**Announcements**

* Self-reflections due tonight! Make sure both you and your partner submit
* Midterm grading this weekend
* Project 3 spec will be released this weekend!

**Motivation**

* Using the data structures we currently know, what if we wanted to represent Facebook friends, where two people are connected if they are friends?
  + Things to consider: friend A be friends with friend B, who can be friends with friend C, who can also be friends with friend A
  + Can we use trees here?

**Graphs Intro**

* Just as in trees, graphs will have nodes and edges that connect them. The nodes are usually called vertices
* Some terminology and graph facts (not comprehensive):
  + Vertex: a node in a graph
  + Edge: an edge between vertices u and v are represented by (u, v), means the two vertices are connected
  + Neighbors/adjacent: for a particular vertex v, neighbors/adjacent vertices of v are one edge away from v
* Some graph facts: (not comprehensive)
  + An edge starting from u and ending at v is usually written as (u, v)
  + Each vertex v has neighbors, which are the vertices that are one edge away from v (this is also called being adjacent)
  + Edges can be directed or undirected
    - Directed: one way streets
    - Undirected: Facebook friends (where being friends is mutual)
  + Trees are a special type of graph
  + Graphs can allow any type of edge connections (including cycles)
* There are many ways to represent graphs, two are listed below:
  + Adjacency list: an array of lists, each item in the array represents a vertex and the corresponding list denotes what is reachable from that vertex, good for sparser graphs
  + Adjacency matrix: if N is the number of vertices, an N\*N boolean matrix. If the (i, j) index is true, then the edge (i, j) exists, good for dense graphs and fast insertion of edges (array indexing!)
  + Vertex objects: each vertex is represented with a Vertex object that has pointers to the neighboring Vertex objects
  + And many more!

**Graph Traversals**

* BFS and DFS still exist, though need a slight modification due to the fact that there can be cycles
  + In trees this wasn’t a problem!
  + Include a ‘visited’ data structure so we know where not to look again!
* There will be other traversals that we’ll learn in the future
* Topological sort: gives an ordering of tasks to do, must be a directed acyclic graph (DAG)