BÁO CÁO THỰC HÀNH AI TUẦN 2

1. File generate_full_space_tree.py

```
from collections import deque
import pydot
import argparse
import os

# Set it to bin folder of graphviz
os.environ["PATH"] += os.pathsep + 'C:\Program Files\Graphviz\bin'

options = [(1, 0), (0, 1), (1, 1), (0, 2), (2, 0)]
Parent = dict()
graph = pydot.Dot(graph_type='graph', strict=False, bgcolor="#fff3af",
                  label="fig: Missionaries and Cannibal State Space Tree", fontcolor="red", fontsize="24", overlap="true")
```

Khai báo các thư viện cần thiết , thiết lập môi trường (đường dẫn).

option =[(number_missionaries, numer_cannibals)], ví dụ (1, 0) là dẫn 1 người , 0 quỷ sang bờ kia sông.

parent là một dictionary rỗng.

graph là một Dot object của thư viện pydot để vẽ đồ thị.

```
i = 0

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

args = vars(arg.parse_args())

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

Nhập tham số depth từ của sổ dòng lện (terminal) , với depth là độ sâu của đồ thị cần tạo.

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

Hàm is_valid_move check bước đi có hợp lệ hay không, bước đi hợp lệ khi số người và quỷ thuộc đoạn [0,3].

```python
def write_image(file_name="state_space"):
    try:
        graph.write_png(f"{file_name}_{max_depth}.png")
    except Exception as e:
        print("Error while writing file", e)
    print(f"File {file_name}_{max_depth}.png successfully written.")
```

Hàm write_image ghi hình đồ thị vào file ảnh .png , nếu ghi không được sẽ báo lỗi, ngược lại báo ghi thành công.

```python
def draw_edge(number_missionaries, number_cannnibals, side, depth_level, node_num):
    u, v = None, None
    if Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)] is not None:
        u = pydot.Node(str(Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)]), label=str(
            Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)][:3]))
        graph.add_node(u)

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

        edge = pydot.Edge(str(Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)]), str(
            (number_missionaries, number_cannnibals, side, depth_level, node_num)), dir='forward')
        graph.add_edge(edge)
    else:
        # For start node
        v = pydot.Node(str((number_missionaries, number_cannnibals, side, depth_level,
                            node_num)), label=str((number_missionaries, number_cannnibals, side)))
        graph.add_node(v)
    return u, v
```

Hàm draw_edge vẽ cạnh của đồ thị với u là node cha và v là node con, nếu v là root(start_state) thì không có cha , hàm trả về 2 node u và v. Mỗi node lưu giá trị

tuple (số lượng người, số lượng quỷ, bờ của dòng sông, độ sâu của đồ thị , số thứ tự node). Bờ của dòng sông gồm 1 là trái , 0 là phải. Độ sâu (depth_level root là 0, tăng dần qua từng cấp, số thứ tự node (node _num) chỉ theo số thứ tự expand node (bắt đầu từ root là 0).

```
def is_start_state(number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == (3, 3, 1)
```

Hàm is_start_state kiểm tra một node có phải state bắt đầu hay không , state bắt đầu tương ứng với 3 người 3 quỷ và thuyền ở bờ trái sông (3, 3, 1).

```
def is_goal_state(number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == (0, 0, 0)
```

Hàm is_goal_state kiểm tra một node có phải goal state hay không , goal state tương ứng với 0 người 0 quỷ ở bờ trái sông và thuyền ở bờ phải sông (0, 0, 0).

```
def number_of_cannibals_exceeds(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)
```

Hàm number_of_cannibals_exceeds kiểm tra số lượng quỷ có vượt qua số lượng người hay không (ở cả 2 bờ sông).

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

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

    while q:

        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
```

Hàm generate là hàm chính để tạo đồ thị. Đầu tiên tạo một hàng đợi q với node đầu tiên là root node (start state). Khi q khác rỗng thì pop q ra (vòng lặp while q), vẽ cạnh u,v vào đồ thị bằng hàm draw_edge với u=parent(v), v ở đây chính là node pop ra từ q. Nếu v là start state thì tô màu blue, nếu v là goal state thì tô màu green và tiếp tục vòng lặp (bỏ qua các bước expand v), nếu v có số lượng quỷ vượt quá người thì tô màu đỏ và cũng tiếp tục vòng lặp (bỏ qua các bước expand v), còn nếu v không phải 3 trường hợp trên thì tô màu orange. Tiếp tục xét nếu độ sâu đã bằng độ sâu lớn nhất (depth_level = max_depth) thì việc tạo cây đã xong, return true(tức đã tạo được cây thành công).Với node v là orange hoặc blue thì ta xem xét expand các node con . Nếu expand được thì cho các node con vào q (expand được khi parent(v) khác None và các node con của v phải khác parent(v)), đồng thời đánh dấu parent của các node con. Còn nếu v không thể expand được thì tô màu gray.

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

Hàm main để gọi hàm generate.

2. File solve.py

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

# Set it to bin folder of graphviz
os.environ["PATH"] += os.pathsep + 'C:\Program Files\Graphviz\bin'

# Dictionaries to backtrack solution nodes
# Parent stores parent of (m , c, s)
# Move stores (x, y, side) i.e number of missionaries, cannibals to be moved from le
# node_list stores pydot.Node object for particular state (m, c, s) so that we can c
Parent, Move, node_list = dict(), dict(), dict()
```

Tương tự file generate_full_space_tree.py, khai báo thư viện, setup environment, khai báo cái dictionary để lưu trữ các thông tin cần thiết.

```
class Solution():

    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
```

Tạo class solution, hàm khởi tạo __init__ khởi tạo các giá trị :

- start_state : trạng thái ban đầu
- goal_state : trạng thái mong muốn
- options : giống ý nghĩa như trong file generate_full_space_stree.py

- boat_side : [bờ phải, bờ trái]
- graph : pydot.Dot object, dùng để vẽ đồ thị
- visited: lưu các node đã đi qua, mặc định rỗng
- solved : bài toán đã được giải quyết hai chưa, mặc định bằng false

```
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)

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

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

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)

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.")
```

Các hàm is_valid_move, is_goal_state, is_start_state, numer_of_cannibals_exceeds, write_image tương tự các hàm trong file generate_full_space_tree.py

```
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()
```

Hàm solve để khởi tạo các giá trị ban đầu , sau đó gọi hàm dfs hay bfs, mặc định là dfs.

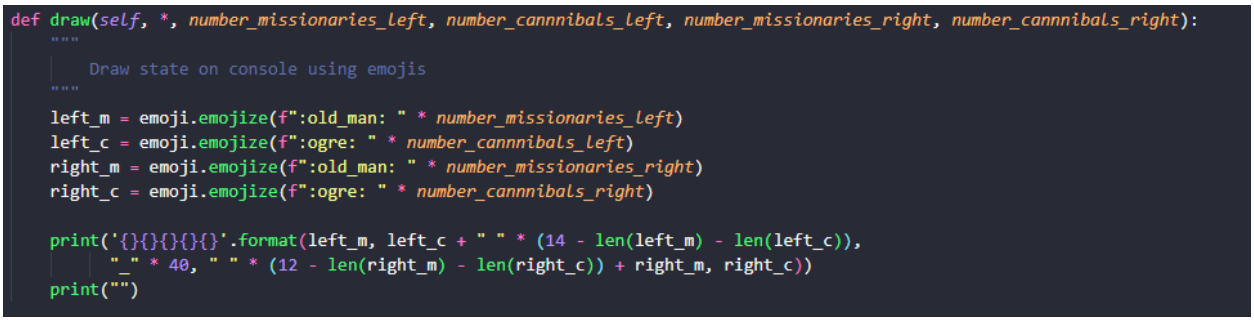
```
def draw_legend(self):
    """
        Utility method to draw legend on graph if  legend flag is ON
    """
    graphlegend = pydot.Cluster(graph_name="legend", label="Legend", fontsize="20", color="gold",
                                fontcolor="blue", style="filled", fillcolor="#f4f4f4")

    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"))
```

Hàm draw_legend để chú thích cho các node vẽ trong đồ thị.

Color	Property
Blue	Start node
Red	Killed node
Yellow	Solution node
Gray	Node that cannot be expanded
Green	Goal node
Gold	Not with child

Chú thích (description) : Mỗi node (m, c, s) đại diện cho một trạng thái khi m là số người, c là số quỷ và s là bờ của sông (1 trái 0 phải). Mục đích là đạt được trạng thái mục tiêu (0,0,0) từ trạng thái ban đầu (3,3,1) bằng các phép biến đổi [(m,c)], mỗi (m,c) đại diện cho việc di chuyển m người , c quỷ từ trái sang phải nếu s=1 và từ phải sang trái nếu s=0.



```
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("")
```

Hàm draw để vẽ các trạng thái trên terminal sử dụng emojis, là hàm bổ trợ cho hàm show solution.

```python
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_missionaries_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_missionaries_left, number_cannnibals_left=number_cannnibals_left,
                  number_missionaries_right=number_missionaries_right, number_cannnibals_right=number_cannnibals_right)

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

Hàm show_solution để in từng bước thực hiện thuật toán và không gian trạng thái qua các bước đến khi đạt được goal state.

```python
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
```

Đoạn code trên để tìm đường đi từ goal_state đến start_state (path) và từng bước trên đường đi như thế nào (steps), nodes để lưu các node (dưới dạng pydot.Dot

object) trên đường đi. Ở đây có nghĩa là khi đã tìm được solution bằng dfs hoặc bfs, ta sẽ tìm đc một đường đi (path) từ start_state đến goal_state (bằng việc truy ngược parent của goal_state lên trên đến start_state, cũng như các bước di chuyển ở mỗi state (Move[state])). Từ đó ta có thể in được các bước di chuyển và trạng thái ở từng bước từ start_stae cho đến khi đạt được goal_state bằng đoạn code dưới đây.

```
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_missionaries_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_missionaries_left, number_cannnibals_left=number_cannnibals_left,
              number_missionaries_right=number_missionaries_right, number_cannnibals_right=number_cannnibals_right)

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

Và đây là kết quả :

```
**********************************************************
😀 😀 😀 😈 😈 😈   ______________________________________

Step 1: Move 1 missionaries  and 1 cannibals from left to right.
😀 😀 😈 😈       ______________________________________       😀 😈

Step 2: Move 1 missionaries  and 0 cannibals from right to left.
😀 😀 😀 😈 😈     ______________________________________           😈

Step 3: Move 0 missionaries  and 2 cannibals from left to right.
😀 😀 😀         ______________________________________      😈 😈 😈

Step 4: Move 0 missionaries  and 1 cannibals from right to left.
😀 😀 😀 😈       ______________________________________       😈 😈

Step 5: Move 2 missionaries  and 0 cannibals from left to right.
😀 😈           ______________________________________    😀 😀 😈 😈

Step 6: Move 1 missionaries  and 1 cannibals from right to left.
😀 😀 😈 😈       ______________________________________       😀 😈

Step 7: Move 2 missionaries  and 0 cannibals from left to right.
😈 😈           ______________________________________    😀 😀 😀 😈

Step 8: Move 0 missionaries  and 1 cannibals from right to left.
😈 😈 😈         ______________________________________      😀 😀 😀

Step 9: Move 0 missionaries  and 2 cannibals from left to right.
😈             ______________________________________  😀 😀 😀 😈 😈

Step 10: Move 1 missionaries  and 0 cannibals from right to left.
😀 😈           ______________________________________    😀 😀 😈 😈

Step 11: Move 1 missionaries  and 1 cannibals from left to right.
               ______________________________________😀 😀 😀 😈 😈 😈

Congratulations!!! you have solved the problem
```

```
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_missionaries, 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
```

Hàm draw_edge tương tự file generate_full_space_tree.py

```
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")
```

Trong hàm bfs, đầu tiên tạo một hàng đợi rỗng, thêm start_state vào hàng đợi, đánh dấu nó đã visited. Lần lượt pop các node trong hàng đợi ra theo thứ tự FIFO (first in first out) bằng vòng lặp while q.Vẽ cạnh u, v vào đồ thị với v là node pop ra từ q và u là parent của v. Nếu v là start_state thì tô màu blue. Nếu v là goal_state thì tô màu green và return True vì đã tìm được goal. Nếu v có số quỷ vượt quá người thì tô màu red và continue vòng lặp while, tức mình sẽ bỏ qua node đó vì node đó sẽ không expand được.Trong trường hợp còn lại thì tô màu 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
```

Với các node blue và orange thì mình xem xét expand các node đó. Nếu các node con nào không nằm trong visited và nó là valid_move thì mình sẽ set can_be_expanded =True, đánh dấu node con đó là đã visited và đánh dấu parent (thêm vào Parent, và parent ở đây chính là v (v ở đây không có nghĩa là pydot.Dot object v)), đánh dấu bước đi tạo ra node con đó (thêm vào Move), node_list cũng như Parent nhưng thay vì Parent là lưu tuple (number_missionaries, number_cannibals, side) thì node_list lưu pydot.Dot object v. Nếu không expand được node con nào thì sẽ tô node cha (v) là gray. Sau tất cả , tức là không tìm đc goal thì return 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
```

Hàm dfs sử dụng đệ quy chứ không dùng stack hay queue để tìm goal_state, ở đây các bước khởi tạo và tô màu cũng tương tự như bfs, trừ màu đỏ (red) sẽ return False chứ không continue như bfs (vì không có vòng lặp).

```
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
```

Phần đánh dấu, tô màu cũng tương tự như bfs, ở đây gọi đệ quy dfs cho các node con.Ở đây có 2 trường hợp:

- Nếu dfs tìm được goal_state ở một mức nào đó thì sẽ trả về True (không tiếp tục expand các node con khác trong mức đó),dẫn đến solution_found ở mức kế trên cũng sẽ là True, từ đó đệ quy trả về True ngược lên lại cho đến start_state, kết quả cuối cùng là tìm được tìm được goal_state và trả về True.
- Nếu dfs không tìm được goal_state ở mức cuối cùng (không thể expand thêm được nữa), thì sẽ đệ quy ngược lại , expand các node khác trong cùng bậc,rồi các bậc ở trên, cứ thế cho đến khi tìm được goal_state, trường hợp tệ nhất không tìm đc goal_state thì sẽ trả về False.

3. File main.py

Đây là file chính để tìm solution bằng dfs hoặc bfs.

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

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)
```

Phần đầu là khai báo thư viện cần thiết cũng như import Class Solution từ file solve.py. Tiếp theo là setup nhập tham số từ cửa sổ dòng lệnh (terminal) , ở đây có các lựa chọn là -m hoặc --method (dsf hoặc bfs , mặc định là bfs) và – l hoặc – legend (dùng để vẽ legend lên graph, mặc định là không có legend).

```python
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")


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

Hàm main là hàm chính để gọi là các hàm đã viết ở bên trên hoặc các file kia , thực hiện các thao tác như tìm solution , ghi ra file ảnh hoặc báo lỗi nếu không tìm được solution.