Topics for this week

Backtracking:
- a brute-force technique that finds a solution by trying all possibilities
- related data structures
- pruning to eliminate certain possibilities from the search
- time complexity: usually really bad (exponential)
Problem: Map Coloring

How to color a map so that no two neighboring states have the same color?

Want: the smallest number of colors (to minimize costs)
Problem: Map Coloring

Idea (brute-force): for a fixed number of colors
- try coloring nodes so that no two neighboring nodes have the same color.
	if get stuck (i.e., no available color), then:
		“backtrack” to the previous node and try another color, then try coloring subsequent nodes.
- if no other choices for the previous node, then go back further.

Another map:
- 3 colors
Backtracking

Brute-force via backtracking:

def backtracking(graph, currentNode, partialColoring, k):
  listofColors = [1, ..., k]
  for color in listofColors:
    assign color to currentNode in partialColoring
    nextNode = the nextNode after currentNode in
                the ordering of the nodes
    if nextNode exists:
      backtracking(graph, nextNode, partialColoring, k)
    else:
      if the partial coloring is valid
      if yes, output it

Running time: \( O(k^n \cdot m) \) # colorings

idea: color the current node

outputs all valid colorings

initial partial coloring
\( \rightarrow \) undefined color for each node

k: # colors

this is literally trying all possible colorings independent
of the neighboring restriction
and checks validity at the very end (once all nodes
assigned colors)
def backtrackColorPrune(graph, currentNode, partialColoring, k):
    create listofColors
    for color in listofColors:
        if color does not appear among the currentNode's neighbors:
            assign color to the currentNode in partial coloring
            next node is the node in the order of nodes after currentNode
            if next node exists:
                backtrackColorPrune(graph, next node, partial coloring, k)
            else:
                output the coloring
Backtracking

Testing:

Time complexity: