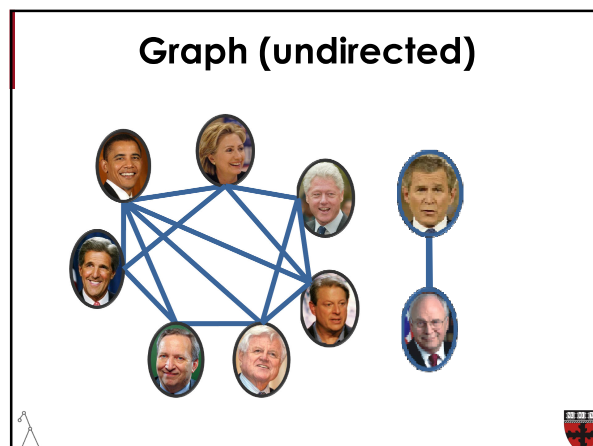
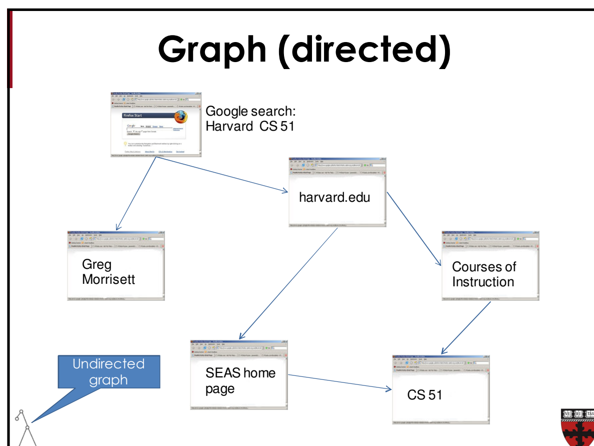
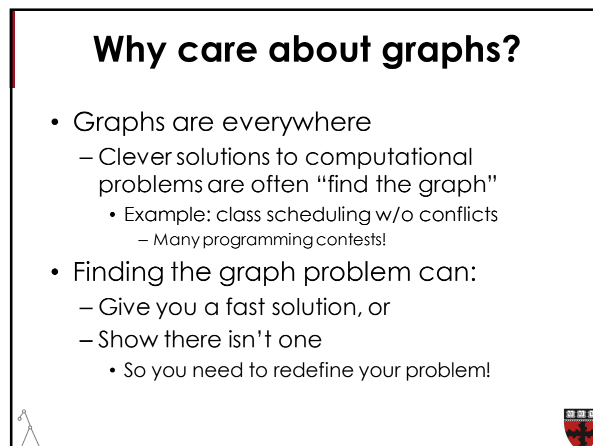
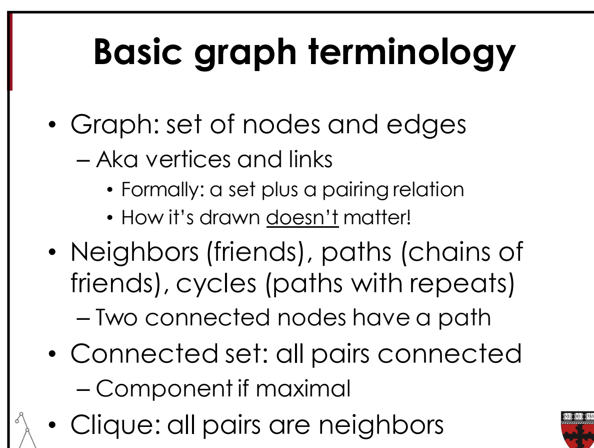
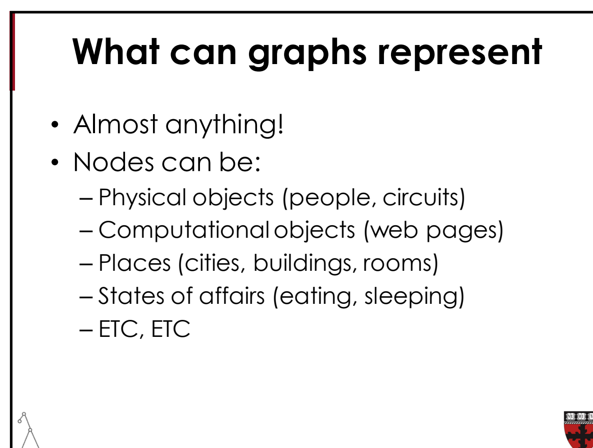
COMPUTER SCIENCE 51
Spring 2009
cs51.seas.harvard.edu
Prof. Greg Morrisett
Prof. Ramin Zabih
computer
science 51


Graph (undirected)


Graph (directed)
Google search:
Harvard CS 51
harvard.edu
Greg
Morrisett
Courses of
Instruction
Undirected
graph
SEAS home
page
CS 51


Why care about graphs?
• Graphs are everywhere
– Clever solutions to computational problems are often "find the graph"
• Example: class scheduling w/o conflicts
– Many programming contests!
• Finding the graph problem can:
– Give you a fast solution, or
– Show there isn't one
• So you need to redefine your problem!


Basic graph terminology
• Graph: set of nodes and edges
– Aka vertices and links
• Formally: a set plus a pairing relation
• How it's drawn doesn't matter!
• Neighbors (friends), paths (chains of friends), cycles (paths with repeats)
– Two connected nodes have a path
• Connected set: all pairs connected
– Component if maximal
• Clique: all pairs are neighbors


What can graphs represent
• Almost anything!
• Nodes can be:
– Physical objects (people, circuits)
– Computational objects (web pages)
– Places (cities, buildings, rooms)
– States of affairs (eating, sleeping)
– ETC, ETC

Example: class scheduling

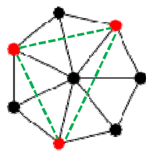
- Some pairs of classes meet at the same time (ex: CS51, CS161)
  - Given a set of conflicting classes, how many courses can you possibly take?
  - Is there a fast way to solve this problem?
- An independent set is a set of nodes where no pair is a neighbor
  - Sort of the "complement" of a clique

Scheduling via graphs

- Each course is a node
  - An edge means the courses conflict
- An independent set is a schedule
  - A set of non-conflicting classes
- Could take K classes iff the graph has an independent set of size K
  - Implies independent sets of size <K
- Is there a fast way to solve this?
  - Better than trying all groups of size K?

Independent set vs. clique

- A fast way to find an independent set of size K can find a K-clique
  - Complement the edges:
- What if clique is hard?
  - "No clever algorithm"
- Then so is independent set!
  - Otherwise we could use it as above
- A fast algorithm for one would give a fast algorithm for the other



Trees

- An undirected graph without cycles is called a tree
  - You've seen this in PS1, and will see many more trees throughout this course
    - In a certain sense, a Scheme program is actually a tree…
- A collection of trees is called (what?)

Directed graphs

- Undirected graphs are special case
- To view an undirected graph as a directed one, replace the edge between A and B with 2 edges
  - A to B, and B to A
    - Facebook "friend request" example
- In a directed graph we refer to a Strongly Connected Component
  - SCC for short, or often "component"

Two CS51 graph problems

- Find "communities" & their members
  - Are you in the same community as the important people?
  - Who's important in CS51?

- Find "important" nodes
  - Is there a graph property that makes certain nodes important?

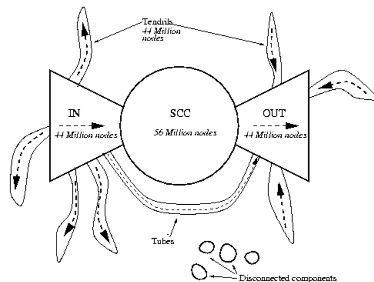
Simple graph theorem

- Take any graph, and replace each SCC by a single node
  - Suppose A is part of an SCC, which we replace by the node Z
    - The edge (B,A) becomes the edge (B,Z)
- Doing this always creates a DAG
  - Proof?
- Very simple facts about graphs can have quite interesting consequences
  - What's our favorite directed graph?

A few web observations

- You can get from Greg's home page to Ramin's by following links
  - Can also get to, e.g., Yahoo, Google, …
- Ditto from NYT to Washington Post
- Not true for all pairs of web pages
  - What's the DAG of the web??

DAG of the web, circa 2000



From: http://www9.org/w9cdrom/160/160.html
Broder et al., 2000

Finding a community

- How do we find a node's SCC?
- Figure out what nodes you can get to
  - If the graph is a tree, you've more or less done this "already" for PS1
- Graphs are harder than trees
  - Because they have cycles!
  - Need to keep track of where we have been already
    - But without using side effects!

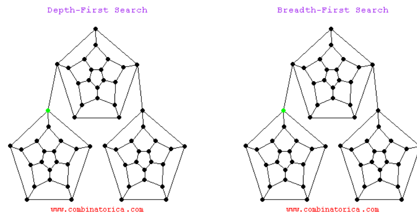
Graph traversal

- We will maintain a set of nodes we can get to from our input node (root)
  - For the moment, let's keep this as a list
  - We will add unseen nodes to this list
    - Neighbors of the node we visit
- We also need to maintain a set of nodes that we have already seen
  - Otherwise, we will get stuck in cycles

Traversal order

- Main issue is the order in which we explore unseen nodes
- Suppose root has 2 neighbors, A and B
  - Can completely explore where we can get from A before we even consider B
  - Or, we can explore A's neighbors, then B's neighbors
- Depth-first versus breadth-first

Examples of DFS vs BFS

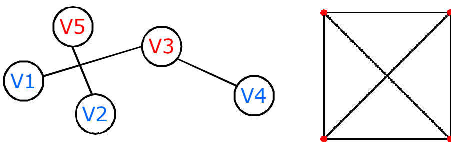


DFS vs BFS tradeoffs

- DFS: less space, generally faster
- BFS: more "fair"
  - B's neighbors waited a long time!
  - We'll see an application where this matters shortly
- These simple algorithms are actually incredibly important
  - Regularly used in important work!
    - Example: Broder's paper

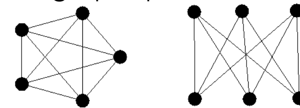
Graph planarity

- A graph is planar if you can draw it in the plane without edges crossing
  - Important for, e.g., chip design



Planarity testing

- Are these graphs planar?



  - How can you tell?
  - Linear-time algorithm via DFS
    - Hopcroft and Tarjan, 1986 Turing Award

Important nodes in an SCC

- We can now find our SCC
  - Let's suppose we have a list of important nodes also (CS51 staff)
  - Easy to determine if one of them is in SCC
- What if we want the closest one?
  - Being great-great-great-grandfriends with Victor, while nice, isn't as good for your GPA as being friends with Gideon
  - "Six degrees of CS51 TF's"
    - Surprisingly unpopular at parties

BFS alternative

- The depth of a node will be the length of the shortest path from the root
- BFS explores all nodes of depth 1, then all nodes of depth 2, etc.
  - This is what makes it so fair
- But BFS has to keep track of a huge number of nodes whose neighbors haven't yet been explored

DFS doesn't suffice

- DFS is very efficient but is very unfair
  - Perhaps we can make DFS more fair?
- Idea: run DFS, but only to a fixed depth (say, 5)
  - Once we consider a node at depth 5, we don't bother looking at its neighbors
  - This will rapidly search for nodes at depth 5 or less, without using much space

Iterative deepening

- Run DFS to depth 1
  - Then to depth 2, then to depth 3, etc
- Efficient, and guaranteed to find your closest head-TF
  - What's the drawback?
- When you run DFS to depth 3, you re-do a lot of work
  - In particular, everything you did when you ran DFS to depth 2!
  - Surprisingly, this doesn't matter much!

Graph summary

- Can easily compute an SCC
  - via BFS/DFS
- Using BFS we can naturally find nearest "important person"
  - But iterative deepening is a smarter way
  - Simple example of "graph search"
- Graphs, and problem reductions, will be your friends in CS51 (I hope!)