Implementation Final

November 10, 2020

0.0.1 PART 2

Implement and solve the problem optimally using an appropriate search algorithm.

[14]:
```
import copy

class CoastState:

    def __init__(self, c, m):
        self.cannibals = c
        self.missionaries = m

    # This is an intermediate state of Coast where the missionaries have to␣
 ↪outnumber the cannibals
    def valid_coast(self):
        if self.missionaries >= self.cannibals or self.missionaries == 0:
            return True
        else:
            return False

    def goal_coast(self):
        if self.cannibals == 3 and self.missionaries == 3:
            return True
        else:
            return False


class GameState:

    def __init__(self, data):
        self.data = data
        self.parent = None

    # Creating the Search Tree
    def building_tree(self):
        children = []
        coast = ""
        across_coast = ""
        temp = copy.deepcopy(self.data)
```

```
        # Intial starting point either Left or Right
        if self.data["boat"] == "left":
            coast = "left"
            across_coast = "right"
        elif self.data["boat"] == "right":
            coast = "right"
            across_coast = "left"

        # MOVING 2 CANNIBALS (CC)
        if temp[coast].cannibals >= 2:
            temp[coast].cannibals = temp[coast].cannibals - 2
            temp[across_coast].cannibals = temp[across_coast].cannibals + 2
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 2 MISSIONARIES (MM)
        if temp[coast].missionaries >= 2:
            temp[coast].missionaries = temp[coast].missionaries - 2
            temp[across_coast].missionaries = temp[across_coast].missionaries +␣
↪2
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 1 CANNIBAL (C)
        if temp[coast].cannibals >= 1:
            temp[coast].cannibals = temp[coast].cannibals - 1
            temp[across_coast].cannibals = temp[across_coast].cannibals + 1
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 1 MISSIONARY (M)
        if temp[coast].missionaries >= 1:
            temp[coast].missionaries = temp[coast].missionaries - 1
            temp[across_coast].missionaries = temp[across_coast].missionaries +␣
↪1
```

```
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 1 CANNIBAL AND 1 MISSIONARY (CM && MM)
        if temp[coast].missionaries >= 1 and temp[coast].cannibals >= 1:
            temp[coast].missionaries = temp[coast].missionaries - 1
            temp[across_coast].missionaries = temp[across_coast].missionaries +␣
↪1
            temp[coast].cannibals = temp[coast].cannibals - 1
            temp[across_coast].cannibals = temp[across_coast].cannibals + 1
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)
        return children


def breadth_first_search():
    left = CoastState(3, 3)
    right = CoastState(0, 0)
    root_data = {"left": left, "right": right, "boat": "left"}

    explored = []
    nodes = []
    path = []
    nodes.append(GameState(root_data))

    while len(nodes) > 0:
        g = nodes.pop(0)
        explored.append(g)
        if g.data["right"].goal_coast():
            path.append(g)
            return g
        else:
            next_children = g.building_tree()
            for x in next_children:
                if (x not in nodes) or (x not in explored):
                    nodes.append(x)
    return None


def print_path(g):
```

```
    path = [g]
    while g.parent:
        g = g.parent
        path.append(g)
    print("                  " + "Left Side" + "                        " + "Right␣
↪Side" + "                    " + "Boat ")
    print(
        "          Cannibals" + "     Missionaries" + "       " + "Cannibals" +␣
↪"       Missionaries" + "    Boat Position")
    counter = 0
    for p in reversed(path):
        print("State " + str(counter) + "    Left C: " + str(p.data["left"].
↪cannibals) + ".    Left M: " + str(
            p.data["left"].missionaries) + ".     |   Right C: " + str(
            p.data["right"].cannibals) + ".     Right M: " + str(p.
↪data["right"].missionaries) + ".     | Boat: " + str(
            p.data["boat"]))
        counter = counter + 1
        print("The counter: ",counter)



def main():
    solution = breadth_first_search()
    print("Using Breath - First Search:")
    print_path(solution)



if __name__ == "__main__":
    main()
```

```
Using Breath - First Search:
                 Left Side                        Right Side
Boat
          Cannibals     Missionaries       Cannibals       Missionaries    Boat
Position
State 0    Left C: 3.    Left M: 3.     |   Right C: 0.     Right M: 0.     |
Boat: left
The counter:  1
State 1    Left C: 1.    Left M: 3.     |   Right C: 2.     Right M: 0.     |
Boat: right
The counter:  2
State 2    Left C: 2.    Left M: 3.     |   Right C: 1.     Right M: 0.     |
Boat: left
The counter:  3
```

```
State 3    Left C: 0.    Left M: 3.     |   Right C: 3.     Right M: 0.     |
Boat: right
The counter:  4
State 4    Left C: 1.    Left M: 3.     |   Right C: 2.     Right M: 0.     |
Boat: left
The counter:  5
State 5    Left C: 1.    Left M: 1.     |   Right C: 2.     Right M: 2.     |
Boat: right
The counter:  6
State 6    Left C: 2.    Left M: 2.     |   Right C: 1.     Right M: 1.     |
Boat: left
The counter:  7
State 7    Left C: 2.    Left M: 0.     |   Right C: 1.     Right M: 3.     |
Boat: right
The counter:  8
State 8    Left C: 3.    Left M: 0.     |   Right C: 0.     Right M: 3.     |
Boat: left
The counter:  9
State 9    Left C: 1.    Left M: 0.     |   Right C: 2.     Right M: 3.     |
Boat: right
The counter:  10
State 10    Left C: 2.    Left M: 0.     |   Right C: 1.     Right M: 3.     |
Boat: left
The counter:  11
State 11    Left C: 0.    Left M: 0.     |   Right C: 3.     Right M: 3.     |
Boat: right
The counter:  12
```

[ ]:

[ ]: