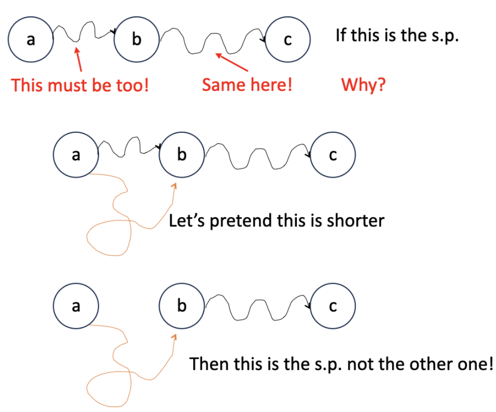
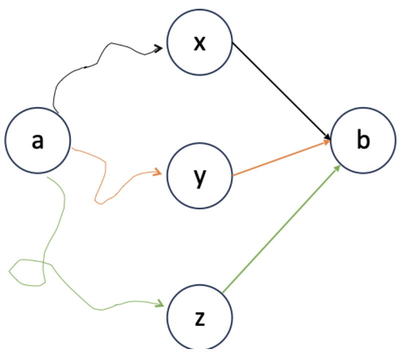
CAS CS 440
Lec 2

Search II

1. BFS (simple path – no vertex is repeated, no cycle), does not care about edge weights
    a. Finds shortest path from the source node if they are unweighted (the more edges in the path, the more expensive and if there are more edges, it means more weights – tries to avoid)
    b. If weighted, the order may not line up, so it is not guaranteed that it finds shortest path
    c. Can modify BFS to find algorithm that finds shortest path to every node from every node
    d. Expands radially outward (neighbor's neighbors …)
2. Depth First Search
    a. Recursively probe "deep" into paths (drills all the way down until it goes back up, enumerates path down to the root)
        i. Can also implement with a stack
    b. Finding simple paths: do not expand a node that is already visited
        i. Algorithm has unbounded time if not!
    c. NOT optimal: returns first path found! (not the shortest path), does care about edge weights
    d. Much better memory complexity: worst case
    e. Can bind depth (hyperparameter): depth limited search
        i. Called diameter of state space
    f. Iterative deepening DFS:
        i. Gradually increase depth limit and rerun DFS (diameter = 0, 1, 2, …)
        ii. Stop when first solution is found
        iii. Optimal when path cost is monotonically-increasing as function of vertex depth (unweighted)
        iv. Still not optimal if there are weights
        v. BFS: worst case O(V^2) + something else
3. Problems with DFS
    a. Path it finds may not be optimal
    b. Just guarantees a path exists
    c. DFS is also blind
        i. Algorithms do not know about the goal state (they are not aware) - not solve our problem

4. Dijkstra's Algorithm
    a. Modification of BFS
    b. Key idea: Shortest paths contain shortest paths
    c. 

    d. 

        i. One of these must be the shortest path from a to b
        ii. Try the smallest (cheapest) first! Might be wrong
        iii. If we're wrong, forget previous choice and remember better one
5. Dijkstra's Algorithm
    a. BFS has the right spirit
    b. Neighborhoods doesn't (necessarily) correspond to longer paths
    c. Idea:
        i. Keep collection of all paths we've explored so far
        ii. Expand the smallest one
            1. Wherever that path leads, that's the shortest path to there
                a. If it wasn't would have seen, shortest path before this one
            2. Expand path = visit neighbors of path
            3. Examine neighbors:
                a. If the neighbor is brand new: yay! Add to collection
                b. If a path to the neighbor already exists:
                    i. Is the path we just found better?
                        1. Keep the better one
                        2. If so, forget old version! If not, forget new path