Middlebury

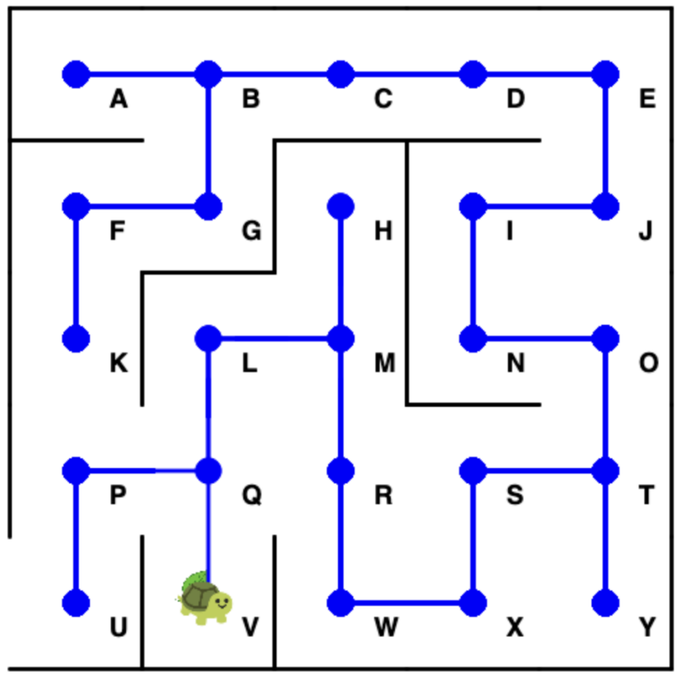
CSCI 200: Math Foundations of Computing

Spring 2026

Lecture 8W: Graph Algorithms I

Goals for today:

1. Implement breadth-first search (BFS) and depth-first-search (DFS) algorithms.
2. Build a spanning tree using DFS and BFS.
3. Build a minimum spanning tree (MST) using Prim's algorithm.

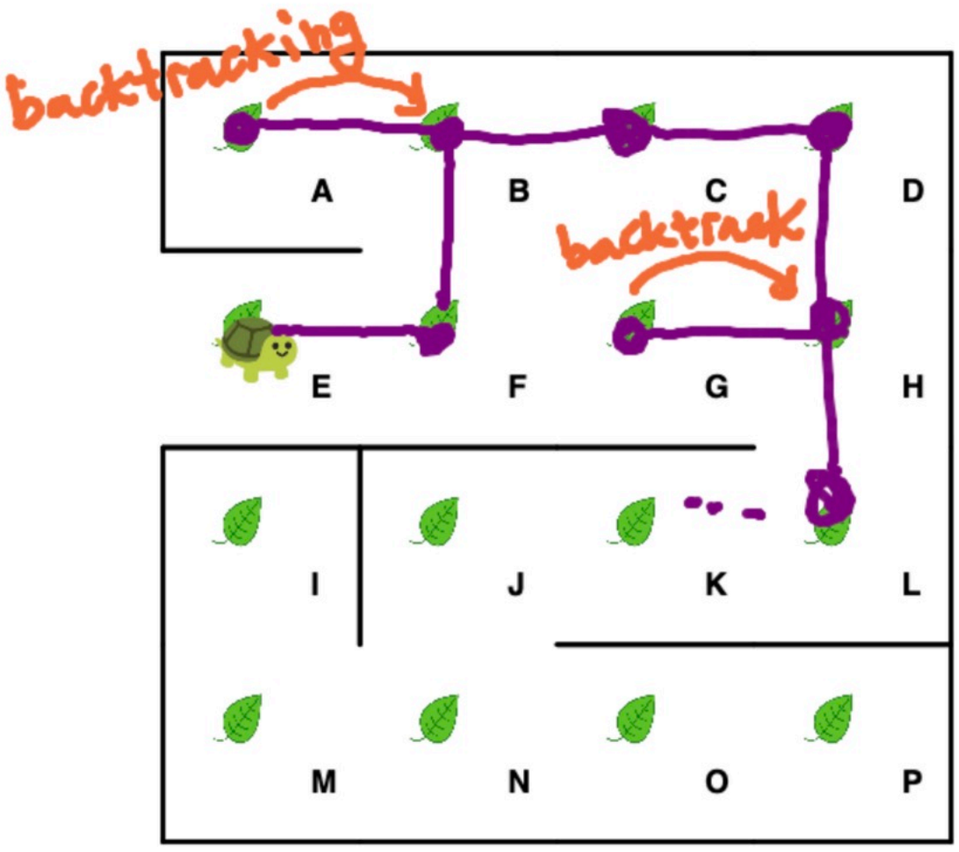


- Click on "game" in the row for today's class at go/cs200.
- Leave the mode as "free" and try to navigate the maze maintaining the tree property (connected, acyclic).

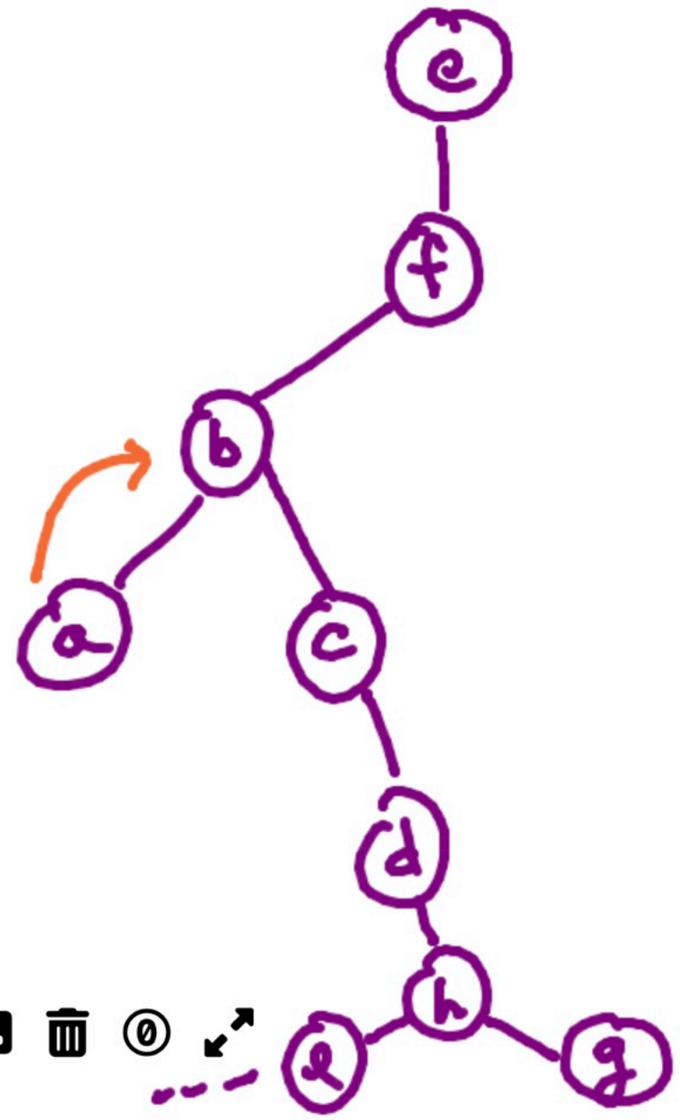
Depth-First Search ("backtracking").

Main idea: Keep traversing edges until you "hit a wall," then go back to parent.

→ Maintain a tree: **connected** and **acyclic**



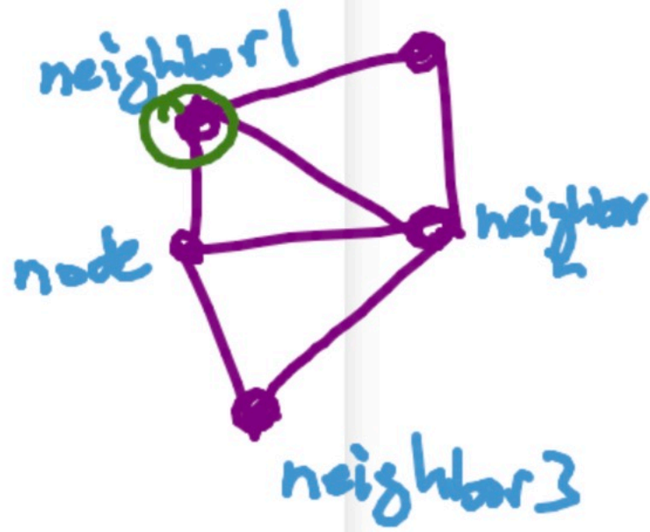
visit adjacent nodes in alphabetical order



Depth-First Search in Python

```python
def dfs(root, nodes, edges):
    adj = get_adjacency_lists(nodes, edges)
    tree_nodes = {root}
    tree_edges = set()

    def visit(node):
        print(node) # to print traversal order
        for neighbor in adj[node]:
            if neighbor not in tree_nodes:
                tree_nodes.add(neighbor)
                tree_edges.add((node, neighbor))
                visit(neighbor)
    visit(root)
    return tree_nodes, tree_edges
```



how many times is this line hit?

recursive step into neighbor

complexity: visit every node
visiting every edge twice

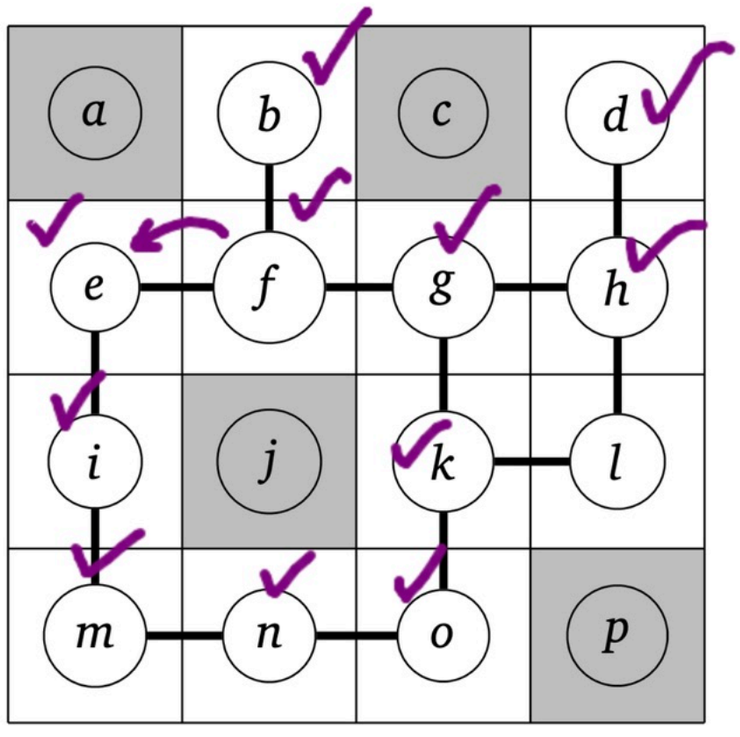
dense graph $|E| \simeq |V|^2$

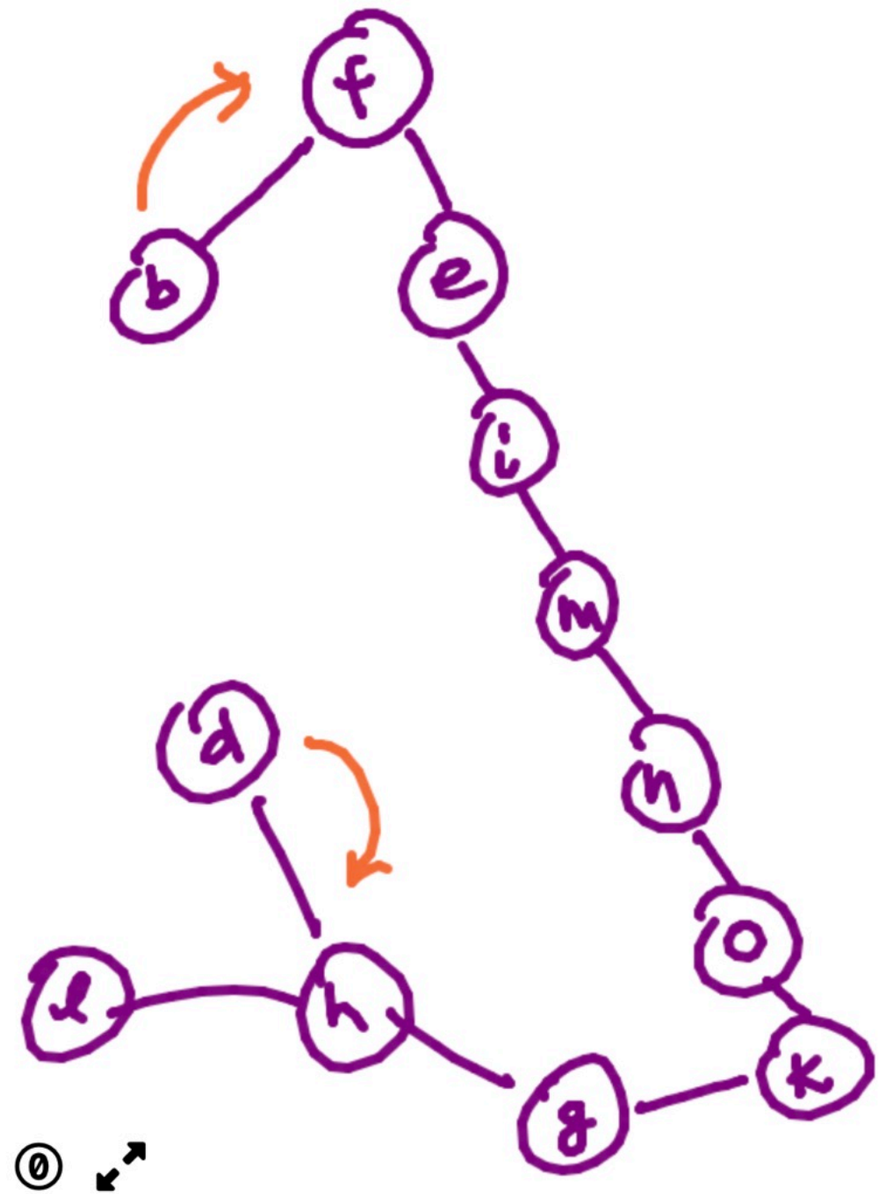
$O(|V| + |E|)$

if graph is sparse
$|E| \ll |V|$

Exercise 1: Build spanning tree of this graph using DFS.

- Start at vertex f.
- Visit neighboring vertices in *alphabetical* order.
- List order of vertices visited.

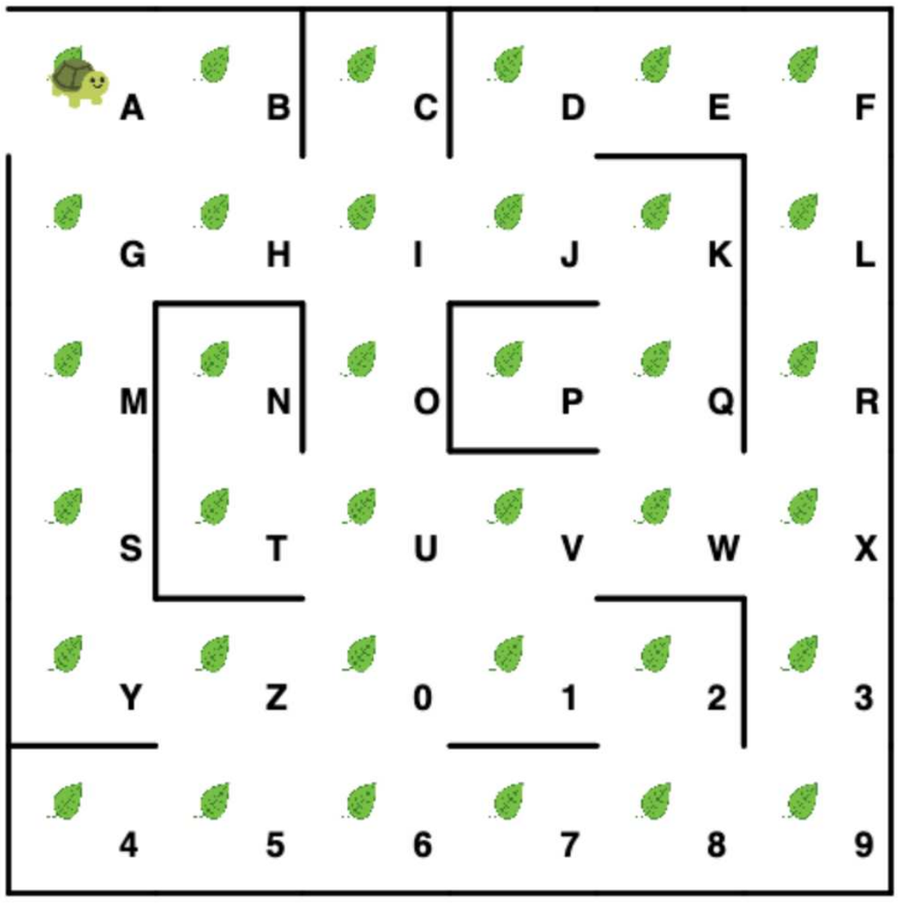




Exercise 2: play the DFS game!

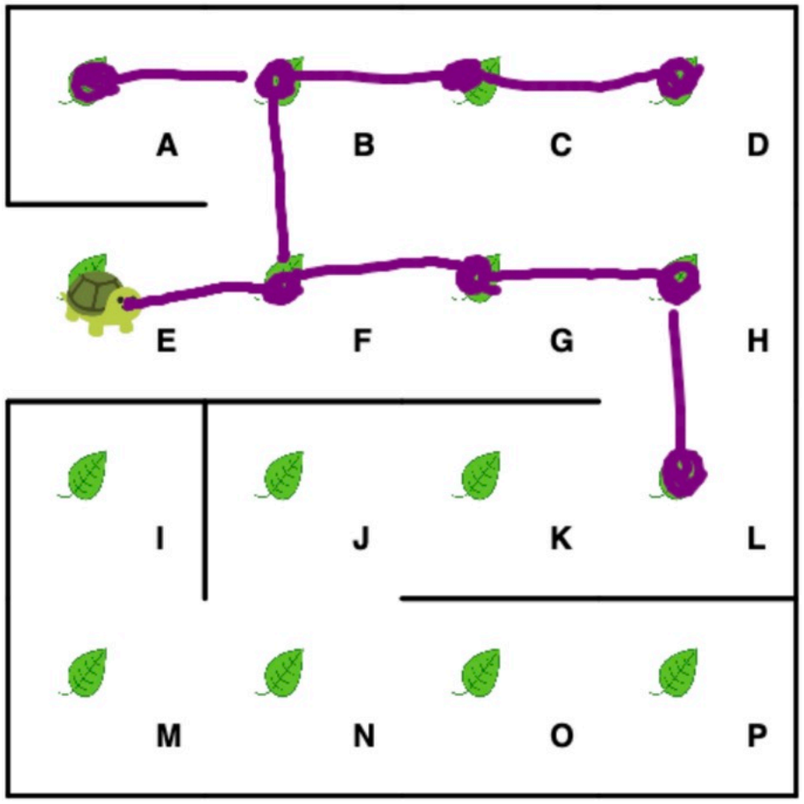
Click on "game" in the row for today's class:

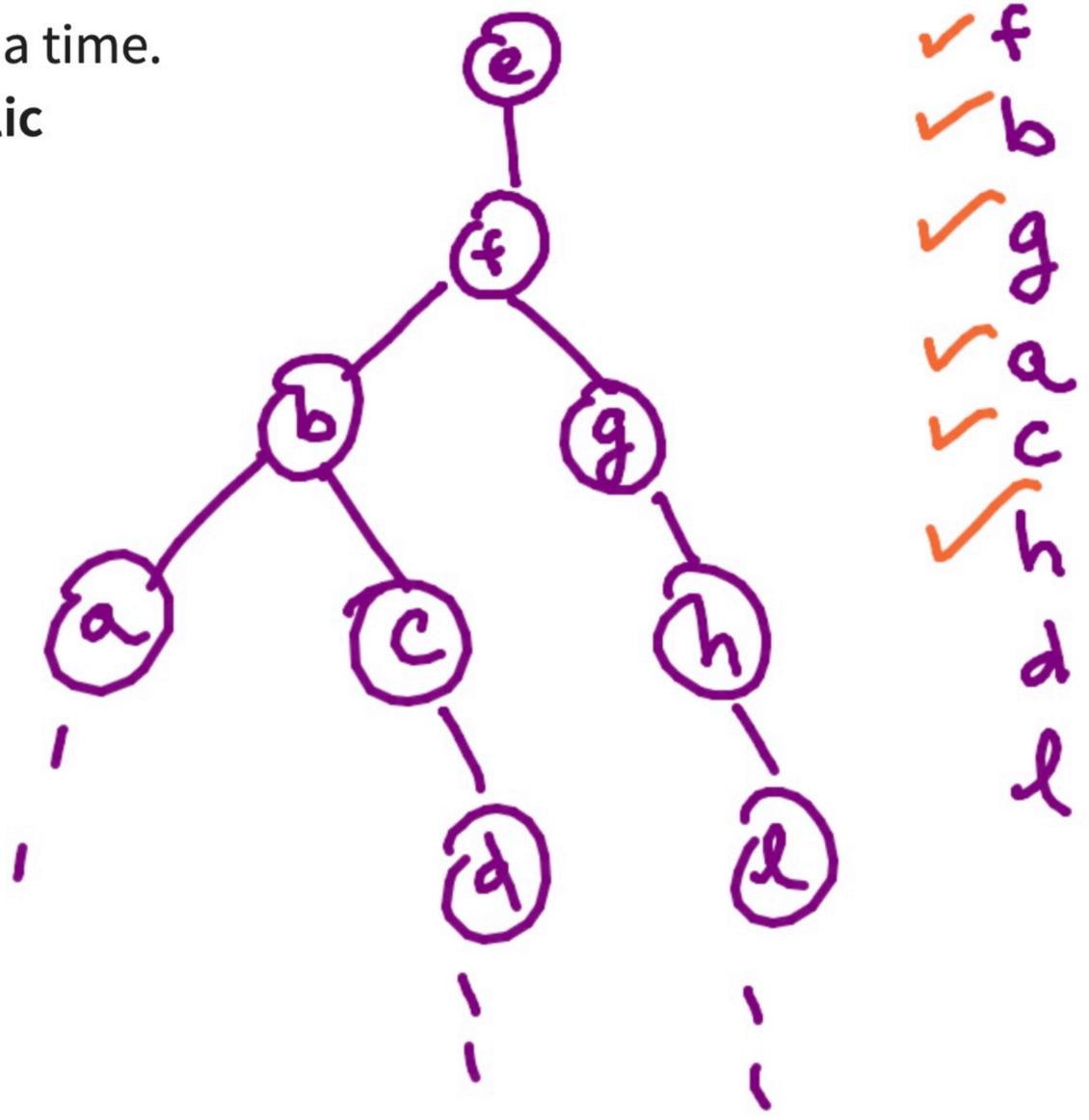
- Set the mode to "DFS" using the dropdown.
- Click the "Start" button and collect all the leaves using DFS on the maze graph (within 20 seconds).
- We'll play until there are > 10 games and the difference between wins and losses is > # people in the class.



Breadth-First Search ("flooding").

Main idea: Visit neighbors one "level" at a time.
$\rightarrow$ maintain a tree: **connected** and **acyclic**





Breadth-First Search in Python

```python
def bfs(root, nodes, edges):
  adj = get_adjacency_lists(nodes, edges)
  tree_nodes = {root}
  tree_edges = set()
  queue = [root]  # queue of unprocessed vertices
  index = 0
  while index < len(queue):
    node = queue[index]
    print(node) # to print traversal order
    index += 1
    for neighbor in adj[node]:
      if neighbor not in tree_nodes:
        queue.append(neighbor)
        tree_nodes.add(neighbor)
        tree_edges.add((node, neighbor))
  return tree_nodes, tree_edges
```

complexity: similar to DFS

$O(|V| + |E|)$

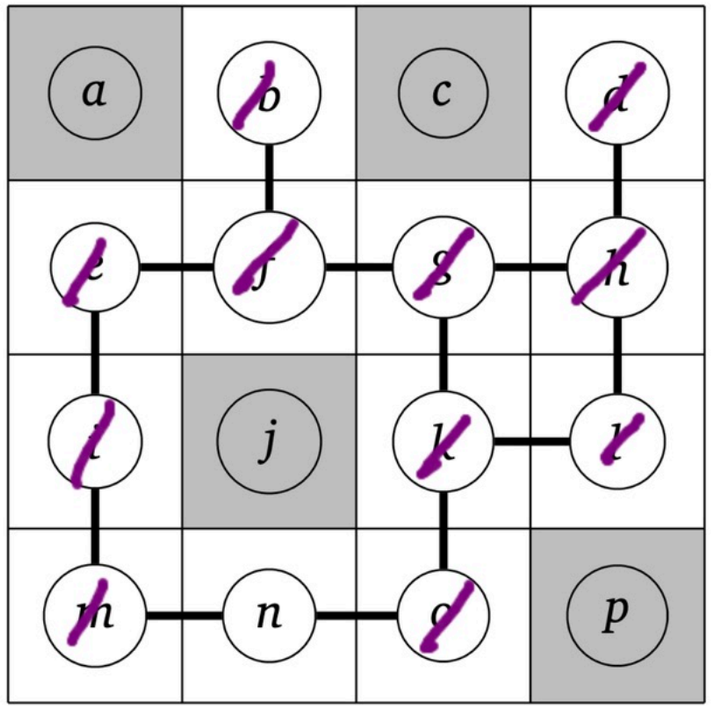
depends on graph

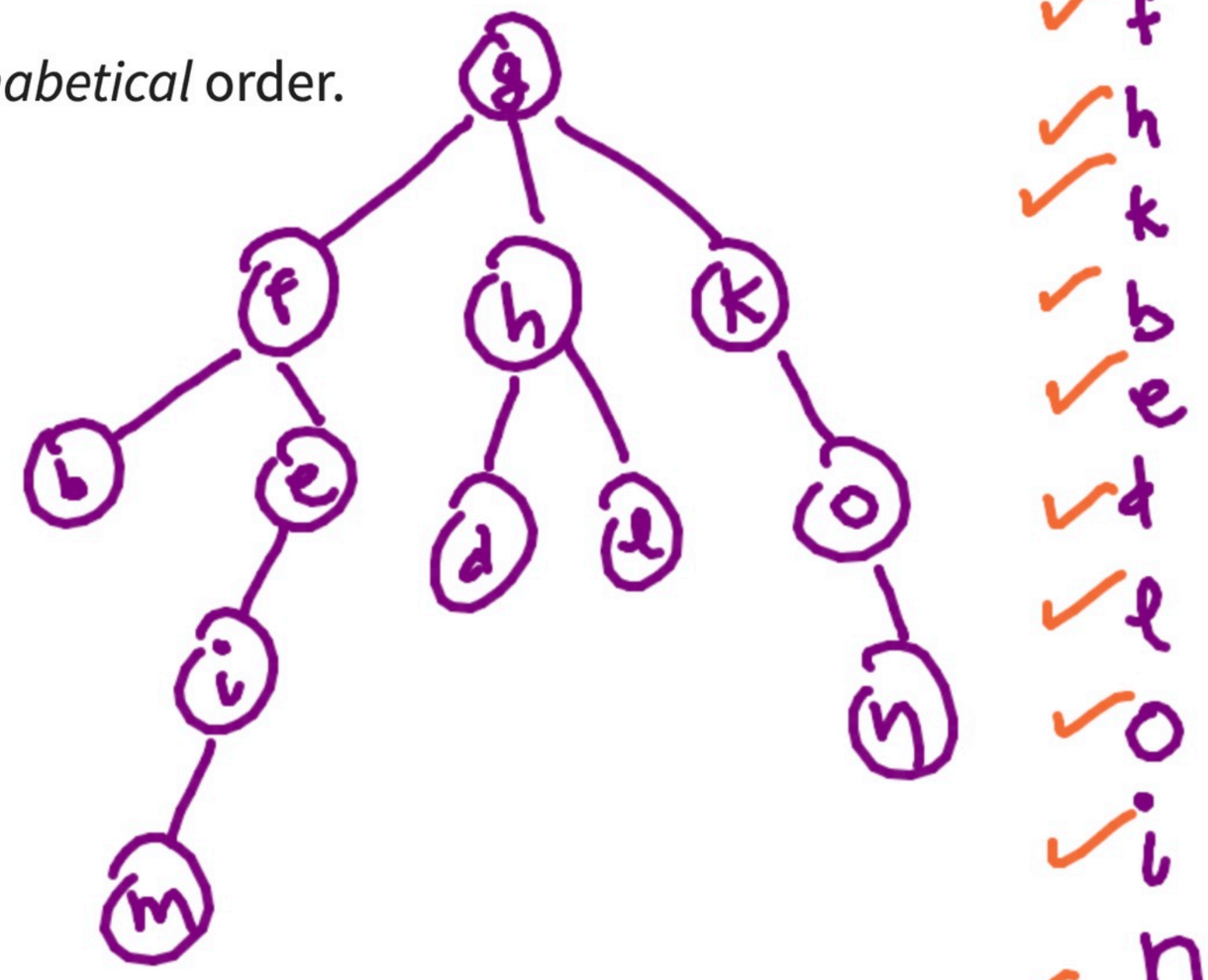
$|E| \approx |V|$ sparse

$|E| \approx |V|^2$ dense

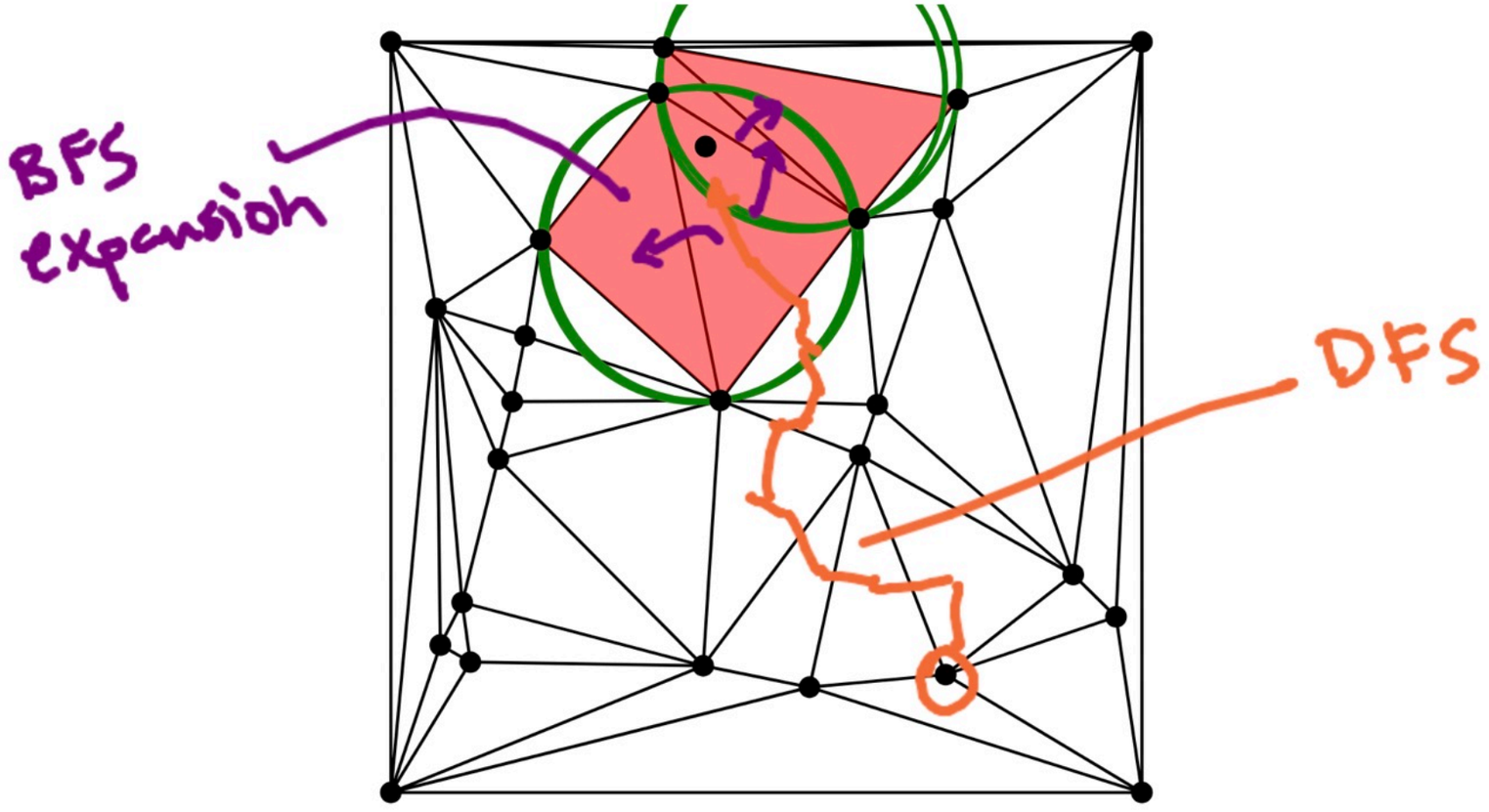
Exercise 3: Build spanning tree of this graph using BFS.

- Start at vertex *g*.
- Visit neighboring vertices in *alphabetical* order.
- List order of vertices visited.





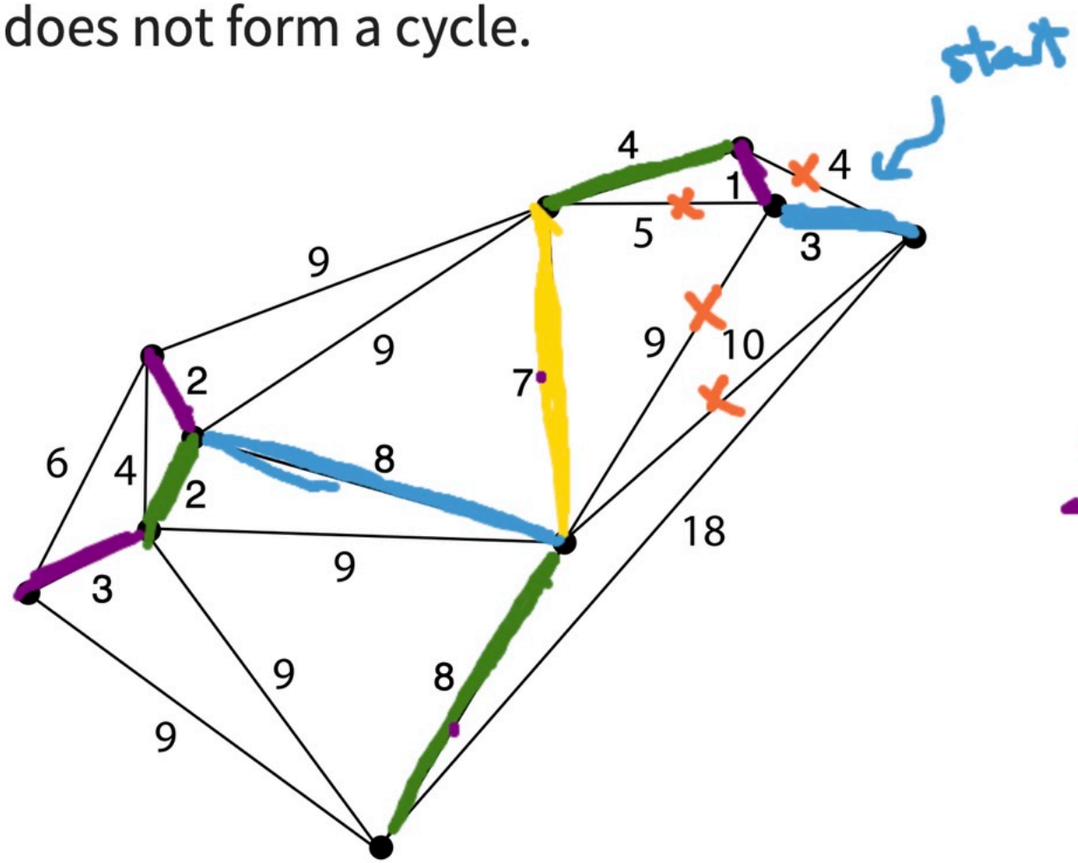
How I use BFS and DFS.



Prim's algorithm for constructing a minimum spanning tree (MST).

Minimum spanning tree: Spanning tree of a graph with smallest sum of edge weights.

Main idea: Add minimum weight edge that is (1) connected to current tree and (2) does not form a cycle.



add up: 38

Finding the shortest path with BFS for an unweighted graph.

Run BFS from source node (root) to target node(s). Shortest path is the path in the resulting spanning tree from the root to the target node(s).