A

COMP338 - Artificial Intelligence

Project #1: Missionaries and Cannibals Problem

Dr. Radi Jarrar

By : Bader Manasra & Yazan Zidan

Bader ID : 1193186

Yazan ID : 1193162

Introduction

Problem description: Missionaries and Cannibals is a common problem in the literature of AI, it is described as follows: assume three missionaries are accompanied by three cannibals, and all six of them have reached one side of a river and they are to pass to the other side. The problem is, there is only one boat that can fit only two persons, and since cannibals might eat missionaries, missionaries should not be outnumbered by cannibals at any point and at any side of the river.

We can solve this problem by using :

1-Best First Search:

The best first search uses the concept of a priority queue and heuristic search. It is a search algorithm that works on a specific rule. The aim is to reach the goal from the initial state via the shortest path. The best First Search algorithm in artificial intelligence is used for for finding the shortest path from a given starting node to a goal node in a graph. The algorithm works by expanding the nodes of the graph in order of increasing the distance from the starting node until the goal node is reached.

2-Depth First Search:

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. It uses last in- first-out strategy and hence it is implemented using a stack.

First : How we use BFS in Missionaries and Cannibals problem?

In other meaning how we do implementation?

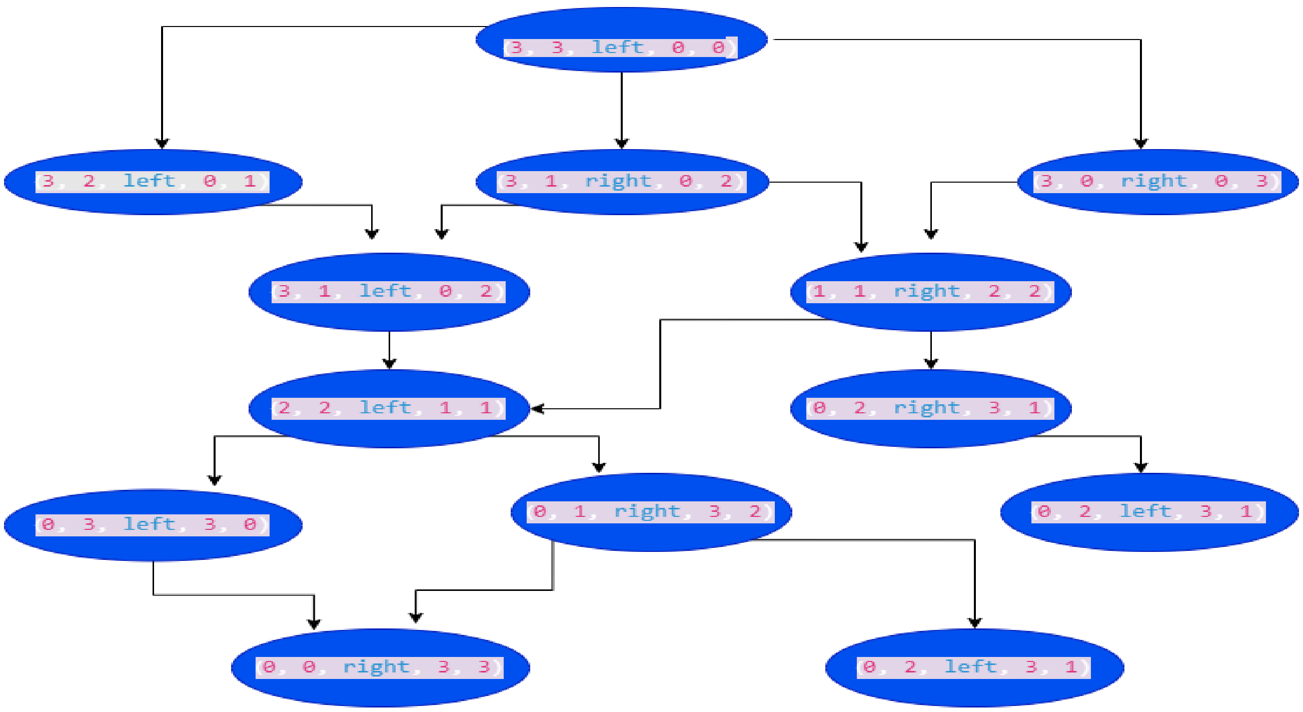
We use BFS algorithm to find solutions . we initialize the initial state, adds it to the queue, and starts exploring states until the goal state is reached or the maximum number of solutions is found.

State Processing : to generate and process valid next states from the current state. It checks if a state is valid, generates next states, and adds them to the queue if they are not visited.

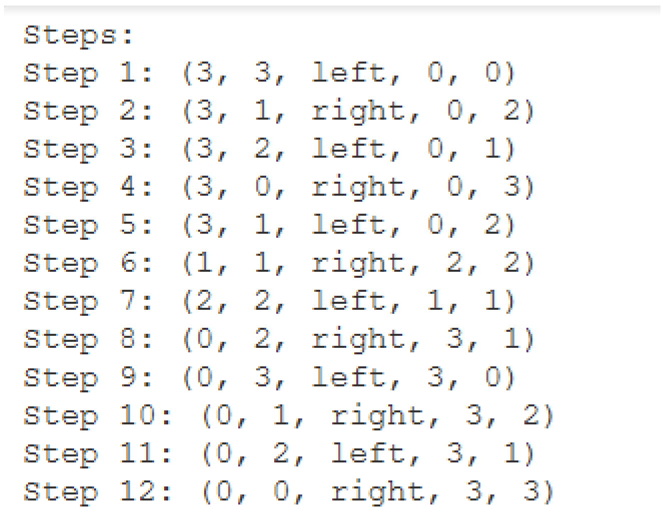
State Validation : to check if a state is valid considering the constraints of the problem. It ensures that the number of missionaries on one side is greater than or equal to the number of cannibals unless there are no missionaries on that side.

State Generation : to generates possible next states based on the current state and constraints of the problem.

drawing of the complete search space:



How output should be in BFS:



```
Steps:
Step 1: (3, 3, left, 0, 0)
Step 2: (3, 1, right, 0, 2)
Step 3: (3, 2, left, 0, 1)
Step 4: (3, 0, right, 0, 3)
Step 5: (3, 1, left, 0, 2)
Step 6: (1, 1, right, 2, 2)
Step 7: (2, 2, left, 1, 1)
Step 8: (0, 2, right, 3, 1)
Step 9: (0, 3, left, 3, 0)
Step 10: (0, 1, right, 3, 2)
Step 11: (0, 2, left, 3, 1)
Step 12: (0, 0, right, 3, 3)
```

Second: How we use DFS in Missionaries and Cannibals problem?

This Java code implements a solution to the classic Missionaries and Cannibals problem using Depth-First Search (DFS). The problem involves three missionaries and three cannibals on one side of a river, and they need to cross the river using a boat that can carry at most two people. The goal is to move all missionaries and cannibals to the other side of the river without ever having more cannibals than missionaries on either side, as the cannibals would eat the missionaries.

The main DFS algorithm to initialize the initial state, pushes it onto a stack, and then iteratively explores states until the goal is reached or the maximum number of solutions is found.

We implement some methods to solve this problem:

Methods:

1-solve(): The main DFS algorithm. It initializes the initial state, pushes it onto a stack, and then iteratively explores states until the goal is reached or the maximum number of solutions is found.

2-getSolutionString(State goalState): Reconstructs and returns the solution path as a string.

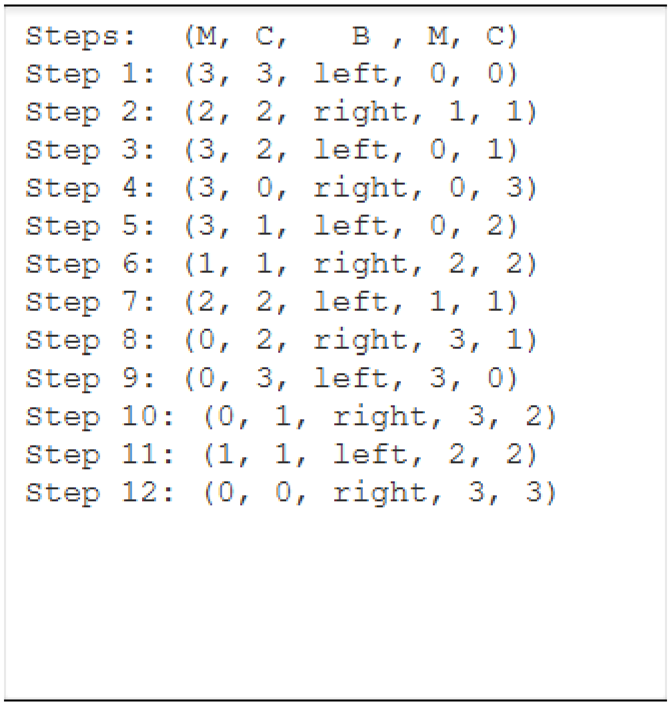
3-isValidIntermediateState(State state): Checks if an intermediate state is valid, considering the constraint of not having more cannibals than missionaries on either side.

4-generateNextStates(State currentState): Generates possible next states from the current state, considering the boat's capacity and the constraints of the problem.

5-isValidState(int missionaries, int cannibals): Checks if a state is valid (within the problem's constraints).

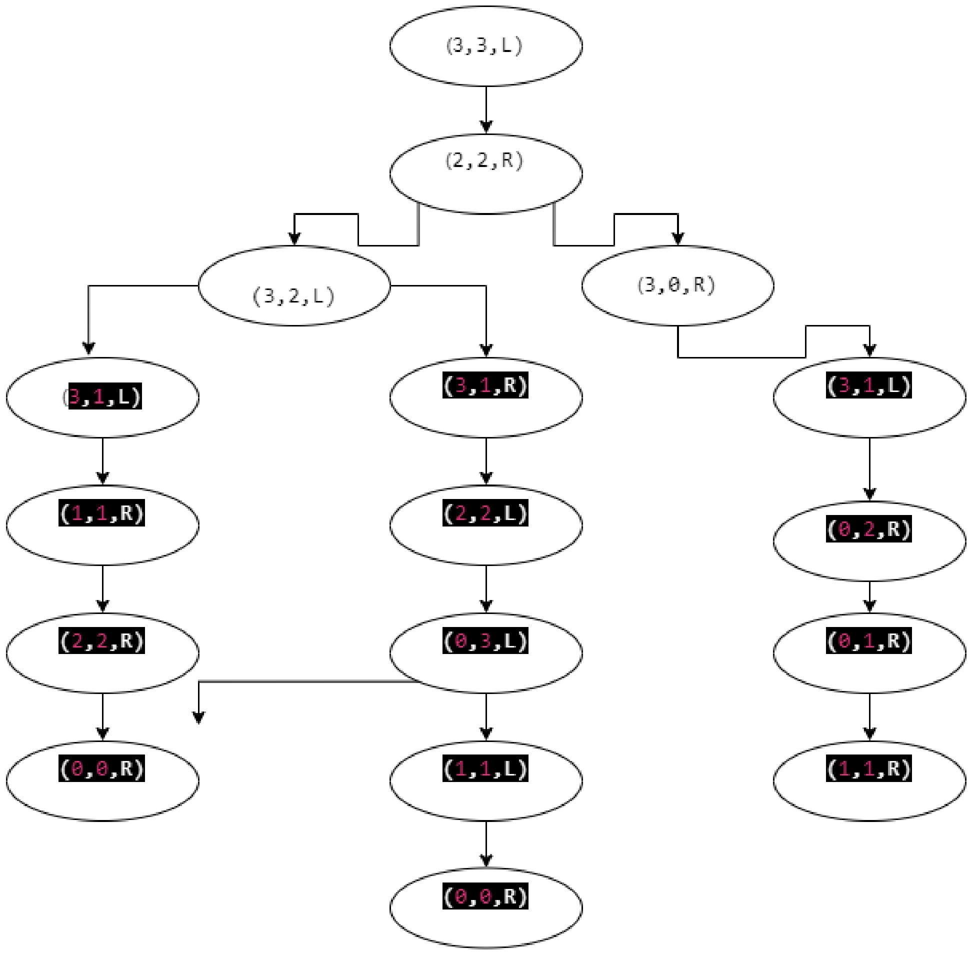
6-stateIsGoal(State state): Checks if the given state is the goal state, where all missionaries and cannibals have crossed to the other side.

How output should be in DFS:



```
Steps:   (M, C,   B , M, C)
Step 1: (3, 3, left, 0, 0)
Step 2: (2, 2, right, 1, 1)
Step 3: (3, 2, left, 0, 1)
Step 4: (3, 0, right, 0, 3)
Step 5: (3, 1, left, 0, 2)
Step 6: (1, 1, right, 2, 2)
Step 7: (2, 2, left, 1, 1)
Step 8: (0, 2, right, 3, 1)
Step 9: (0, 3, left, 3, 0)
Step 10: (0, 1, right, 3, 2)
Step 11: (1, 1, left, 2, 2)
Step 12: (0, 0, right, 3, 3)
```

drawing of the complete search space:


(3,3,L)
(2,2,R)
(3,2,L)
(3,0,R)
(3,1,L)
(3,1,R)
(3,1,L)
(1,1,R)
(2,2,L)
(0,2,R)
(2,2,R)
(0,3,L)
(0,1,R)
(0,0,R)
(1,1,L)
(1,1,R)
(0,0,R)