21110324 – Lương Đăng Khôi

*Note: Để thuận tiện hơn người và quỷ sẽ được đổi cách gọi thành người và quái.

** File đầu tiên: generate_full_space_tree.py*

```
options = [(1, 0), (0, 1), (1, 1), (0, 2), (2, 0)]
```

- options là list chứa chứa các tupple là tất cả các trạng thái của con thuyền (x, y) với x là số người, và y là số quái và thuyền chỉ có 2 vị trí nên tổng x và y không được lớn hơn 2.

```
Parent = dict()
```

- Parent có kiểu dữ liệu dictionary để lưu trữ các node cha, tạm thời dict này đang trống.

```
graph = pydot.Dot(graph_type='graph',strict=False,
bgcolor="#fff3af",
label="fig: Missionaries and Cannibal State Space Tree",
fontcolor="red", fontsize="24", overlap="true")
```

- graph là đồ thị được tạo bởi thư viện pydot với các thuộc tính: graph_type là loại đồ thị(ở trên là dạng graph thông thường), strict là sự chặt chẽ của cú pháp(ở code trên thì cú pháp không chặt chẽ), bgcolor là màu nền(mã màu được chọn là #fff3af, đây là màu được biểu diễn dạng HEX), label là tên của đồ thị(ở trên tiêu đề được đặt là fig: Missonaries and Cannibal State Space Tree), fontcolor là màu chữ(màu được chọn là đỏ, có mã màu trong hệ rgb là (255, 0, 0)), fontsize là cỡ chữ(cỡ chữ trên là 24), overlap là tham số biểu thị sự chồng chéo của các node(được đặt là true, tức cho phép điều này diễn ra).

```
arg = argparse.ArgumentParser()
arg.add_argument("-d", "--depth", required=False,
                 help="MAximum depth upto which you want to
generate Space State Tree")
```

- Tạo một đối tượng ArgumentParser và gán nó cho biến arg. Đối tượng này sẽ được sử dụng để quản lý các đối số dòng lệnh.
- Sau đó thêm một số dòng lệnh cho chương trình cụ thể, đối số có các thuộc tính sau:
+ '-d' và '--depth' là các tuỳ chọn để chỉ định đối số và d nghĩa là depth.
+ required là chỉ định bắt buộc của đối số, ở trên là False nghĩa là không bắt buộc tức chương trình hoàn toàn có thể chạy là không có cần được cung cấp giá trị cho đối số này.
+help là mô tả giúp hiểu rõ ý nghĩa của đối số. Ở trên, đối số này được sử dụng để chỉ định một giá trị số nguyên, là độ sâu tối đa cho việc tạo ra Space State Tree.

```
args = vars(arg.parse_args())
```

- Dòng này để phân tíchvà lấy ra giá trị của các đối số dòng lệnh mà bạn đã cung cấp khi chạy chương trình. Kết quả là một đối tượng chứa thông tin về các đối số dưới dạng các cặp "tên đối số - giá trị". Bằng cách sử dụng vars(), nó được chuyển đổi thành một từ điển (dictionary) Python, nơi bạn có thể truy cập giá trị của các đối số bằng tên của chúng.

```
max_depth = int(args.get("depth", 20))
```

- Dòng này nhằm mục đích lấy giá trị của đối số "depth" từ từ điển args. Hàm .get("depth", 20) sẽ thử lấy giá trị của "depth" nếu nó tồn tại trong từ điển, nếu không thì sẽ trả về giá trị mặc định là 20. Sau đó, giá trị này được chuyển đổi thành một số nguyên bằng cách sử dụng int(), và kết quả được gán vào biến max_depth.
- Ta đang cố lấy giá trị của đối số "depth" từ các đối số dòng lệnh mà người dùng cung cấp khi chạy chương trình. Nếu đối số "depth" không được cung cấp, giá trị mặc định là 20 sẽ được sử dụng cho biến max_depth.

Các bước chuẩn bị đã hoàn tất. Ta phân tích tiếp hàm main để biết chương trình hoạt động như thế nào và các hàm nào được gọi đầu tiên, chức năng của các hàm đó.

```
if __name__ == "__main__":
    if generate():
        write_image()
```

- main gọi hàm generate() đầu tiên và tuỳ thuộc vào giá trị trả về của generate mà có quyết định tiếp hàm write_image() hay không.

Ở hàm generate:

```
global i
q = deque()
node_num = 0
q.append((3, 3, 1, 0, node_num))

Parent[(3, 3, 1, 0, node_num)] = None
```

- Đầu tiên gọi biến toàn cục i
- Khởi tạo hàng đợi q
- node_num là biến chứa giá trị số thứ tự các node trong cây
- Đưa trạng thái ban đầu (3 người, 3 quái) vào q và trạng thái ban đầu có đánh dấu bờ xuất phát là 1 và độ sâu 0.
-Parent[(3, 3, 1, 0, node_num)] = None: Dùng để lưu trữ thông tin về node cha của trạng thái ban đầu. Ban đầu, node cha của trạng thái ban đầu là None.

Tiếp theo ta chạy vòng lặp while với điều kiện dừng là q rỗng.

```
            number_missionaries, number_cannnibals, side,
depth_level, node_num = q.popleft()
            # print(number_missionaries, number_cannnibals)
            # Draw Edge from u -> v
            # Where u = Parent[v]
            # and v = (number_missionaries, number_cannnibals,
side, depth_level)
            u, v = draw_edge(number_missionaries,
number_cannnibals, side, depth_level, node_num)

            if is_start_state(number_missionaries,
number_cannnibals, side):
                v.set_style("filled")
```

```
                v.set_fillcolor("blue")
                v.set_fontcolor("white")
            elif is_goal_state(number_missionaries,
number_cannnibals, side):
                v.set_style("filled")
                v.set_fillcolor("green")
                continue
                # return True
            elif
number_of_cannibals_exceeds(number_missionaries,
number_cannnibals):
                v.set_style("filled")
                v.set_fillcolor("red")
                continue
            else:
                v.set_style("filled")
                v.set_fillcolor("orange")

            if depth_level == max_depth:
                return True

            op = -1 if side == 1 else 1

            can_be_expanded = False

            # i = node_num
            for x, y in options:
                next_m, next_c, next_s = number_missionaries +
op * x, number_cannnibals + op * y, int(not side)

                if Parent[(number_missionaries,
number_cannnibals, side, depth_level, node_num)] is None
or(next_m, next_c, next_s)\
                != Parent[(number_missionaries,
number_cannnibals, side, depth_level, node_num)][:3]:
                    if is_valid_move(next_m, next_c):
                        can_be_expanded = True
                        i += 1
                        q.append((next_m, next_c, next_s,
depth_level + 1, i))
                        # Keep track of parent
                        Parent[(next_m, next_c, next_s,
depth_level + 1, i)] =\
```

```
                        (number_missionaries,
number_cannnibals, side, depth_level, node_num)
            if not can_be_expanded:
                v.set_style("filled")
                v.set_fillcolor("gray")
        return False
```

- Trong vòng lặp chương trình lấy trạng thái từ đầu hàng đợi và xử lý.
- Hàm draw_edge được gọi để vẽ các cạnh trong cây trạng thái và cập nhật node hiện tại (u) và node tiếp theo (v).
- Tiếp theo, các điều kiện được kiểm tra để xác định màu sắc của node trong biểu đồ:
+ Nếu đây là trạng thái xuất phát, nó được tô màu xanh.
+ Nếu đây là trạng thái kết thúc, nó được tô màu xanh lá cây.
+ Nếu trạng thái không hợp lệ (ví dụ: số người quái vượt quá số người), nó được tô màu đỏ.
+ Nếu không thuộc các trường hợp trên, nó được tô màu cam.
- Công việc trên được thực hiện bởi cụm câu lệnh điều kiện trước vòng lặp for và các hàm con có tên và nhiệm vụ như sau:
+ *is_start_state(number_missionaries, number_cannibals, side)*: Hàm này kiểm tra xem một trạng thái có phải là trạng thái xuất phát không. Trạng thái xuất phát được định nghĩa là (3 người, 3 quái) ở bờ xuất phát (side = 1).
+ *is_goal_state(number_missionaries, number_cannibals, side)*: Hàm này kiểm tra xem một trạng thái có phải là trạng thái kết thúc không. Trạng thái kết thúc được định nghĩa là (0 thám hiểm, 0 người ăn thịt) ở bờ đích (side = 0).
+ *number_of_cannibals_exceeds(number_missionaries, number_cannibals)*: Hàm này kiểm tra xem có bất kỳ trạng thái nào trong đó số người ăn thịt vượt quá số thám hiểm hay không. Nếu có, nó trả về True, ngược lại trả về False.
- Nếu depth_level đạt đến max_depth, chương trình trả về true kết thúc quá trình tạo cây.
- Biến op được sử dụng để xác định bờ đối diện (1 hoặc 0) để di chuyển.
- Vòng lặp for x, y in options: xử lý các tùy chọn di chuyển. Nó kiểm tra xem di chuyển có hợp lệ và có thể mở rộng trạng thái không. Nếu có thể mở rộng, trạng thái mới được thêm vào hàng đợi và thông tin về node cha của trạng thái mới được cập nhật trong Parent.
- Nếu không có cách mở rộng nào, node hiện tại được tô màu xám, và vòng lặp tiếp tục.
- Nếu không còn trạng thái nào để duyệt (tất cả trạng thái đều đã được kiểm tra), chương trình trả về False để thể hiện rằng không có giải pháp tới max_depth.

** Tiếp theo ta phân tích cả 2 file còn lại gồm main.py và solve.py do 2 file này có sự liên kết*

File main này là chương trình điều khiển để xử lý các đối số dòng lệnh và tạo đối tượng Solution.

```
from solve import Solution
import argparse
import itertools
```

- Đầu tiên là gọi class Solution trong file solve.py và gọi các thư viện argparse và thư viện iterpools.
+ Thư viện argparse quản lý đối số dòng lệnh, giúp chương trình tương tác với người dùng và xác định cách chạy chương trình.
+ Thư viện iterpools cung cấp các công cụ cho việc xử lý iterator và chuỗi dữ liệu, chẳng hạn như tạo hoán vị, tổ hợp, lặp qua các phần tử.

```
arg = argparse.ArgumentParser()
arg.add_argument("-m", "--method", required=False,
help="Specify which method to use")
arg.add_argument("-l", "--legend", required=False,
help="Specify if you want to display legend on graph")

args = vars(arg.parse_args())

solve_method = args.get("method", "bfs")
legend_flag = args.get("legend", False)
```

- arg = argparse.ArgumentParser(): Đây là dòng code tạo một đối tượng ArgumentParser. Đối tượng này sẽ được sử dụng để định nghĩa và quản lý các đối số dòng lệnh cho chương trình.
- arg.add_argument("-m", "--method", required=False, help="Specify which method to use"): Dòng này định nghĩa một đối số dòng lệnh có tên "-m" và tên đầy đủ "--method". Đối số này không bắt buộc (do required=False) và cho phép người dùng chỉ định một phương pháp giải quyết bằng cách truyền giá trị sau nó. Mô tả "Specify which method to use" được hiển thị khi bạn chạy chương trình và sử dụng tùy chọn --help.
- arg.add_argument("-l", "--legend", required=False, help="Specify if you want to display legend on graph"): Dòng này định nghĩa một đối số dòng lệnh có tên "-l" và tên đầy đủ "--legend". Đối số này không bắt buộc và cho phép người dùng chỉ định xem có muốn hiển thị hình chú thích (legend) trên biểu đồ hay không.
- args = vars(arg.parse_args()): Dòng này sử dụng arg.parse_args() để phân tích và lấy ra giá trị của các đối số dòng lệnh mà bạn đã cung cấp khi chạy chương trình. Kết quả là một từ điển (dictionary) chứa các giá trị của các đối số dòng lệnh, với tên của đối số là khóa (key) và giá trị của đối số là giá trị tương ứng trong từ điển.
- solve_method = args.get("method", "bfs"): Dòng này lấy giá trị của đối số "method" từ từ điển args. Nếu người dùng không cung cấp giá trị cho đối số "method", thì giá trị mặc định là "bfs" sẽ được sử dụng cho biến solve_method. Điều này cho phép chương trình biết được phương pháp giải quyết nào được chọn.
- legend_flag = args.get("legend", False): Tương tự như trên, dòng này sử dụng từ điển args để lấy giá trị của đối số "legend". Nếu đối số "legend" không được cung cấp, giá trị mặc định là False và sẽ được sử dụng và gán cho biến legend_flag. Điều này cho phép bạn xác định xem bạn muốn hiển thị huy hiệu (legend) trên biểu đồ hay không.
-- Tóm lại, mã code này thực hiện các công việc sau:
+ Sử dụng argparse để định nghĩa và quản lý các đối số dòng lệnh cho chương trình.
+ Cung cấp hai đối số dòng lệnh: "method" để chọn phương pháp giải quyết và "legend" để quyết định xem có hiển thị huy hiệu (legend) trên biểu đồ hay không.

+ Sử dụng arg.parse_args() để phân tích đối số dòng lệnh được cung cấp khi chạy chương trình.
+ Gán giá trị của các đối số dòng lệnh vào các biến solve_method và legend_flag, sử dụng giá trị mặc định "bfs" và False nếu các đối số không được cung cấp.
- Trên là các đối số, các biến cần thiết để cấu thành hàm main.

- Công việc đầu tiên của hàm main là gọi class Solution của file solve.py nên tiếp theo ta sẽ phân tích file solve.py (file solve.py có nhiệm vụ chính là định nghĩa đối tượng Solution và sẽ được main.py gọi khi cần như phần khai báo các thư viện ở đầu file main.py đã trình bày ở trên).

```
import os
import emoji
import pydot
import random
from collections import deque
```

- Thư viện os cung cấp các chức năng để tương tác với hệ điều hành, bao gồm việc thực hiện các thao tác trên tệp và thư mục, kiểm tra sự tồn tại của tệp, thay đổi thư mục làm việc hiện tại, và nhiều chức năng khác. Trong mã code, os có thể được sử dụng để kiểm tra và tạo các tệp và thư mục.
- Thư viện imoji cung cấp các biểu tượng cảm xúc và ký tự đặc biệt dùng để trang trí văn bản và tạo các biểu đồ hoặc in ra màn hình với biểu tượng cảm xúc. Trong mã code, có thể được sử dụng để thêm biểu tượng cảm xúc vào thông báo hoặc kết quả của chương trình
- Thư viện pydot là một giao diện Python cho Graphviz, một công cụ sử dụng để vẽ đồ thị và biểu đồ. Trong mã code, pydot được sử dụng để vẽ biểu đồ trạng thái và cạnh trong bài toán "vận chuyển người và quái qua sông".
- Module random cung cấp các chức năng để làm việc với số ngẫu nhiên. Trong mã code, random có thể được sử dụng để tạo ngẫu nhiên các giá trị hoặc lựa chọn một phần tử ngẫu nhiên từ một danh sách.

- Từ thư viện collections ta import deque giúp tạo một cấu trúc dữ liệu hàng đợi.

```
    def __init__(self):
        # Start state (3M, 3C, Left)
        # Goal State (0M, 0C, Right)
        # Each state gives the number of missionaries and
cannibals on the left side

        self.start_state = (3, 3, 1)
        self.goal_state = (0, 0, 0)
        self.options = [(1, 0), (0, 1), (1, 1), (0, 2), (2,
0)]

        self.boat_side = ["right", "left"]

        self.graph = pydot.Dot(graph_type='graph',
bgcolor="#fff3af",
```

```
                                label="fig: Missionaries and
Cannibal State Space Tree",fontcolor="red",
fontsize="24")
        self.visited = {}
        self.solved = False
```

- Đầu tên ta khởi tạo trạng thái ban đầu (3, 3, 1)
- Tiếp theo khởi tạo nút kết thúc (0, 0, 0)
- Khởi tạo các trạng thái của con thuyền (x, y) với x là số người, và y là số quái.
- Khởi tạo vị trí của thuyền gồm phải và trái.
- Tiếp theo dùng hàm Dot trong thư viện pydot để tạo đồ thị như trong file đầu tiên với các thuộc tính như trên.
- Khởi tạo biến visited để chứa các node đã đi qua.
- solved là một biến kiểu bool dùng để đánh dấu bài toán đã hoàn tất giải hay chưa.

```
    def is_valid_move(self, number_missionaries,
number_cannnibals):
        """
        Checks if number constraints are satisfied
        """
        return (0 <= number_missionaries <= 3) and (0 <=
number_cannnibals <= 3)
```

- Đây là phương thức để kiểm tra xem một trạng thái cụ thể có tuân theo các ràng buộc về số lượng.

```
    def is_goal_state(self, number_missionaries,
number_cannnibals, side):
        return (number_missionaries, number_cannnibals, side)
== self.goal_state
```

- Phương thức này dùng để kiểm tra xem một trạng thái cụ thể có phải là trạng thái mục tiêu hay không.

```
    def is_start_state(self, number_missionaries,
number_cannnibals, side):
        return (number_missionaries, number_cannnibals, side)
== self.start_state
```

- Phương thức dùng để kiểm tra xem một trạng thái cụ thể có phải là trạng thái ban đầu hay không.

```
    def number_of_cannibals_exceeds(self, number_missionaries,
number_cannnibals):
        number_missionaries_right = 3 - number_missionaries
        number_cannnibals_right = 3 - number_cannnibals
        return (number_missionaries > 0 and number_cannnibals
> number_missionaries) \
               or (number_missionaries_right > 0 and
number_cannnibals_right > number_missionaries_right)
```

- Kiểm tra xem số người điều hành vượt quá số dân đảo cụ thể.

```
    def write_image(self, file_name="state_space.png"):
        try:
            self.graph.write_png(file_name)
        except Exception as e:
            print("Error while writing file", e)
        print(f"File {file_name} successfully written.")
```
- Ghi biểu đồ trạng thái vào một tệp hình ảnh.

```
    def solve(self, solve_method="dfs"):
        self.visited = dict()
        Parent[self.start_state] = None
        Move[self.start_state] = None
        node_list[self.start_state] = None

        return self.dfs(*self.start_state, 0) if solve_method
== "dfs" else self.bfs()
```
- Phương thức giải quyết bài toán sử dụng thuật toán tìm kiếm theo chiều sâu (DFS) hoặc tìm kiếm theo chiều rộng (BFS). Thực hiện việc khởi tạo và quản lý trạng thái ban đầu và các trạng thái liên quan.

```
    def draw_legend(self):
        """
            Utility method to draw legend on graph if  legend
flag is ON
        """
        graphlegend = pydot.Cluster(graph_name="legend",
label="Legend", fontsize="20", color="gold",
                                    fontcolor="blue",
style="filled", fillcolor="#f4f4f4")


        node1 = pydot.Node("1", style="filled",
fillcolor="blue", label="Start Node", fontcolor="white",
width="2", fixedsize="true")
        graphlegend.add_node(node1)

        node2 = pydot.Node("2", style="filled",
fillcolor="red", label="Killed Node", fontcolor="black",
width="2", fixedsize="true")
        graphlegend.add_node(node2)

        node3 = pydot.Node("3", style="filled",
fillcolor="yellow", label="Solution nodes", width="2",
fixedsize="true")
        graphlegend.add_node(node3)
```

```
        node4 = pydot.Node("4", style="filled",
fillcolor="gray", label="Can't be expanded",  width="2",
fixedsize="true")
        graphlegend.add_node(node4)

        node5 = pydot.Node("5", style="filled",
fillcolor="green", label="Goal node", width="2",
fixedsize="true")
        graphlegend.add_node(node5)

        node7 = pydot.Node("7", style="filled",
fillcolor="gold", label="Node with child", width="2",
fixedsize="true")
        graphlegend.add_node(node7)


        description = "Each node (m, c, s) represents a
\nstate where 'm' is the number of\n missionaries,\'n' the
cannibals \
              and \n's' the side of the boat\n"
        " where  '1' represents the left \nside and '0' the
right side \n\nOur objective is to reach goal state (0, 0, 0)\
        \nfrom start state (3, 3, 1) by some \noperators =
[(0, 1), (0, 2), (1, 0), (1, 1), (2, 0),]\n"\
                "each tuples (x, y) inside operators
\nrepresents the number of missionaries and\
        \ncannibals to be moved from left to right \nif c == 1
and viceversa"

        node6 = pydot.Node("6", style="filled",
fillcolor="gold", label= description, shape="plaintext",
fontsize="20", fontcolor="red")
        graphlegend.add_node(node6)

        self.graph.add_subgraph(graphlegend)

        self.graph.add_edge(pydot.Edge(node1, node2,
style="invis"))
        self.graph.add_edge(pydot.Edge(node2, node3,
style="invis"))
        self.graph.add_edge(pydot.Edge(node3, node4,
style="invis"))
```

```
        self.graph.add_edge(pydot.Edge(node4, node5,
style="invis"))
        self.graph.add_edge(pydot.Edge(node5, node7,
style="invis"))
        self.graph.add_edge(pydot.Edge(node7, node6,
style="invis"))
```

- Đầu tiên phương thức tạo một đối tượng pydot.Cluster để tạo một nhóm trong biểu đồ với tên là legend. Đây là nơi chứa các nút giải thích huy hiệu.
- Sau đó, phương thức tạo các nút đại diện cho các màu và trạng thái khác nhau trong huy hiệu. Các node này được thêm vào phân nhóm legend với các thuộc tính sau:

+ node1: đại diện cho trạng thái ban đầu (start node) với màu nền xanh (blue) và văn bản trắng (white).
+ node2: đại diện cho trạng thái không thể mở rộng (killed node) với màu nền đỏ (red) và văn bản màu đen (black).
+ node3: đại diện cho các trạng thái giải quyết (solution nodes) với màu nền vàng (yellow).
+ node4: Nút đại diện cho các trạng thái không thể mở rộng (can't be expanded) với màu nền xám (gray).
+ node5: đại diện cho trạng thái mục tiêu (goal node) với màu nền xanh lá cây (green).
+ node7: đại diện cho các trạng thái có con (node with child) với màu nền vàng (yellow).

- Method này cũng tạo một node đặc biệt là node6 để hiển thị mô tả chi tiết về ý nghĩa của các màu và trạng thái trong huy hiệu. Nút này chứa văn bản mô tả và có màu vàng (gold) với font màu đỏ (red) và hình dạng là "plaintext".
- Cuối cùng, các node được thêm vào phân nhóm "legend" và được kết nối bằng các cạnh ẩn (invisible edges) để hiển thị sự kết nối giữa chúng.

```
    def draw(self, *, number_missionaries_left,
number_cannnibals_left, number_missionaries_right,
number_cannnibals_right):
        """
            Draw state on console using emojis
        """
        left_m = emoji.emojize(f":old_man: " *
number_missionaries_left)
        left_c = emoji.emojize(f":ogre: " *
number_cannnibals_left)
        right_m = emoji.emojize(f":old_man: " *
number_missionaries_right)
        right_c = emoji.emojize(f":ogre: " *
number_cannnibals_right)
```

```
        print('{}{}{}{}{}'.format(left_m, left_c + " " * (14 -
len(left_m) - len(left_c)), "_" * 40, " " * (12 - len(right_m)
- len(right_c)) + right_m, right_c))
        print("")
```

- Phương thức này có nhiệm vụ vẽ biểu diễn trạng thái của trò chơi trên màn hình console sử dụng các biểu tượng emoji để hiển thị số lượng người(missionaries) và quái (cannibals) ở cả hai bên của sông.
- Phương thức nhận các tham số như sau:
+ number_missionaries_left: Số lượng người bên trái sông.
+ number_cannibals_left: Số lượng quái bên trái sông.
+ number_missionaries_right: Số lượng người bên phải sông.
+ number_cannibals_right: Số lượng quái bên phải sông.
- Phương thức sử dụng các biểu tượng emoji để hiển thị số lượng người và quái trên cả hai bên của sông. Emoji được sử dụng là ":old_man:" (người) và ":ogre:" (quái).
- Đầu tiên, phương thức sẽ xây dựng chuỗi văn bản cho phần bên trái và bên phải của sông, sử dụng emoji để biểu thị số lượng ngườivà quái.
- Đối với bên trái, chuỗi văn bản này được tạo bằng cách ghép nối emoji ":old_man:" và ":ogre:" với số lượng tương ứng của người và quái.
- Đối với bên phải, chuỗi văn bản này được tạo bằng cách ghép nối dấu gạch ngang "_" để biểu thị sông trống và sau đó thêm emoji ":old_man:" để biểu thị số lượng người và emoji ":ogre:" để biểu thị số lượng quái.
- Sau đó, hàm in ra trạng thái của cả hai bên của sông lên màn hình console, sử dụng các chuỗi văn bản đã tạo. Phần bên trái và bên phải của sông được in cùng nhau để hiển thị trạng thái hoàn chỉnh của trò chơi.

```
    def show_solution(self):
        # Recursively start from Goal State
        # And find parent until start state is reached

        state = self.goal_state
        path, steps, nodes = [] ,[], []

        while state is not None:
            path.append(state)
            steps.append(Move[state])
            nodes.append(node_list[state])

            state = Parent[state]

        steps, nodes = steps[::-1], nodes[::-1]

        number_missionaries_left, number_cannnibals_left = 3,
3
        number_missionaries_right, number_cannnibals_right =
0, 0
```

```
        print("*" * 60)
        self.draw(number_missionaries_left=number_missionaries
_left, number_cannnibals_left=number_cannnibals_left,
                  number_missionaries_right=number_missionarie
s_right, number_cannnibals_right=number_cannnibals_right)

        for i, ((number_missionaries, number_cannnibals,
side), node) in enumerate(zip(steps[1:], nodes[1:])):

            if node.get_label() != str(self.start_state):
                node.set_style("filled")
                node.set_fillcolor("yellow")

            print(f"Step {i + 1}: Move {number_missionaries}
missionaries  and {number_cannnibals} \
                      cannibals from {self.boat_side[side]} to
{self.boat_side[int(not side)]}.")

            op = -1 if side == 1 else 1

            number_missionaries_left =
number_missionaries_left + op * number_missionaries
            number_cannnibals_left = number_cannnibals_left +
op * number_cannnibals

            number_missionaries_right =
number_missionaries_right - op * number_missionaries
            number_cannnibals_right = number_cannnibals_right
- op * number_cannnibals

            self.draw(number_missionaries_left=number_missiona
ries_left, number_cannnibals_left=number_cannnibals_left,
                      number_missionaries_right=number_mission
aries_right, number_cannnibals_right=number_cannnibals_right)

        print("Congratulations!!! you have solved the
problem")
        print("*" * 60)
```

- Phương thức này bắt đầu từ trạng thái kết thúc (đạt được sau khi giải quyết bài toán) và tiến lui từng bước về phía trạng thái ban đầu bằng cách theo dõi các bước di chuyển đã lưu trữ trong biến Move và trạng thái cha của từng trạng thái đã được lưu trữ trong biến Parent. Điều này giúp tạo ra một dãy các trạng thái từ trạng thái kết thúc đến trạng thái ban đầu.

- Đối với mỗi bước di chuyển trong dãy trạng thái, phương thức thực hiện các thao tác sau:
+ Đánh dấu trạng thái với màu và kiểu nền thích hợp để biểu thị trạng thái hiện tại của trò chơi. Các loại màu và kiểu nền được sử dụng để biểu thị các loại trạng thái khác nhau, chẳng hạn như trạng thái ban đầu, trạng thái kết thúc, trạng thái không thể mở rộng, và các trạng thái trung gian trong quá trình giải quyết.
+ In ra màn hình console thông tin về bước di chuyển hiện tại, bao gồm số lượng người và quái được di chuyển và hướng di chuyển (từ bên nào đến bên nào).
+ Vẽ biểu đồ trạng thái của trò chơi trên màn hình console để hiển thị trạng thái hiện tại. Điều này giúp ta theo dõi trạng thái của trò chơi và quá trình di chuyển từ trạng thái này đến trạng thái khác.
- Phương thức tiếp tục lặp lại các bước trên cho đến khi đạt được trạng thái ban đầu (trạng thái đầu tiên) trong dãy trạng thái. Khi đó, toàn bộ giải pháp đã được hiển thị.
- Tóm lại, phương thức này có nhiệm vụ hiển thị giải pháp cho trò chơi "Missionaries and Cannibals" sau khi đã tìm thấy nó. Phương thức này sẽ in ra các bước để di chuyển từ trạng thái ban đầu đến trạng thái kết thúc, cũng như vẽ biểu đồ trạng thái của trò chơi trên màn hình console để theo dõi quá trình di chuyển.

```
    def draw_edge(self, number_missionaries,
number_cannnibals, side, depth_level):
        u, v = None, None
        if Parent[(number_missionaries, number_cannnibals,
side)] is not None:
            u = pydot.Node(str(Parent[(number_missionaries,
number_cannnibals, side)] + (depth_level - 1, )),
                           label=str(Parent[((number_missionar
ies, number_cannnibals, side))]))
            self.graph.add_node(u)

            v = pydot.Node(str((number_missionaries,
number_cannnibals, side, depth_level)),
                           label=str((number_missionaries,
number_cannnibals, side)))
            self.graph.add_node(v)

            edge = pydot.Edge(str(Parent[(number_missionaries,
number_cannnibals, side)] + (depth_level - 1, )),
                              str((number_missionaries,
number_cannnibals, side, depth_level) ), dir='forward')
            self.graph.add_edge(edge)
        else:
            # For start node
            v = pydot.Node(str((number_missionaries,
number_cannnibals, side, depth_level)),
                           label=str((number_missionaries,
number_cannnibals, side)))
```

```
            self.graph.add_node(v)
        return u, v
```

- Phương thức này nhận các tham số đầu vào gồm: number_missionaries, number_cannibals, side, và depth_level.
- Trước hết phương thức khởi tạo 2 biến u và v với u = v = None tiếp theo dùng câu lệnh điều kiện kiểm tra xem trạng thái cha có tồn tại không. Nếu có, nó tạo một đối tượng pydot.Node để biểu diễn trạng thái cha và đối tượng pydot.Node để biểu diễn trạng thái hiện tại trên biểu đồ.
- Sau đó, phương thức tạo một đối tượng pydot.Edge để biểu diễn cạnh từ trạng thái cha đến trạng thái hiện tại. Cạnh này sẽ có hướng từ trạng thái cha đến trạng thái hiện tại trên biểu đồ.
- Cuối cùng, phương thức thêm trạng thái cha, trạng thái hiện tại và cạnh vào biểu đồ.
- Tóm lại, phương thức này có nhiệm vụ vẽ cạnh (edge) trên biểu đồ trạng thái của trò chơi "Missionaries and Cannibals". Cụ thể, phương thức này vẽ một cạnh từ trạng thái cha (parent state) đến trạng thái hiện tại trên biểu đồ để biểu diễn quá trình di chuyển giữa các trạng thái khác nhau trong quá trình giải quyết bài toán.

```
    def bfs(self):
        q = deque()
        q.append(self.start_state + (0, ))
        self.visited[self.start_state] = True

        while q:
            number_missionaries, number_cannnibals, side,
depth_level = q.popleft()
            # Draw Edge from u -> v
            # Where u = Parent[v]
            # and v = (number_missionaries, number_cannnibals,
side, depth_level)
            u, v = self.draw_edge(number_missionaries,
number_cannnibals, side, depth_level)


            if self.is_start_state(number_missionaries,
number_cannnibals, side):
                v.set_style("filled")
                v.set_fillcolor("blue")
                v.set_fontcolor("white")
            elif self.is_goal_state(number_missionaries,
number_cannnibals, side):
                v.set_style("filled")
                v.set_fillcolor("green")
                return True
```

```
            elif
self.number_of_cannibals_exceeds(number_missionaries,
number_cannnibals):
                v.set_style("filled")
                v.set_fillcolor("red")
                continue
            else:
                v.set_style("filled")
                v.set_fillcolor("orange")

            op = -1 if side == 1 else 1

            can_be_expanded = False

            for x, y in self.options:
                next_m, next_c, next_s = number_missionaries +
op * x, number_cannnibals + op * y, int(not side)
                if (next_m, next_c, next_s) not in
self.visited:
                    if self.is_valid_move(next_m, next_c):
                        can_be_expanded = True
                        self.visited[(next_m, next_c, next_s)]
= True
                        q.append((next_m, next_c, next_s,
depth_level + 1))

                        # Keep track of parent and
corresponding move
                        Parent[(next_m, next_c, next_s)] =
(number_missionaries, number_cannnibals, side)
                        Move[(next_m, next_c, next_s)] = (x,
y, side)
                        node_list[(next_m, next_c, next_s)] =
v

            if not can_be_expanded:
                v.set_style("filled")
                v.set_fillcolor("gray")
        return False
```

- Đây là phương thức thực hiện thuật toán duyệt theo chiều rộng(Breadth First Search) đã học ở các bài trước để tìm lời giải cho vấn đề đã được nêu ra.
- Bắt đầu với trạng thái ban đầu à trạng thái (3M, 3C, Left) và độ sâu (depth_level) bằng 0. Trạng thái ban đầu được thêm vào hàng đợi (queue) và được đánh dấu là đã thăm (visited).

- Tiến hành duyệt theo BFS trong while loop cho đến khi queue rỗng:
+ Lấy trạng thái đầu tiên ra khỏi hàng đợi (dequeue).
+ Vẽ cạnh trên biểu đồ giữa trạng thái hiện tại và trạng thái cha (nếu có).
+ Kiểm tra xem trạng thái hiện tại có phải là trạng thái ban đầu không. Nếu đúng, nó được tô màu xanh biểu thị trạng thái khởi đầu.
+ Kiểm tra xem trạng thái hiện tại có phải là trạng thái mục tiêu không. Nếu đúng, nó được tô màu xanh lá cây và phương thức trả về **True** để cho biết lời giải đã được tìm thấy.
+ Kiểm tra xem trạng thái hiện tại có vi phạm ràng buộc về số lượng người Cannibals vượt quá số lượng Missionaries không. Nếu đúng, nó được tô màu đỏ.
+ Nếu không thuộc các trường hợp trên, trạng thái hiện tại được tô màu cam.
- Sau khi kiểm tra trạng thái hiện tại, phương thức tiếp tục duyệt qua tất cả các tùy chọn (operators) cho phép (di chuyển một số lượng người Missionaries và Cannibals từ bờ này sang bờ kia) và kiểm tra xem trạng thái con đã được thăm chưa. Nếu trạng thái con chưa được thăm và là một trạng thái hợp lệ, nó được thêm vào hàng đợi với độ sâu (depth_level) tăng lên 1. Trạng thái con này cũng được đánh dấu là đã thăm và thông tin về trạng thái cha và di chuyển được lưu trữ.
- Nếu không có trạng thái con nào có thể mở rộng (không thể di chuyển tiếp), trạng thái hiện tại được tô màu xám để biểu thị rằng nó không thể mở rộng thêm.
- Nếu trong quá trình duyệt, không tìm thấy lời giải (không có cách di chuyển từ trạng thái ban đầu đến trạng thái mục tiêu), phương thức trả về False.

```
    def dfs(self, number_missionaries, number_cannnibals,
side, depth_level):
        self.visited[(number_missionaries, number_cannnibals,
side)] = True

        # Draw Edge from u -> v
        # Where u = Parent[v]
        u, v = self.draw_edge(number_missionaries,
number_cannnibals, side, depth_level)


        if self.is_start_state(number_missionaries,
number_cannnibals, side):
            v.set_style("filled")
            v.set_fillcolor("blue")
        elif self.is_goal_state(number_missionaries,
number_cannnibals, side):
            v.set_style("filled")
            v.set_fillcolor("green")
            return True
        elif
self.number_of_cannibals_exceeds(number_missionaries,
number_cannnibals):
            v.set_style("filled")
            v.set_fillcolor("red")
```

```
            return False
        else:
            v.set_style("filled")
            v.set_fillcolor("orange")

        solution_found = False
        operation = -1 if side == 1 else 1

        can_be_expanded = False

        for x, y in self.options:
            next_m, next_c, next_s = number_missionaries +
operation * x, number_cannnibals + operation * y, int(not
side)

            if (next_m, next_c, next_s) not in self.visited:
                if self.is_valid_move(next_m, next_c):
                    can_be_expanded = True
                    # Keep track of Parent state and
corresponding move
                    Parent[(next_m, next_c, next_s)] =
(number_missionaries, number_cannnibals, side)
                    Move[(next_m, next_c, next_s)] = (x, y,
side)
                    node_list[(next_m, next_c, next_s)] = v

                    solution_found = (solution_found or
self.dfs(next_m, next_c, next_s, depth_level + 1))

                    if solution_found:
                        return True

        if not can_be_expanded:
            v.set_style("filled")
            v.set_fillcolor("gray")

        self.solved = solution_found
        return solution_found
```

- Phương thức này thực hiện thuật toán duyệt theo chiều sâu(Depth First Search) để tìm lời giải cho bài toán.
- Nếu trạng thái hiện tại là trạng thái mục tiêu (0M, 0C, Right), phương thức trả về True để cho biết lời giải đã được tìm thấy.
- Nếu trạng thái hiện tại vi phạm ràng buộc về số lượng người Cannibals vượt quá

số lượng Missionaries, trạng thái này được đánh dấu màu đỏ để biểu thị rằng nó không hợp lệ và phương thức trả về False.
- Nếu không thuộc các trường hợp trên, trạng thái hiện tại được tô màu cam để biểu thị rằng nó là một trạng thái trung gian trong quá trình giải quyết.
- Tiếp theo, phương thức kiểm tra tất cả các tùy chọn (operators) cho phép (di chuyển một số lượng người Missionaries và Cannibals từ bờ này sang bờ kia) và thử mở rộng trạng thái hiện tại bằng cách thay đổi số lượng người và bờ xuất phát của thuyền.

- Nếu một trạng thái con hợp lệ và chưa được thăm, phương thức gọi đệ quy để thử tìm lời giải từ trạng thái con đó. Nếu đệ quy trả về True, nghĩa là lời giải đã được tìm thấy, phương thức cũng trả về True. Nếu không tìm thấy lời giải từ trạng thái con, phương thức tiếp tục kiểm tra các trạng thái con khác.
- Nếu không còn trạng thái con nào có thể mở rộng, trạng thái hiện tại được đánh dấu màu xám để biểu thị rằng nó không thể mở rộng thêm.
- Phương thức trả về True nếu lời giải đã được tìm thấy, ngược lại trả về False.

Giờ ta sẽ trở lại với file main.py để phân tích tiếp

```
def main():
    s = Solution()

    if(s.solve(solve_method)):

        # Display SOlution on console
        s.show_solution()

        output_file_name = f"{solve_method}"
        # Draw legend if legend_flag is set
        if legend_flag:
            if legend_flag[0].upper() == 'T' :
                output_file_name += "_legend.png"
                s.draw_legend()
            else:
                output_file_name += ".png"
        else:
             output_file_name += ".png"

        # Write State space tree
        s.write_image(output_file_name)
    else:
        raise Exception("No solution found")
```

- Trong file main có hàm main được định nghĩa như trên.
- Đầu tiên tạo một đối tượng Solution mới được gọi là s, đại diện cho bài toán "Missionaries and Cannibals" và chuẩn bị môi trường để giải quyết nó.
- Kiểm tra xem bài toán có thể được giải quyết bằng phương pháp được chọn (được xác định bởi biến solve_method) hay không. Nếu có lời giải, chương trình tiếp tục,

nếu không, nó sẽ ném một ngoại lệ với thông báo "No solution found".
- Nếu có lời giải, chương trình sẽ tiến hành hiển thị lời giải trên màn hình console bằng cách gọi phương thức show_solution của đối tượng s.
- Tiếp theo, chương trình xác định tên tệp đầu ra dự kiến cho biểu đồ trạng thái (state space tree). Tên tệp này sẽ được xây dựng dựa trên phương pháp giải quyết (solve_method) và có thể bao gồm chữ cái "legend" nếu cờ legend_flag được đặt.
- Nếu cờ legend_flag được đặt và có giá trị bắt đầu bằng 'T' (ngữ cảnh không rõ), chương trình sẽ thêm "_legend" vào tên tệp để biểu thị rằng biểu đồ bao gồm một hình chú thích (legend). Sau đó, nó gọi phương thức draw_legend để vẽ hình chú thích.
- Cuối cùng, chương trình sử dụng phương thức write_image của đối tượng s để ghi biểu đồ trạng thái vào một tệp hình ảnh PNG với tên tệp được xác định ở bước trước. Tên tệp này sẽ được sử dụng để lưu trữ biểu đồ trạng thái của cây không gian trạng thái.