Day 6: Depth-First Search

- Searches add structure to graphs and turn them into trees. (?)
- Some vocabulary:
  - **Expanding [a node]:** applying relevant rules to produce more nodes for exploring
  - **Testing for success:** Validate that you've found a solution
  - **Backtracking:** Undo actions when you've reached a dead end
    - Easily done with recursive algorithms
- Two main approaches: Breadth-first and Depth-First
- Breadth-first
  - Uses a queue for to-be-explored states
  - Explores one layer of a tree before the next
  - Always finds a solution if it exists
  - Always finds the shortest solution
- Depth-first
  - Uses a stack for to-be-explored states
  - Explores all the way down before going sideways at all
  - Will infinite loop for cyclical graphs
  - If you add a hashmap/cache, you can make it work for cyclical graphs but it might not find the shortest solution
- ProTip: use function parameters and stack for the search
- ProTip: in your state, you can store a list of the path you've taken to get there, thereby allowing you to not only know that there is a solution but what the solution is.
- DFS/BFS are weak methods. How can you make them stronger?

        - Go through the tree in a specific order (ex. take away the largest possible coin first)
        - Prune the tree, for example (pennies):
            - If the amount of money is greater than the number of coins, or more than 25x the number of coins, it's impossible
            - First, take away pennies till it's a multiple of 5, then do nickles/dimes/quarters
        - Use heuristics to improve the search
            - Search backwards (start at goal and go to start)
                - Depending on the problem, this may be way more efficient, the same, or worse
                - Factors to consider:
                    - You want to be moving from a smaller set to a larger set
                    - Is the branching factor symmetric? You want to keep the search as small as possible
                    - Does the reasoning need to be human like?
- Considerations for BFS vs. DFS
    - Topology of the space?
        - cyclical vs. dissipative problems (uses up resources as they go, likely DFS is better)
    -
    - Do you really need to check for cycles?
        - It might be cheaper to just repeat your work
        - Some problems are so dense that you're very unlikely to hit a loop.
    - Use a heuristic that can compute the "goodness" (ie. distance to solution) of any state to figure out which states to explore first.
        - Needs to be smooth across all states
        - Needs to be fast to measure

- This can (and almost certainly is) approximate
- In theory, this doesn't actually improve the performance that much (ie. BigO notation), but in practice it's a huge improvement.