# 18 November

## Announcements

- lots of extensions used on HW 5: maybe makes sense because we only have two homeworks left!
- One will be released today, due next Monday
- One released Friday Nov 29 and due Friday Dec 6
- Final exam is Friday the 13th (!)
- Labs happening this week, next week, and first week of December
  - Pro tip: treat labs like exams (don't just immediately google / ask a friend); pay attention to where you get stuck, how you get un-stuck and come up with strategies
- Project will be released by tomorrow; you will *fork* a repository

## Graph Traversals!

- Traversal is the key to searching
- Need to guarantee:
  - we traverse the entire graph (don't miss anything)
  - don't get stuck in loops (track where we've already explored)
- Use cases:
  - solving or playing games
  - generating directions in map
  - web crawling / scraping
  - finding specific files in filesystem or calculating total size of files on disk
  - mazes
- For now, we will assume our graph is in object-oriented representation (collection of `Node` objects with `data` and `neighbors`)

### Search Problem

Inputs:

- start node (somewhat arbitrary, depends on problem)
- goal node (usually a **property** of that node, since recall that nodes are defined by their data and incoming/outgoing edges)

Output:

- Node object, or `null` if node does not exist in this graph
- Or, entire path from start to goal

## Breadth-First Traverse

- Example on board for intuition
- Corn maze: you have lots of friends, walkie-talkies. Split group at every fork, and wait to hear from everyone before you do the next split
  - Will minimize walking and how often you have to split the group

Algorithm:

1. from start node, visit all neighbors
2. visit all neighbors-of-neighbors
3. . . .
4. profit!

Talk to neighbors and think about what data structures we could use to track:

- nodes we have already visited

- nodes in the "frontier" we are currently visiting

Answers: set (hashmap), queue

More complete traversal algorithm (procedural):

```java
public void BFS(Node s, T goal_data) {
  // create data structures and initialize start node
  HashMap<Node, boolean> visited = new HashMap<Node, boolean>();
  Queue<Node> frontier = new LinkedList<>();
  visited.put(s, "True");

  while (! frontier.isEmpty()) {
    Node curr = frontier.poll();
    if (curr.data == goal_data) return curr;

    for (Node n : curr.neighbors) {
      if (!visited[n]) {
        visited.put(n, "True");
        frontier.add(n);
      }
    }
    return null;
}
```

- Example on board
- Note that this traversal strategy gives nodes in *topological order*

## Depth First Traversal / Search

- Intuition: closer to how we actually do corn mazes
- Explore as far as possible along each branch before backtracking

Recursive psuedocode:

```
DFS(G, n):
    label n as visited
    for all neighbors w of v:
      if w not visited:
        DFS(G, w)
```

This implicitly uses the call stack as a stack to perform backtracking! So we can also implement procedurally with an actual stack (psuedocode):

```
DFS(G, n):
  let S be a stack
  S.push(n)
  label n as visited
  while S is not empty:
    v = S.pop()
    label v as visited
    for all neighbors w of v:
      if w not visited:
        S.push(w)
```

- Write down order if we print node labels of example graph (on board)
- Note that depending on when we print labels, we can get a pre-order (nodes in order they were first visited) or post-order (order they were last visited)