Section 9 Overview

Agenda:

- Mini Assignment
  - Spanning Trees & Forests
  - Topsort
- MST Algorithm Overview
  - Prim-Jarnik
  - Kruskal's
- Code Review
- Functional Programming Problems
- Interview Questions

Mini Assignment

1. Spanning Trees and Forests
   a. A spanning tree and a minimum spanning tree are both a subset of edges in a graph that span every vertex (forming a tree). A ***minimum spanning tree is the spanning tree with the least possible total edge weight*** for a given graph.

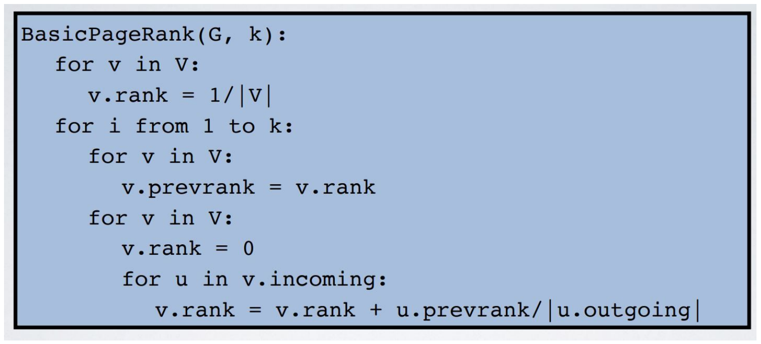
   b. A spanning forest is the collection of STs for unconnected graphs, while a MSF is the collection of MSTs.
2. Topsort
   a. an algorithm that only works on directed acyclic graphs and provides a valid ordering of vertices in the DAG, ensuring that for each vertex, all of its prerequisites come before it in the ordering.
   b. iterates over vertices in a DAG. If a vertex has no prerequisites (if it is a source), we visit it. After visiting a vertex, we remove all of its outgoing edges because the prerequisite they represent has been satisfied. This will create new source nodes that we then visit.
   c. can only be used on DAGs. Cycles will mess up the algorithm.

Spanning Trees

1. **Prim-Jarnik's Algorithm**
   a. Greedy algorithm to find MST using priority queue approach
   b. Start by decorating nodes with infinity cost, set start node to 0, keep removing the smallest cost node from the PQ and updating its neighbors accordingly (curr node cost + edge)
   c. O((|V| + |E|) * log(|V|))
   d. Optional but cool animation here: https://visualgo.net/en/mst, can choose an animation of Prim's or Kruskal's on an MST

2. **Kruskal's Algorithm**
    a. Union Find
        i. Intuition - 'clouds' used to make sure that the edges that are added do not create cycles
        ii. Path compression - have every node point to the parent of the cloud it's in to decrease the amount of looping required to see if the two nodes on opposite sides of an edge belong to the same cloud

3. **PageRank**



```
BasicPageRank(G, k):
  for v in V:
    v.rank = 1/|V|
  for i from 1 to k:
    for v in V:
      v.prevrank = v.rank
    for v in V:
      v.rank = 0
      for u in v.incoming:
        v.rank = v.rank + u.prevrank/|u.outgoing|
```

I. O(|V| + |E|)
ii. Rank web pages by importance using the underlying assumption that "more important websites are likely to include more links from other websites"
iii. Webpages represented as nodes
iv. Ranked according to the number of links it has to other existing webpages
v. The more webpages it's linked to, the higher the rank
vi. Updates rankings each iteration

4. **Code Review**
    a. Went over peer's radix sort and gave constructive feedback

5. **Functional Programming**
    a. A way of structuring programs using mathematical functions
    b. What are the key characteristics of this programming paradigm?
        i. Functions are stateless
        ii. Functions are deterministic
        iii. Functions are first-class citizens
    c. Higher Order Functions
        i. map(function, list)
            1. Applies function to every element in the list
        ii. reduce (function, list, accumulator)
            1. A binary function, i.e. takes in two arguments
            2. Applies function with the accumulator as the first argument and each element of the list as the second argument

d. **Practice Problems**
   i. Return abs values of a list **(**map and if/else)
      1. Lambda x : math.sqrt(x**2)
      2. map(Lambda x : math.sqrt(x**2) , list]
      3. f(x) = Lambda x: if x > 0 return x else return -1*x
      4. map(f(x), list)
   ii. Reverse a list (uses reduce, acc is a list)
      1. Lambda x, y : x + y[-1]
      2. reduce(reverse, list,[])

Optional Interview Questions
Reverse a string in Java (without using stringBuilder)

In-place solution:

```
	public String reverse(String s) {
		char[] str = s.toCharArray();
		int mid = str.length / 2;
		// If we went from i=0:str.length, we’d end up un-reversing
		// everything
		for (int i = 0; i < mid; i++) {
			char tmp = str[i];
			str[i] = str[str.length - i];
			str[str.length - i] = tmp;
		}
		return new String(str);
	}
```

Reverse a singly linked list
http://www.geeksforgeeks.org/write-a-function-to-reverse-the-nodes-of-a-linked-list/

```
	public void reverse(Node head) {
		prev = None
		current = head
		while(current is not None):
			next = current.next
			current.next = prev
			prev = current
			current = next
		head = prev
	}
```

Reverse a doubly linked list
http://www.geeksforgeeks.org/reverse-a-doubly-linked-list/

```
    public void reverse(Node head) {
        Node tmp = null;
        Node curr = head;

        while (curr != null) {
            tmp = curr.prev
            curr.prev = curr.next
            curr.next = tmp
            curr = curr.prev
        }

        // Edge cases like empty list and list with one node
        if (tmp != null) {
            head = tmp.prev
        }
    }
```