Project one

COMP338 - Artificial Intelligence

Missionaries and Cannibals Problem

-Student's name:

*Sereen Hamed _ 1200967

*Ziad Masalma_1202199

-Instructor: Dr. Radi Jarrar

-Section: 1

Contents

Background Information:

BFS Algorithm

Breadth First Search (BFS) is a Fundamental search algorithm used to explore nodes and edges of a graph. Its often used as a building block in other algorithms.

The BFS algorithm is particularly useful for **Finding the shortest path on unweighted graphs.**

A BFS algorithm starts at some arbitrary node of a graph and explores the neighbor nodes first, before moving to the next level neighbors.

The BFS algorithm uses **a queue** data structure to track which node to visit next. Upon reaching a new node the algorithm adds it to the queue to visit it later.

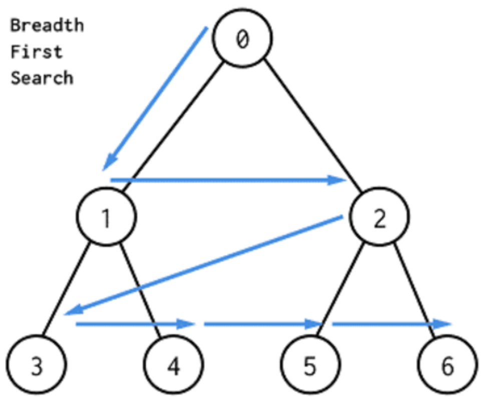
DFS Algorithm

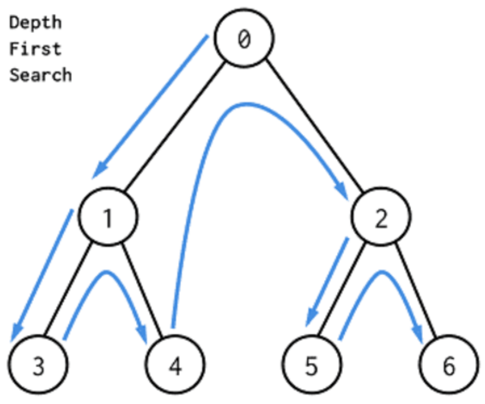
Depth First Search (DFS) is the most Fundamental search algorithm used to explore nodes and edges of a graph. It's often used as a building block in other algorithms too.

By itself the DFS algorithm isn't all that useful, but when **augmented to perform other tasks such as Count connected components, determine connectivity, or find bridges/articulation points** then DFS really shines.

As the name suggests, a DFS plugins **depth first into** a graph without regard for which edge it takes next until it cannot go any further at which point it backtracks and continues.

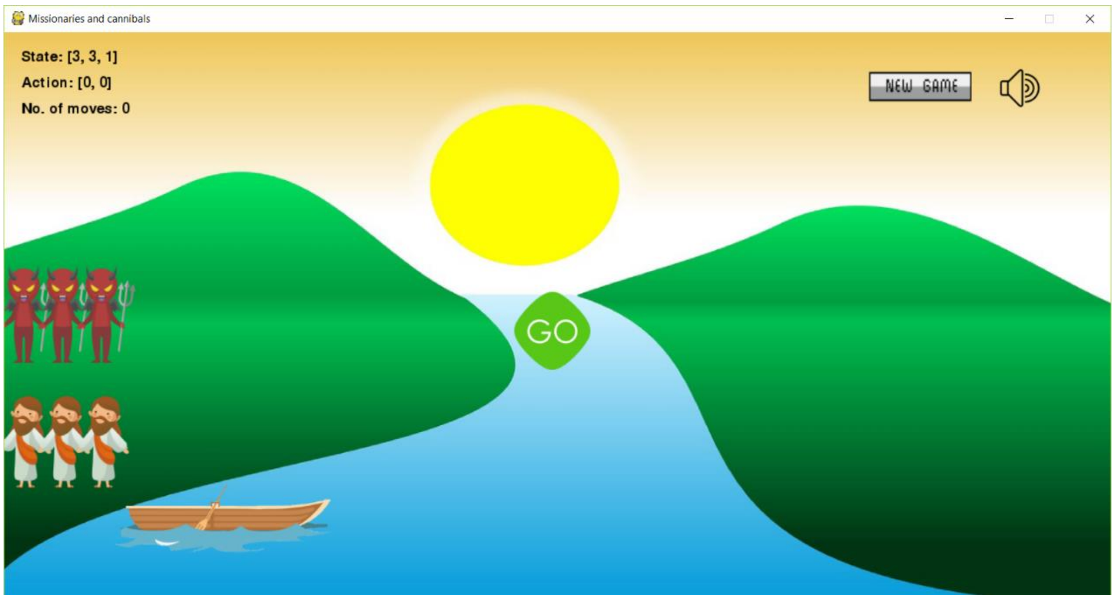
DFS, unlike BFS, delves deeply into a graph's nodes, prioritizing exploration along a path before backtracking. It uses **a stack** or recursion to navigate, focusing on depth-first traversal, beneficial for tasks like sorting, cycle detection, and maze solving in graphs.





Explains the Missionaries and Cannibals problem.

A famous brainteaser, the Missionaries and Cannibals issue has three missionaries and three cannibals stranded on one side of a river. They will need to use a boat that can accommodate no more than two people to get across to the opposite side. To prevent the missionaries from being eaten, it is necessary to design a sequence of crossings that guarantee they are never outnumbered by the cannibals on either side of the river. The goal is to get all six of them over the river while staying within the rules—that is, without having more cannibals than missionaries on either side at any given time and using the boat's restricted capacity.



State: [3, 3, 1]
Action: [0, 0]
No. of moves: 0

problem formulation for the mentioned problem.

- **Initial state: (3,3,1,0,0)**
  Three missionaries and three cannibals start on the left side (starting side) of the river and the boat also. The other side is empty.

- **Goal test: (0,0,0,3,3)**
  when all missionaries and cannibals are on the right side (opposite side) of the river.

- **Actions:**
  The available actions are moving individuals and the boat across the river like that:
  - Move 1 missionary and 0 cannibals from one side to the other.
  - Move 0 missionaries and 1 cannibal from one side to the other.

  - Move 1 missionary and 1 cannibal from one side to the other.

- **Transition Model:**
  The **“is_valid_state(state)”** method in our code verifies the validity of a state by examining constraints like:
  - restrictions on the number of cannibals and missionaries on each side, ranging from 0 to 3.
  - safety measures to keep missionaries from being attacked by cannibals.
  - The boat should be positioned either on the left, zero, or one (1).

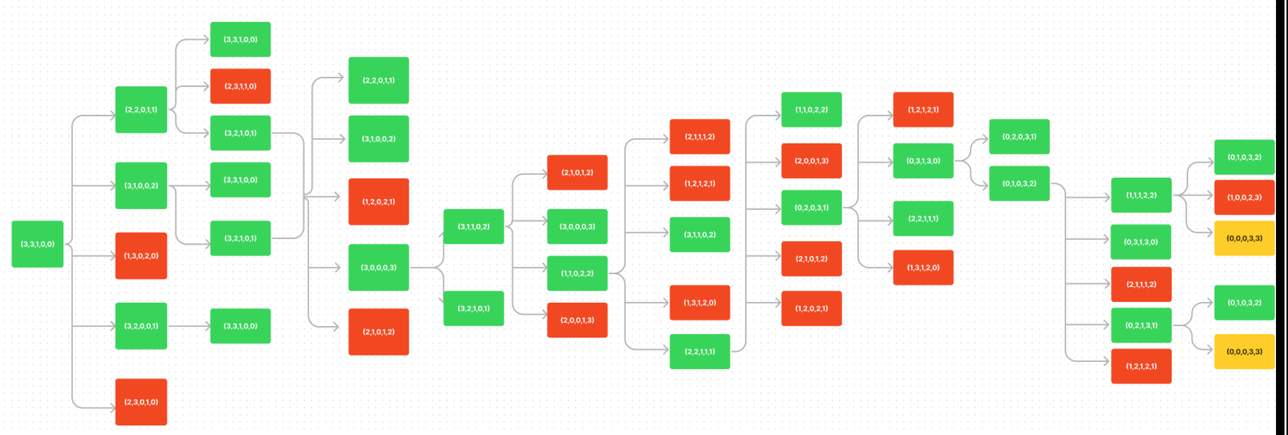
  The **“generate_next_states(state)”** function determines the legal states that can be reached from a specified state. It creates new states by considering limitations and valid actions.

  Transferring one or two missionaries, cannibals, or both from one side to the other is one possible course of action.

  There can only be two people on the boat at a time.

- **Path cost**: Each valid action (moving individuals and the boat) from one state to another represents one step or one unit of cost. We have in the two algorithm **(12 steps).**

A drawing of the complete search space (state space)

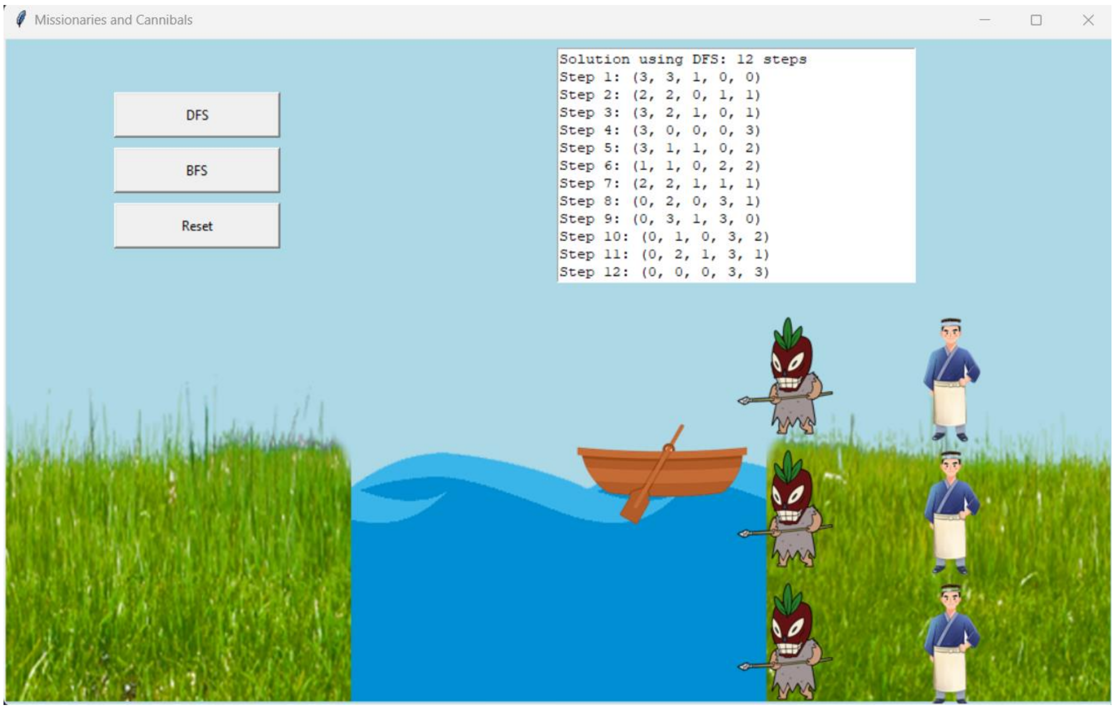


This is the link of this graph:

https://www.figma.com/file/w4BVgQm26sWG3JsuKg5Aw4/Untitled?type=whiteboard&node-id=0%3A1&t=WnhbkhTZOSCleHds-1

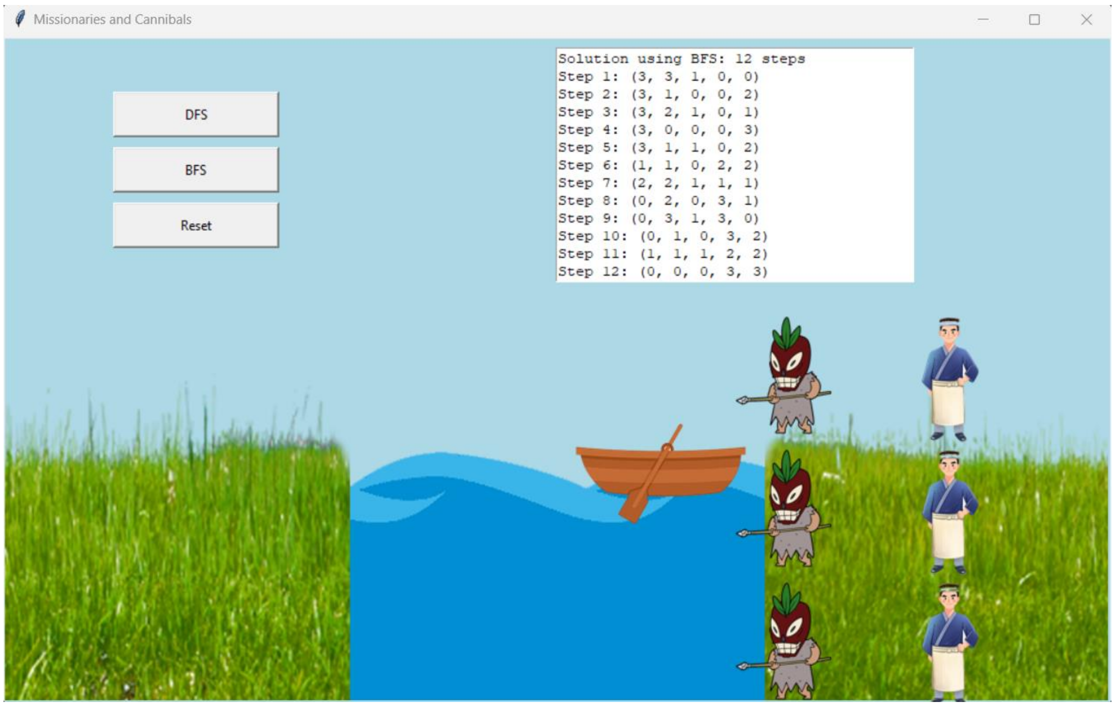
Present the optimal solution.

Solution using DFS algorithm.



```
Solution using DFS: 12 steps
Step 1: (3, 3, 1, 0, 0)
Step 2: (2, 2, 0, 1, 1)
Step 3: (3, 2, 1, 0, 1)
Step 4: (3, 0, 0, 0, 3)
Step 5: (3, 1, 1, 0, 2)
Step 6: (1, 1, 0, 2, 2)
Step 7: (2, 2, 1, 1, 1)
Step 8: (0, 2, 0, 3, 1)
Step 9: (0, 3, 1, 3, 0)
Step 10: (0, 1, 0, 3, 2)
Step 11: (0, 2, 1, 3, 1)
Step 12: (0, 0, 0, 3, 3)
```

Solution using BFS algorithm.



```
Solution using BFS: 12 steps
Step 1: (3, 3, 1, 0, 0)
Step 2: (3, 1, 0, 0, 2)
Step 3: (3, 2, 1, 0, 1)
Step 4: (3, 0, 0, 0, 3)
Step 5: (3, 1, 1, 0, 2)
Step 6: (1, 1, 0, 2, 2)
Step 7: (2, 2, 1, 1, 1)
Step 8: (0, 2, 0, 3, 1)
Step 9: (0, 3, 1, 3, 0)
Step 10: (0, 1, 0, 3, 2)
Step 11: (1, 1, 1, 2, 2)
Step 12: (0, 0, 0, 3, 3)
```