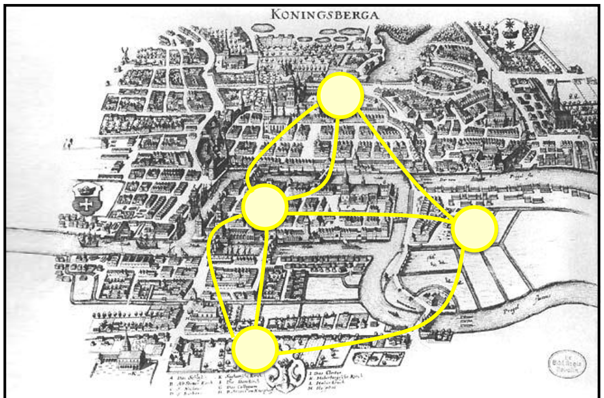


GRAPHS

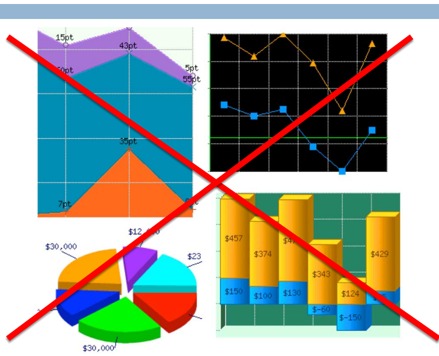
Lecture 17
CS2110

Announcements

2

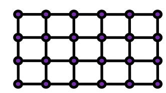
- A5 Heaps Due October 27
- Prelim 2 in ~3 weeks: Thursday Nov 15
- A4 being graded right now
- Mid-Semester College Transitions Survey on Piazza

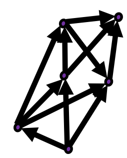
These aren't the graphs we're looking for

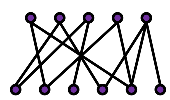


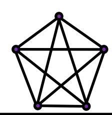
Graphs

- 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



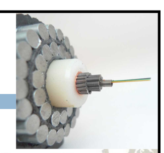


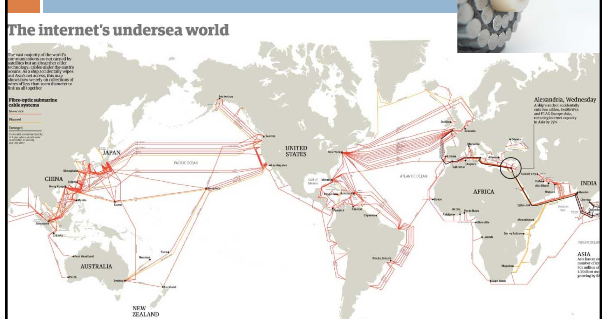




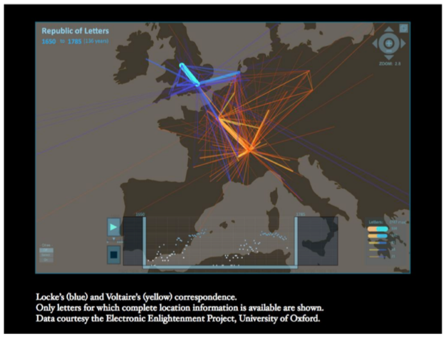
This is a graph

This is a graph

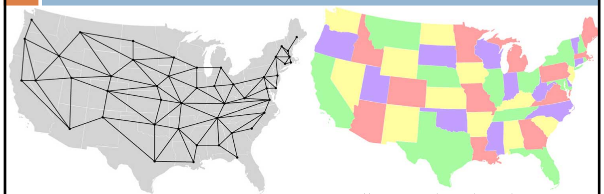




A Social Network Graph



Viewing the map of states as a graph

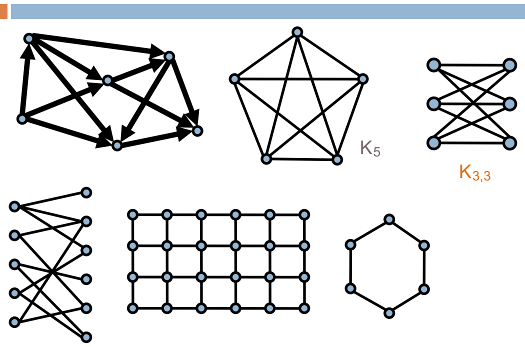


http://www.cs.cmu.edu/~bryant/boolean/maps.html

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

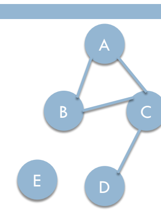
Do the same thing for a map of the world showing countries

Graphs



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



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

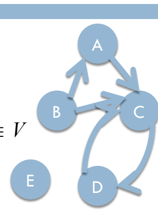
$|V| = 5$
$|E| = 4$

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 $u,v \in V$

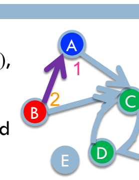


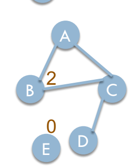
- Every undirected graph can be easily converted to an equivalent directed graph via a simple transformation:
  - Replace every undirected edge with two directed edges in opposite directions
- … but not vice versa

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

Graph terminology

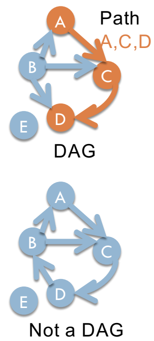
- Vertices u and v are called
  - the source and sink of the directed edge (u, v), respectively
  - the **endpoints** of (u, v) or $\{u, v\}$
- Two vertices are adjacent if they are connected by an edge
- The outdegree of a vertex u in a directed graph is the number of edges for which u is the source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



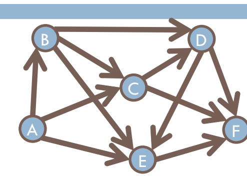


More graph terminology

- 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



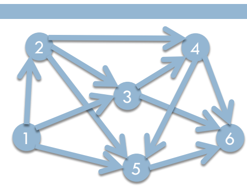
Is this a DAG?



Yes!
It is 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

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

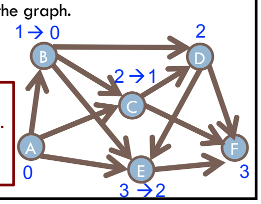
Topological sort

```
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;
}
```

k= 0 → 1

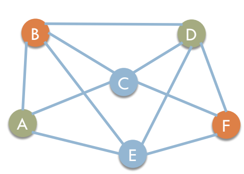
A 0
B 1
C
D
E
F

1. Abstract algorithm
2. Don't really want to change the graph.
3. Will have to use some data structures to support this efficiently.



Graph coloring

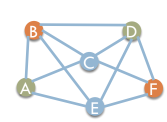
- A coloring of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color



- How many colors are needed to color this graph?

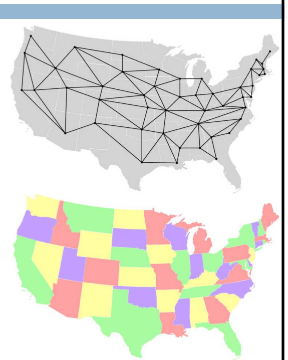
An application of coloring

- 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



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!

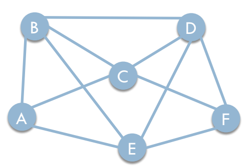
Basically the first time a computer was needed to check something. Caused a lot of controversy.

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.

Planarity

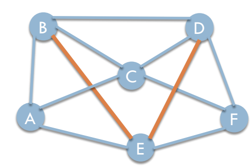
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?

Planarity

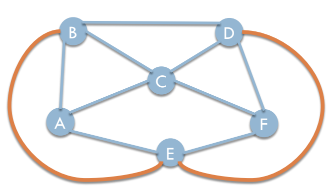
- A graph is planar if it can be drawn in the plane without any edges crossing



- Is this graph planar?
  - Yes!

Planarity

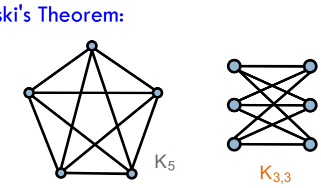
- A graph is planar if it can be drawn in the plane without any edges crossing



- 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)

John Hopcroft & Robert Tarjan

25

Turing Award in 1986 "for fundamental achievements in the design and analysis of algorithms and data structures"

One of their fundamental achievements was a linear-time algorithm for determining whether a graph is planar.

David Gries & Jinyun Xue

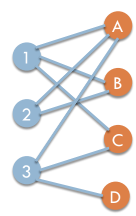
26

Tech Report, 1988

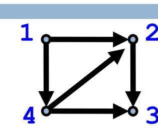
Abstract: We give a rigorous, yet, we hope, readable, presentation of the Hopcroft-Tarjan linear algorithm for testing the planarity of a graph, using more modern principles and techniques for developing and presenting algorithms that have been developed in the past 10-12 years (their algorithm appeared in the early 1970's). Our algorithm not only tests planarity but also constructs a planar embedding, and in a fairly straightforward manner. The paper concludes with a short discussion of the advantages of our approach.

Bipartite graphs

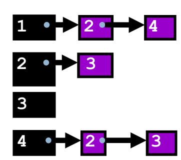
- A directed or undirected graph is bipartite if the vertices can be partitioned into two sets such that no edge connects two vertices in the same set
- The following are equivalent
  - G is bipartite
  - G is 2-colorable
  - G has no cycles of odd length



Representations of graphs



Adjacency List



Adjacency Matrix

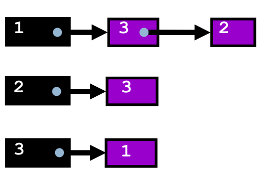
	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	0	0	0
4	0	1	1	0

Graph Quiz

Which of the following two graphs are DAGs?

Directed Acyclic Graph

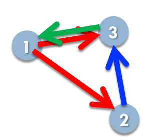
Graph 1:



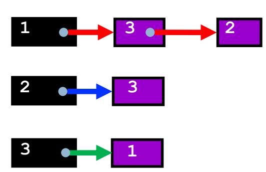
Graph 2:

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

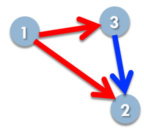
Graph 1



Is this a DAG?



Graph 2



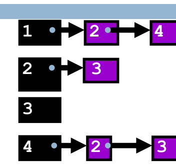
Is this a DAG?

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

Adjacency Matrix vs. Adjacency List

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

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



Matrix	Property	List
$O(v^2)$	Space	$O(v + e)$
$O(v^2)$	Time to enumerate all edges	$O(v + e)$
$O(1)$	Time to answer "Is there an edge from $u1$ to $u2$?"	$O(d(u1))$
dense graphs	better for	sparse graphs

Graph algorithms

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