

Graphs - I

CS 2110, Spring 2017

Leonhard Euler. 1736
7 Bridges of Königsberg

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

This is

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

And so is this

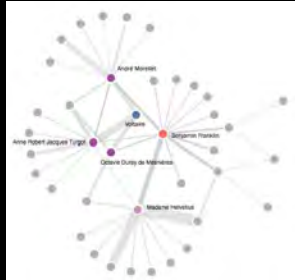
The internet's undersea world

This carries Internet traffic across the oceans

An older social graph

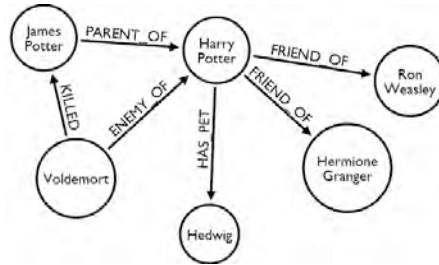
Locke's (blue) and Voltaire's (yellow) correspondence.
Only letters for which complete location information is available are shown.
Data courtesy the Electronic Enlightenment Project, University of Oxford.

An older social graph



Voltaire and Benjamin Franklin

A fictional social graph



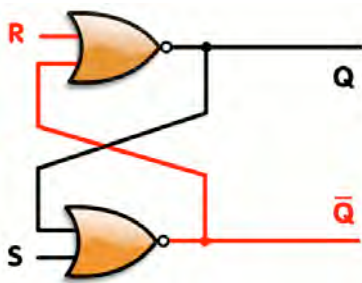
A transport graph: NY subway system



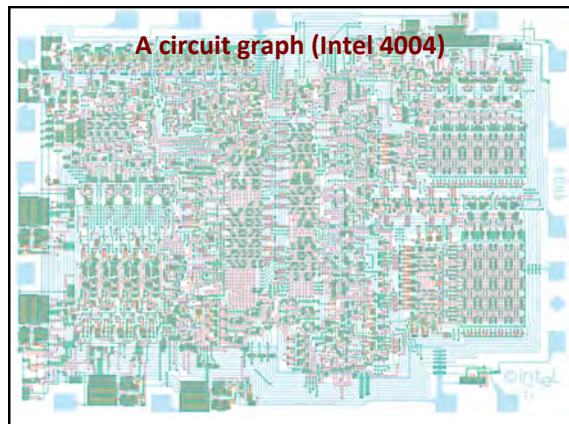
Another transport graph

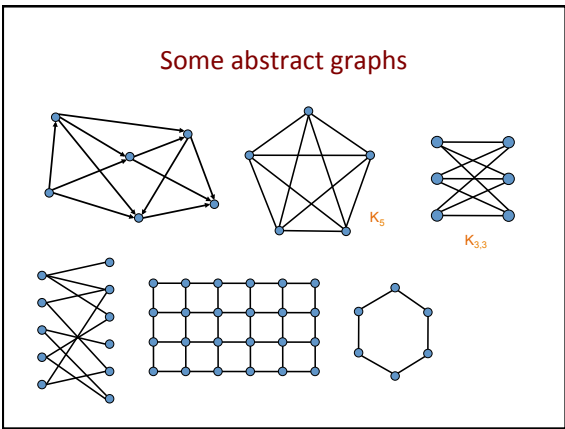
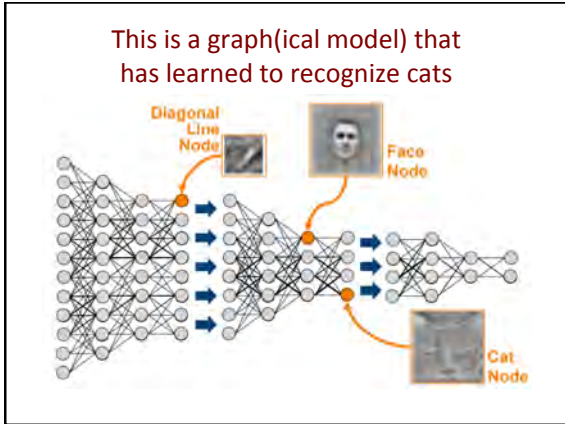
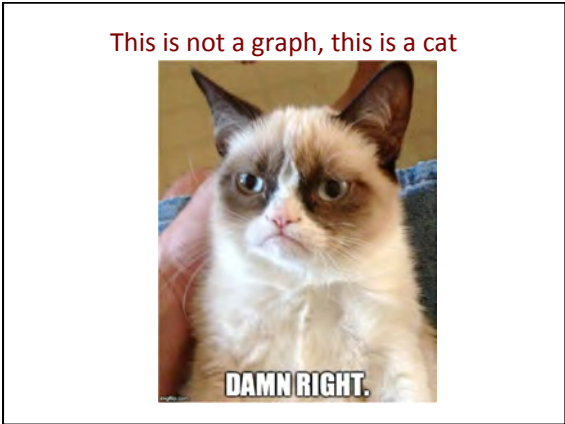


A circuit graph (flip-flop)



A circuit graph (Intel 4004)





- 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 (finite) 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 by n
- $|E|$ = size of E , often denoted by m

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

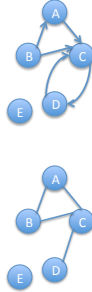
Undirected Graphs

- 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 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

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

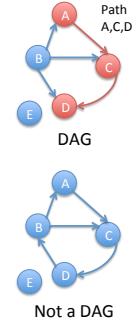
Graph terminology

- 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

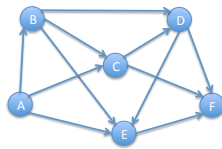


More graph terminology

- A **path** is a sequence $v_0, v_1, v_2, \dots, v_p$ of vertices such that for $0 \leq i < p$,
 - $(v_i, v_{i+1}) \in E$ 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 $v_0, v_1, v_2, \dots, v_p$ 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 **directed acyclic graph** is called a **DAG**

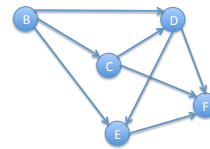


Is this a DAG?



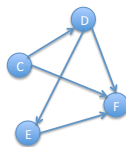
- **Intuition:**
 - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an **algorithm**
 - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is this a DAG?



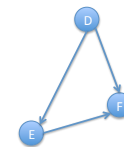
- **Intuition:**
 - If it's a DAG, there must be a vertex with indegree zero
- This idea leads to an **algorithm**
 - A digraph is a DAG if and only if we can iteratively delete indegree-0 vertices until the graph disappears

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

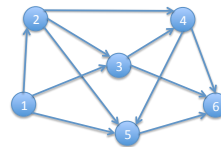
Is this a DAG?

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

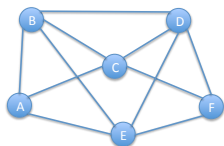
Topological sort



- 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

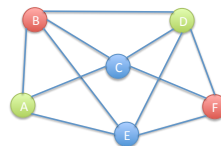
- A **coloring** of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color



- How many colors are needed to color this graph?

Graph coloring

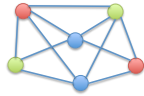
- A **coloring** of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color



- How many colors are needed to color this graph?

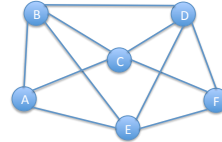
An application of coloring

- Vertices are 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

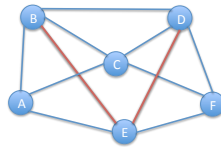
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?

Planarity

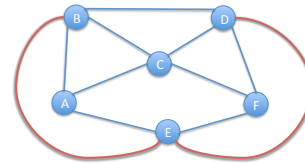
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?
– Yes!

Planarity

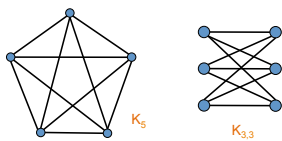
- A graph is planar if it **can** be drawn in the plane without any edges crossing



- 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 countries so that no two adjacent countries have the same color?

Question asked as early as 1852.

1879. Kemp publishes a theorem that only 4 colors are needed!

1880. Julius Peterson finds a flaw in the 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

Another 4-colored planar graph

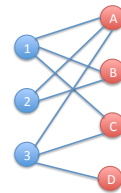


<http://www.cs.cmu.edu/~bryant/boolean/maps.html>

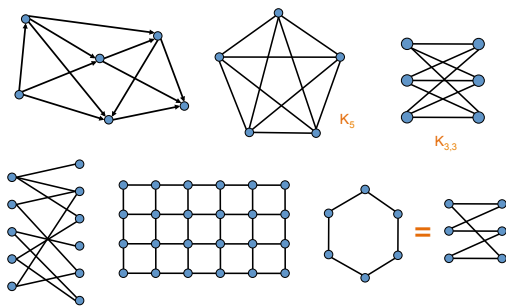
Bipartite graphs

- 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

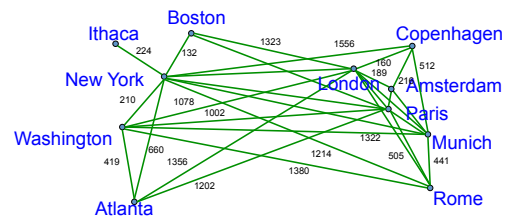
- The following are equivalent
 - G is bipartite
 - G is 2-colorable
 - G has no cycles of odd length



Some abstract graphs

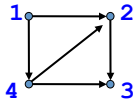


Traveling salesperson

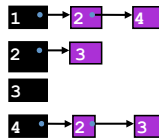


Find a path of minimum distance that visits every city

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

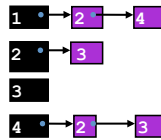
Adjacency matrix or adjacency List?

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

	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
- 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