**Announcements**

* Midterm 2 scores have been released!
  + Regrades open Tuesday 8/1 at noon and close Saturday 8/5 at noon
  + Not for asking for more partial credit!
* One-on-one meetings! shoutkey.com/leek
* Project 3 spec has been released, to be done in your partnerships (not super groups)
* Guerrilla section this Sunday 3-5 PM

**Motivation**

* We want to somehow traverse all the nodes that are within our graph, but maybe there are different orderings that we want to pick!

**Traversal Pseudocode**

Instantiate fringe

Add starting vertex onto fringe

While fringe is not empty:

V = remove vertex from the fringe

Do something with V

For all neighbors of V:

Add neighbors into the fringe

**Depth-First Search**

* “Depth-first” before anything, explore one path completely before turning back
* If you have three boxes to search, explore one box completely before going to the next
* Fringe is a stack (last in first out)
  + Think of a stack of plates or stack of papers, can only remove what’s at the top!

**Breadth-First Search**

* “Breadth-first”, explore all paths equally
* If you have three boxes to search, explore each box layer by layer in equal layers
* Fringe is a queue (first in first out)
  + Think of a line at the grocery store

**Topological Sort**

* If you have a graph that represents the courses you have to take, with a node x pointing to a node y if x is a prerequisite of y, a topological sort will give you an order of classes to take
* Can only be run on a directed acyclic graph (DAG)
  + Directed: must have an ordering, one must come before the other
  + Acyclic: cannot have cycles because if x requires y and y requires x, which one is the prerequisite?
* Usually graphs have multiple valid topological sorts
* Algorithm:
  + Keep an array that is initialized to the in-degrees of all vertices, this will signify the number of prerequisites that need to be satisfied for a particular vertex
  + Repeat until all vertices have been explored:
    - Pick a vertex whose indegree is 0 (must pick something that doesn’t have any more prerequisites to satisfy)
    - For each of the neighbors of this vertex, decrease the number of prerequisites that need to be satisfied by 1
      * The current vertex is a prerequisite of all of its neighbors, once we have explored the current vertex, all the neighbors have less prerequisites to go!

**Dijkstra’s Algorithm**

* Motivation: what if we want a graph to describe cities and distances between them, need to introduce weighted edges! What if we wanted to find the shortest path between San Francisco and New York City? We can use DFS/BFS but how do we determine shortest?
* Used to find the shortest path from a starting vertex to all other vertices
* Fringe is a priority queue ordered by how far a particular vertex is from the starting vertex
* Not guaranteed to work correctly with negative edges!
* Algorithm:
  + Keep track of predecessors and minimum distance from the start
  + Add the starting value to the fringe
  + While the fringe isn’t empty
    - Remove the vertex v with the minimum distance from the start
    - For each neighbor w of v
      * If w is not on the fringe, add it with the current minimum distance and update predecessors and minimum distance from the start
      * Else if w’s current minimum distance is greater than v’s current minimum distance + distance between v and w
        + Update the values in predecessors and minimum distance from the start
* Runtime:
  + Because we have all of our vertices on the fringe, each operation on the fringe will take logV time
  + We will have to add each vertex once and go through at most all E edges, each one of these operations will take logV amount of time, so the total runtime is (V+E)(logV)