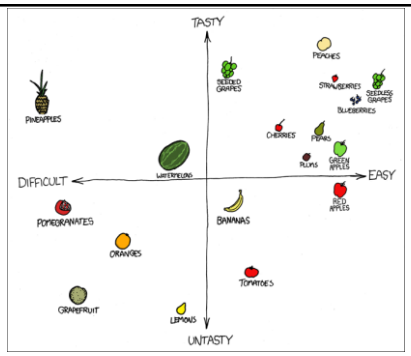
Graphs



(picture above from http://xkcd.com/388/)

7/29/2009

Many thanks to David Sun for some of the included slides!

Important Dates

- Project Work
  - Monday's Lab 8/3/2009
- Project 3 due
  - You may have 1 or 2 partners.
    - NO EXCEPTIONS!
  - Due Tuesday 8/11/2009 – 10pm
- Final Review
  - Sunday 8/09/2009 – 1-4pm in 306 Soda
- Final
  - Thursday 8/13/2009 – 5-6pm in 10 Evans

How to calculate your grade

Points	Grade
190-200	A+
180-190	A
170-180	A-
160-170	B+
150-160	B
140-150	B-
130-140	C+
120-130	C
110-120	C-
100-110	D+
90-100	D
80-90	D-
< 80	F

Assignment category	# points	percent
Project 1	6	3%
Project 2	12	6%
Project 3	18	9%
All other homework scaled to:	24	12%
All quizzes scaled to:	20	10%
Midterm 1	24	12%
Midterm 2	36	18%
Final	60	30%

What does "scaled to" mean?

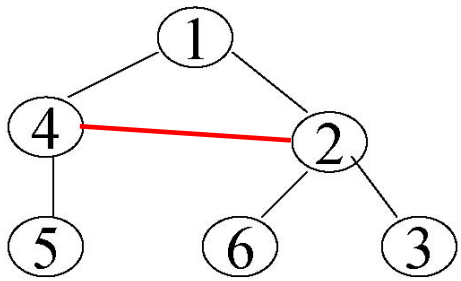
$$\frac{\sum \textit{your homework scores}}{\sum \textit{homework points possible}} \; x \;\; 24 \text{ "course" points}$$

$$\frac{\textit{your midterm score}}{\textit{possible midterm points } (28)} \; x \;\; 36 \text{ "course" points}$$

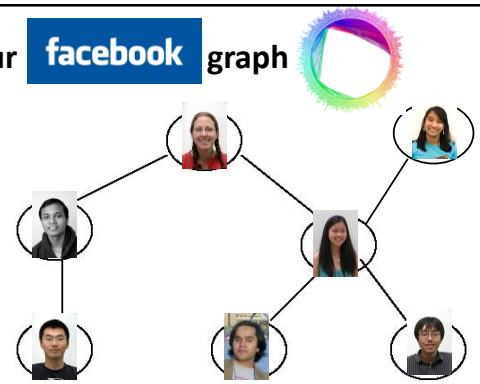
Tree Definitions: Overview

- A ***tree*** consists of a *set of nodes* and a *set of edges* that connect *pairs of nodes*.
- There is *exactly one* path between any two nodes of the tree.
- A ***path*** is a connected sequence of zero or more edges.

How Trees (grow-up) become Graphs



Our facebook graph



Graph Vocab in facebook Terms

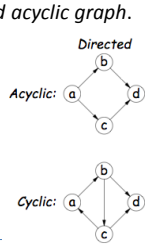
- Undirected Graph:
  - If I am friends with Jon, Jon is friends with me.
- Directed Graph: (non-mutual friendships)
  - I can be friends with Kaushik even if he is not friends with me
  - David can be friends with me even if I am not friends with him.
  - I can be friends with George AND George can be friends with me.

Graph Vocab in facebook Terms

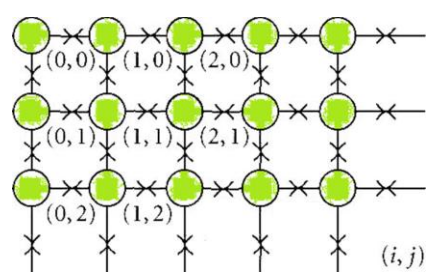
- Weighted Graph:
  - I can give each friendship (link) a value based upon how good of friends we are.

Cycles

- A *cycle* is a *path* without repeated edges leading from a vertex back itself.
- A graph is called ***cyclic*** if it has a a cycle, else *acyclic.*
- DAG: *directed acyclic graph*.

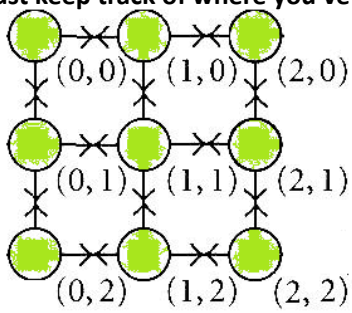


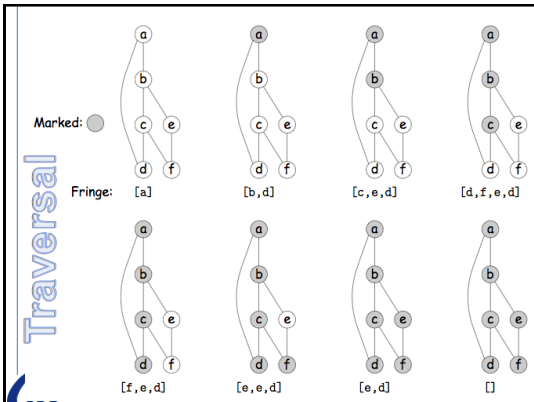
You've used a Graph before



Paint Bucket was a PAIN because…

You must keep track of where you've been





Iterative Depth First Traversal

```
Stack<Vertex> fringe;
fringe = stack containing {v};

while (! fringe.isEmpty()) {
  Vertex v = fringe.pop ();
      if (! marked(v)) {
            mark(v);
            VISIT(v);
            For each edge (v,w) {
                  if (! marked (w))
                        fringe.push (w);
            }
      }
}
```

Recursive Depth First Traversal

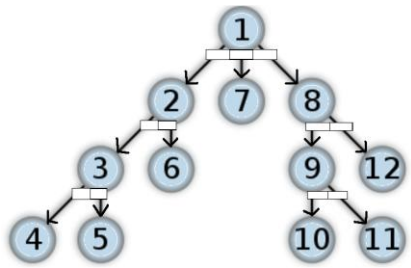
```
void traverse (Graph G) {
  For each vertex v in G {
            traverse (G, v);
  }
}

void traverse (Graph G, vertex v) {
  if (v is unmarked) {
            mark(v);
            VISIT(v);
            For each edge (v,w) in G {
                  traverse(G,w)
            }
  }
}
```

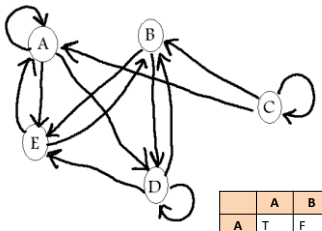
Ways to Represent Graphs

- Array of Neighbors
- Adjacency Matrix
- Hash Table of Vertices

For both Trees and Graphs: Keep an array of your children/neighbors



Adjacency Matrix



	A	B	C	D	E
A	T	F	F	T	T
B	F	F	F	T	T
C	T	T	T	F	F
D	F	T	F	T	T
E	T	T	F	F	F

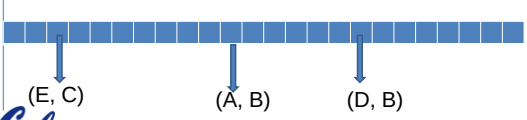
Adjacency Matrix

- insert(u, v)
- remove(u, v)
- member (u, v)
- printEdges()

	A	B	C	D	E
A	T	F	F	T	T
B	F	F	F	T	T
C	T	T	T	F	F
D	F	T	F	T	T
E	T	T	F	F	F

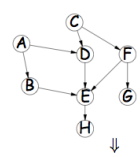
Hash Table of Vertices

- insert(u, v)
- remove(u, v)
- member (u, v)
- printEdges()



Topological Sort

- Problem:
  - given a DAG, find a *linear ordering* of all the *vertices* that's consistent with the edges: if *(u,v)* is an edge, then *u* appears before *v* in the ordering.
- A topological sort of a DAG can be viewed as an *ordering* of its vertices along a horizontal line so that all directed edges go from *left to right.*



⇓

```
[A,B,C,F,D,G,E,H], or
[A,C,B,D,F,E,G,H], or
[A,B,C,F,D,E,H,G], or
⋮
```

Shortest Path Problem

- Suppose a graph represents a highway map, and each road is labeled with the amount of time it takes to drive from one interchange to the next. What's the fastest way to drive from Berkeley to Los Angeles?

Shortest Paths: Dijkstra's Algorithm

- Problem: Given a graph (directed or undirected) with *non-negative* edge weights, compute shortest paths from given source vertex, *s*, to all vertices.
  - "Shortest" = sum of weights along path is smallest.
  - For each vertex, keep estimated distance from *s*, . . .
  - . . . and of preceding vertex in shortest path from s.