Wolf, Cabbage, and Goat Problem

**PEAS Assessment:**

Performance:

- Legality: the goal must be reached without violating the constraints listed in the problem. The goat cannot be left alone with the wolf or the cabbage.
- Efficiency: An optimal solution must be found, that is to reach the goal state in the fewest number of moves possible.

Environment:

- Two river banks, boat, rules, farmer

Properties:

- Actuators: row with wolf, row with goat, row with cabbage, row alone
- Sensors: Location of the farmer, location of the wolf, goat, and cabbage

**State-Space-Model:**

Initial State:

- The farmer, wolf, goat, and cabbage are all on the left bank, the right bank is empty. → {F, W, G, C | }

Actions:

- move() → move(wolf), move(goat), move(cabbage), move(self)

Transition Model:

- Moving the boat containing an entity or alone from one bank to the other.

Goat Test:

- All entities are uneaten on the right bank of the river. → { | F, W, G, C}

Path Cost:

- Each action has a cost of 1, and the total path cost is equal to the number of trips taken to reach the goal.

**Results Tables:**

| Metric | Result |
|---|---|
| Domain | WGC |
| Algorithm | BFS |
| Solution cost | 7 |
| Depth | 7 |
| Nodes generated | 13 |
| Nodes expanded | 11 |
| Max frontier | 2 |

| Metric | Result |
|---|---|
| Domain | WGC |
| Algorithm | IDS |
| Solution cost | 7 |
| Depth | 7 |
| Nodes generated | 78 |
| Nodes expanded | 35 |
| Max frontier | 10 |

**Analysis and Insight**

In the context of the Wolf, Goat, Cabbage problem, the BFS algorithm seems to be the most straightforward and efficient approach. With the ability to detect and avoid repeated states, it

only had to generate 13 nodes to find the solution. The max frontier size of two is a reflection of the small problem it was faced with, as it only had to hold two nodes at a time. If this was a larger problem, each layer could be much larger and the frontier would have to hold many more nodes. It is in a case like this that the IDS would likely become more memory efficient, holding the nodes of one path in the frontier rather than the entire layer. In this case, it seems the main drawback was the redundancy of having to generate all of the nodes on layers 1-6 each iteration until the solution at depth 7 was found.

**Git Repo:**

https://github.com/joeoneal/cs4300/tree/b03bc98c927611885bb3067cef0415022d0a49a3/a1

**Addendum: Raw Text Outputs**

**BFS)**

Domain: WGC | Algorithm: BFS

Solution Cost: 7 | Depth: 7

Nodes generated: 13 | Nodes expanded: 11 | Max frontier: 2

State key: [Wolf, Goat, Cabbage, Farmer]

Path:

1) Move Goat (0, 0, 0, 0) --> (0, 1, 0, 1)
2) Move Alone (0, 1, 0, 1) --> (0, 1, 0, 0)
3) Move Wolf (0, 1, 0, 0) --> (1, 1, 0, 1)
4) Move Goat (1, 1, 0, 1) --> (1, 0, 0, 0)
5) Move Cabbage (1, 0, 0, 0) --> (1, 0, 1, 1)
6) Move Alone (1, 0, 1, 1) --> (1, 0, 1, 0)
7) Move Goat (1, 0, 1, 0) --> (1, 1, 1, 1)

**IDS)**

Domain: WGC | Algorithm: IDS

Solution Cost: 7 | Depth: 7

Nodes generated: 78 | Nodes expanded: 35 | Max frontier: 10

State key: [Wolf, Goat, Cabbage, Farmer]

Path:

1) Move Goat (0, 0, 0, 0) --> (0, 1, 0, 1)
2) Move Alone (0, 1, 0, 1) --> (0, 1, 0, 0)
3) Move Wolf (0, 1, 0, 0) --> (1, 1, 0, 1)
4) Move Goat (1, 1, 0, 1) --> (1, 0, 0, 0)
5) Move Cabbage (1, 0, 0, 0) --> (1, 0, 1, 1)
6) Move Alone (1, 0, 1, 1) --> (1, 0, 1, 0)

7) Move Goat (1, 0, 1, 0) --> (1, 1, 1, 1)