Git commands for this class

git init - Initialize a local repository

git branch – Create a new branch/version or copy

git add – Add a file/change to the staging area

git commit – Commit changes (from stages) “Commit message what the change is”,

git merge – Merge a branch into a target branch

git checkout – Switch to a branch

The difference between github and git is that github is a server that hosts git remotes.

Remote – a copy of your repository

There is a “local remote” and “remote”

LA’s should review the develop to master pull requests

LA’s should be one of the collaborators on GitHub

Push – Push is pushing any commits from your local to the remote repository

Pull – Pull is simply a fetch followed by a merge

Fetch – Fetching

You can do a git fetch at any time to update your remote-tracking branches under refs/remotes/<remote>/ . This operation never changes any of your own local branches under refs/heads , and is safe to do without changing your working copy

GitFlow

No one should be pushing to master. You should be pushing to your feature branch which would then be merged to develop branch and then to master.

Master branch – Working product

Develop branch – Next release

Hotfix Branch – There is a bug in master and hotfix branches are meant to be done quick (Branched from master and goes into master)

Graphs – The most used data type

Linked List is a specialized graph

Graphs

* Vertices
  + Holds the data
* Edges
  + Allow movement to other vertices
  + Can have “weights”

A Linked List and a Tree and a Heap is just a specialized graph

How can we use graphs?

* Networks/Network topologies
* Advanced Math
* Functional Programming
* Machine Learning
  + Neural Networks
* Graph Databases
* Game Development

Main Types of Graphs

* Weighted vs. Unweighted Graphs
* Weighted Graphs
  + Edges have numeric values (distance, importance, etc)
* Unweighted
  + Edges are all effectively equivalent in their traversal
* Directed vs. Undirected Graphs
* Directed (Digraph) Graph
  + You can only move along an edge in one direction (via arrows)
* Undirected Graph
  + You can move along an edge in both directions

Typical Graph Functions

* Standards
  + Add/remove vertex
  + Add/remove edge
  + Weighted?
  + Directed?
  + Find Vertex
  + Find Edge
  + Next Edge
  + Is Connected?
    - Given any vertex, can you get to any other vertex through legal edge movements
  + Is Fully Connected?
    - Can every vertex reach every other vertex through a single edge traversal?
  + Path Exists?
    - Does at least one path exist between two given vertices
* Dijkstra’s Algorithm
  + If a path between two vertices exists, find the shortest path from one vertex to the other

How to Represent Graphs

* Adjacency List
  + List of Linked Lists
  + Each vertex is the head of a linked list (or a list, array, etc.)
    - The linked list holds their edges (and weights, if they exist)
* Adjacency Matrix
  + Matrix of vertices
    - Row and column titles are the vertices
    - Row-column indices are edge weights (or edge existence)

List vs Matrix

* List
  + More common
  + Usually easier to visualize/conceptualize
  + Adding edges is slow
  + Many algorithms are slow or very individualized
  + Usually take up less space
* Matrix
  + Most algorithms are extremely fast in comparison
  + Searches and traversals very mathematical
  + Adding vertices is slow

Library

* NetworkX (<https://networkx.github.io/>)
  + Graphs, digraphs, multigraphs
  + Well tested (>90% coverage)
  + Very 00 compliant

Ask Customer what they want for the API

* Directional Methods (North, South, etc)
* She is looking for the first two user stories

What customer wants

* Wednesday