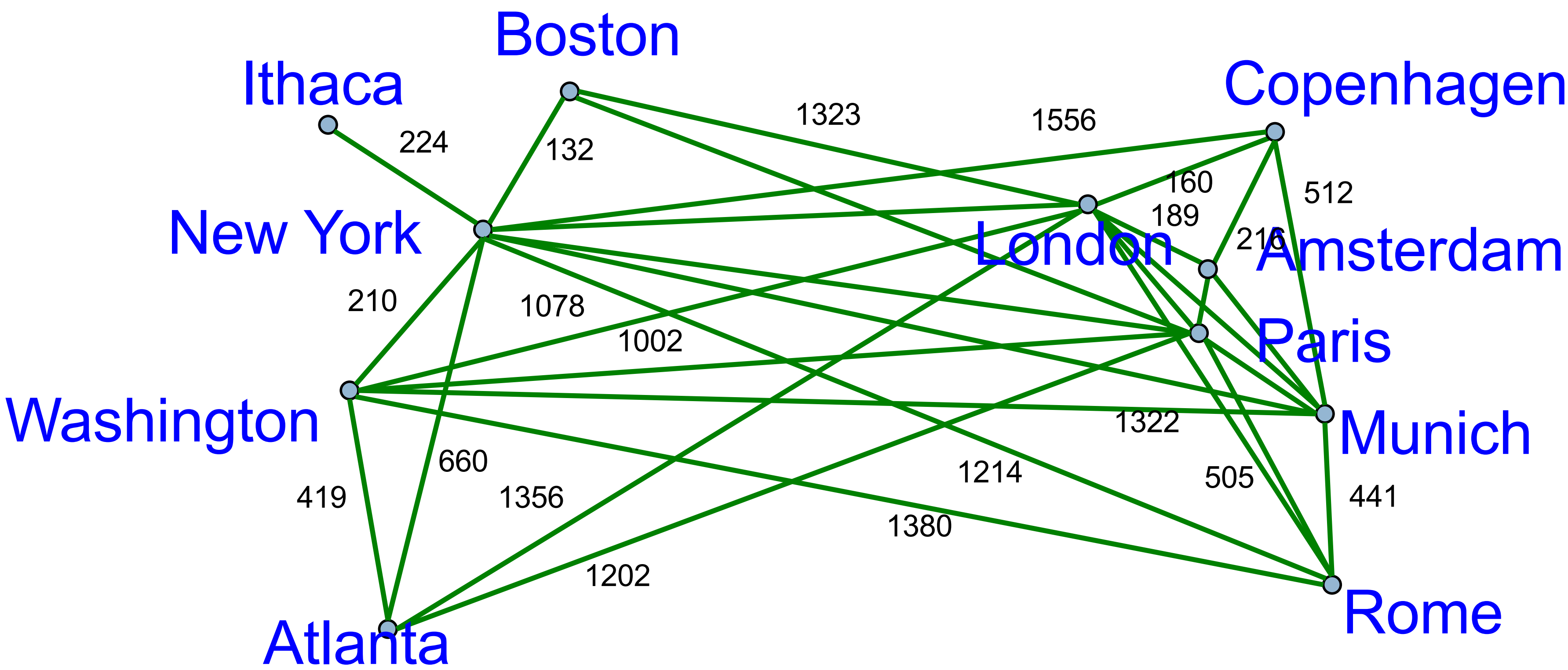
Bipartite graphs

- 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

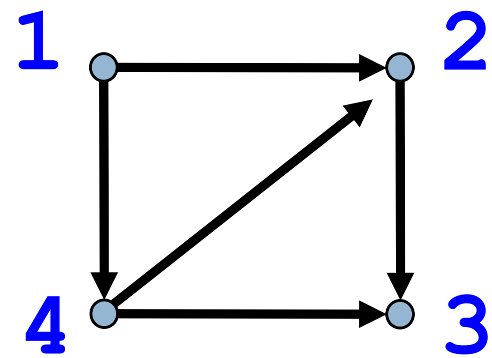


Traveling salesperson

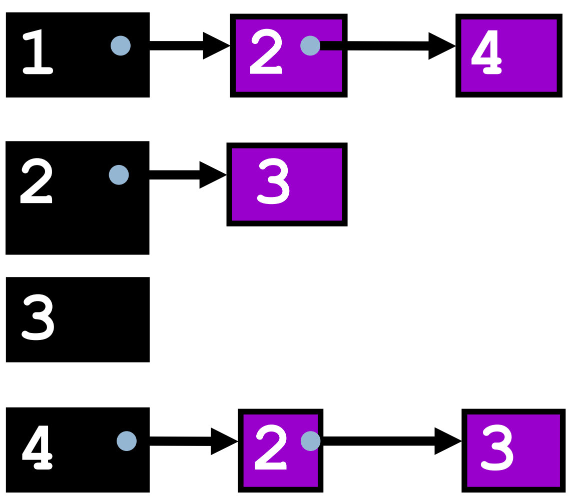


Find a path of minimum distance that visits every city

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?

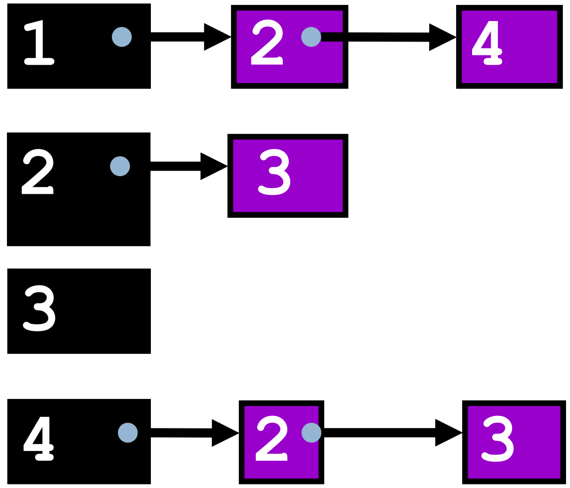
- n = number of vertices
- m = number of edges
- $d(u)$ = degree of u = no. of edges leaving u

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

- Adjacency Matrix
  - Uses space $O(n^2)$
  - Enumerate all edges in time $O(n^2)$
  - Answer "Is there an edge from u to v?" in $O(1)$ time
  - Better for dense graphs (lots of edges)

Adjacency matrix or adjacency list?

- n = number of vertices
- e = number of edges
- $d(u)$ = degree of u = no. edges leaving u



- Adjacency List
  - Uses space $O(e + n)$
  - Enumerate all edges in time $O(e + n)$
  - 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
  - Jarnik/Prim/Dijkstra algorithm
  - Kruskal's algorithm