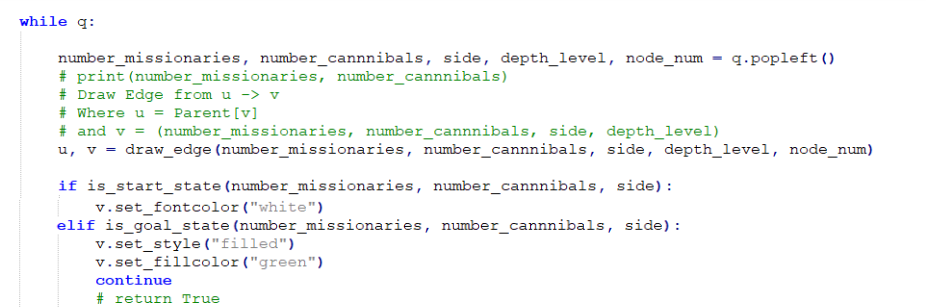
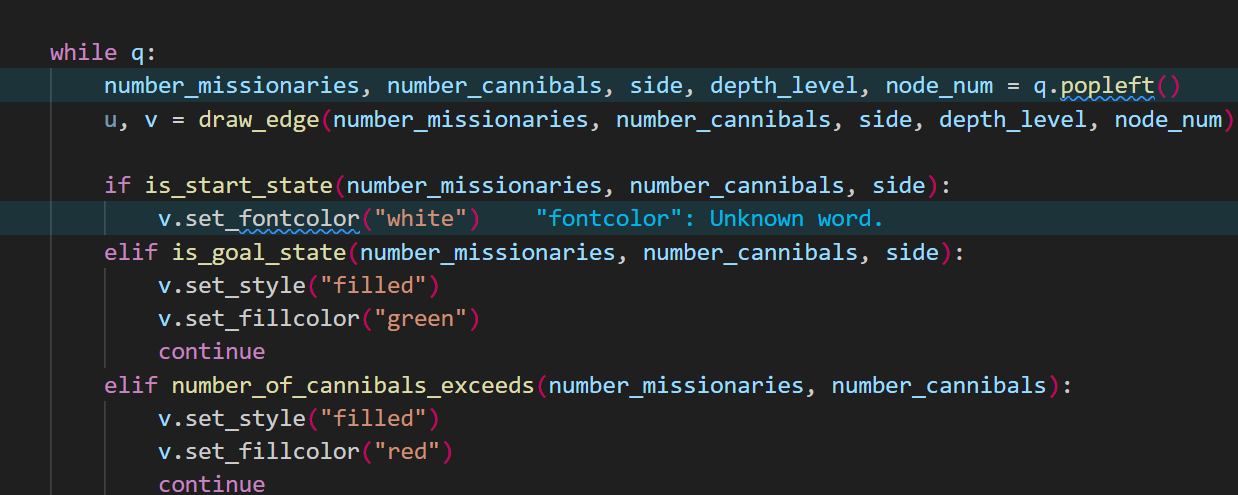
In generate\_full\_space\_tree.py, the error was about we can't find i.  
As this problem, we just easy delete the command **#return True** is done.

The origin is



After fixed:



Next, in solve.py, the code is trying to call get\_label() on a list object. The error is because nodes contain list objects instead of graph nodes as we expect. So, we can fix it by :  
Removed the nodes list since it wasn't being used correctly

* Removed the node styling code since it was causing the error
* Simplified the move tracking and state representation
* Fixed string formatting for better readability
* Added reverse() to show path from start to goal instead of goal to start
* Added safety checks for Move and Parent dictionary lookups

Origin code :



After fixed:



# Explain source code after fix 1.Generate\_full\_space\_tree.py

After fixing, the generate\_full\_space\_tree.py generates a visual representation (state space tree) of the classic "Missionaries and Cannibals" puzzle.

1. The Puzzle Context:
   * The "Missionaries and Cannibals" puzzle involves moving 3 missionaries and 3 cannibals across a river using a boat
   * The goal is to move everyone to the other side
   * Main rule: Cannibals can never outnumber missionaries on either side (or the missionaries are in danger)
2. Key Functionality:
   * The code creates a visual graph showing all possible states and moves in the puzzle
   * Each node represents a state with format (missionaries, cannibals, side)
   * Side is represented as 1 (starting side) or 0 (destination side)
   * The graph is color-coded:
     + Orange: Valid intermediate states
     + Red: Invalid states (where cannibals outnumber missionaries)
     + Green: Goal state (everyone reached other side)
     + Grey: Dead-end states
     + White: Starting state (3,3,1)
3. Implementation Details:
   * Uses pydot to generate the graph visualization
   * Implements breadth-first search (using deque) to explore all possible states
   * Allows setting a maximum depth for the tree via command line argument
   * Generates a PNG file showing the complete state space tree
   * Tracks parent states to build the tree structure
   * Validates moves using rules like:
     + Can't have negative people
     + Can't have more than 3 people on either side
     + Cannibals can't outnumber missionaries
4. Valid Moves:
   * The options list defines possible boat moves: [(1, 0), (0, 1), (1, 1), (0, 2), (2, 0)]
     + First number: missionaries to move
     + Second number: cannibals to move
     + For example, (1,1) means move 1 missionary and 1 cannibal To use this code, you would:
5. Ensure Graphviz is installed (it's required for visualization)
6. Run the script, optionally specifying a depth limit: python generate\_full\_space\_tree.py --depth 10
7. Get a PNG file showing all possible states and transitions in the puzzle

# Solve.py

This code implements a solution to the classic "Missionaries and Cannibals" puzzle using both Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms

The code solves the following puzzle:

* 1. Problem Statement:
     + There are 3 missionaries and 3 cannibals on one side of a river
     + They need to cross to the other side using a boat
     + The boat can carry at most 2 people
     + At no point can cannibals outnumber missionaries on either side (or the cannibals will eat the missionaries)
  2. Key Features:
     + Uses both DFS and BFS search algorithms to find a solution
     + Creates a visual state space tree using pydot\_ng (GraphViz)
     + Displays the solution steps using emoji visualizations
     + Tracks the path to solution using Parent and Move dictionaries
  3. State Representation:
     + Each state is represented as a tuple (m, c, s) where:
       - m: number of missionaries on the left side
       - c: number of cannibals on the left side
       - s: boat side (1 for left, 0 for right)
     + Start state: (3, 3, 1)
     + Goal state: (0, 0, 0)
  4. Available Moves:
     + The possible moves are: (1,0), (0,1), (1,1), (0,2), (2,0)
     + Each tuple represents (missionaries, cannibals) to move
  5. Visualization Features:
     + Creates a colored graph showing the search space
     + Blue: Start node
     + Green: Goal node
     + Red: Invalid states (cannibals eat missionaries)
     + Yellow: Solution path
     + Gray: Dead ends
     + Orange: Expandable nodes
  6. Solution Display:
     + Shows step-by-step solution with emoji representations
     + Displays missionaries as 👴 and cannibals as 👹
     + Shows the river crossing process with a visual representation To use this code, you would:

1. Initialize the Solution class
2. Call the solve method with either "dfs" or "bfs"
3. Use show\_solution() to see the step-by-step solution
4. Use write\_image() to save the state space tree visualization