Họ và tên: Phan Hồng Trâm

MSSV: 21110414

Báo cáo

Thực hành Nhập môn Trí tuệ nhân tạo tuần 2

Giải thích code:

Để build bài toán cần 3 file code, mỗi file có những chức năng và hàm hỗ trợ.

1. File **generate_full_space_tree.py**

Chức năng chính của file này là tạo ra một cây tìm kiếm và vẽ cây tìm kiếm này ra một file ảnh, bao gồm một số hàm sau:

- **`is_valid_move(number_missionaries, number_cannibals):`** dùng để kiểm tra xem một nước đi cụ thể có tuân theo ràng buộc của bài toán "Missionaries and Cannibals"hay không. Bài toán yêu cầu rằng số lượng Missionaries và Cannibals ở mỗi bên của sông không được vượt quá 3 và không thấp hơn 0.

```
def is_valid_move(number_missionaries, number_cannnibals):
        """
        Checks if number constraints are satisfied
        """
        return (0 <= number_missionaries <= 3) and (0 <= number_cannnibals <= 3)
        # Kiểm tra số lượng của Missionaries (number_missionaries) và số lượng Cannibals (number_cannnibals) có nằm trong khoảng từ 0 đến 3 không
```

- **`write_image(file_name="state_space")`**: dùng để xuất đồ thị đã vẽ thành một file ảnh

```
def write_image(file_name="state_space"):
        try:
            graph.write_png(f"{file_name}_{max_depth}.png") #Thử ghi đồ thị đã vẽ thành một file ảnh với tên "{file_name}_{max_depth}.png"
            # Sử dụng write_png cho đối tượng graph, nếu không lỗi thì chương trình sẽ nhảy vào except để xét ngoại lệ
        except Exception as e:
            print("Error while writing file", e)
        print(f"File {file_name}_{max_depth}.png successfully written.") # Sau khi ghi file thành công, chương trình in ra thông báo
```

- **`draw_edge(number_missionaries, number_cannibals, side, depth_level, node_num):`** được sử dụng để vẽ cạnh (edge) trong cây tìm kiếm, nối hai trạng thái (nodes) với nhau

```
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
```

- Dòng 2 đến dòng 4: Khởi tạo các biến u và v với giá trị ban đầu là None. Chúng sẽ được sử dụng để đại diện cho hai đỉnh (nodes) trong cây tìm kiếm.

- Dòng 5 đến dòng 14: Kiểm tra xem có một đỉnh cha (parent node) cho đỉnh hiện tại không. Nếu có, tức là đây không phải là đỉnh gốc, ta tiến hành vẽ cạnh và thêm hai đỉnh u và v vào đồ thị. Đầu tiên, ta tạo một đỉnh u với tên là **`Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)]`** và nhãn (label) **`là Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)][:3].`** Sau đó, ta tạo một đỉnh v với tên và nhãn là **`(number_missionaries, number_cannnibals, side)`** tương ứng. Cuối cùng, ta tạo một cạnh edge để nối đỉnh u và v với nhau, với hướng (direction) là **`'forward'`**. Tất cả các đỉnh và cạnh được thêm vào đồ thị.

- Dòng 15 đến dòng 18: Trường hợp không có đỉnh cha (parent node) tức là đỉnh gốc của cây tìm kiếm. Ta chỉ cần tạo một đỉnh v với tên và nhãn là **`(number_missionaries, number_cannnibals, side)`** tương ứng và thêm đỉnh này vào đồ thị.

- Dòng 19: Hàm trả về đối tượng u và v (có thể là None nếu không có đỉnh cha).

- **`is_start_state(number_missionaries, number_cannibals, side)`**: kiểm tra xem một trạng thái cụ thể có phải là trạng thái ban đầu của bài toán "Missionaries and Cannibals" không. Trạng thái ban đầu thường được đặt là (3, 3, 1).

```
def is_start_state(number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == (3, 3, 1) # Nếu trạng thái ban đầu = (3,3,1) return true, nếu không return false
```

- **`is_goal_state(number_missionaries, number_cannibals, side)`**: kiểm tra xem trạng thái nào đó có phải là trạng thái kết thúc không, trạng thái kết thúc là (0,0,0) và tương tự như hàm kiểm tra trạng thái bắt đầu

```
def is_goal_state(number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == (0, 0, 0) # Nếu trạng thái kết thúc = (0, 0, 0) return true, nếu không return false
```

- **`number_of_cannibals_exceeds(number_missionaries, number_cannibals):`** kiểm tra xem số lượng người ăn thịt (cannibals) có vượt quá số lượng nhà truyền giáo (missionaries) trên bất kỳ bờ nào của sông không. Nếu có, hàm trả về True, ngược lại trả về False.

```
def number_of_cannibals_exceeds(number_missionaries, number_cannnibals):
    number_missionaries_right = 3 - number_missionaries # Số người truyền giáo bên bờ phải bằng 3 trừ số người truyền giáo bên bờ trái
    number_cannnibals_right = 3 - number_cannnibals # Số con quỷ bờ phải bằng 3 trừ số con quỷ bờ trái
    return (number_missionaries > 0 and number_cannnibals > number_missionaries)
 or (number_missionaries_right > 0 and number_cannnibals_right > number_missionaries_right)
 # Nếu số quỷ nhiều hơn số người truyền giáo thì hàm trả về True, ngược lại là False
```

- **`generate():`** đây là hàm chính, giúp hình thành lời giải của bài toán. Hàm trả về True nếu đi đến được độ sâu sâu nhất của cây, ngược lại trả về False. Ngoài ra, ta tạo thêm tham số dòng lệnh -d quyết định độ sâu của cây ta muốn tạo (mặc định là 20).

```
def generate():
        global i # Đánh dấu i là biến toàn cục.
        q = deque() # Tạo một hàng đợi (queue) bằng deque để lưu trữ các trạng thái của cây tìm kiếm.
        node_num = 0 # Khởi tạo biến node_num với giá trị 0 để đánh dấu các nút trong cây.
        q.append((3, 3, 1, 0, node_num)) # Thêm trạng thái ban đầu vào hàng đợi. Trạng thái ban đầu bao gồm 3 nhà truyền giáo và 3 người ăn thịt
                                         # trên bờ một, chiều sâu bằng 0 và đánh dấu node số 0

        Parent[(3, 3, 1, 0, node_num)] = None # Gán giá trị Node cho nút ban đầu (node gốc) trong dict Parent.

        while q: # Bắt đầu vòng lặp, lặp cho đến khi hàng đợi rỗng

            number_missionaries, number_cannnibals, side, depth_level, node_num = q.popleft() '''Lấy trạng thái tiếp theo từ hàng đợi và gán
            các giá trị tương ứng cho các biến. Nếu không có trạng thái nào trong hàng đợi, vòng lặp kết thúc'''

            # 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) '''Gọi hàm draw_edge để vẽ cạnh từ đỉnh cha (u)
            đến đỉnh hiện tại (v) và lưu trả về 2 đỉnh này. Hàm draw_edge được gọi là hàm tạo biểu đồ cây tìm kiếm'''

            # Kiểm tra trạng thái hiện tại và thực hiện thay đổi màu sắc cho đỉnh
            if is_start_state(number_missionaries, number_cannnibals, side): # Nếu là trạng thái ban đầu, đánh dấu màu xanh dương
                v.set_style("filled")
                v.set_fillcolor("blue")
                v.set_fontcolor("white")
            elif is_goal_state(number_missionaries, number_cannnibals, side): # Nếu là trạng thái kết thúc, đánh dấu màu xanh lá cây và kết thúc loop
                v.set_style("filled")
                v.set_fillcolor("green")
                continue
                # return True
            elif number_of_cannibals_exceeds(number_missionaries, number_cannnibals): ''' Nếu trạng thái vi phạm ràng buộc (số người ăn thịt lớn hơn
            số nhà truyền giáo trên bất kì bờ nào), đánh dấu màu đỏ và tiếp tục vòng lặp'''
                v.set_style("filled")
                v.set_fillcolor("red")
                continue
            else: # Nếu không thuộc trường hợp nào ở trên, đánh dấu màu cam
                v.set_style("filled")
                v.set_fillcolor("orange")
```

```
            if depth_level == max_depth: # Nếu độ sâu của cây tìm kiếm bằng max_depth, kết thúc vòng lặp và trả về True
                return True

            op = -1 if side == 1 else 1 # Xác định phía bờ sông đối diện

            can_be_expanded = False # Khởi tạo biến can_be_expanded là False để kiểm tra xem trạng thái hiện tại có được mở rộng hay không

            # i = node_num
            for x, y in options: # Duyệt qua tất cả các tùy chọn di chuyển từ trạng thái hiện tại và kiểm tra tính hợp lệ của các di chuyển
                next_m, next_c, next_s = number_missionaries + op * x, number_cannnibals + op * y, int(not side) # Lấy giá trị mới dựa trên di chuyển
                                                                                                                  # và trạng thái hiện tại

                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)
                        ''' Kiểm tra xem trạng thái cha (đỉnh cha) có giá trị None và trạng thái mới khác với trạng thái cha không.
                        Nếu có và di chuyển là hợp lệ, đánh dấu  can_be_expanded = True (trạng thái có thể được mở rộng),
                        tăng giá trị của i lên và thêm trạng thái mới vào hàng đợi '''

            if not can_be_expanded: # nếu không có trạng thái nào có thể được mở rộng (can_be_expanded vẫn là False), đánh dấu màu xám cho đỉnh.
                v.set_style("filled")
                v.set_fillcolor("gray")
        return False # Trả về False để kết thúc hàm sau khi duyệt hết toàn bộ cây tìm kiếm.
```

- Cuối cùng, nếu tập lệnh python chạy là tập chính thì nếu hàm **`generate()`** trả về giá trị đúng, ta xuất hình ảnh đồ thị ra.

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

2. File **solve.py**

Trong file này, chúng ta viết một lớp Solution cùng một số các thuộc tính và phương thức để thuận tiện hơn cho việc đi tìm lời giải của bài toán. Các thuộc tính của lớp nằm ở trong hàm **`__init__()`** bao gồm:

- **`self.start_state = (3, 3, 1):`** trạng thái ban đầu của bài toán.
- **`self.goal_state = (0, 0, 0):`** trạng thái mục tiêu của bài toán.
- **`self.options = [(1, 0), (0, 1), (1, 1), (0, 2), (2, 0)]:`** các toán tử chuyển trạng thái của bài toán.
- **`self.boat_side = ["right", "left"]:`** vị trí của con thuyền (bờ phải hoặc bờ trái).
- **`self.graph = pydot.Dot(graph_type='graph', bgcolor="#fff3af", label="fig: Missionaries and Cannibal State Space Tree", fontcolor="red", fontsize="24"):`** đồ thị lưu các node trạng thái của bài toán, mặc định ban đầu đồ thị có một node, nền màu da, chữ màu đỏ. . .
- **`self.visited = {}:`** một list lưu trữ các trạng thái đã duyệt.
- **`self.solved = False:`** cho biết bài toán có giải được hay không.

```
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
```

Các phương thức trong lớp này bao gồm:

- **is_valid_move(self, number_missionaries, number_cannibals):** Hàm kiểm tra xem một nước đi nào đó có thoả ràng buộc hay không. Hàm này nhận hai tham số là number_missionaries (số lượng nhà truyền giáo) và number_cannnibals (số lượng người ăn thịt).

```
def is_valid_move(self, number_missionaries, number_cannnibals): # Hàm check xem có tồn tại đường đi không
    """
    Checks if number constraints are satisfied
    """
    return (0 <= number_missionaries <= 3) and (0 <= number_cannnibals <= 3) # Số truyền giáo và con quỷ >=0 và <=3 thì trả về True,
                                                                             # ngược lại trả False.
```

- **is_goal_state(self, number_missionaries, number_cannibals, side):** là khai báo của hàm is_goal_state. Hàm này nhận ba tham số: number_missionaries (số lượng nhà truyền giáo), number_cannnibals (số lượng người ăn thịt) và side (bên mà thuyền đang đậu).

```
def is_goal_state(self, number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == self.goal_state # So sánh trạng thái truyền vào với trạng thái mục tiêu,
                                                                             # Nếu bằng trả về True, ngược lại False
```

- **is_start_state(self, number_missionaries, number_cannibals, side):** kiểm tra xem một trạng thái cụ thể (gồm số lượng nhà truyền giáo, số lượng người ăn thịt và bên mà thuyền đang đậy) có phải là trạng thái ban đầu hay không.

```
def is_start_state(self, number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == self.start_state # So sánh trạng thái truyền vào với trạng thái ban đầu,
                                                                              # Nếu bằng trả về True, ngược lại False
```

- **number_of_cannibals_exceeds(self, number_missionaries, number_cannibals):** kiểm tra xem trạng thái có vi phạm ràng buộc về số lượng người ăn thịt hay không.

```
def number_of_cannibals_exceeds(self, number_missionaries, number_cannnibals):
    number_missionaries_right = 3 - number_missionaries # Tạo biến number_missionaries_right để tính số lượng người ăn thịt còn lại bên phải
    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 này trả về kết quả của việc kiểm tra xem có vi phạm ràng buộc về số lượng người ăn thịt hay không. Có hai điều kiện
           để kiểm tra: Một là bờ mà thuyền đang đậu (bờ hiện tại) có nhiều quỷ hơn số lượng nhà truyền giáo không,
           hai là bờ bên phải (nếu thuyền đậu bên trái) có nhiều quỷ hơn số lượng nhà truyền giáo không.
           Trả về True nếu 1 trong 2 thỏa còn cả 2 không thỏa trả về False'''
```

- **`write_image(self, file_name="state_space.png"):`** được sử dụng để ghi biểu đồ của không gian trạng thái vào một hình ảnh và lưu vào tệp tin. Hàm này nhận một tham số là file_name, đại diện cho tên tệp tin đầu ra mặc định là "state_space.png". Tên tệp tin này sẽ được sử dụng để lưu biểu đồ.

```
def write_image(self, file_name="state_space.png"):
    try: # sử dụng try-except để xử lý lỗi.
        self.graph.write_png(file_name) # Ghi biểu đồ của không gian trạng thái vào một tệp hình ảnh với tên là file_name.
                                        # Phương thức write_png được gọi từ đối tượng self.graph
    except Exception as e:
    '''Bắt đầu khối except để xử lý các ngoại lệ hoặc lỗi xảy ra trong khối try. Bất kỳ ngoại lệ nào cũng được bắt,
    (thể hiện bằng Exception as e), và ngoại lệ này được gán vào biến e'''

        print("Error while writing file", e) # In ra thông báo lỗi và hiển thị thông tin về ngoại lệ 'e' để mô tả lỗi xảy ra.
    print(f"File {file_name} successfully written.")
    ''' Nếu không có lỗi, in ra thông báo tập hình ảnh đã được ghi thành công. Giá trị {file_name} được thay thế bằng tên thực tế
    của tệp tin mà bạn truyền vào.'''
```

- **`def solve(self, solve_method="dfs"):`** có mục đích giải quyết bài toán sử dụng phương pháp tìm kiếm theo chiều sâu ("dfs") hoặc tìm kiếm theo chiều rộng ("bfs")

```
def solve(self, solve_method="dfs"): # Hàm này nhận một tham số solve_method, cho biết phương pháp giải quyết bạn muốn sử dụng
    self.visited = dict() # Tạo một dict rỗng visited trong đối tượng self để theo dõi trạng thái đã được thăm hay chưa.
    Parent[self.start_state] = None # Gán giá trị None cho trạng thái ban đầu (nút gốc) trong dict Parent
    Move[self.start_state] = None # Gán giá trị None cho trạng thái ban đầu (nút gốc) trong dict Move
    node_list[self.start_state] = None # Gán giá trị None cho trạng thái ban đầu (nút gốc) trong danh sách node_list

    return self.dfs(*self.start_state, 0) if solve_method == "dfs" else self.bfs()
    '''Dựa vào giá trị solve_method, hàm solve sẽ gọi entweder self.dfs (tìm kiếm theo chiều sâu) hoặc self.bfs (tìm kiếm theo chiều rộng)'''
```

- **`draw_legend(self):`** được sử dụng để vẽ chú thích (legend) cho biểu đồ (graph). Chú thích này giúp giải thích ý nghĩa của các màu và nhãn trên biểu đồ
    1. Tạo một cụm (cluster) mới cho chú thích với graph_name là legend. Điều này giúp nhóm các node và edge của chú thích lại với nhau.
    2. Tạo node với nhãn 1, đại diện cho trạng thái ban đầu, có màu nền màu xanh, và nhãn Start Node. Điều này mô tả nút bắt đầu của tìm kiếm.
    3. Tạo node với nhãn 2, đại diện cho trạng thái đã thăm và bị loại bỏ, có màu nền đỏ và nhãn Killed Node. Điều này mô tả trạng thái mà tìm kiếm đã xem xét và bỏ qua.
    4. Tạo node với nhãn 3, đại diện cho trạng thái là lời giải, có màu nền màu vàng và nhãn Solution nodes. Điều này mô tả trạng thái mà đạt được lời giải.
    5. Tạo node với nhãn 4, đại diện cho trạng thái không thể được mở rộng (hoặc không hợp lệ), có màu nền màu xám và nhãn Can't be expanded.
    6. Tạo node với nhãn 5, đại diện cho trạng thái mục tiêu, có màu nền màu xanh lá cây và nhãn Goal node.
    7. Tạo node với nhãn 6 để cung cấp mô tả chi tiết về ý nghĩa của các trạng thái và toán tử.
    8. Thêm các node vừa tạo vào cụm graphlegend.

9. Tạo các edge ẩn (style="invis") giữa các node trong chú thích để xác định thứ tự mà chú thích sẽ được hiển thị.
10. Thêm cụm graphlegend và các edge vào biểu đồ self.graph.

```
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"))
```

- **`draw(self, *, number_missionaries_left, number_cannibals_left, number_missionaries_right, number_cannibals_right):`** Đây là khai báo hàm draw. Hàm này nhận bốn tham số gắn sao lưu thông tin về số lượng nhà truyền giáo và người ăn thịt trên cả hai bờ của sông.

```
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)
    ''' Dòng này sử dụng thư viện emoji để tạo biểu tượng người đàn ông (:old_man:) nhiều lần tương ứng với number_missionaries_left,
    sau đó gán kết quả cho biến left_m. Nhằm hiển thị số lượng nhà truyền giáo trên bờ bên trái. '''
    left_c = emoji.emojize(f":ogre: " * number_cannnibals_left)
    ''' Tương tự như dòng trước, nhưng ở đây ta dùng biểu tượng người khổng lồ (:ogre:) để biểu diễn số lượng quỷ bên bờ trái'''
    right_m = emoji.emojize(f":old_man: " * number_missionaries_right)
    ''' Tương tự như left_m, nhưng áp dụng cho bờ phải'''
    right_c = emoji.emojize(f":ogre: " * number_cannnibals_right)
    ''' Tương tự như left_c, nhưng áp dụng cho bờ phải'''
    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))
    ''' Dòng này in biểu đồ của trạng thái bờ sông lên màn hình. Biểu đồ bao gồm hai bờ sông với số lượng nhà truyền giáo (old man) và số quỷ (ogre) tương ứng.
    Các dấu gạch ngang tạo sự phân tách giữa 2 bờ. Sự điều chỉnh dựa vào số lượng kí tự để đảm bảo biểu đồ hiển thị đúng vị trí.'''
    print("") # Dòng này in một dòng trống để tạo khoảng cách giữa các biểu đồ trạng thái liên tiếp.
```

- **`draw_edge(self, number_missionaries, number_cannibals, side, depth_level):`** vẽ hai trạng thái nào đó và cạnh nối hai trạng thái đó với nhau. Nếu là trạng thái bắt đầu, hàm chỉ vẽ node mà không vẽ cạnh nối. Hàm này

nhận bốn tham số là số lượng nhà truyền giáo, số lượng người ăn thịt, bên nào của sông (0 hoặc 1), và mức độ độ sâu trong cây tìm kiếm.

```
def draw_edge(self, number_missionaries, number_cannnibals, side, depth_level):
    u, v = None, None # Khởi tạo 2 biến u, v với giá trị là None
    if Parent[(number_missionaries, number_cannnibals, side)] is not None: # Kiểm tra xem đỉnh cha của trạng thái hiện tại có tồn tại không
        u = pydot.Node(str(Parent[(number_missionaries, number_cannnibals, side)] + (depth_level - 1, )),
                       label=str(Parent[((number_missionaries, number_cannnibals, side))]))
        '''Tạo đỉnh u trong biểu đồ với nhãn là trạng thái của đỉnh cha. Nhãn này giúp đại diện cho trạng thái của đỉnh cha và sẽ được hiển thị trong biểu đồ'''

        self.graph.add_node(u) # Thêm đỉnh u vào biểu đồ

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

        edge = pydot.Edge(str(Parent[(number_missionaries, number_cannnibals, side)] + (depth_level - 1, )),
                          str((number_missionaries, number_cannnibals, side, depth_level) ), dir='forward')
        ''' Tạo một cạnh từ đỉnh cha u đến đỉnh hiện tại v, thuộc tính dir='forward xác định hướng của cạnh '''
        self.graph.add_edge(edge) # Thêm cạnh vào biểu đồ
    else: # Nếu đỉnh cha không tồn tại
        # For start node
        v = pydot.Node(str((number_missionaries, number_cannnibals, side, depth_level)),
                       label=str((number_missionaries, number_cannnibals, side)))
        '''Tạo đỉnh v với nhãn là trạng thái ban đầu'''
        self.graph.add_node(v) # Thêm đỉnh v vào biểu đồ
    return u, v # Trả về các đỉnh u (cha) và v (hiện tại) để sử dụng trong việc vẽ biểu đồ
```

- **`bfs(self)`**: hàm này dùng để thực hiện thuật toán BFS.

```
def bfs(self):
    q = deque() # Tạo 1 hàng đợi rỗng để lưu trữ các trạng thái trong quá trình duyệt BFS
    q.append(self.start_state + (0, )) # Thêm trạng thái ban đầu vào hàng đợi, kèm theo mức độ sâu bằng 0
    self.visited[self.start_state] = True # Đánh dấu trạng thái ban đầu là đã được thăm

    while q: # Bắt đầu vòng lặp, lặp đến khi hàng đợi rỗng
        number_missionaries, number_cannnibals, side, depth_level = q.popleft() # lấy trạng thái tiếp theo từ hàng đợi và gán giá trị tương ứng cho các biến
        # 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)
        '''Vẽ cạnh từ trạng thái cha (u) đến trạng thái hiện tại (v) và lưu trả về 2 đỉnh  '''


        if self.is_start_state(number_missionaries, number_cannnibals, side): # Kiểm tra trạng thái hiện tại có phải trạng thái ban đầu không
            v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled" để màu nền có thể được đặt
            v.set_fillcolor("blue") # Đặt màu nền của đỉnh v thành xanh để biểu thị trạng thái ban đầu
            v.set_fontcolor("white") # Đặt font chữ của đỉnh v thành trắng để văn bản trên đỉnh dễ đọc
        elif self.is_goal_state(number_missionaries, number_cannnibals, side): # Kiểm tra trạng thái hiện tại có phải trạng thái mục tiêu không
            v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled"
            v.set_fillcolor("green") # Đặt màu nền cho đỉnh v thành xanh lá cây để biểu diễn trạng thái mục tiêu
            return True # Trả về True
        elif self.number_of_cannibals_exceeds(number_missionaries, number_cannnibals): # Kiểm tra có vi phạm ràng buộc về số quỷ không
            v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled"
            v.set_fillcolor("red") # Đặt màu nền của đỉnh v thành đỏ để biểu thị trạng thái vi phạm
            continue # tiếp tục vòng lặp và không xem xét trạng thái này nữa
        else: # Trường hợp không thuộc trường hợp nào ở trên
            v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled"
            v.set_fillcolor("orange")  # Đặt màu nền cho đỉnh v thành cam để biểu thị trạng thái bình thường

        op = -1 if side == 1 else 1 # Xác định phía bên kia của sông

        can_be_expanded = False # Khởi tạo biến can_be_expanded là False để kiểm tra xem trạng thái hiện tại có thể mở rộng hay không
```

```
        for x, y in self.options: # Duyệt qua tất cả các tùy chọn di chuyển từ trạng thái hiện tại
            next_m, next_c, next_s = number_missionaries + op * x, number_cannnibals + op * y, int(not side)
            #Tính toán trạng thái mới dựa trên dựa trên di chuyển và trạng thái hiện tại

            if (next_m, next_c, next_s) not in self.visited: # Kiểm tra trạng thái mới đã thăm chưa
                if self.is_valid_move(next_m, next_c): # Kiểm tra di chuyển này có hợp lệ không
                    can_be_expanded = True # Đánh dấu can_be_expanded là True để biểu thị rằng trạng thái mới có thể mở rộng
                    self.visited[(next_m, next_c, next_s)] = True # Đánh dấu trạng thái mới đã được thăm
                    q.append((next_m, next_c, next_s, depth_level + 1)) # Thêm trạng thái mới vào hàng đợi với độ sâu tăng thêm 1

                    # Keep track of parent and corresponding move
                    Parent[(next_m, next_c, next_s)] = (number_missionaries, number_cannnibals, side) # Ghi lại đỉnh cha của trạng thái mới
                    Move[(next_m, next_c, next_s)] = (x, y, side) # Ghi lại đường đi để đến trạng thái mới
                    node_list[(next_m, next_c, next_s)] = v # Lưu trạng thái hiện tại vào danh sách các trạng thái

        if not can_be_expanded: # Nếu không có trạng thái nào có thể được mở rộng
            v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled"
            v.set_fillcolor("gray") # Đặt màu nền cho đỉnh v thành xám để biểu thị trạng thái không thể được mở rộng
    return False # Trả về false để kết thúc trạng thái tìm kiếm BFS nếu không có trạng thái nào thỏa mãn.
```

- **`dfs(self, number_missionaries, number_cannibals, side, depth_level):`** hàm này dùng để thực hiện thuật toán DFS. Khác với hàm bfs(self), ta thêm các tham số để có thể gọi đệ quy hàm này nhiều lần.

```
def dfs(self, number_missionaries, number_cannnibals, side, depth_level):
    self.visited[(number_missionaries, number_cannnibals, side)] = True # Đánh dấu trạng thái hiện tại là đã được thăm.

    # Draw Edge from u -> v
    # Where u = Parent[v]
    u, v = self.draw_edge(number_missionaries, number_cannnibals, side, depth_level) #Vẽ cạnh từ trạng thái cha đến trạng thái hiện tại và lưu trả về hai đỉnh.


    if self.is_start_state(number_missionaries, number_cannnibals, side): # Kiểm tra xem trạng thái hiện tại có phải trạng thái ban đầu không.
        v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled" để màu nền có thể được đặt.
        v.set_fillcolor("blue") # Đặt màu nền của đỉnh v thành màu xanh để biểu thị trạng thái ban đầu.
    elif self.is_goal_state(number_missionaries, number_cannnibals, side): # Kiểm tra xem trạng thái hiện tại có phải là trạng thái mục tiêu không.
        v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled".
        v.set_fillcolor("green") # Đặt màu nền của đỉnh v thành xanh lá cây để biểu thị trạng thái mục tiêu.
        return True # Trả về True để kết thúc tìm kiếm vì đã tìm thấy lời giải.
    elif self.number_of_cannibals_exceeds(number_missionaries, number_cannnibals): # Kiểm tra xem có vi phạm ràng buộc về số lượng người ăn thịt không.
        v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled".
        v.set_fillcolor("red") # Đặt màu nền của đỉnh v thành đỏ để biểu thị trạng thái vi phạm
        return False # Trả về False để kết thúc tìm kiếm vì trạng thái hiện tại không thỏa mãn.
    else: # Trường hợp không thuộc trường hợp nào ở trên
        v.set_style("filled") # Đặt kiểu cho đỉnh v thành "filled".
        v.set_fillcolor("orange") # Đặt màu nền của đỉnh v thành màu cam để biểu thị trạng thái bình thường.

    solution_found = False # Khởi tạo biến solution_found với giá trị False để theo dõi xem có lời giải nào được tìm thấy hay không.
    operation = -1 if side == 1 else 1 # Xác định phía bên kia của sông.

    can_be_expanded = False # Khởi tạo biến can_be_expanded với giá trị False để kiểm tra xem trạng thái hiện tại có thể mở rộng hay không.
```

```
    for x, y in self.options: # Duyệt qua tất cả các tùy chọn di chuyển từ trạng thái hiện tại.
        next_m, next_c, next_s = number_missionaries + operation * x, number_cannnibals + operation * y, int(not side)
        # Tính toán trạng thái mới dựa trên di chuyển và trạng thái hiện tại.

        if (next_m, next_c, next_s) not in self.visited: # Kiểm tra xem trạng thái mới đã được thăm chưa.
            if self.is_valid_move(next_m, next_c): # Kiểm tra xem di chuyển đến trạng thái mới có hợp lệ không.
                can_be_expanded = True # Đánh dấu can_be_expanded là True (trạng thái hiện tại có thể được mở rộng)
                # Keep track of Parent state and corresponding move
                Parent[(next_m, next_c, next_s)] = (number_missionaries, number_cannnibals, side) # Ghi lại đỉnh cha của trạng thái mới.
                Move[(next_m, next_c, next_s)] = (x, y, side)  # Ghi lại di chuyển để đến trạng thái mới.
                node_list[(next_m, next_c, next_s)] = v # Lưu trạng thái hiện tại vào danh sách các trạng thái.

                solution_found = (solution_found or self.dfs(next_m, next_c, next_s, depth_level + 1))
                # Gọi đệ quy hàm dfs để tìm kiếm từ trạng thái mới, và cập nhật biến solution_found nếu tìm thấy lời giải.

                if solution_found: # Nếu kiếm được lời giải
                    return True # Trả về True

    if not can_be_expanded:
        '''Nếu không có trạng thái nào có thể được mở rộng
        (điều này có nghĩa rằng chúng ta đã kiểm tra tất cả các tùy chọn di chuyển và không có trạng thái mới nào hợp lệ để mở rộng).'''
        v.set_style("filled") # Đặt kiểu của đỉnh v thành "filled".
        v.set_fillcolor("gray")  # Đặt màu nền của đỉnh v thành màu xám để biểu thị rằng trạng thái này không thể được mở rộng.

    self.solved = solution_found # Cập nhật biến solved với giá trị của solution_found.
    return solution_found # Trả về solution_found để kết thúc tìm kiếm và thông báo kết quả.
```

3. File **main.py**

Nhiệm vụ chính của file này là tạo ra một đối tượng của lớp Solution và gọi đến các phương thức của lớp đó để giải bài toán. Trong file này, ta còn có thêm một số tham số dòng lệnh đi kèm:

- **-m:** phương thức nào được sử dụng (bfs hoặc dfs).
- **-l:** quyết định xem có chú thích cho đồ thị hay không (True hoặc False).

```
arg = argparse.ArgumentParser() # Tạo một đối tượng arg của lớp ArgumentParser để quản lý các dòng lệnh.
arg.add_argument("-m", "--method", required=False, help="Specify which method to use")
''' Thêm một đối số -m hoặc -method cho chương trình. Đối số này cho phép người dùng chọn phương thức giải quyết bằng cách
chỉ định tên phương thức sau khi sử dụng -m hoặc --method trên dòng lệnh. Đối số không bắt buộc (required=False) và có mô tả ("help")
giúp người dùng hiểu cách sử dụng.'''
arg.add_argument("-l", "--legend", required=False, help="Specify if you want to display legend on graph")
'''Tương tự như đối số phương thức, đối số -l hoặc --legend cho phép người dùng chỉ định xem có muốn hiển thị chú thích (legend) trên biểu đồ hay không.
Đối số không bắt buộc (required=False) và có mô tả ("help") giúp người dùng hiểu cách sử dụng.'''
args = vars(arg.parse_args())
'''Sử dụng parge_args() để phân tích các đối số trên dòng lệnh và lưu chúng vào biến args. Hàm này trả về một dict chứa các đối số và giá
trị tương ứng của chúng'''

solve_method = args.get("method", "bfs")
# Sử dụng arg.get("method","bfs") để lấy giá trị đối số "method" từ biến args. Nếu đối số "method" không được cung cấp, giá trị mặc định là "bfs".
legend_flag = args.get("legend", False)
# Tương tự như trên, lấy giá trị của đối số "legend" từ biến args. Nếu không có đối số "legend", giá trị mặc định là False.
```

Trong đây chương trình xử lý các đối số dòng lệnh :

1. **`arg = argparse.ArgumentParser():`** Tạo một đối tượng arg của lớp ArgumentParser để quản lý các đối số dòng lệnh.
2. **`arg.add_argument("-m", "–method", required=False, help="Specify which method to use"):`** Thêm một đối số -m hoặc –method cho chương trình. Đối số này cho phép người dùng chọn phương thức giải quyết bằng cách chỉ định tên phương thức sau khi sử dụng -m hoặc –method trên dòng lệnh. Đối số không bắt buộc (required=False) và có mô tả ("help") giúp người dùng hiểu cách sử dụng.
3. **`. arg.add_argument("-l", "–legend", required=False, help="Specify if you want to display legend on graph")`**: Tương tự như đối số phương thức, đối số -l hoặc –legend cho phép người dùng chỉ định xem có muốn hiển thị chú thích (legend) trên biểu đồ hay không. Đối số không bắt buộc (required=False) và có mô tả ("help") giúp người dùng hiểu cách sử dụng.
4. **`args = vars(arg.parse_args()):`** Sử dụng parse_args() để phân tích các đối số trên dòng lệnh và lưu chúng vào biến args. Hàm này trả về một từ điển chứa các đối số và giá trị tương ứng của chúng.

Tiếp theo, chương trình trích xuất các giá trị đối số từ biến **`args`**:

1. **`solve_method = args.get("method", "bfs"):`** Sử dụng **`args.get("method", "bfs")`** để lấy giá trị của đối số "method" từ biến args. Nếu đối số "method" không được cung cấp, giá trị mặc định là "bfs".
2. **`legend_flag = args.get("legend", False):`** Tương tự như trên, lấy giá trị của đối số " legend" từ biến args. Nếu không có đối số "legend", giá trị mặc định là False.

```
def main():
    s = Solution()  # Tạo một đối tượng của lớp Solution gọi là s, để sử dụng các phương thức và thuộc tính của lớp này để giải bài toán.

    if(s.solve(solve_method)):
    #Gọi phương thức solve() của đối tượng s với đối số solve_method. Nếu phương thức solve() trả về True, tức là đã tìm thấy lời giải, thực hiện các bước sau:


        s.show_solution() # Hiển thị lời giải trên màn hình

        output_file_name = f"{solve_method}" #Xây dựng tập đầu ra (output_file_name) dựa trên phương thức giải (solve_method) đã chọn.
        # Draw legend if legend_flag is set
        if legend_flag:  # Kiểm tra biến legend_flag
            if legend_flag[0].upper() == 'T' : # Nếu legend_flag được đặt và có giá trị bắt đầu bằng "T" (hoặc "t")
                output_file_name += "_legend.png"  # Chú thích legend vào tệp đầu ra
                s.draw_legend() # Vẽ chú thích
            else:
                output_file_name += ".png" # Sử dụng tên tệp mặc định (output_file_name += ".png").
        else:
             output_file_name += ".png"

        # Write State space tree
        s.write_image(output_file_name) # Ghi biểu đồ cây trạng thái vào tập đầu ra.
    else: # Nếu không tìm thấy lời giải (kết quả trả về từ solve() là False)
        raise Exception("No solution found") # Ném một ngoại lệ với thông báo "No solution found".
```

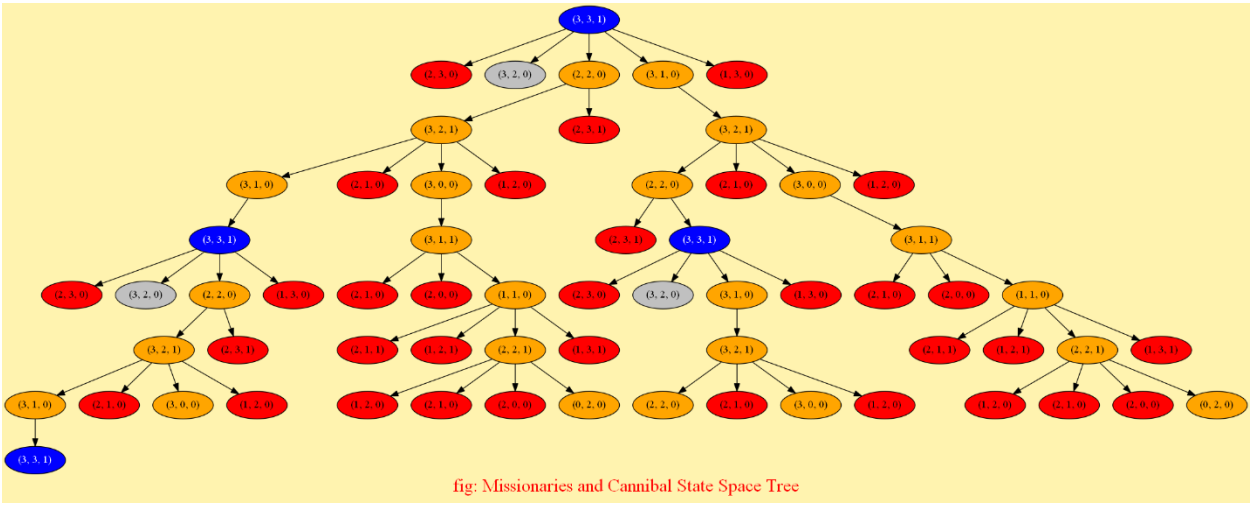
4. Kết quả

- Khởi tạo cây không gian trạng thái:

Gõ lệnh python generate_full_space_tree.py -d 8 trên terminal Ta thu được kết quả :

```
❯ python3 generate_full_space_tree.py -d 8
File state_space_8.png successfully written.
```

Nếu ta gõ lệnh trên terminal python generate_full_space_tree.py -d 20, tức là thay đổi độ sâu tối đa của cây tìm kiếm thành 20, ta thu được cây tìm kiếm như hình 5: Có thể thấy được rằng khi tăng độ sâu tối đa của cây tìm kiếm lên 20, ta có thể thu được lời giải của bài toán. Ngược lại, nếu độ sâu tối da của cây tìm kiếm là 8, ta không thể tìm được lời giải của bài toán.



Hình 1: Cây không gian trạng thái cho bài toán với độ sâu tối đa là 8

- **Giải bài toán với DFS**

Thực thi lệnh python main.py -m dfs, ta thu được kết quả trên màn hình console ở hình dưới và file dfs.png. Do thuật toán DFS ưu tiên duyệt trạng thái theo chiều sâu, cho nên ta có thể thấy lời giải của bài toán nghiêng hẳn về bên tay phải (các node màu vàng).

Hình 2: Màn hình console cho thuật toán DFS và không chú thích đồ thị

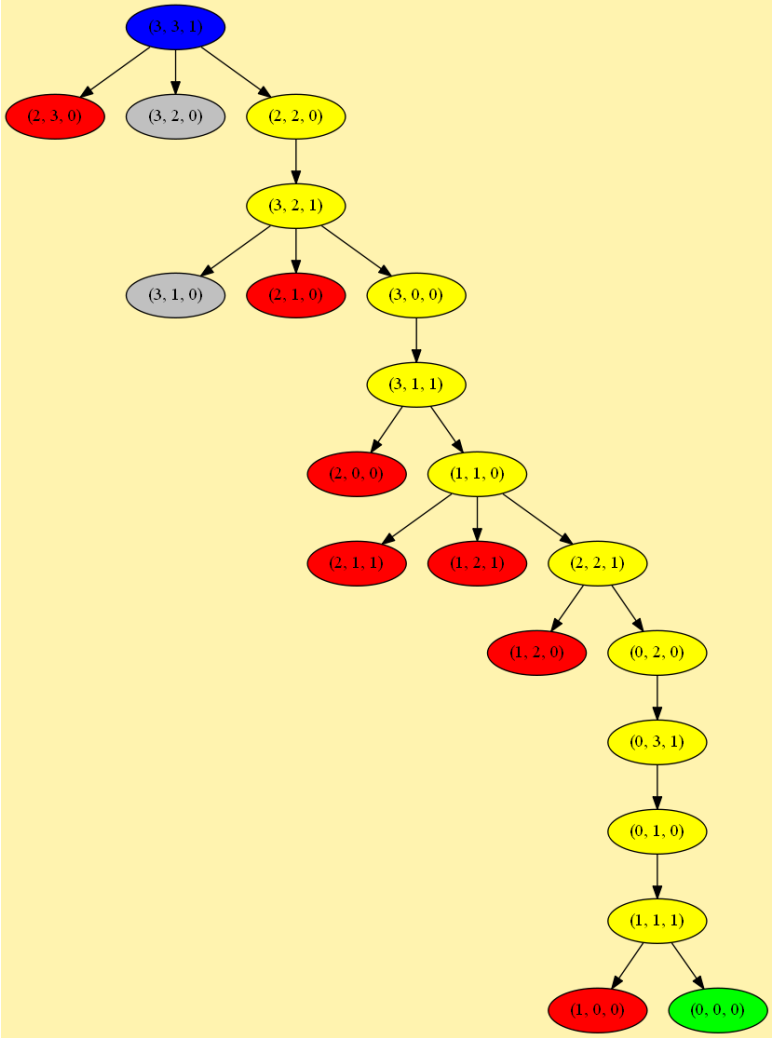


fig: Missionaries and Cannibal State Space Tree

(a): Không chú thích đồ thị

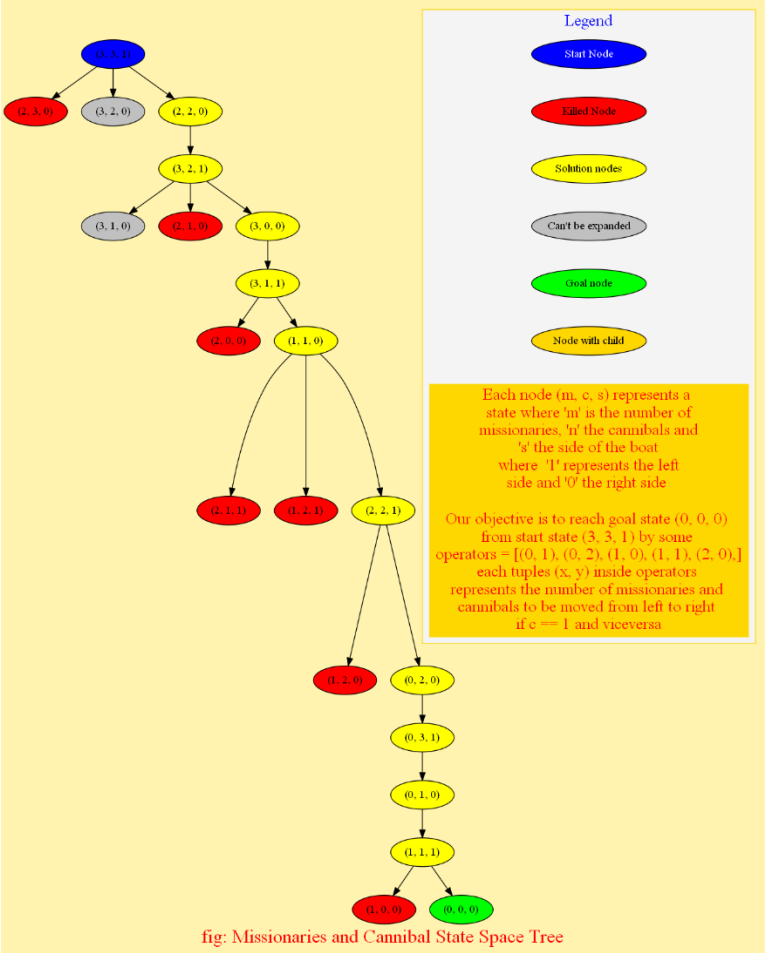


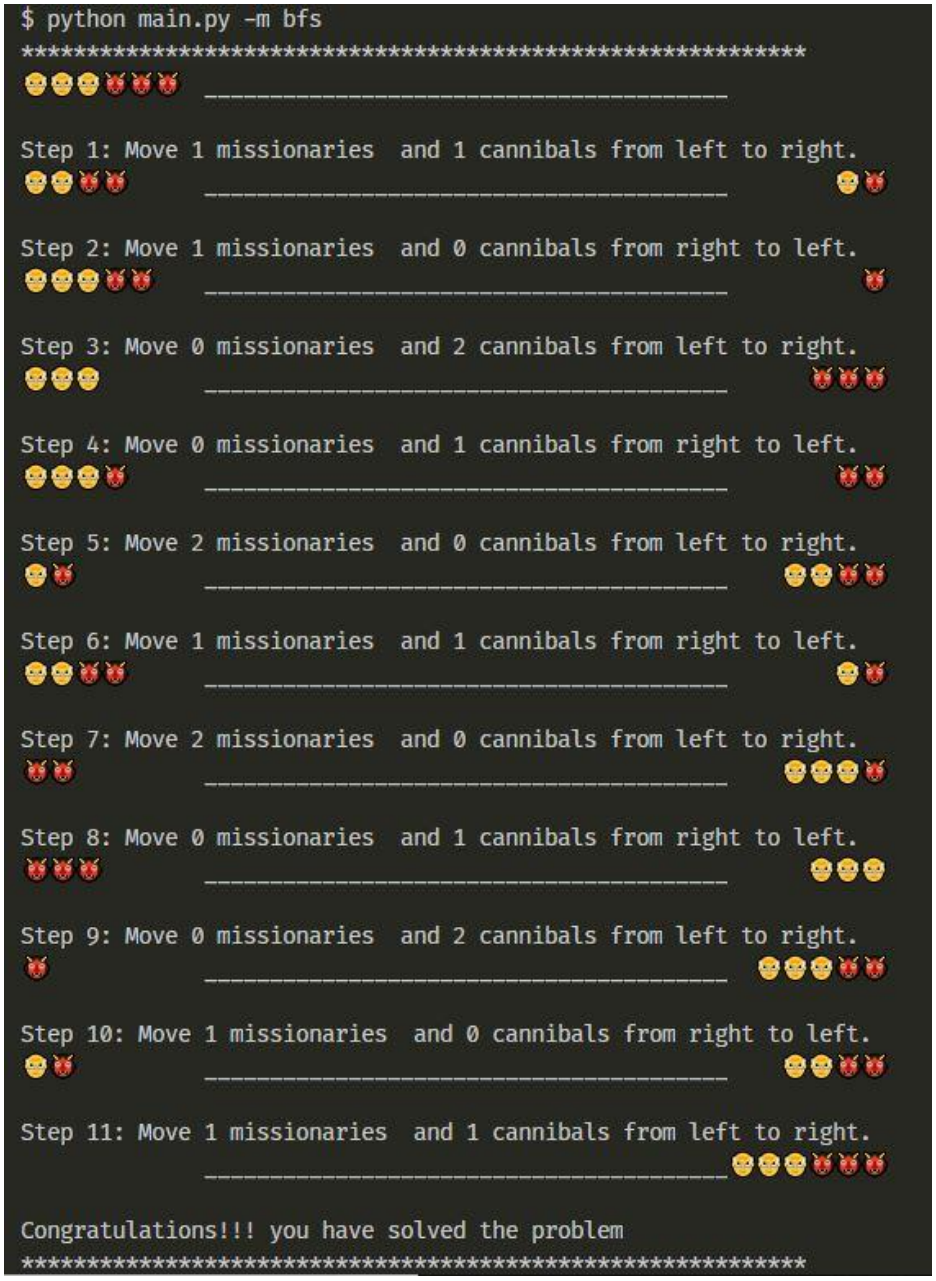
fig: Missionaries and Cannibal State Space Tree

(b): Có chú thích đồ thị

Hình: Cây tìm kiếm trạng thái bằng thuật toán DFS

- **Giải bài toán với BFS**

Thực thi lệnh python main.py -m bfs, ta thu được kết quả t và file bfs.png như hình dưới. Do thuật toán BFS ưu tiên duyệt trạng thái theo chiều rộng, cho nên ta có thể thấy các node được mở rộng ra theo nhiều hướng hơn so với thuật toán DFS.



Hình 4: Màn hình console cho thuật toán BFS và không chú thích đồ thị

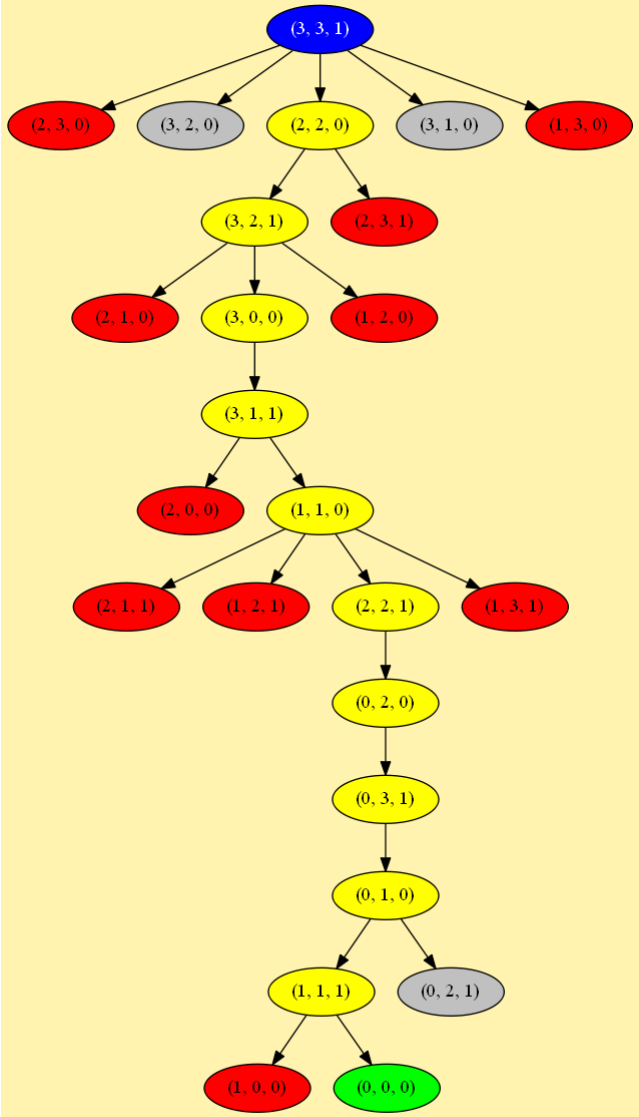
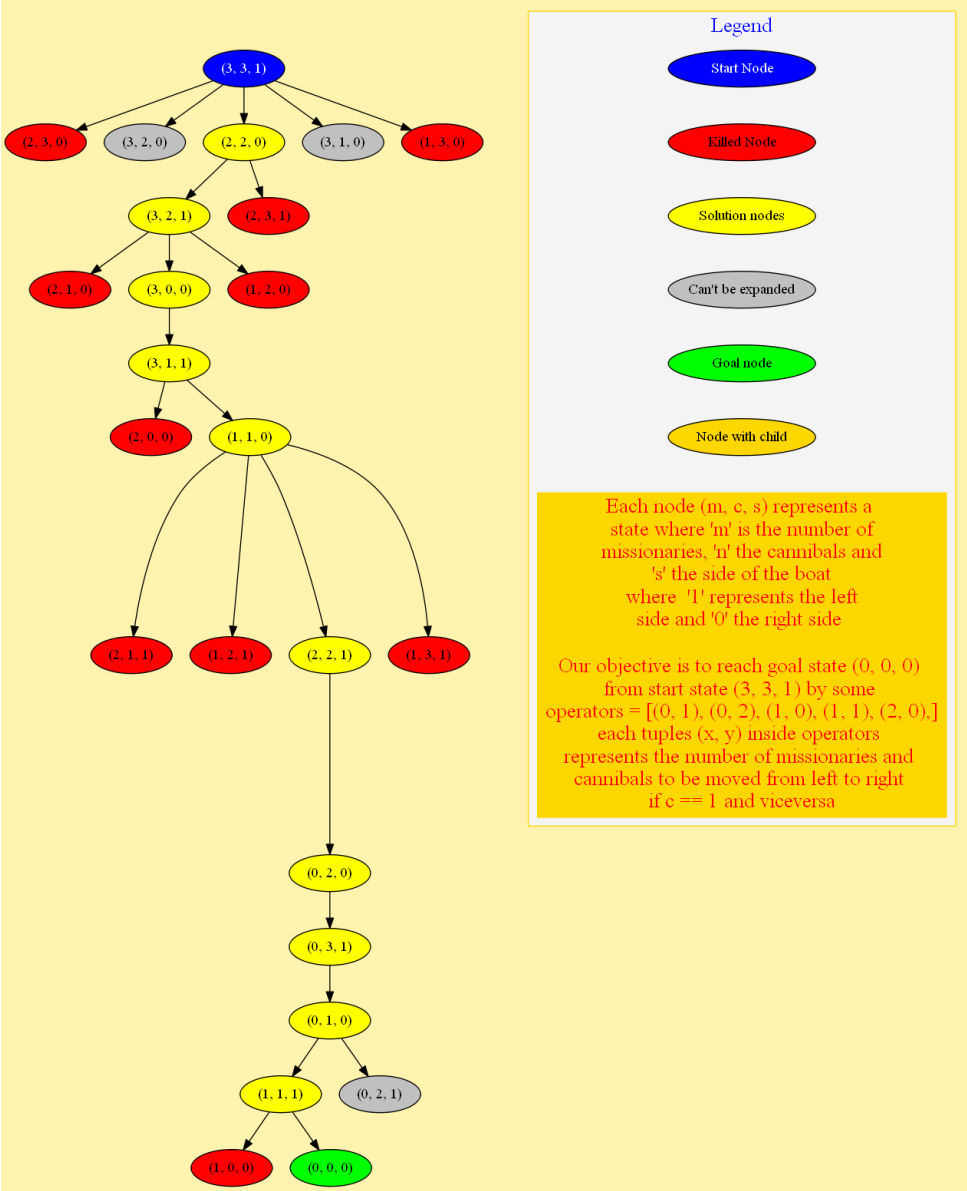


fig: Missionaries and Cannibal State Space Tree

(a): Không chú thích đồ thị



(b) Có chú thích đồ thị

Hình: Cây tìm kiếm trạng thái bằng thuật toán BFS

5. Nhận xét

Cả hai thuật toán đều tìm được đáp án cho bài toán, nhưng thuật toán DFS tìm được đáp án nhanh hơn so với thuật toán BFS. Điều này có thể giải thích bằng cách nhìn vào cây tìm kiếm của hai thuật toán. Thuật toán DFS tìm kiếm theo chiều sâu, nên nó sẽ mở rộng các node ở mức thấp nhất trước, sau đó mới mở rộng các node ở mức cao hơn. Trong khi đó, thuật toán BFS tìm kiếm theo chiều rộng, nên nó sẽ mở rộng các node ở mức thấp nhất theo thứ tự từ trái sang phải,

sau đó mới mở rộng các node ở mức cao hơn. Do đó, thuật toán DFS tìm được đáp án nhanh hơn so với thuật toán BFS.