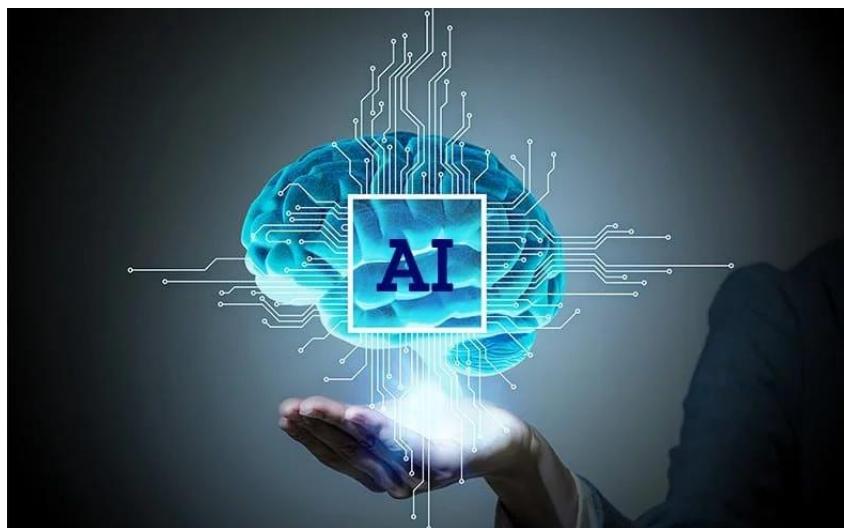


BỘ GIÁO DỤC VÀ ĐÀO TẠO  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
KHOA TOÁN - TIN HỌC

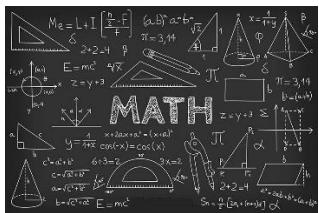


**BÀI TẬP THỰC HÀNH TUẦN 2**



MÔN HỌC: Nhập môn AI  
Sinh Viên: Trần Công Hiếu - 21110294  
Lớp: 21TTH\_KDL

TP.HCM, ngày 06 tháng 11 năm 2023



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM  
KHOA TOÁN – TIN HỌC



## BÀI BÁO CÁO BÀI TẬP THỰC HÀNH TUẦN 2 HKL - NĂM HỌC: 2023-2024

---

**MÔN: NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

**SINH VIÊN: TRẦN CÔNG HIẾU**

**MSSV: 21110294**

**LỚP: 21TTH\_KDL**

NHẬN XÉT

Tp.HCM, Ngày.....Tháng.....Năm 2021

Giảng viên Bộ môn

# **MỤC LỤC**

<b>I. CÀI ĐẶT VÀ PHÁT HIỆN LỖI.....</b>	<b>5</b>
1. Lỗi chưa cài thư viện pydot. ....	5
2. Chưa cài đặt Graphviz.....	5
3. Lỗi logic try except. ....	8
4. Cài đặt các yêu cầu trong file requirement.txt.....	9
<b>II. TRÌNH BÀY CHI TIẾT.....</b>	<b>10</b>
1. Giải thích file generate_full_space_tree.py. ....	10
2. Giải thích file main.py.....	22
3. Giải thích file solve.py.....	25
<b>III. CHẠY ĐOẠN MÃ. .....</b>	<b>47</b>
1. Khởi tạo cây không gian trạng thái. ....	47
2. Cây DFS. ....	48
3. Cây BFS. ....	51

# I. CÀI ĐẶT VÀ PHÁT HIỆN LỖI.

## 1. Lỗi chưa cài thư viện pydot.

```
Traceback (most recent call last):
  File "c:\Users\PC-LENOVO\Downloads\HBM_US-er\HBM5\Thực hành Nhập môn AI\Tuần 2\generate_full_space_tree.py", line 7, in <module>
    import pydot
ModuleNotFoundError: No module named 'pydot'
```

Cách khắc phục: pip install pydot..

## 2. Chưa cài đặt Graphviz.

```
C:\Users\PC-LENOVO\AppData\Local\Programs\Python\Python310\python.exe "C:\Users\PC-LENOVO\Downloads\test code lab 2 ai\generate_full_space_tree.py"
Error while writing file [WinError 2] "dot" not found in path.
File state_space_20.png successfully written.

Process finished with exit code 0
```

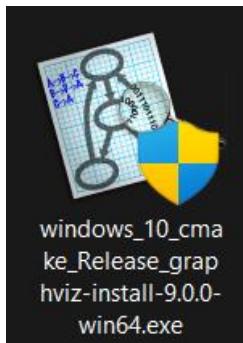
Cách khắc phục: install Graphviz.

- Truy cập vào đường link như bên dưới, chọn mục Download.

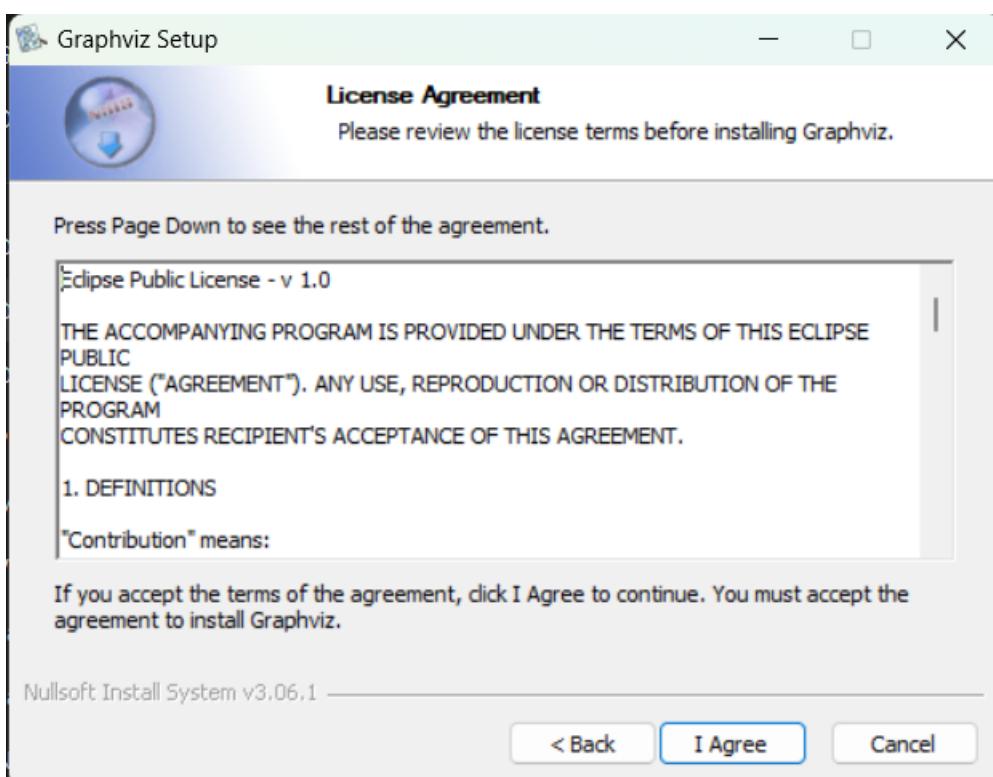
The screenshot shows the Graphviz download page on graphviz.org. The left sidebar has a 'Download' section with links to Source Code, Documentation, and various language and format options. The main content area is titled 'Windows' and lists several stable Windows install packages built with Microsoft Visual Studio 16 2019. Each package is accompanied by a ZIP archive and EXE installer link, both with SHA256 checksums. To the right of the content area is a sidebar with links to View page source, Edit this page, Create child page, Create documentation issue, Print entire section, Source Code, Executable Packages, Linux, Windows, Mac, Solaris, and Other Unix.

- Stable Windows install packages, built with Microsoft Visual Studio 16 2019:
  - graphviz-9.0.0
    - graphviz-9.0.0 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
    - graphviz-9.0.0 (64-bit) EXE installer [sha256]
    - graphviz-9.0.0 (32-bit) EXE installer [sha256]
  - graphviz-8.1.0
    - graphviz-8.1.0 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
    - graphviz-8.1.0 (64-bit) EXE installer [sha256]
    - graphviz-8.1.0 (32-bit) EXE installer [sha256]
  - graphviz-8.0.5
    - graphviz-8.0.5 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
    - graphviz-8.0.5 (64-bit) EXE installer [sha256]
    - graphviz-8.0.5 (32-bit) EXE installer [sha256]
  - graphviz-8.0.3
    - graphviz-8.0.3 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
    - graphviz-8.0.3 (64-bit) EXE installer [sha256]
    - graphviz-8.0.3 (32-bit) EXE installer [sha256]
  - graphviz-8.0.2
    - graphviz-8.0.2 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
    - graphviz-8.0.2 (32-bit) EXE installer [sha256]
    - graphviz-8.0.2 (64-bit) EXE installer [sha256]

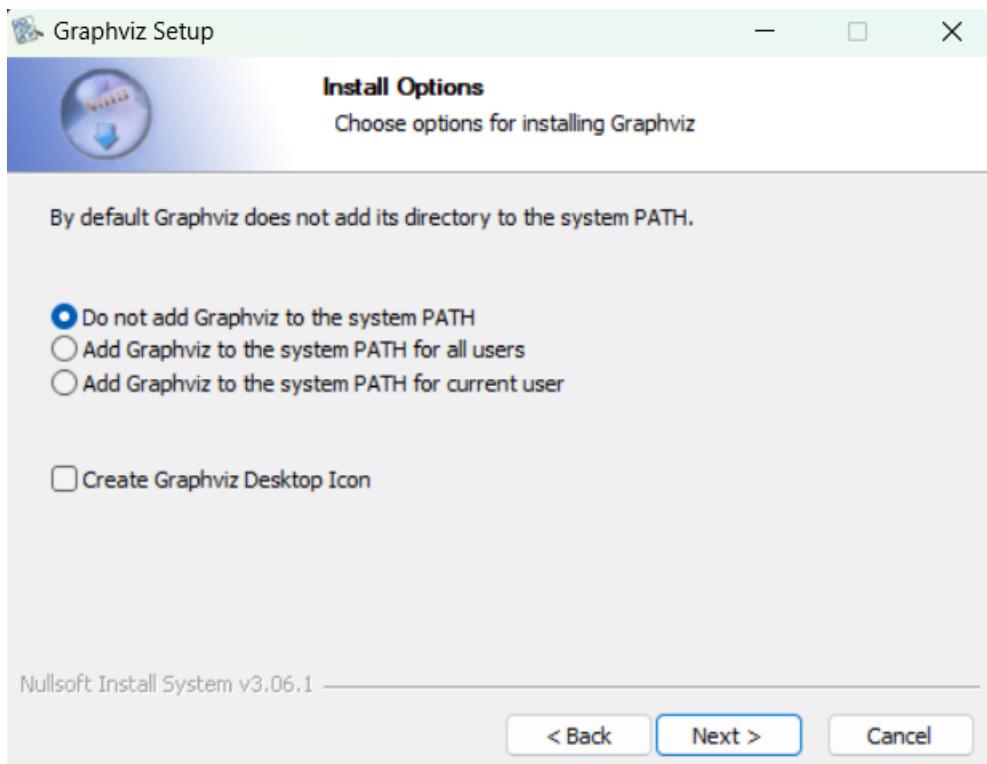
- Chọn phần mềm tương thích rồi download. Sau đó sẽ có file như sau:



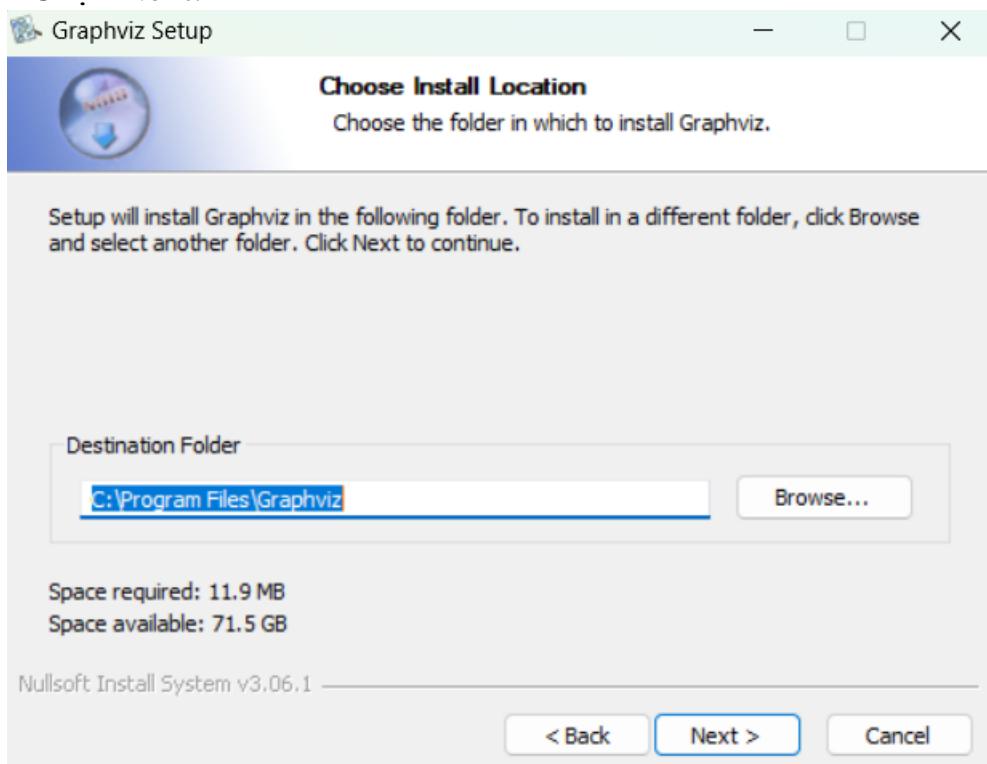
- Nhấn đúp chuột để mở, sẽ xuất hiện bảng sau:



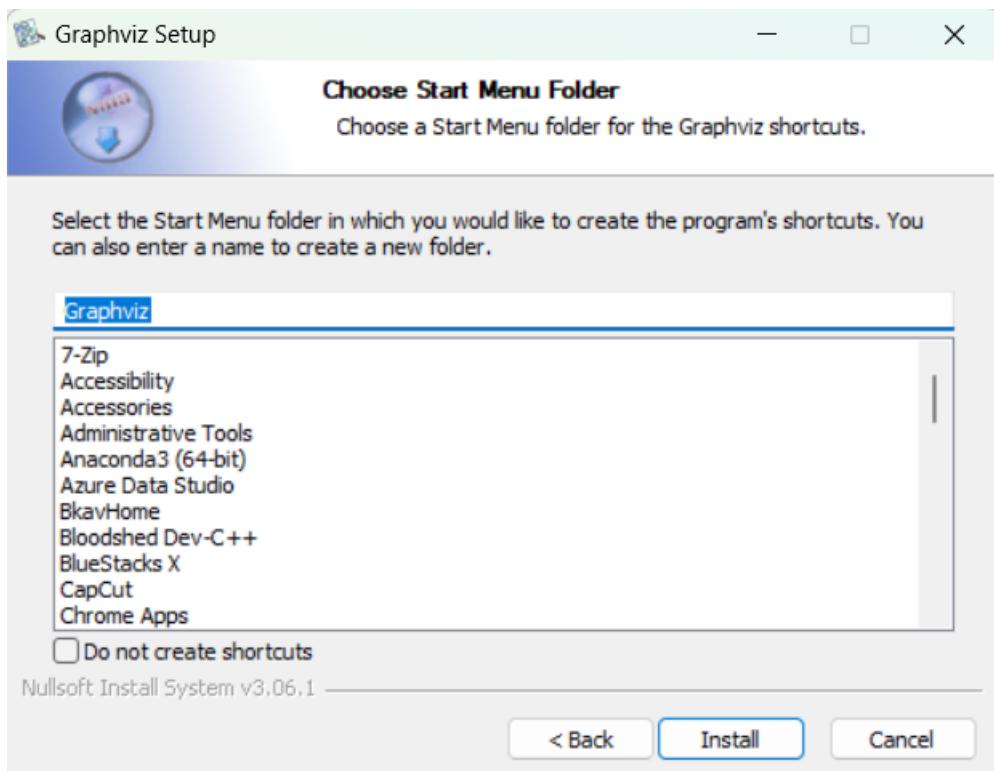
- Chọn agree.



- Chọn Next.



- Chọn đường dẫn tới thư mục muốn lưu, sau đó chọn Next.



- Nhấn Install là hoàn thành.

### 3. Lỗi logic try except.

- Dòng lệnh print nên để trong try để khi bắt lỗi ở graph.write\_png() dòng lệnh print mang thông báo thành công sẽ không xuất hiện mà sẽ xuống để thực thi lệnh print trong except.

(Code ban đầu)

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

```
C:\Users\PC-LENOVO\AppData\Local\Programs\Python\Python310\python.exe
Error while writing file [WinError 2] "dot" not found in path.
File state_space_20.png successfully written.

Process finished with exit code 0
```

Kết quả chạy cho thấy điều đã nói, kết quả chương trình vừa báo lỗi, vừa báo thành công trong 1 công việc, tạo ra lỗi logic.

- Cách khắc phục:

(Code sau khi sửa)

```
42     def write_image(file_name="state_space"):
43         try:
44             graph.write_png(f"{file_name}_{max_depth}.png")
45             print(f"File {file_name}_{max_depth}.png successfully written.")
46         except Exception as e:
47             print("Error while writing file", e)
48         #print(f"File {file_name}_{max_depth}.png successfully written.")
```

Kết quả sẽ cho thấy rõ là error.

```
C:\Users\PC-LENOVO\AppData\Local\Programs\Python\Python310\python.exe
Error while writing file [WinError 2] "dot" not found in path.

Process finished with exit code 0
```

Ngược lại, sẽ thông báo successfully written.

```
C:\Users\PC-LENOVO\AppData\Local\Programs\Python\Python310\python.exe
File state_space_20.png successfully written.

Process finished with exit code 0
```

Lỗi logic tương tự cũng xảy ra ở file solve.py (dòng 64). Và cách khắc phục cũng tương tự như đoạn mã trên ở file generate\_full\_space\_tree.py (dòng 48).

#### 4. Cài đặt các yêu cầu trong file requirement.txt.

```
PS C:\Users\PC-LENOVO\Downloads\test_code_lab2> pip install -r requirements.txt
Attempting uninstall: pyparsing
  Found existing installation: pyparsing 3.0.9
  Uninstalling pyparsing-3.0.9:
    Successfully uninstalled pyparsing-3.0.9
Attempting uninstall: pydot
  Found existing installation: pydot 1.4.2
  Uninstalling pydot-1.4.2:
    Successfully uninstalled pydot-1.4.2
Successfully installed emoji-0.5.4 pydot-1.4.1 pyparsing-2.4.5
PS C:\Users\PC-LENOVO\Downloads\test_code_lab2>
```

## II. TRÌNH BÀY CHI TIẾT.

### 1. Bài toán:

Có 3 người truyền giáo và 3 con quỷ ở bờ bên trái của một con sông, cùng với con thuyền có thể chở được 1 hoặc 2 người. Nếu số quỷ nhiều hơn số người truyền giáo trong một bờ thì số quỷ sẽ ăn thịt số người truyền giáo. Tìm các để đưa tất cả qua bờ sông bên kia (bên phải) sao cho số người không ít hơn số quỷ ở cùng 1 bờ (bên trái hay bên phải), nghĩa là không ai bị ăn thịt. Gọi  $(a, b, k)$  với  $0 \leq a, b \leq 3$ , trong đó  $a$  là số người,  $b$  là số con quỷ ở bên bờ bên trái,  $k = 1$  nếu thuyền ở bờ bên trái và  $k = 0$  nếu thuyền ở bờ bên phải. Khi đó, không gian trạng thái của bài toán được xác định như sau:

- Trạng thái ban đầu là  $(3, 3, 1)$ .
- Thuyền chở qua sông 1 người, hoặc 1 con quỷ, hoặc 1 người và 1 con quỷ, hoặc 2 người, hoặc 2 con quỷ  $\Rightarrow$  các phép toán chuyển từ trạng thái này sang trạng thái khác là:  $(1, 0), (0, 1), (1, 1), (2, 0), (0, 2)$  (trong đó  $(x, y)$  là số người và số quỷ di chuyển từ bờ bên trái qua bờ bên phải hay ngược lại).
- Trạng thái kết thúc là  $(0, 0, 0)$ .

### 1. Giải thích file generate\_full\_space\_tree.py.

Code này thực hiện một số việc cần thiết trước khi tạo và hiển thị cây không gian trạng thái cho bài toán bằng sử dụng thư viện PyDot để vẽ biểu đồ cây.

```
6   from collections import deque
7   import pydot
8   import argparse
9   import os
10
11  # Set it to bin folder of graphviz
12  os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin'
13
14  options = [(1, 0), (0, 1), (1, 1), (0, 2), (2, 0)]
15  Parent = dict()
```

“from collections import deque”: Import module deque từ thư viện collections, deque được sử dụng để tạo một hàng đợi (queue) nhằm theo dõi các trạng thái cần xử lý.

“import pydot”: Import thư viện pydot. Pydot cung cấp các công cụ cho việc tạo các node và cạnh trong biểu đồ, đặt màu, nhãn, kiểu nét và nhiều thuộc tính khác. Thuận tiện tạo biểu đồ và quản lý dữ liệu.

“import argparse”: Import thư viện argparse, một thư viện được sử dụng để xử lý tham số dòng lệnh.

“import os”: Import thư viện os để thao tác với biến môi trường PATH.

“os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin' ”: Thêm đường dẫn đến thư mục bin của Graphviz vào biến môi trường PATH. Graphviz là một phần mềm được sử dụng để tạo và vẽ biểu đồ, và thư viện PyDot sử dụng nó để tạo biểu đồ.

“options = [(1, 0), (0, 1), (1, 1), (0, 2), (2, 0)]”: Một danh sách options chứa các phép biến đổi có thể cho bài toán. Mỗi phần tử trong danh sách là một tuple, trong đó phần tử đầu tiên của tuple là số người và phần tử thứ hai của tuple là số quỷ trên thuyền di chuyển từ bờ bên trái qua bờ bên phải hay ngược lại.

“Parent = dict()”: Khởi tạo một từ điển trống có tên là Parent. Từ điển này được sử dụng để theo dõi trạng thái cha của mỗi trạng thái trong cây không gian trạng thái.

```
16 graph = pydot.Dot(graph_type='graph', strict=False, bgcolor="#ffff3af",
17                      label="fig: Missionaries and Cannibal State Space Tree",
18                      fontcolor="yellow", fontsize="24", overlap="true")
19 # To track node
20 i = 0
```

“graph = pydot.Dot(...)”: Tạo đối tượng biểu đồ graph thông qua thư viện PyDot. Đối tượng biểu đồ này được sử dụng để vẽ và lưu biểu đồ cây không gian trạng thái. Các thuộc tính của biểu đồ được đặt trong mã nguồn này, bao gồm màu nền, nhãn, màu chữ, kích thước chữ, và hỗn hợp.

“pydot.Dot(...)”: Đây là cách tạo một đối tượng biểu đồ mới trong PyDot.

+ “graph\_type='graph'”: Đối số graph\_type xác định loại biểu đồ (Tại có nhiều loại thuộc tính như graph, digraph, subgraph, cluster, ...). Trong trường hợp này, nó được đặt thành 'graph', cho biểu đồ có định dạng tổng quan (Biểu diễn một mạng lưới các nút và các cạnh trong đồ thị).

+ “strict=False”: Đối số strict chỉ định xem biểu đồ có nên tuân thủ các quy tắc strict (nguyên tắc của Graphviz như “Không có cạnh lặp”, “Không có nút lặp”, “Không có cạnh không hợp”,...) hay không. Trong trường hợp này, nó được đặt thành False, vì biểu đồ không cần phải tuân theo các quy tắc nghiêm ngặt của Graphviz.

+ “ bgcolor="#ffff3af" ”: Đối số bgcolor xác định màu nền của biểu đồ. Trong trường hợp này, màu nền được đặt thành #ffff3af, một mã màu hex đại diện cho màu nền màu vàng nhạt.

+ “ label="fig: Missionaries and Cannibal State Space Tree" ”: Đối số label đặt nhãn cho biểu đồ. Trong trường hợp này, biểu đồ được gắn nhãn là "fig: Missionaries and Cannibal State Space Tree" (Dòng chữ màu đỏ nằm cuối 2 tám ảnh kết quả).

+ “fontcolor="red"”: Đối số fontcolor xác định màu chữ của label trên là màu đỏ.

+ “fontsize="24"”: Đối số fontsize xác định kích thước chữ sử dụng trong biểu đồ. Trong trường hợp này, kích thước chữ được đặt thành 24.

+ “overlap="true"”: Đối số overlap xác định xem các nút có thể trùng nhau (overlap) hay không. Trong trường hợp này, nó được đặt thành “true”, cho phép các nút có thể trùng nhau.

“i = 0”: Khởi tạo biến i với giá trị ban đầu là 0. Biến này sẽ được sử dụng để theo dõi số lượng nút (trạng thái) trong cây không gian trạng thái.

```
22     arg = argparse.ArgumentParser()
23     arg.add_argument( *name_or_flags: "-d", "--depth", required=False,
24                         help="MAximum depth upto which you want to generate Space State Tree")
25
26     args = vars(arg.parse_args())
27
28     depth_value = args.get("depth", 20)
29     if depth_value is not None:
30         max_depth = int(depth_value)
31     else:
32         # Xử lý khi không tìm thấy khóa "depth"
33         max_depth = 20 # Giá trị mặc định
```

“arg = argparse.ArgumentParser()”: Tạo một đối tượng arg của lớp ArgumentParser từ thư viện argparse. Đối tượng này được sử dụng để xử lý các tham số dòng lệnh.

“arg.add\_argument(...)”: Định nghĩa một tùy chọn dòng lệnh -d hoặc --depth để cho phép người dùng chỉ định độ sâu tối đa cho cây không gian trạng thái. Tham số này có thể không được cung cấp và sẽ có giá trị mặc định là 20 nếu không được chỉ định.

+ “-d” là tùy chọn ngắn (short option) cho tham số --depth. Tùy chọn ngắn thường là một ký tự, và người dùng có thể sử dụng nó để truyền tham số thông qua dòng lệnh, ví dụ: -d 10.

+ “--depth” là tùy chọn dài (long option) cho tham số --depth. Tùy chọn dài thường là một chuỗi có thể dễ dàng ghi nhớ, và người dùng cũng có thể sử dụng nó để truyền tham số thông qua dòng lệnh, ví dụ: --depth 10.

+ “required=False”: Xác định rằng tham số --depth không bắt buộc. Nếu người dùng không truyền giá trị cho --depth, thì giá trị mặc định là None.

+ “help="MAximum depth upto which you want to generate Space State Tree" ”: là mô tả (description) cho tham số --depth. Mô tả này sẽ được hiển thị khi người dùng sử dụng tùy chọn --help để yêu cầu trợ giúp. Nó giúp người dùng hiểu rõ ý nghĩa của tham số --depth, trong trường hợp này, là giới hạn độ sâu tối đa của cây trạng thái không gian mà bạn muốn tạo.

Nếu ta chạy ứng dụng với lệnh python your\_script.py --depth 5, thì argparse sẽ phân tích dòng lệnh và gán giá trị 5 cho tham số --depth.

“args = vars(arg.parse\_args())”: Phân tích các tham số dòng lệnh được truyền vào và lưu chúng trong một từ điển args.

“depth\_value = args.get("depth", 20)”: Lấy giá trị của tham số depth từ từ điển args. Nếu không tìm thấy, sẽ sử dụng giá trị mặc định là 20.

“if depth\_value is not None:”: Đây là một điều kiện kiểm tra xem biến depth\_value có giá trị (khác None) hay không. Ta kiểm tra xem người dùng đã truyền giá trị cho --depth thông qua dòng lệnh hay chưa.

“max\_depth = int(depth\_value)”: Nếu depth\_value có giá trị (khác None), thì ta chuyển đổi giá trị của depth\_value thành một số nguyên bằng cách sử dụng hàm int(). Điều này giúp chuyển đổi giá trị từ dạng chuỗi (string) sang dạng số nguyên, để sử dụng trong các tính toán hoặc so sánh sau này.

Ngược lại nếu là “else:”, tức depth\_value là None, người dùng không truyền giá trị cho --depth thông qua dòng lệnh, thì chúng ta gán giá trị mặc định cho max\_depth là 20.

```
36     def is_valid_move(number_missionaries, number_cannibals):
37         """
38             Checks if number constraints are satisfied
39         """
40         return (0 <= number_missionaries <= 3) and (0 <= number_cannibals <= 3)
```

Định nghĩa hàm is\_valid\_move với tham số truyền vào là number\_missionaries (số người truyền giáo) và number\_cannibals (số quỷ). Kiểm tra xem một nước đi cụ thể trong bài có hợp lý hay không. Nếu tham số truyền vào có giá trị nằm ngoài đoạn từ 0 đến 3 thì hàm return về False, ngược lại thì là True.

```
42     def write_image(file_name="state_space"):
43         try:
44             graph.write_png(f"{file_name}_{max_depth}.png")
45             print(f"File {file_name}_{max_depth}.png successfully written.")
46         except Exception as e:
47             print("Error while writing file", e)
48         #print(f"File {file_name}_{max_depth}.png successfully written.")
```

Định nghĩa hàm write\_image với tham số truyền vào là file\_name có giá trị mặc định là “state\_space”, hàm mục đích là để tạo bản vẽ của đồ thị và lưu tệp dưới định dạng “.png”. Thủ ghi hình ảnh bằng cách gọi phương thức write\_png trên đối tượng graph. Đường dẫn tệp hình ảnh sẽ được tạo bằng cộng chuỗi file\_name và max\_depth, sau đó nối với phần mở rộng “.png”.

Nếu quá trình ghi hình ảnh không gặp lỗi, hàm in ra dòng "File {tên\_tệp\_hình\_ảnh} successfully written." để thông báo rằng việc ghi tệp hình ảnh đã thành công.

Nếu có lỗi xảy ra trong quá trình ghi hình ảnh, hàm sẽ bắt lỗi và in ra dòng thông báo "Error while writing file" cùng với thông báo lỗi cụ thể (e là biến chứa thông tin về lỗi).

```

50     def draw_edge(number_missionaries, number_cannibals, side, depth_level, node_num):
51         u, v = None, None
52         if Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)] is not None:
53             u = pydot.Node(str(Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)]),
54                             label=str(Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)][:-3]))
55             graph.add_node(u)
56
57             v = pydot.Node(str((number_missionaries, number_cannibals, side, depth_level, node_num)),
58                             label=str((number_missionaries, number_cannibals, side)))
59             graph.add_node(v)
60
61             edge = pydot.Edge(str(Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)]),
62                               str((number_missionaries, number_cannibals, side, depth_level, node_num)), dir='forward')
63             graph.add_edge(edge)
64         else:
65             # For start node
66             v = pydot.Node(str((number_missionaries, number_cannibals, side, depth_level, node_num)),
67                             label=str((number_missionaries, number_cannibals, side)))
68             graph.add_node(v)
69
    return u, v

```

Định nghĩa hàm `draw_edge` nhận các tham số `number_missionaries`, `number_cannibals`, `side` (bờ sông mà trạng thái đại diện, trạng thái bao gồm số lượng người truyền giáo và quỷ đang ở, thường là "left" ( $k=1$ ) hoặc "right" ( $k=0$ )), `depth_level` (mức độ sâu của trạng thái trong cây tìm kiếm) và `node_num` (số thứ tự của trạng thái trong cùng một mức độ sâu).

“`if Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)] is not None`”: kiểm tra xem có một phần tử tương ứng với khóa `(number_missionaries, number_cannibals, side, depth_level, node_num)` trong từ điển `Parent` hay không, và liệu giá trị của phần tử đó có khác `None` hay không.

Nếu có phần tử và khác `None` thì khi đó `pydot.Node()` này sẽ tạo một đối tượng, đại diện cho 1 nút trong biểu diễn đồ thị. Gán nó cho `u` với:

“`str(Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)])`”: Đây là cách truy xuất giá trị từ từ điển `Parent`. Khóa của từ điển là một bộ năm giá trị. Điều này giúp xác định trạng thái cha (nếu có) của trạng thái hiện tại.

“`label=str(Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)][:-3])`”: Đây là cách lấy ba phần đầu tiên của giá trị trong từ điển `Parent` và gán chúng làm nhãn cho nút. Nhãn này sẽ hiển thị trên biểu đồ đồ thị để đại diện cho nút cha. Lấy ba ký tự đầu có thể là một cách để rút gọn và hiển thị trạng thái một cách ngắn gọn. Ví dụ, nếu `Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)]` trả về một giá trị là `(2, 1, 0, 3, 4)` và `[:-3]` lấy giá trị đầu, thì

label của nút u sẽ được gán bằng "2, 1, 0". Điều này làm cho nút cha trở nên dễ nhận biết trên biểu đồ đồ thị. Sau đó add node này vào đồ thị bằng câu lệnh “graph.add\_node(u)”.

Tiếp tục tạo đối tượng thông qua pydot.Node() và gán cho v. Node này mang thông tin về trạng thái hiện tại. Thêm nút này vào đồ thị bằng câu lệnh “graph.add\_node(v)”.

“edge = pydot.Edge(...)" : tạo cạnh trong biểu diễn đồ thị bằng pydot.Edge() cho 2 trạng thái được đưa vào là trạng thái nút cha và trạng thái nút hiện tại. “dir='forward'" : Tham số này định nghĩa hướng của cạnh, trong trường hợp này, nó là 'forward', nghĩa là cạnh sẽ trỏ từ trạng thái cha đến trạng thái hiện tại trên biểu đồ. Sau đó thêm cạnh vào biểu diễn đồ thị bằng “graph.add\_edge(edge)”.

Ngược lại, nếu trong từ điển Parent của khóa 5 tham số mà trả về None thì tức là đây là node start. Lúc này ta tạo node v lưu trạng thái hiện tại rồi add node vào biểu đồ. Sau cùng trả về node u và v, phục vụ cho thao tác sau này.

```
71     def is_start_state(number_missionaries, number_cannibals, side):  
72         return (number_missionaries, number_cannibals, side) == (3, 3, 1)
```

Hàm is\_start\_state có nhiệm vụ kiểm tra xem một trạng thái cụ thể có phải là trạng thái ban đầu (trạng thái xuất phát) của bài toán hay không. Sau đó trả về True hoặc False.

```
74     def is_goal_state(number_missionaries, number_cannibals, side):  
75         return (number_missionaries, number_cannibals, side) == (0, 0, 0)
```

Hàm is\_goal\_state có nhiệm vụ kiểm tra xem một trạng thái cụ thể có phải là trạng thái kết thúc (trạng thái đích) của bài toán hay không. Sau đó trả về True hoặc False.

```
77     def number_of_cannibals_exceeds(number_missionaries, number_cannibals):  
78         number_missionaries_right = 3 - number_missionaries  
79         number_cannibals_right = 3 - number_cannibals  
80         return (number_missionaries > 0 and number_cannibals > number_missionaries) \\\n81             or (number_missionaries_right > 0 and number_cannibals_right > number_missionaries_right)
```

Hàm number\_of\_cannibals\_exceeds có nhiệm vụ kiểm tra xem số lượng quỷ (cannibals) có vượt quá số lượng người truyền giáo (missionaries) trong một trạng thái cụ thể của bài toán hay không. Hàm nhận hai tham số là number\_missionaries và number\_cannibals. Hàm thực hiện các bước sau để kiểm tra xem số lượng quỷ có vượt quá số lượng

người truyền giáo hay không bằng cách tính số lượng người truyền giáo và quỷ ở bờ sông đối diện (bờ sông bên phải):

number\_missionaries\_right là số lượng người truyền giáo ở bờ sông đối diện.

number\_cannibals\_right là số lượng quỷ ở bờ sông đối diện.

Hàm kiểm tra hai điều kiện:

(number\_missionaries > 0 and number\_cannibals > number\_missionaries): Điều này kiểm tra xem số lượng người truyền giáo và quỷ ở bờ sông hiện tại có vượt quá số lượng người truyền giáo hay không.

(number\_missionaries\_right > 0 and number\_cannibals\_right > number\_missionaries\_right): Điều này kiểm tra xem số lượng người truyền giáo và quỷ ở bờ sông đối diện có vượt quá số lượng người truyền giáo hay không.

Nếu một trong các điều kiện này đúng, hàm sẽ trả về True. Ngược lại, hàm sẽ trả về False, ngũ ý rằng số lượng quỷ không vượt quá số lượng người truyền giáo ở cả hai bờ sông.

```
83     def generate():
84         global i
85         q = deque()
86         node_num = 0
87         q.append((3, 3, 1, 0, node_num))
88
89         Parent[(3, 3, 1, 0, node_num)] = None
90
91         while q:
92
93             number_missionaries, number_cannibals, side, depth_level, node_num = q.popleft()
94             # print(number_missionaries, number_cannibals)
95             # Draw Edge from u -> v
96             # Where u = Parent[v]
97             # and v = (number_missionaries, number_cannibals, side, depth_level)
98             u, v = draw_edge(number_missionaries, number_cannibals, side, depth_level, node_num)
```

global i: Đây là một khai báo cho biến toàn cục i, nhưng mã này không sử dụng i trong phần đoạn code mà bạn đã cung cấp, nên nó có thể không cần thiết.

“q = deque()”: Đây là một hàng đợi (deque) được sử dụng để duyệt qua các trạng thái trong quá trình giải quyết bài toán. Hàng đợi này được sử dụng để theo dõi các trạng thái cần xét trong thứ tự đã đăng ký.

“node\_num” = 0: Biến node\_num được sử dụng để đánh dấu các trạng thái với một số duy nhất. Ban đầu, nó được thiết lập thành 0.

“q.append((3, 3, 1, 0, node\_num))”: Dòng mã này đưa trạng thái ban đầu vào hàng đợi. Trạng thái ban đầu có số người truyền giáo và quỷ đều là 3 (3 missionaries, 3 cannibals), và bờ sông bên trái có thuyền (side = 1), depth\_level đánh dấu mức độ sâu trong cây tìm kiếm, nhưng ở đây, nó được thiết lập thành 0. node\_num là số định danh cho trạng thái, và ban đầu được đặt là 0.

“Parent[(3, 3, 1, 0, node\_num)] = None”: Đây là dòng mã để xác định rằng trạng thái ban đầu không có trạng thái cha (parent), nên nó được thiết lập thành None.

Sau đó, bên trong vòng lặp while q: (khi hàng đợi không rỗng), trạng thái hiện tại (number\_missionaries, number\_cannibals, side, depth\_level, node\_num) được rút ra từ hàng đợi, và hàm draw\_edge được gọi để tạo và vẽ cạnh giữa trạng thái cha (nếu có) và trạng thái hiện tại.

```
100      if is_start_state(number_missionaries, number_cannibals, side):
101          v.set_style("filled")
102          v.set_fillcolor("blue")
103          v.set_fontcolor("white")
104      elif is_goal_state(number_missionaries, number_cannibals, side):
105          v.set_style("filled")
106          v.set_fillcolor("green")
107          continue
108      # return True
109      elif number_of_cannibals_exceeds(number_missionaries, number_cannibals):
110          v.set_style("filled")
111          v.set_fillcolor("red")
112          continue
113      else:
114          v.set_style("filled")
115          v.set_fillcolor("orange")
```

“if is\_start\_state(number\_missionaries, number\_cannibals, side):”: Dòng mã này kiểm tra xem trạng thái hiện tại có phải là trạng thái ban đầu (trạng thái xuất phát) của bài toán hay không bằng cách gọi hàm is\_start\_state với các tham số tương ứng. Nếu trạng thái là trạng thái xuất phát, nó sẽ được tô màu xanh làm nền (filled with blue), với văn bản màu trắng.

“elif is\_goal\_state(number\_missionaries, number\_cannibals, side):”: Dòng mã này kiểm tra xem trạng thái hiện tại có phải là trạng thái

mục tiêu (trạng thái kết thúc) của bài toán hay không bằng cách gọi hàm `is_goal_state` với các tham số tương ứng. Nếu trạng thái là trạng thái mục tiêu, nó sẽ được tô màu xanh lá cây làm nền (filled with green) và có thể kết thúc quá trình xử lý, nhưng trong mã này đã có lệnh `continue`, nên nó sẽ bỏ qua các phần mã còn lại và tiếp tục vòng lặp.

“`elif number_of_cannibals_exceeds(number_missionaries, number_cannibals):`”: Dòng mã này kiểm tra xem số lượng quỷ có vượt quá số lượng người truyền giáo trong trạng thái hiện tại hay không, bằng cách gọi hàm `number_of_cannibals_exceeds` với các tham số tương ứng. Nếu điều kiện này đúng, trạng thái sẽ được tô màu đỏ làm nền (filled with red) và sau đó sẽ bị bỏ qua (bằng `continue`) trong quá trình xử lý.

“`else:`”: Nếu trạng thái không thuộc vào các trường hợp trên, nó sẽ được tô màu cam làm nền (filled with orange).

```
117         if depth_level == max_depth:  
118             return True  
119  
120             op = -1 if side == 1 else 1  
121  
122             can_be_expanded = False
```

“`if depth_level == max_depth:`”: Dòng mã này kiểm tra xem mức độ sâu của trạng thái hiện tại có bằng với mức độ sâu tối đa (được xác định bởi biến `max_depth`) hay không. Nếu có, hàm sẽ trả về `True`, nghĩa là已经达到 mức độ tối đa và có thể kết thúc quá trình xử lý.

“`op = -1 if side == 1 else 1`”: Dòng mã này thiết lập giá trị cho biến `op` bằng cách sử dụng một biểu thức điều kiện (ternary condition). Biểu thức này có dạng `x if condition else y`, nghĩa là nếu điều kiện `condition` đúng, thì biến `op` sẽ được gán giá trị `x`, còn nếu điều kiện sai, thì biến `op` sẽ được gán giá trị `y`. Trong trường hợp này, nếu `side` (bờ sông) bằng 1, thì `op` sẽ được gán giá trị -1, còn nếu `side` không bằng 1 (tức là `side` khác 1), thì `op` sẽ được gán giá trị 1. Tùy thuộc vào giá trị của `side`, `op` sẽ có giá trị là -1 hoặc 1.

“`can_be_expanded = False`”: Dòng mã này thiết lập biến `can_be_expanded` thành `False`. Biến này có thể được sử dụng sau đó trong vòng `for` để kiểm tra xem một trạng thái có thể mở rộng (mở rộng có nghĩa là có thể điều chỉnh trạng thái đó để tạo ra các trạng thái con mới) hay không. Trong trường hợp này, nó được đặt thành `False` ban đầu, nghĩa là không có trạng thái nào có thể mở rộng.

trạng thái hiện tại không thể được mở rộng mặc dù giá trị của op đã được tính toán.

```
124     # i = node_num
125     for x, y in options:
126         next_m, next_c, next_s = number_missionaries + op * x, number_cannibals + op * y, int(not side)
127
128         if Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)] is None or (next_m, next_c, next_s) \ 
129         != Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)][-3]:
130             if is_valid_move(next_m, next_c):
131                 can_be_expanded = True
132                 i += 1
133                 q.append((next_m, next_c, next_s, depth_level + 1, i))
134                 # Keep track of parent
135                 Parent[(next_m, next_c, next_s, depth_level + 1, i)] = \
136                 (number_missionaries, number_cannibals, side, depth_level, node_num)
137
138             if not can_be_expanded:
139                 v.set_style("filled")
140                 v.set_fillcolor("gray")
141
142     return False
```

Vòng lặp for nhằm mở rộng các trạng thái và kiểm tra xem chúng có thể được mở rộng hay không.

Trong vòng lặp for x, y in options:, mã duyệt qua tất cả các cách có thể di chuyển (được biểu diễn bởi các cặp (x, y) trong options), trong đó x và y là số lượng người truyền giáo và quỷ được di chuyển. Mỗi cách di chuyển được thử nghiệm để xem có thể mở rộng trạng thái hiện tại thành trạng thái mới hay không.

“next\_m, next\_c, next\_s = number\_missionaries + op \* x, number\_cannibals + op \* y, int(not side)”:

Dòng mã này tính toán trạng thái mới bằng cách thêm (hoặc trừ) số người truyền giáo và quỷ tương ứng vào trạng thái hiện tại. Biến next\_s là bờ phía (side) mới, được tính toán bằng cách đảo ngược giá trị side hiện tại.

Đoạn mã “if Parent[...]:” kiểm tra xem trạng thái cha của trạng thái hiện tại có tồn tại và khác với trạng thái hiện tại không. Nếu điều kiện này đúng, có nghĩa rằng trạng thái hiện tại là một trạng thái mới hoặc là một trạng thái không trùng với trạng thái cha. Điều này cần thiết để tránh lặp lại các trạng thái đã xem xét trước đó.

“if is\_valid\_move(next\_m, next\_c)”:

Dòng mã này kiểm tra xem trạng thái mới (next\_m, next\_c, next\_s) có hợp lệ hay không bằng cách gọi hàm is\_valid\_move. Nếu trạng thái là hợp lệ (không vi phạm ràng buộc), thì biến can\_be\_expanded được thiết lập thành True, ngụ ý rằng trạng thái hiện tại có thể được mở rộng thành các trạng thái con mới.

“`i += 1`”: Biến `i` (số định danh của trạng thái) được tăng lên một đơn vị để đảm bảo mỗi trạng thái mới có một số duy nhất.

“`q.append((next_m, next_c, next_s, depth_level + 1, i))`”: Trạng thái mới với các thông tin như số người truyền giáo, quý, phia bờ thuyền, mức độ sâu tăng lên 1 và số định danh mới `i` được thêm vào hàng đợi `q` để xem xét trong các vòng lặp tiếp theo.

“`Parent[(next_m, next_c, next_s, depth_level + 1, i)] = (number_missionaries, number_cannibals, side, depth_level, node_num)`”: Dòng mã này thiết lập trạng thái cha của trạng thái mới trong từ điển `Parent`. Điều này giúp theo dõi quá trình di chuyển từ trạng thái cha đến trạng thái con.

“`if not can_be_expanded:`”: Nếu biến `can_be_expanded` vẫn là `False` sau khi kiểm tra tất cả các lựa chọn di chuyển, ngụ ý rằng trạng thái hiện tại không thể được mở rộng thành bất kỳ trạng thái con nào. Trong trường hợp này, trạng thái hiện tại sẽ được tô màu xám (filled with gray), và hàm sẽ trả về `False`, ngụ ý rằng không còn trạng thái con nào để xem xét.

```
142 ► if __name__ == "__main__":
143     if generate():
144         write_image()
```

Dòng mã `if __name__ == "__main__":`: kiểm tra xem tệp Python đang chạy là tệp chính hay không. Nếu tệp này đang được thực thi trực tiếp (chứ không phải là một module được nhập), thì mã trong khối này sẽ được thực thi. “`if generate():`”: Dòng mã này gọi hàm `generate()` để bắt đầu quá trình tạo cây tìm kiếm hoặc thực hiện các công việc cần thiết. Nếu hàm `generate()` trả về `True`, có nghĩa rằng quá trình đã hoàn thành một cách thành công, và sau đó sẽ tiếp tục thực hiện dòng mã tiếp theo.

`write_image()`: Dòng mã này gọi hàm `write_image()` để tạo và lưu biểu đồ đồ thị dưới dạng hình ảnh. Hàm này có thể được sử dụng để trực quan hóa cây tìm kiếm hoặc dữ liệu liên quan đến bài toán. Sau khi thực hiện hàm này, một hình ảnh biểu đồ đồ thị sẽ được tạo ra và lưu trữ.

## 2. Giải thích file main.py.

```
5   from solve import Solution
6   import argparse
7   import itertools
8
9   arg = argparse.ArgumentParser()
10  arg.add_argument( *name_or_flags: "-m", "--method", required=False, help="Specify which method to use")
11  arg.add_argument( *name_or_flags: "-l", "--legend", required=False, help="Specify if you want to display legend on graph")
12
13  args = vars(arg.parse_args())
```

“from solve import Solution”: Đoạn mã này nhập một lớp có tên “Solution” từ một tệp gọi là “solve.” Điều này cho phép bạn sử dụng lớp Solution trong chương trình của mình. Lớp Solution có thể chứa logic để giải quyết một vấn đề cụ thể.

“import argparse”: Đây là một lệnh nhập thư viện argparse. Thư viện này giúp bạn dễ dàng xử lý và phân tích các đối số được truyền vào chương trình từ dòng lệnh.

“import itertools”: Lệnh này nhập thư viện itertools, một thư viện Python cung cấp các công cụ cho việc tạo và xử lý các vòng lặp, chuỗi và dãy số.

“arg = argparse.ArgumentParser()”: Tạo một đối tượng tham số dòng lệnh bằng cách gọi lớp argparse.ArgumentParser(). Đối tượng này được sử dụng để định nghĩa cách ta muốn xử lý các đối số dòng lệnh.

“arg.add\_argument("-m", "--method", required=False, help="Specify which method to use")”: Thêm một đối số dòng lệnh cho tùy chọn “-m” hoặc “--method” vào đối tượng tham số dòng lệnh. Đối số này không bắt buộc (do required=False), và thông tin hướng dẫn sẽ được hiển thị khi bạn chạy chương trình.

“arg.add\_argument("-l", "--legend", required=False, help="Specify if you want to display legend on graph")”: Tương tự như trên, đây là đối số dòng lệnh cho tùy chọn “-l” hoặc “--legend” với thông tin hướng dẫn.

```
13  args = vars(arg.parse_args())
14
15  solve_method = args.get("method", "bfs")
16  legend_flag = args.get("legend", False)
```

“args = vars(arg.parse\_args())”: Dòng này thực hiện việc phân tích các đối số dòng lệnh được truyền vào chương trình bằng cách gọi phương thức parse\_args() từ đối tượng arg. Kết quả của phương thức này là một đối

tương tạo từ các đối số dòng lệnh, và vars() được sử dụng để chuyển đổi nó thành một từ điển.

“solve\_method = args.get("method", "bfs")”: Dòng này trích xuất giá trị của đối số “--method” hoặc “-m” từ từ điển args bằng cách sử dụng phương thức get(). Nếu đối số không được xác định (tức là không được truyền vào dòng lệnh), giá trị mặc định là “bfs” sẽ được sử dụng.

“legend\_flag = args.get("legend", False)”: Tương tự như trên, dòng này trích xuất giá trị của đối số “--legend” hoặc “-l” từ từ điển args. Nếu đối số này không được xác định, giá trị mặc định là False sẽ được sử dụng.

Kết quả của các dòng này là việc lấy được giá trị của các đối số dòng lệnh (nếu có) và gán chúng vào các biến solve\_method và legend\_flag. Nếu các đối số không được truyền vào, giá trị mặc định sẽ được sử dụng để thiết lập các biến này.

```
19     def main():
20         s = Solution()
21
22         if(s.solve(solve_method)):
23
24             # Display Solution on console
25             s.show_solution()
26
27             output_file_name = f"{solve_method}"
28             # Draw legend if legend_flag is set
29             if legend_flag:
30                 if legend_flag[0].upper() == 'T':
31                     output_file_name += "_legend.png"
32                     s.draw_legend()
33                 else:
34                     output_file_name += ".png"
35             else:
36                 output_file_name += ".png"
37
38             # Write State space tree
39             s.write_image(output_file_name)
40         else:
41             raise Exception("No solution found")
```

“s = Solution()”: Tạo một đối tượng Solution bằng cách gọi lớp Solution, đã được nhập từ tệp “solve.”

“if s.solve(solve\_method)”: Kiểm tra xem việc giải quyết vấn đề bằng phương pháp đã được xác định trong solve\_method (có thể là “bfs” hoặc giá trị mặc định “bfs”) có thành công hay không. Điều này được thực hiện bằng cách gọi phương thức solve() từ đối tượng Solution. Nếu giải quyết thành công, chương trình sẽ thực hiện các hành động bên trong khối if. Nếu không, nó sẽ ném một ngoại lệ với thông báo “No solution found.”

“s.show\_solution()”: Gọi phương thức show\_solution() từ đối tượng Solution để hiển thị kết quả giải quyết vấn đề trên màn hình.

“output\_file\_name = f”{solve\_method}””: Tạo một biến output\_file\_name để lưu trữ tên tệp đầu ra dựa trên phương pháp giải quyết (solve\_method). Ban đầu, biến này chỉ chứa giá trị của solve\_method.

“if legend\_flag”: Kiểm tra xem biến legend\_flag có được đặt hoặc không. Nếu legend\_flag đã được đặt, nghĩa là đối số “--legend” hoặc “-l” đã được truyền vào dòng lệnh, chương trình sẽ thực hiện các hành động trong khối if.

“if legend\_flag[0].upper() == ‘T’”: Kiểm tra xem ký tự đầu tiên của giá trị trong legend\_flag đã được chuyển thành chữ hoa có phải là ‘T’ hay không. Nếu là ‘T’, tức là đối số “--legend” hoặc “-l” được đặt thành True, chương trình sẽ thêm “\_legend.png” vào output\_file\_name và gọi phương thức draw\_legend() từ đối tượng Solution để vẽ chú thích.

“else”: Nếu ký tự đầu tiên trong legend\_flag không phải là ‘T’, nghĩa là đối số “--legend” hoặc “-l” không được đặt thành True, chương trình chỉ thêm “.png” vào output\_file\_name.

“s.write\_image(output\_file\_name)”: Gọi phương thức write\_image() từ đối tượng Solution để ghi tệp hình ảnh đại diện cho cây trạng thái. Tên tệp này đã được xác định ở các dòng trên, tùy thuộc vào giá trị của legend\_flag.

“else”: Nếu việc giải quyết vấn đề không thành công (không tìm thấy giải pháp), chương trình sẽ ném một ngoại lệ (Exception) với thông báo “No solution found.”

```
44 ► | if __name__ == "__main__":
45     main()
```

Dòng `if __name__ == "__main__":` là một điều kiện kiểm tra để xác định xem chương trình đang chạy như một tập lệnh (script) riêng lẻ hay không. Nếu tập lệnh này được chạy trực tiếp (thay vì được nhập như một mô-đun vào một tập lệnh khác), thì khối mã bên trong khối if sẽ được thực thi.\

### 3. Giải thích file solve.py.

```
6  import os
7  import emoji
8  import pydot
9  import random
10 from collections import deque
11
12 # Set it to bin folder of graphviz
13 os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin'
14
15 # Dictionaries to backtrack solution nodes
16 # Parent stores parent of (m , c , s)
17 # Move stores (x, y, side) i.e number of missionaries,
18 #cannibals to be moved from left to right or right to left for particular state
19 # node_list stores pydot.Node object for particular state (m, c, s) so that we can color the solution nodes
20 Parent, Move, node_list = dict(), dict(), dict()
```

“import os”: Dòng này import module os, cho phép làm việc với các chức năng liên quan đến hệ điều hành, chẳng hạn như làm việc với thư mục và tệp tin.

“import emoji”: Dòng này import module emoji, giúp bạn làm việc với emojis trong Python, chẳng hạn như thêm emojis vào chuỗi hoặc thao tác với dữ liệu liên quan đến emoji.

“import pydot”: Dòng này import module pydot, một giao diện Python đến phần mềm tạo đồ thị Graphviz. Sử dụng nó để tạo và thao tác với đồ thị và biểu đồ một cách tự động.

“import random”: Dòng này import module random, mà cung cấp các hàm để tạo ra các số ngẫu nhiên và thực hiện các phép toán ngẫu nhiên. Nó thường được sử dụng cho mục đích mô phỏng, trò chơi và các ứng dụng khác đòi hỏi tính ngẫu nhiên.

“from collections import deque”: Dòng này import lớp deque từ module collections. Deque là một cấu trúc dữ liệu hàng đợi đa năng với khả năng thêm và xóa phần tử ở cả hai đầu một cách hiệu quả. Nó thường được sử dụng cho việc quản lý hàng đợi và ngăn xếp trong Python.

“os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin' ”: Thêm đường dẫn đến thư mục bin của Graphviz vào biến môi trường PATH. Graphviz là một phần mềm được sử dụng để tạo và vẽ biểu đồ, và thư viện PyDot sử dụng nó để tạo biểu đồ.

```

15     # Dictionaries to backtrack solution nodes
16     # Parent stores parent of (m , c, s)
17     # Move stores (x, y, side) i.e number of missionaries,
18     #cannibals to be moved from left to right or right to left for particular state
19     # node_list stores pydot.Node object for particular state (m, c, s) so that we can color the solution nodes
20     Parent, Move, node_list = dict(), dict(), dict()

```

**Parent:** Đây là một từ điển được sử dụng để theo dõi các trạng thái cha mẹ của trạng thái hiện tại. Ví dụ: Parent[(m, c, s)] sẽ cho biết trạng thái cha mẹ của (m, c, s).

**Move:** Đây cũng là một từ điển được sử dụng để lưu trữ thông tin về số lượng người truyền giáo và quỷ (missionaries và cannibals) cần di chuyển từ bờ trái sang bờ phải hoặc ngược lại để đạt được trạng thái hiện tại. Nó có thể lưu trữ số lượng người cần di chuyển cùng với hướng di chuyển (bên trái sang bên phải hoặc ngược lại). Ví dụ: Move[(m, c, s)] có thể lưu trữ (x, y, side) để biểu thị số lượng người truyền giáo x và quỷ y cần di chuyển từ side (bên trái hoặc bên phải) để đạt được trạng thái (m, c, s).

**node\_list:** Đây là một từ điển khác được sử dụng để lưu trữ các đối tượng pydot.Node. Mục đích chính của từ điển này là để thao tác với biểu đồ và đánh dấu các trạng thái cụ thể là các nút trên biểu đồ.

```

22 class Solution():
23
24     def __init__(self):
25         # Start state (3M, 3C, Left)
26         # Goal State (0M, 0C, Right)
27         # Each state gives the number of missionaries and cannibals on the left side
28
29         self.start_state = (3, 3, 1)
30         self.goal_state = (0, 0, 0)
31         self.options = [(1, 0), (0, 1), (1, 1), (0, 2), (2, 0)]
32
33         self.boat_side = ["right", "left"]
34
35         self.graph = pydot.Dot(graph_type='graph',
36                                bgcolor="#ffff3af",
37                                label="fig: Missionaries and Cannibal State Space Tree", fontcolor="red", fontsize="24")
38         self.visited = {}
39         self.solved = False

```

Ta định nghĩa một lớp Solution trong Python.

“def \_\_init\_\_(self):”: Đây là hàm khởi tạo (constructor) của lớp Solution. Nó được gọi khi một đối tượng của lớp được tạo. Trong hàm này, các thuộc tính và biến cố định ban đầu được khởi tạo.

**self.start\_state** và **self.goal\_state**: Đây là các biến instance lưu trữ trạng thái ban đầu và trạng thái mục tiêu của bài. Mỗi trạng thái được biểu diễn bằng một bộ 3 giá trị: (số lượng người truyền giáo, số lượng quỷ, vị trí

của thuyền). Ví dụ, (3, 3, 1) đại diện cho trạng thái ban đầu với 3 truyền giáo, 3 quỷ ở bờ trái (vị trí thuyền là "trái"), và (0, 0, 0) đại diện cho trạng thái mục tiêu với 0 truyền giáo, 0 quỷ ở bờ phải (vị trí thuyền là "phải").

“self.options = [...]”: Đây là một danh sách các tùy chọn di chuyển có sẵn cho các thuyền giáo và quỷ khi họ băng qua sông. Mỗi tùy chọn được biểu diễn bằng một bộ 2 giá trị, ví dụ (1, 0) đại diện cho di chuyển 1 thuyền giáo và 0 quỷ từ bờ này sang bờ kia.

self.boat\_side: Đây là danh sách các bên (sides) có thể của thuyền, trong trường hợp này "right" (bên phải) và "left" (bên trái). Thuyền sẽ nằm ở một trong hai bên này tại mỗi trạng thái.

self.graph: Đây là một đối tượng biểu đồ PyDot, được sử dụng để tạo biểu đồ trạng thái và biểu đồ không gian trạng thái của bài toán. Các trạng thái và các liên kết giữa chúng sẽ được thêm vào biểu đồ này.

self.visited: Đây là một từ điển dùng để theo dõi các trạng thái đã được kiểm tra (visited). Mục đích của nó là để tránh việc kiểm tra lại các trạng thái đã xem xét và đảm bảo rằng thuật toán không bị lặp lại trong việc tìm kiếm đường đi.

self.solved: Biến này đánh dấu xem bài toán đã được giải quyết thành công (đã tìm thấy đường đi từ trạng thái ban đầu đến trạng thái mục tiêu) hay chưa. Ban đầu, nó được đặt thành False, và nếu giải quyết thành công, nó sẽ được đặt thành True.

```
40     def is_valid_move(self, number_missionaries, number_cannibals):
41         """
42             Checks if number constraints are satisfied
43         """
44         return (0 <= number_missionaries <= 3) and (0 <= number_cannibals <= 3)
45
46     def is_goal_state(self, number_missionaries, number_cannibals, side):
47         return (number_missionaries, number_cannibals, side) == self.goal_state
48
49     def is_start_state(self, number_missionaries, number_cannibals, side):
50         return (number_missionaries, number_cannibals, side) == self.start_state
51
52     def number_of_cannibals_exceeds(self, number_missionaries, number_cannibals):
53         number_missionaries_right = 3 - number_missionaries
54         number_cannibals_right = 3 - number_cannibals
55         return (number_missionaries > 0 and number_cannibals > number_missionaries) \
56                 or (number_missionaries_right > 0 and number_cannibals_right > number_missionaries_right)
```

Định nghĩa phương thức is\_valid\_move với tham số thuyền vào là number\_missionaries (số người truyền giáo) và number\_cannibals (số quỷ). Kiểm tra xem một nước đi cụ thể trong bài có hợp lý hay không. Nếu

tham số truyền vào có giá trị nằm ngoài đoạn từ 0 đến 3 thì hàm return về False, ngược lại thì là True.

“is\_goal\_state(self, number\_missionaries, number\_cannibals, side)”: Phương thức này nhận ba tham số: number\_missionaries, number\_cannibals và side. Nó kiểm tra xem trạng thái được biểu diễn bởi các tham số này có trùng khớp với trạng thái mục tiêu (self.goal\_state) trong lớp Solution. Nếu trạng thái này trùng khớp với trạng thái mục tiêu, phương thức trả về True, ngược lại trả về False.

“is\_start\_state(self, number\_missionaries, number\_cannibals, side)”: Phương thức này cũng nhận ba tham số tương tự và kiểm tra xem trạng thái này có trùng khớp với trạng thái xuất phát (self.start\_state) trong lớp Solution. Nếu trạng thái này trùng khớp với trạng thái xuất phát, phương thức trả về True, ngược lại trả về False.

Hàm number\_of\_cannibals\_exceeds có nhiệm vụ kiểm tra xem số lượng quỷ (cannibals) có vượt quá số lượng người truyền giáo (missionaries) trong một trạng thái cụ thể của bài toán hay không. Hàm nhận hai tham số là number\_missionaries và number\_cannibals. Hàm thực hiện các bước sau để kiểm tra xem số lượng quỷ có vượt quá số lượng người truyền giáo hay không bằng cách tính số lượng người truyền giáo và quỷ ở bờ sông đối diện (bờ sông bên phải):

number\_missionaries\_right là số lượng người truyền giáo ở bờ sông đối diện.

number\_cannibals\_right là số lượng quỷ ở bờ sông đối diện.

Hàm kiểm tra hai điều kiện:

(number\_missionaries > 0 and number\_cannibals > number\_missionaries): Điều này kiểm tra xem số lượng người truyền giáo và quỷ ở bờ sông hiện tại có vượt quá số lượng người truyền giáo hay không.

(number\_missionaries\_right > 0 and number\_cannibals\_right > number\_missionaries\_right): Điều này kiểm tra xem số lượng người truyền giáo và quỷ ở bờ sông đối diện có vượt quá số lượng người truyền giáo hay không.

Nếu một trong các điều kiện này đúng, hàm sẽ trả về True. Ngược lại, hàm sẽ trả về False, ngụ ý rằng số lượng quy không vượt quá số lượng người truyền giáo ở cả hai bờ sông.

```
58     def write_image(self, file_name="state_space.png"):
59         try:
60             self.graph.write_png(file_name)
61             print(f"File {file_name} successfully written.")
62         except Exception as e:
63             print("Error while writing file", e)
64         #print(f"File {file_name} successfully written.")
```

Định nghĩa phương thức write\_image với tham số truyền vào là file\_name có giá trị mặc định là “state\_space”, phương thức mục đích là để tạo bản vẽ của đồ thị và lưu tệp dưới định dạng “.png”. Thủ ghi hình ảnh bằng cách gọi phương thức write\_png trên đối tượng graph. Đường dẫn tệp hình ảnh sẽ được tạo bằng công thức file\_name và max\_depth, sau đó nối với phần mở rộng “.png”.

Nếu quá trình ghi hình ảnh không gặp lỗi, hàm in ra dòng "File {tên\_tệp\_hình\_ảnh} successfully written." để thông báo rằng việc ghi tệp hình ảnh đã thành công.

Nếu có lỗi xảy ra trong quá trình ghi hình ảnh, hàm sẽ bắt lỗi và in ra dòng thông báo "Error while writing file" cùng với thông báo lỗi cụ thể (e là biến chứa thông tin về lỗi).

```
66     def solve(self, solve_method="dfs"):
67         self.visited = dict()
68         Parent[self.start_state] = None
69         Move[self.start_state] = None
70         node_list[self.start_state] = None
71
72         return self.dfs(*self.start_state, 0) if solve_method == "dfs" else self.bfs()
```

“self.visited = dict()”: Đoạn mã này khởi tạo một từ điển trống để theo dõi các trạng thái đã được kiểm tra. Trong quá trình tìm kiếm, các trạng thái sẽ được thêm vào từ điển này để đảm bảo không có trạng thái nào được kiểm tra lại.

Parent[self.start\_state] = None, Move[self.start\_state] = None, và node\_list[self.start\_state] = None: Đoạn mã này khởi tạo ba từ điển Parent, Move, và node\_list để theo dõi thông tin về cha mẹ, di chuyển, và đối tượng nút cho trạng thái ban đầu (self.start\_state). Điều này sẽ hữu ích

trong việc xây dựng đường đi từ trạng thái ban đầu đến trạng thái mục tiêu sau này.

return self.dfs(\*self.start\_state, 0) if solve\_method == "dfs" else self.bfs():  
Đoạn mã này sử dụng biểu thức tam phân (ternary expression) để chọn giữa hai phương thức tìm kiếm. Nếu solve\_method được chỉ định là "dfs" (tìm kiếm theo chiều sâu), thì nó gọi phương thức dfs với trạng thái ban đầu và trả về kết quả của dfs. Nếu solve\_method không phải là "dfs" (nghĩa là "bfs" hoặc bất kỳ giá trị khác), thì nó gọi phương thức bfs và trả về kết quả của bfs. Tùy thuộc vào giá trị của solve\_method, ta có thể chọn giữa tìm kiếm theo chiều sâu hoặc tìm kiếm theo chiều rộng để giải quyết bài toán.

```
74     def draw_legend(self):
75         """
76             Utility method to draw legend on graph if legend flag is ON
77         """
78         graphlegend = pydot.Cluster(graph_name="legend", label="Legend", fontsize="20", color="gold",
79                                     fontcolor="blue", style="filled", fillcolor="#f4f4f4")
80
81
82         node1 = pydot.Node("1", style="filled", fillcolor="blue", label="Start Node", fontcolor="white", width="2", fixedsize="true")
83         graphlegend.add_node(node1)
84
85         node2 = pydot.Node("2", style="filled", fillcolor="red", label="Killed Node", fontcolor="black", width="2", fixedsize="true")
86         graphlegend.add_node(node2)
87
88         node3 = pydot.Node("3", style="filled", fillcolor="yellow", label="Solution nodes", width="2", fixedsize="true")
89         graphlegend.add_node(node3)
90
91         node4 = pydot.Node("4", style="filled", fillcolor="gray", label="Can't be expanded", width="2", fixedsize="true")
92         graphlegend.add_node(node4)
93
94         node5 = pydot.Node("5", style="filled", fillcolor="green", label="Goal node", width="2", fixedsize="true")
95         graphlegend.add_node(node5)
96
97         node7 = pydot.Node("7", style="filled", fillcolor="gold", label="Node with child", width="2", fixedsize="true")
98         graphlegend.add_node(node7)
```

Trong phương thức draw\_legend của lớp Solution, tạo một phần giải thích (legend) cho biểu đồ bằng cách thêm các nút và nút nối vào một cụm (cluster) trong biểu đồ. Đoạn mã này tạo một phần giải thích bằng cách sử dụng PyDot để biểu diễn màu sắc và ý nghĩa của các nút trên biểu đồ. Dưới đây là giải thích chi tiết:

“graphlegend = pydot.Cluster(...)”: Đoạn mã này tạo một cụm (cluster) mới trong biểu đồ. Cụm này sẽ chứa các nút giải thích của bạn và được đặt tên là "legend." Cụm này sẽ có một tiêu đề "Legend" và có các thuộc tính về màu sắc và kiểu định dạng.

node1, node2, node3, node4, node5, node7: Đoạn mã này tạo các nút để giải thích các trạng thái khác nhau trên biểu đồ. Mỗi nút biểu diễn một ý nghĩa cụ thể và được đặt màu và kiểu định dạng khác nhau:

node1: Biểu diễn nút "Start Node" với màu nền màu xanh (blue), văn bản trắng, và có nhãn "Start Node." Nút này đại diện cho trạng thái ban đầu.

node2: Biểu diễn nút "Killed Node" với màu nền đỏ (red), văn bản màu đen, và có nhãn "Killed Node." Nút này đại diện cho các trạng thái bị loại bỏ hoặc không hợp lệ.

node3: Biểu diễn nút "Solution nodes" với màu nền màu vàng (yellow) và có nhãn "Solution nodes." Nút này đại diện cho các trạng thái là một phần của giải pháp.

node4: Biểu diễn nút "Can't be expanded" với màu nền màu xám (gray) và có nhãn "Can't be expanded." Nút này đại diện cho các trạng thái không thể mở rộng hoặc duyệt.

node5: Biểu diễn nút "Goal node" với màu nền màu xanh lá cây (green) và có nhãn "Goal node." Nút này đại diện cho trạng thái mục tiêu.

node7: Biểu diễn nút "Node with child" với màu nền màu vàng (gold) và có nhãn "Node with child." Nút này có thể đại diện cho các trạng thái có các trạng thái con kết nối với nó.

Tất cả các nút này được thêm vào cụm graphlegend bằng cách sử dụng phương thức add\_node. Khi bạn vẽ biểu đồ chung với phần giải thích, nó sẽ hiển thị các nút này và các mô tả tương ứng để giúp hiểu rõ hơn nội dung của biểu đồ trạng thái.

```

101      description = "Each node (m, c, s) represents a \nstate where 'm' is the number of\nmissionaries,\n102      and 's' is the side of the boat\n"
103      " where '1' represents the left \nside and '0' the right side \n\nOur objective is to reach goal state (0, 0, 0)\n"
104      "\nfrom start state (3, 3, 1) by some \noperators = [(0, 1), (0, 2), (1, 0), (1, 1), (2, 0), ]\n"
105      "each tuples (x, y) inside operators \nrepresents the number of missionaries and\n
106      \ncannibals to be moved from left to right \nif c == 1 and viceversa"
107
108      node6 = pydot.Node( name="6", style="filled", fillcolor="gold", label=description, shape="plaintext", fontsize="20", fontcolor="red")
109      graphlegend.add_node(node6)
110
111      self.graph.add_subgraph(graphlegend)
112
113      self.graph.add_edge(pydot.Edge(node1, node2, style="invis"))
114      self.graph.add_edge(pydot.Edge(node2, node3, style="invis"))
115      self.graph.add_edge(pydot.Edge(node3, node4, style="invis"))
116      self.graph.add_edge(pydot.Edge(node4, node5, style="invis"))
117      self.graph.add_edge(pydot.Edge(node5, node7, style="invis"))
118      self.graph.add_edge(pydot.Edge(node7, node6, style="invis"))

```

description: Đây là một chuỗi mô tả chung về bài toán và biểu diễn trạng thái trong bài toán. Mô tả giải thích rằng mỗi trạng thái được biểu diễn bởi bộ ba giá trị (m, c, s), trong đó 'm' là số người truyền giáo, 'c' là số quỷ, và 's' là vị trí của thuyền. Mô tả cũng nêu rõ mục tiêu của bài toán là đạt đến trạng thái mục tiêu (0, 0, 0) từ trạng thái xuất phát (3, 3, 1) bằng

cách sử dụng các phép toán được định nghĩa trong danh sách operators. Mô tả cung cấp thông tin về ý nghĩa của các trạng thái và mô tả ý nghĩa của các phép toán.

node6: Đoạn mã này tạo một nút với mô tả (description) và các thuộc tính liên quan. Nút này được đặt màu vàng (gold), có hình dạng là "plaintext," có màu văn bản là đỏ (red), và có một nhãn dạng chuỗi mô tả (description). Nút này được tạo để hiển thị thông tin mô tả chung về bài toán và cách trạng thái được biểu diễn.

graphlegend.add\_node(node6): Đoạn mã này thêm nút node6 vào cụm graphlegend, là phần giải thích của biểu đồ. Nút này sẽ hiển thị mô tả chung của bài toán.

Các đoạn mã dưới đây sử dụng phương thức add\_edge để thêm các cạnh ảo (edges) giữa các nút giải thích để chỉ định một dãy liên kết từ "Start Node" đến "Node with child." Các cạnh này được đặt với kiểu "invis" (ẩn), nghĩa là chúng không được hiển thị trên biểu đồ, nhưng chúng định rõ mối quan hệ giữa các nút trong phần giải thích.

```
120     def draw(self, *, number_missionaries_left, number_cannibals_left, number_missionaries_right, number_cannibals_right):
121         """
122             Draw state on console using emojis
123         """
124         left_m = emoji.emojize(":old_man: " * number_missionaries_left)
125         left_c = emoji.emojize(":ogre: " * number_cannibals_left)
126         right_m = emoji.emojize(":old_man: " * number_missionaries_right)
127         right_c = emoji.emojize(":ogre: " * number_cannibals_right)
128
129         print(''{}:{}:{}:{}''.format(*args:left_m, left_c + " " * (14 - len(left_m) - len(left_c)), " " * 40, " " * (12 - len(right_m) - len(right_c)) + right_m, right_c))
130         print("")
```

“\*”: Trong định nghĩa của phương thức draw, sử dụng \* trước các tham số để chỉ định rằng các tham số sau đó sẽ được truyền dưới dạng keyword arguments. Điều này đồng nghĩa với việc ta cần gọi phương thức draw bằng cách chỉ định tên của các tham số cụ thể, chứ không phải theo thứ tự.

left\_m, left\_c, right\_m, right\_c: chuỗi emoji để biểu diễn số lượng thầy (old man) và số lượng dân cannibals (ogre) trên cả hai bờ của sông. Số lượng thầy và dân cannibals được biểu diễn bằng cách nhân chuỗi emoji với số lượng tương ứng. Các biến left\_m, left\_c, right\_m, và right\_c chứa chuỗi này.

“left\_m = emoji.emojize(f":old\_man: " \*  
number\_missionaries\_left)": Dòng này tạo chuỗi left\_m bằng cách sử dụng emoji :old\_man: để biểu diễn người truyền giáo (người đàn ông) và nhân chuỗi này với number\_missionaries\_left (số lượng người truyền giáo trên bờ trái của sông). Kết quả là left\_m là một chuỗi biểu diễn số lượng người

truyền giáo trên bờ trái bằng emoji "ogi" (người truyền giáo) lặp lại number\_missionaries\_left lần. Ví dụ, nếu number\_missionaries\_left bằng 2, thì left\_m sẽ là chuỗi "ogi ogi" (2 người truyền giáo trên bờ trái).

“left\_c = emoji.emojiize(f":ogre: \* number\_cannibals\_left)”: Tương tự như trên, dòng này tạo chuỗi left\_c bằng cách sử dụng emoji :ogre: để biểu diễn quỷ và nhân chuỗi này với number\_cannibals\_left (số lượng quỷ trên bờ trái của sông). Kết quả là left\_c là một chuỗi biểu diễn số lượng quỷ trên bờ trái bằng emoji "ogi" lặp lại number\_cannibals\_left lần.

“right\_m = emoji.emojiize(f":old\_man: \* number\_missionaries\_right)": Tương tự như trên, dòng này tạo chuỗi right\_m bằng cách sử dụng emoji :old\_man: để biểu diễn người truyền giáo và nhân chuỗi này với number\_missionaries\_right (số lượng người truyền giáo trên bờ phải của sông). Kết quả là right\_m là một chuỗi biểu diễn số lượng người truyền giáo trên bờ phải bằng emoji "ogi" lặp lại number\_missionaries\_right lần.

“right\_c = emoji.emojiize(f":ogre: \* number\_cannibals\_right)": Tương tự như trên, dòng này tạo chuỗi right\_c bằng cách sử dụng emoji :ogre: để biểu diễn quỷ và nhân chuỗi này với number\_cannibals\_right (số lượng quỷ trên bờ phải của sông). Kết quả là right\_c là một chuỗi biểu diễn số lượng quỷ trên bờ phải bằng emoji "ogi" lặp lại number\_cannibals\_right lần.

print(...): Đoạn mã này dùng để in ra màn hình console biểu diễn trạng thái của bài toán. Màn hình được chia thành hai phần, một phần bên trái và một phần bên phải, biểu diễn hai bờ của sông. Mỗi bờ bao gồm số lượng người truyền giáo và quỷ và sử dụng emoji để biểu diễn. Dấu \_ được sử dụng để biểu diễn sông, và khoảng trắng được sử dụng để cân đối khoảng cách giữa người truyền giáo và quỷ trên cả hai bờ.

```

132     def show_solution(self):
133         # Recursively start from Goal State
134         # And find parent until start state is reached
135
136         state = self.goal_state
137         path, steps, nodes = [], [], []
138
139         while state is not None:
140             path.append(state)
141             steps.append(Move[state])
142             nodes.append(node_list[state])
143
144             state = Parent[state]
145
146         steps, nodes = steps[::-1], nodes[::-1]

```

state = self.goal\_state: bắt đầu với trạng thái hiện tại là trạng thái mục tiêu (goal state), tức là (0, 0, 0).

“path, steps, nodes = [], [], []”: tạo ba danh sách trống, path để lưu trạng thái trong giải pháp, steps để lưu các bước di chuyển tương ứng và nodes để lưu các đối tượng nút tương ứng với trạng thái.

Sử dụng một vòng lặp while để thực hiện việc truy ngược từ trạng thái mục tiêu đến trạng thái xuất phát:

“path.append(state)”: Trạng thái hiện tại (state) được thêm vào danh sách path, để lưu trạng thái trong giải pháp.

“steps.append(Move[state])”: Bước di chuyển tương ứng với trạng thái hiện tại (state) được thêm vào danh sách steps. Bước di chuyển này đã được lưu trong từ điển Move.

“nodes.append(node\_list[state])”: Đối tượng nút tương ứng với trạng thái hiện tại (state) được thêm vào danh sách nodes. Đối tượng nút này đã được lưu trong từ điển node\_list.

“state = Parent[state]”: Cập nhật trạng thái hiện tại (state) bằng trạng thái cha (parent state) của nó, tức là trạng thái trước đó trên đường đi đến trạng thái mục tiêu. Điều này làm tiếp tục vòng lặp cho đến khi bạn đạt đến trạng thái xuất phát.

Sau khi vòng lặp kết thúc, ta có danh sách path chứa trạng thái từ trạng thái mục tiêu đến trạng thái xuất phát theo thứ tự, danh sách steps chứa các bước di chuyển tương ứng, và danh sách nodes chứa các đối tượng nút để biểu diễn trạng thái.

```

148     number_missionaries_left, number_cannibals_left = 3, 3
149     number_missionaries_right, number_cannibals_right = 0, 0
150
151     print("*" * 60)
152     self.draw(number_missionaries_left=number_missionaries_left, number_cannibals_left=number_cannibals_left,
153               number_missionaries_right=number_missionaries_right, number_cannibals_right=number_cannibals_right)

```

“number\_missionaries\_left, number\_cannibals\_left = 3, 3”: Thiết lập số lượng người truyền giáo và quỷ ở bờ trái của sông thành 3 người truyền giáo và 3 quỷ.

“number\_missionaries\_right, number\_cannibals\_right = 0, 0”: Thiết lập số lượng người truyền giáo và quỷ ở bờ phải của sông thành 0 người truyền giáo và 0 quỷ.

“print("\*" \* 60)": In một dòng ký tự \* có độ dài 60 lên màn hình console để tạo ra một dấu gạch ngang dùng để phân chia trạng thái ban đầu với các trạng thái giải pháp.

self.draw(...): Gọi phương thức draw trong lớp Solution để hiển thị trạng thái ban đầu của bài toán. Trạng thái này được biểu diễn bởi các biểu tượng emoji và ký tự \_ trên màn hình console. Các tham số number\_missionaries\_left, number\_cannibals\_left, number\_missionaries\_right, và number\_cannibals\_right được sử dụng để chỉ định số lượng người truyền giáo và quỷ ở cả hai bờ của sông.

```

155     for i, ((number_missionaries, number_cannibals), side), node) in enumerate(zip(steps[1:], nodes[1:])):
156
157         if node.get_label() != str(self.start_state):
158             node.set_style("filled")
159             node.set_fillcolor("yellow")
160
161         print(f"Step {i + 1}: Move {number_missionaries} missionaries and {number_cannibals} \
162               cannibals from {self.boat_side[side]} to {self.boat_side[int(not side)]}.")
163
164         op = -1 if side == 1 else 1
165
166         number_missionaries_left = number_missionaries_left + op * number_missionaries
167         number_cannibals_left = number_cannibals_left + op * number_cannibals
168
169         number_missionaries_right = number_missionaries_right - op * number_missionaries
170         number_cannibals_right = number_cannibals_right - op * number_cannibals
171
172         self.draw(number_missionaries_left=number_missionaries_left, number_cannibals_left=number_cannibals_left,
173                   number_missionaries_right=number_missionaries_right, number_cannibals_right=number_cannibals_right)
174
175     print("Congratulations!!! you have solved the problem!")
176     print("*" * 60)

```

“Vòng lặp for i, ((number\_missionaries, number\_cannibals, side), node) in enumerate(zip(steps[1:], nodes[1:]))”: Lặp qua từng bước trong giải pháp, bỏ qua bước đầu tiên vì trạng thái xuất phát đã được hiển thị ban đầu. Với mỗi bước, lấy thông tin về số lượng người truyền giáo, quỷ, và bên (side) cùng với đối tượng nút tương ứng.

“if node.get\_label() != str(self.start\_state)”: Kiểm tra nếu đối tượng nút không tương ứng với trạng thái xuất phát (start state), thì đánh dấu nút đó bằng màu vàng để chỉ ra rằng nó là một bước di chuyển trong giải pháp.

“print(f"Step {i + 1}: Move {number\_missionaries} missionaries and {number\_cannibals} cannibals from {self.boat\_side[side]} to {self.boat\_side[int(not side)]}.")”: In ra màn hình thông tin về bước di chuyển hiện tại trong giải pháp. Đây bao gồm số lượng người truyền giáo và quỷ được di chuyển từ một bên (side) sang bên kia của sông.

“op = -1 if side == 1 else 1”: Tính toán giá trị op để xác định hướng di chuyển của thuyền. Nếu thuyền ở bên trái (side == 1), op sẽ là -1, ngược lại, op sẽ là 1.

Cập nhật số lượng người truyền giáo và quỷ trên cả hai bờ của sông dựa trên hướng di chuyển op. Số lượng truyền giáo và quỷ trên bờ trái và bờ phải của sông được điều chỉnh tương ứng.

“self.draw(...)”: Gọi phương thức draw để hiển thị trạng thái mới sau bước di chuyển. Các tham số number\_missionaries\_left, number\_cannibals\_left, number\_missionaries\_right, và number\_cannibals\_right được cập nhật để thể hiện trạng thái mới.

Cuối cùng, in thông báo "Congratulations!!! you have solved the problem" để thông báo rằng đã giải quyết thành công bài toán, và in một dòng dấu gạch ngang để phân chia trạng thái cuối cùng của giải pháp với thông báo.

```

178     def draw_edge(self, number_missionaries, number_cannibals, side, depth_level):
179         u, v = None, None
180         if Parent[(number_missionaries, number_cannibals, side)] is not None:
181             u = pydot.Node(str(Parent[(number_missionaries, number_cannibals, side)] + (depth_level - 1, )), 
182                             label=str((number_missionaries, number_cannibals, side)))
183             self.graph.add_node(u)
184
185             v = pydot.Node(str((number_missionaries, number_cannibals, side, depth_level)),
186                             label=str((number_missionaries, number_cannibals, side)))
187             self.graph.add_node(v)
188
189             edge = pydot.Edge(str(Parent[(number_missionaries, number_cannibals, side)] + (depth_level - 1, )), 
190                               str((number_missionaries, number_cannibals, side, depth_level))), dir='forward')
191             self.graph.add_edge(edge)
192         else:
193             # For start node
194             v = pydot.Node(str((number_missionaries, number_cannibals, side, depth_level)),
195                             label=str((number_missionaries, number_cannibals, side)))
196             self.graph.add_node(v)
197
198     return u, v

```

Định nghĩa hàm `draw_edge` nhận các tham số `number_missionaries`, `number_cannibals`, `side` (bờ sông mà trạng thái đại diện, trạng thái bao gồm số lượng người truyền giáo và quỷ đang ở, thường là "left" ( $k=1$ ) hoặc "right" ( $k=0$ )), `depth_level` (mức độ sâu của trạng thái trong cây tìm kiếm) và `node_num` (số thứ tự của trạng thái trong cùng một mức độ sâu).

“`if Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)] is not None`”: kiểm tra xem có một phần tử tương ứng với khóa `(number_missionaries, number_cannibals, side, depth_level, node_num)` trong từ điển `Parent` hay không, và liệu giá trị của phần tử đó có khác `None` hay không.

Nếu có phần tử và khác `None` thì đó `pydot.Node()` này sẽ tạo một đối tượng, đại diện cho 1 nút trong biểu diễn đồ thị. Gán nó cho `u` với:

“`str(Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)])`”: Đây là cách truy xuất giá trị từ từ điển `Parent`. Khóa của từ điển là một bộ năm giá trị. Điều này giúp xác định trạng thái cha (nếu có) của trạng thái hiện tại.

“`label=str(Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)][:3])`”: Đây là cách lấy ba phần đầu tiên của giá trị trong từ điển `Parent` và gán chúng làm nhãn cho nút. Nhãn này sẽ hiển thị trên biểu đồ đồ thị để đại diện cho nút cha. Lấy ba ký tự đầu có thể là một cách để rút gọn và hiển thị trạng thái một cách ngắn gọn. Ví dụ, nếu `Parent[(number_missionaries, number_cannibals, side, depth_level, node_num)]` trả về một giá trị là `(2, 1, 0, 3, 4)` và `[:3]` lấy giá trị đầu, thì

label của nút u sẽ được gán bằng "2, 1, 0". Điều này làm cho nút cha trở nên dễ nhận biết trên biểu đồ đồ thị. Sau đó add node này vào đồ thị bằng câu lệnh “graph.add\_node(u)”.

Tiếp tục tạo đối tượng thông qua pydot.Node() và gán cho v. Node này mang thông tin về trạng thái hiện tại. Thêm nút này vào đồ thị bằng câu lệnh “graph.add\_node(v)”.

“edge = pydot.Edge(...)”: tạo cạnh trong biểu diễn đồ thị bằng pydot.Edge() cho 2 trạng thái được đưa vào là trạng thái nút cha và trạng thái nút hiện tại. “dir='forward'”: Tham số này định nghĩa hướng của cạnh, trong trường hợp này, nó là 'forward', ngụ ý rằng cạnh sẽ trỏ từ trạng thái cha đến trạng thái hiện tại trên biểu đồ. Sau đó thêm cạnh vào biểu diễn đồ thị bằng “graph.add\_edge(edge)”.

Ngược lại, nếu trong từ điển Parent của khóa 5 tham số mà trả về None thì tức là đây là node start. Lúc này ta tạo node v lưu trạng thái hiện tại rồi add node vào biểu đồ. Sau cùng trả về node u và v, phục vụ cho thao tác sau này.

```
199     def bfs(self):
200         q = deque()
201         q.append(self.start_state + (0, ))
202         self.visited[self.start_state] = True
203
204         while q:
205             number_missionaries, number_cannibals, side, depth_level = q.popleft()
206             # Draw Edge from u -> v
207             # Where u = Parent[v]
208             # and v = (number_missionaries, number_cannibals, side, depth_level)
209             u, v = self.draw_edge(number_missionaries, number_cannibals, side, depth_level)
```

“q = deque()”: Tạo một hàng đợi (queue) rỗng bằng cách sử dụng đối tượng deque từ thư viện collections. Hàng đợi này sẽ được sử dụng để duyệt qua các trạng thái của bài toán theo chiều rộng.

“q.append(self.start\_state + (0, ))”: Thêm trạng thái ban đầu vào hàng đợi q. Trạng thái ban đầu được biểu diễn bởi tuple self.start\_state, trong đó có thông tin về số lượng truyền giáo, quỷ, và bên của thuyền, và thêm 0 vào tuple để biểu diễn số bước di chuyển từ trạng thái ban đầu đến trạng thái hiện tại, với giá trị ban đầu là 0.

“self.visited[self.start\_state] = True”: Đánh dấu trạng thái ban đầu (tuple self.start\_state) là đã được duyệt bằng cách đặt giá trị True trong từ

điển visited. Điều này đảm bảo rằng không sẽ duyệt lại trạng thái ban đầu trong quá trình tìm kiếm.

“while q:”: Đây là một vòng lặp, và nó sẽ tiếp tục chạy cho đến khi hàng đợi q trở thành rỗng, tức là không còn trạng thái nào để duyệt.

“number\_missionaries, number\_cannibals, side, depth\_level = q.popleft()”: Lấy trạng thái tiếp theo từ hàng đợi bằng cách sử dụng .popleft(), đồng thời gán giá trị của các biến number\_missionaries, number\_cannibals, side, và depth\_level tương ứng với các phần tử của trạng thái này. Điều này cho phép bạn xử lý trạng thái này trong quá trình tìm kiếm.

“u, v = self.draw\_edge(number\_missionaries, number\_cannibals, side, depth\_level)”: Gọi phương thức draw\_edge để vẽ cạnh giữa trạng thái cha u và trạng thái con v. Phương thức này được sử dụng để biểu diễn mối quan hệ giữa các trạng thái trong biểu đồ của bài toán.

```
212     if self.is_start_state(number_missionaries, number_cannibals, side):
213         v.set_style("filled")
214         v.set_fillcolor("blue")
215         v.set_fontcolor("white")
216     elif self.is_goal_state(number_missionaries, number_cannibals, side):
217         v.set_style("filled")
218         v.set_fillcolor("green")
219         return True
220     elif self.number_of_cannibals_exceeds(number_missionaries, number_cannibals):
221         v.set_style("filled")
222         v.set_fillcolor("red")
223         continue
224     else:
225         v.set_style("filled")
226         v.set_fillcolor("orange")
```

“if is\_start\_state(number\_missionaries, number\_cannibals, side)”: Dòng mã này kiểm tra xem trạng thái hiện tại có phải là trạng thái ban đầu (trạng thái xuất phát) của bài toán hay không bằng cách gọi hàm is\_start\_state với các tham số tương ứng. Nếu trạng thái là trạng thái xuất phát, nó sẽ được tô màu xanh làm nền (filled with blue), với văn bản màu trắng.

“elif is\_goal\_state(number\_missionaries, number\_cannibals, side)”: Dòng mã này kiểm tra xem trạng thái hiện tại có phải là trạng thái mục tiêu (trạng thái kết thúc) của bài toán hay không bằng cách gọi hàm

is\_goal\_state với các tham số tương ứng. Nếu trạng thái là trạng thái mục tiêu, nó sẽ được tô màu xanh lá cây làm nền (filled with green) và có thể kết thúc quá trình xử lý, nhưng trong mã này đã có lệnh continue, nên nó sẽ bỏ qua các phần mã còn lại và tiếp tục vòng lặp.

“elif number\_of\_cannibals\_exceeds(number\_missionaries, number\_cannibals):”: Dòng mã này kiểm tra xem số lượng quỷ có vượt quá số lượng người truyền giáo trong trạng thái hiện tại hay không, bằng cách gọi hàm number\_of\_cannibals\_exceeds với các tham số tương ứng. Nếu điều kiện này đúng, trạng thái sẽ được tô màu đỏ làm nền (filled with red) và sau đó sẽ bị bỏ qua (bằng continue) trong quá trình xử lý.

“else:”: Nếu trạng thái không thuộc vào các trường hợp trên, nó sẽ được tô màu cam làm nền (filled with orange).

```
228         op = -1 if side == 1 else 1
229
230         can_be_expanded = False
```

“op = -1 if side == 1 else 1”: Dòng mã này thiết lập giá trị cho biến op bằng cách sử dụng một biểu thức điều kiện (ternary condition). Biểu thức này có dạng x if condition else y, nghĩa là nếu điều kiện condition đúng, thì biến op sẽ được gán giá trị x, còn nếu điều kiện sai, thì biến op sẽ được gán giá trị y. Trong trường hợp này, nếu side (bờ sông) bằng 1, thì op sẽ được gán giá trị -1, còn nếu side không bằng 1 (tức là side khác 1), thì op sẽ được gán giá trị 1. Tùy thuộc vào giá trị của side, op sẽ có giá trị là -1 hoặc 1.

“can\_be\_expanded = False”: Dòng mã này thiết lập biến can\_be\_expanded thành False. Biến này có thể được sử dụng sau đó trong vòng for để kiểm tra xem một trạng thái có thể mở rộng (mở rộng có nghĩa là có thể điều chỉnh trạng thái đó để tạo ra các trạng thái con mới) hay không. Trong trường hợp này, nó được đặt thành False ban đầu, ngũ ý rằng trạng thái hiện tại không thể được mở rộng mặc dù giá trị của op đã được tính toán.

```

232     for x, y in self.options:
233         next_m, next_c, next_s = number_missionaries + op * x, number_cannibals + op * y, int(not side)
234         if (next_m, next_c, next_s) not in self.visited:
235             if self.is_valid_move(next_m, next_c):
236                 can_be_expanded = True
237                 self.visited[(next_m, next_c, next_s)] = True
238                 q.append((next_m, next_c, next_s, depth_level + 1))
239
240             # Keep track of parent and corresponding move
241             Parent[(next_m, next_c, next_s)] = (number_missionaries, number_cannibals, side)
242             Move[(next_m, next_c, next_s)] = (x, y, side)
243             node_list[(next_m, next_c, next_s)] = v
244
245         if not can_be_expanded:
246             v.set_style("filled")
247             v.set_fillcolor("gray")
248
249     return False

```

Vòng lặp “for x, y in self.options:”: Duyệt qua tất cả các tùy chọn di chuyển có thể, trong đó x và y biểu diễn số lượng người truyền giáo và quý có thể được di chuyển trong một bước.

“next\_m, next\_c, next\_s = number\_missionaries + op \* x, number\_cannibals + op \* y, int(not side)”: Tính toán trạng thái tiếp theo (next\_m, next\_c, next\_s) dựa trên số lượng người truyền giáo và quý hiện tại (number\_missionaries, number\_cannibals) và hướng di chuyển (op và side). Cập nhật next\_s bằng cách lấy giá trị đối lập của side (tức là, từ 1 sang 0 hoặc ngược lại).

“if (next\_m, next\_c, next\_s) not in self.visited:”: Kiểm tra xem trạng thái tiếp theo chưa được duyệt bằng cách kiểm tra nó có trong từ điển self.visited hay không. Nếu nó chưa được duyệt, tiếp tục kiểm tra các điều kiện khác để xem xét trạng thái này.

“if self.is\_valid\_move(next\_m, next\_c):”: Kiểm tra xem trạng thái tiếp theo có hợp lệ không bằng cách sử dụng phương thức is\_valid\_move, nếu nó hợp lệ, bạn tiếp tục xử lý trạng thái này.

“can\_be\_expanded = True”: Đặt biến can\_be\_expanded thành True, để chỉ ra rằng trạng thái hiện tại có thể được mở rộng (expanded) và thêm vào hàng đợi.

“self.visited[(next\_m, next\_c, next\_s)] = True”: Đánh dấu trạng thái tiếp theo là đã được duyệt.

“q.append((next\_m, next\_c, next\_s, depth\_level + 1))”: Thêm trạng thái tiếp theo vào hàng đợi với độ sâu tăng lên 1 so với trạng thái hiện tại.

“Parent[(next\_m, next\_c, next\_s)] = (number\_missionaries, number\_cannibals, side)”: Cập nhật danh sách Parent để lưu trạng thái cha của trạng thái tiếp theo.

“Move[(next\_m, next\_c, next\_s)] = (x, y, side)”: Cập nhật danh sách Move để lưu bước di chuyển từ trạng thái hiện tại đến trạng thái tiếp theo.

“node\_list[(next\_m, next\_c, next\_s)] = v”: Cập nhật danh sách node\_list để liên kết trạng thái tiếp theo với đối tượng nút (node) tương ứng.

Tiếp đó, kiểm tra xem trạng thái hiện tại có thể được mở rộng hay không. Nếu không có trạng thái con nào có thể mở rộng từ trạng thái hiện tại, đánh dấu trạng thái hiện tại là đã được xem xét và không thể mở rộng thêm.

“if not can\_be\_expanded”: Sau khi kiểm tra tất cả các tùy chọn di chuyển và không thấy trạng thái con mới nào có thể mở rộng, kiểm tra biến can\_be\_expanded. Nếu can\_be\_expanded vẫn giữ giá trị ban đầu False, điều này có nghĩa là trạng thái hiện tại (được biểu diễn bởi v) không thể mở rộng sang bất kỳ trạng thái con nào.

“v.set\_style("filled")”: Đặt kiểu của đối tượng nút v thành "filled" để làm cho nó có màu nền (điều này thường dùng để biểu diễn trạng thái đã được xem xét hoặc không thể mở rộng nữa).

“v.set\_fillcolor("gray")”: Đặt màu nền của đối tượng nút v thành màu xám để biểu diễn rằng trạng thái này không thể mở rộng nữa.

“return False”: Cuối cùng, trả về False để cho biết rằng không tìm thấy giải pháp từ trạng thái hiện tại và tất cả các trạng thái có thể đã được duyệt qua.

```
250     def dfs(self, number_missionaries, number_cannibals, side, depth_level):
251         self.visited[(number_missionaries, number_cannibals, side)] = True
252
253         # Draw Edge from u -> v
254         # Where u = Parent[v]
255         u, v = self.draw_edge(number_missionaries, number_cannibals, side, depth_level)
```

Thực hiện thuật toán tìm kiếm theo chiều sâu (DFS - Depth-First Search) để tìm giải pháp cho bài toán.

“self.visited[(number\_missionaries, number\_cannibals, side)] = True”: Đánh dấu trạng thái hiện tại (biểu diễn bởi tuple

(number\_missionaries, number\_cannibals, side)) là đã được duyệt qua bằng cách thêm nó vào từ điển visited. Điều này đảm bảo rằng không sẽ duyệt lại trạng thái này trong quá trình tìm kiếm.

“ $u, v = self.draw\_edge(number\_missionaries, number\_cannibals, side, depth\_level)$ ”: Gọi phương thức draw\_edge để vẽ cạnh giữa trạng thái cha u và trạng thái con v. Phương thức này được sử dụng để biểu diễn mối quan hệ giữa các trạng thái trong biểu đồ của bài toán.

Phương thức dfs sẽ tiếp tục gọi đệ quy để duyệt các trạng thái con từ trạng thái hiện tại và kiểm tra xem chúng có thể dẫn đến giải pháp hay không. Trạng thái con mới sẽ được duyệt qua trong quá trình tìm kiếm.

```
258     if self.is_start_state(number_missionaries, number_cannibals, side):
259         v.set_style("filled")
260         v.set_fillcolor("blue")
261     elif self.is_goal_state(number_missionaries, number_cannibals, side):
262         v.set_style("filled")
263         v.set_fillcolor("green")
264         return True
265     elif self.number_of_cannibals_exceeds(number_missionaries, number_cannibals):
266         v.set_style("filled")
267         v.set_fillcolor("red")
268         return False
269     else:
270         v.set_style("filled")
271         v.set_fillcolor("orange")
```

“if is\_start\_state(number\_missionaries, number\_cannibals, side)”: Dòng mã này kiểm tra xem trạng thái hiện tại có phải là trạng thái ban đầu (trạng thái xuất phát) của bài toán hay không bằng cách gọi hàm is\_start\_state với các tham số tương ứng. Nếu trạng thái là trạng thái xuất phát, nó sẽ được tô màu xanh làm nền (filled with blue), với văn bản màu trắng.

“elif is\_goal\_state(number\_missionaries, number\_cannibals, side)”: Dòng mã này kiểm tra xem trạng thái hiện tại có phải là trạng thái mục tiêu (trạng thái kết thúc) của bài toán hay không bằng cách gọi hàm is\_goal\_state với các tham số tương ứng. Nếu trạng thái là trạng thái mục tiêu, nó sẽ được tô màu xanh lá cây làm nền (filled with green) và có thể kết thúc quá trình xử lý, nhưng trong mã này đã có lệnh continue, nên nó sẽ bỏ qua các phần mã còn lại và tiếp tục vòng lặp.

“elif number\_of\_cannibals\_exceeds(number\_missionaries, number\_cannibals)”: Dòng mã này kiểm tra xem số lượng quỷ có vượt

quá số lượng người truyền giáo trong trạng thái hiện tại hay không, bằng cách gọi hàm `number_of_cannibals_exceeds` với các tham số tương ứng. Nếu điều kiện này đúng, trạng thái sẽ được tô màu đỏ làm nền (filled with red) và sau đó sẽ bị bỏ qua (bằng `continue`) trong quá trình xử lý.

“else:”: Nếu trạng thái không thuộc vào các trường hợp trên, nó sẽ được tô màu cam làm nền (filled with orange).

```

273         solution_found = False
274         operation = -1 if side == 1 else 1
275
276         can_be_expanded = False

```

“`solution_found = False`”: Khởi tạo biến `solution_found` thành `False`. Biến này được sử dụng để theo dõi xem trong quá trình tìm kiếm DFS có tìm thấy giải pháp hay không. Nếu `solution_found` vẫn giữ giá trị `False`, điều này có nghĩa là chưa tìm thấy giải pháp.

“`operation = -1 if side == 1 else 1`”: Tính toán giá trị `operation` để xác định hướng di chuyển của thuyền. Nếu `side` (bên của thuyền) là 1 (bên trái), thì `operation` sẽ là `-1`, ngược lại, nó sẽ là `1`. Giá trị `operation` này sẽ được sử dụng để cập nhật trạng thái của thuyền và dân cannibals khi di thuyền từ một bờ sang bờ khác.

“`can_be_expanded = False`”: Khởi tạo biến `can_be_expanded` thành `False`. Biến này sẽ được sử dụng để theo dõi xem trạng thái hiện tại có thể mở rộng (tạo ra các trạng thái con) hay không. Nếu `can_be_expanded` vẫn giữ giá trị `False`, điều này có nghĩa là không có trạng thái con để tiếp tục mở rộng.

```

278     for x, y in self.options:
279         next_m, next_c, next_s = number_missionaries + operation * x, number_cannibals + operation * y, int(not side)
280
281         if (next_m, next_c, next_s) not in self.visited:
282             if self.is_valid_move(next_m, next_c):
283                 can_be_expanded = True
284                 # Keep track of Parent state and corresponding move
285                 Parent[(next_m, next_c, next_s)] = (number_missionaries, number_cannibals, side)
286                 Move[(next_m, next_c, next_s)] = (x, y, side)
287                 node_list[(next_m, next_c, next_s)] = v
288
289                 solution_found = (solution_found or self.dfs(next_m, next_c, next_s, depth_level + 1))
290
291             if solution_found:
292                 return True

```

“`for x, y in self.options:`”: Đoạn mã này duyệt qua tất cả các tùy chọn có thể thực hiện trong một bước di chuyển, được lưu trữ trong `self.options`.

Mỗi tùy chọn có thể thay đổi số lượng misionaries (number\_missionaries) và số lượng cannibals (number\_cannibals) trên một bờ sông (side).

“next\_m, next\_c, next\_s = number\_missionaries + operation \* x, number\_cannibals + operation \* y, int(not side)”": Đoạn mã này tính toán số lượng missionaries, cannibals, và bờ sông (side) sau khi thực hiện một bước di chuyển. x và y là số lượng missionaries và cannibals sẽ di chuyển (hoặc rời bỏ) trong tùy chọn hiện tại. operation là một biến kiểm soát di chuyển đi hoặc về (có thể là -1 hoặc 1). next\_s là bờ sông mới sau bước di chuyển.

“if (next\_m, next\_c, next\_s) not in self.visited”": Đoạn mã này kiểm tra xem trạng thái (next\_m, next\_c, next\_s) đã được duyệt qua trước đó hay chưa. Nếu trạng thái này chưa được duyệt, nghĩa là nó là một trạng thái mới, ta có thể tiếp tục thực hiện các bước tiếp theo.

“if self.is\_valid\_move(next\_m, next\_c)”": Đoạn mã này kiểm tra xem bước di chuyển từ trạng thái hiện tại đến trạng thái mới có hợp lệ không. Thông thường, bài toán Missionaries and Cannibals có một số ràng buộc, ví dụ: không được để lại nhiều hơn một lượng cannibals hơn missionaries trên bất kỳ bờ nào.

“self.is\_valid\_move(next\_m, next\_c)”": Điều kiện kiểm tra xem một bước di chuyển từ trạng thái hiện tại (được đại diện bởi next\_m, next\_c, next\_s) có hợp lệ hay không. Có thể là kiểm tra xem bước di chuyển có làm cho số lượng người và số lượng người cannibal ở mỗi bên của sông không vượt quá một ngưỡng cụ thể hoặc có bất kỳ ràng buộc nào khác của vấn đề cụ thể.

“can\_be\_expanded = True”": Gán giá trị True cho biến can\_be\_expanded, đánh dấu rằng có thể mở rộng nút hiện tại để tìm kiếm thêm các nút con.

“Parent[(next\_m, next\_c, next\_s)] = (number\_missionaries, number\_cannibals, side)”": Lưu trữ thông tin về trạng thái cha của nút hiện tại. Trong trường hợp này, Parent là một từ điển (dictionary) với các khóa là trạng thái của nút con và giá trị tương ứng là thông tin về trạng thái của nút cha (số lượng người truyền giáo, số lượng quỷ, và bên của sông).

“Move[(next\_m, next\_c, next\_s)] = (x, y, side)”": Lưu trữ thông tin về bước di chuyển từ trạng thái cha đến trạng thái con. Move cũng là một từ điển, trong đó khóa là trạng thái của nút con và giá trị tương ứng là thông

tin về bước di chuyển (số lượng người truyền giáo và số lượng quý di chuyển từ x sang y trên side).

“node\_list[(next\_m, next\_c, next\_s)] = v”: Lưu trữ giá trị v cho trạng thái con trong node\_list, có thể là một danh sách hoặc tập hợp các trạng thái đã xem xét.

“solution\_found = (solution\_found or self.dfs(next\_m, next\_c, next\_s, depth\_level + 1))”: Gọi đệ quy hàm dfs trên trạng thái con. Nếu hàm dfs trả về giá trị True, solution\_found sẽ được gán bằng True, đánh dấu rằng đã tìm thấy một giải pháp. Điều này sẽ tiếp tục cho đến khi tìm thấy giải pháp hoặc đã kiểm tra tất cả các trạng thái con mà không tìm thấy giải pháp.

“if solution\_found:”: Kiểm tra xem có tìm thấy giải pháp hay không. Nếu có, hàm sẽ trả về True và kết thúc.

```
294     if not can_be_expanded:  
295         v.set_style("filled")  
296         v.set_fillcolor("gray")  
297  
298     self.solved = solution_found  
299     return solution_found
```

“if not can\_be\_expanded:”: Điều kiện kiểm tra xem nút hiện tại có thể được mở rộng hay không. Nếu can\_be\_expanded là False, điều này يعني rằng không còn trạng thái con nào có thể được kiểm tra (đã kiểm tra tất cả các trạng thái con), và do đó nút này đã hoàn thành việc kiểm tra. Trong trường hợp này, nút đóng bằng cách đặt màu nền là màu xám để đánh dấu rằng nó đã được kiểm tra và không cần xem xét nữa.

“v.set\_style("filled") và v.set\_fillcolor("gray")”: Đây là các hàm (method) dùng để thiết lập kiểu và màu nền của nút (node) trong biểu đồ hoặc cấu trúc dữ liệu mà bạn đang làm việc. Bằng cách gọi hàm này, bạn đặt kiểu của nút thành "filled" và đặt màu nền của nó thành màu xám ("gray").

“self.solved = solution\_found”: Cập nhật thuộc tính solved của đối tượng hiện tại bằng giá trị của solution\_found. Thuộc tính solved thường được sử dụng để đánh dấu xem vấn đề đã được giải quyết (hoặc tìm thấy giải pháp) hay chưa.

“return solution\_found: Trả về giá trị của solution\_found, thường là True nếu đã tìm thấy giải pháp hoặc False nếu không tìm thấy giải pháp. Kết quả này sẽ được truyền lên trình gọi gốc để đánh dấu rằng vấn đề đã được giải quyết hoặc không có giải pháp nếu True, và ngược lại.

### III. CHẠY ĐOAN MÃ.

#### 1. Khởi tạo cây không gian trạng thái.

python generate\_full\_space\_tree.py -d 10 (với d là độ sâu (depth=10))

```
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2> python generate_full_space_tree.py -d 10
File state_space_10.png successfully written.
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2>
```

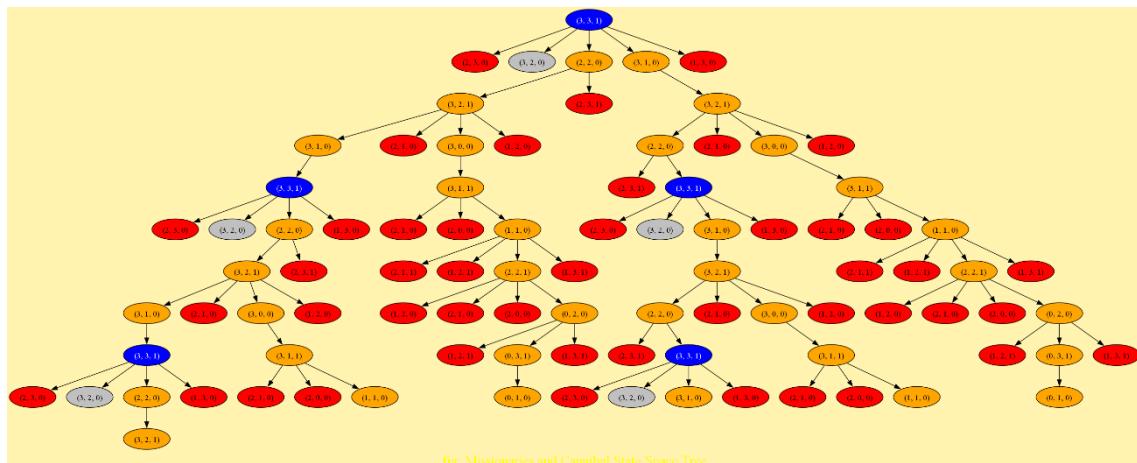


fig: Missionaries and Cannibals State Space Tree

```
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2> python generate_full_space_tree.py -d 19
File state_space_19.png successfully written.
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2>
```

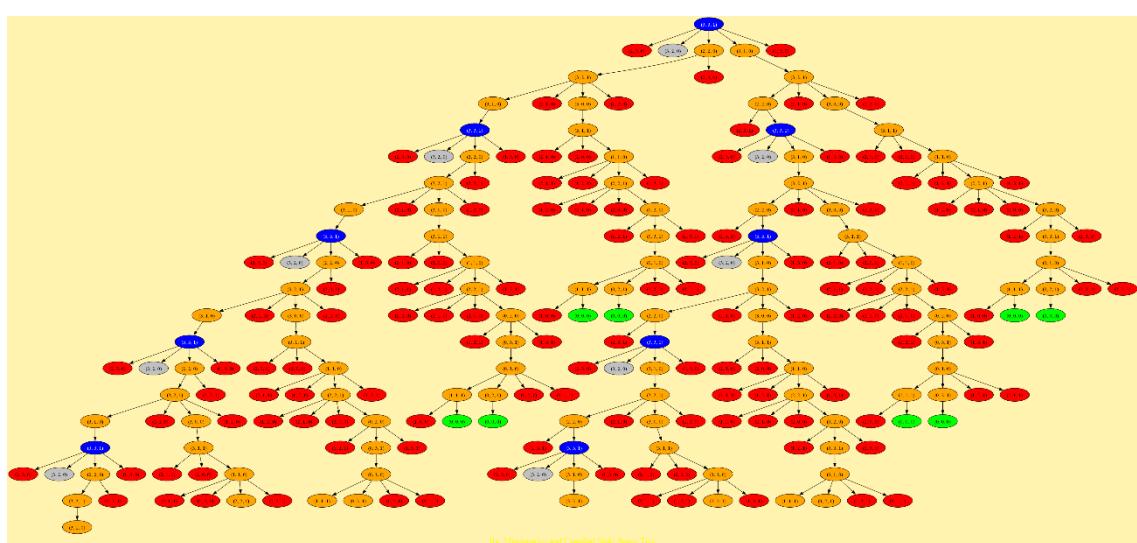


fig: Missionaries and Cannibals State Space Tree

## 2. Cây DFS.

- DFS: python main.py -m dfs

```
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2> python main.py -m dfs
*****
ΘΘΘΘΘΘ ----- ΘΘΘΘΘΘ
Step 1: Move 1 missionaries and 1 cannibals from left to right.
ΘΘΘΘΘΘ ----- ΘΘΘΘΘΘ
Step 2: Move 1 missionaries and 0 cannibals from right to left.
ΘΘΘΘΘΘ ----- ΘΘΘΘΘΘ
Step 3: Move 0 missionaries and 2 cannibals from left to right.
ΘΘΘΘΘΘ ----- ΘΘΘΘΘΘ
Step 10: Move 1 missionaries and 0 cannibals from right to left.
ΘΘΘΘΘΘ ----- ΘΘΘΘΘΘ
Step 11: Move 1 missionaries and 1 cannibals from left to right.
ΘΘΘΘΘΘ ----- ΘΘΘΘΘΘ

Congratulations!!! you have solved the problem
*****
File dfs.png successfully written.
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2>
```

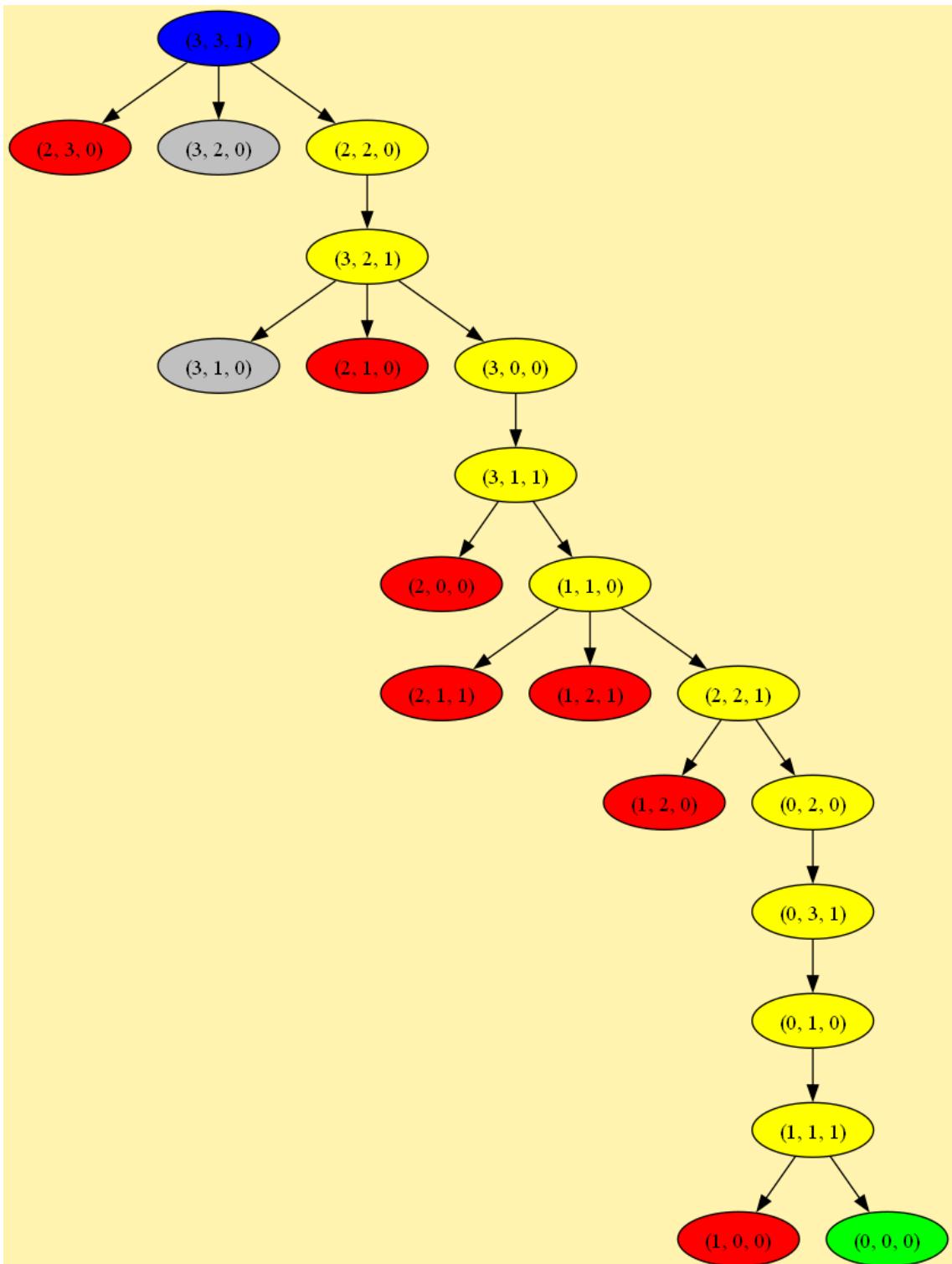
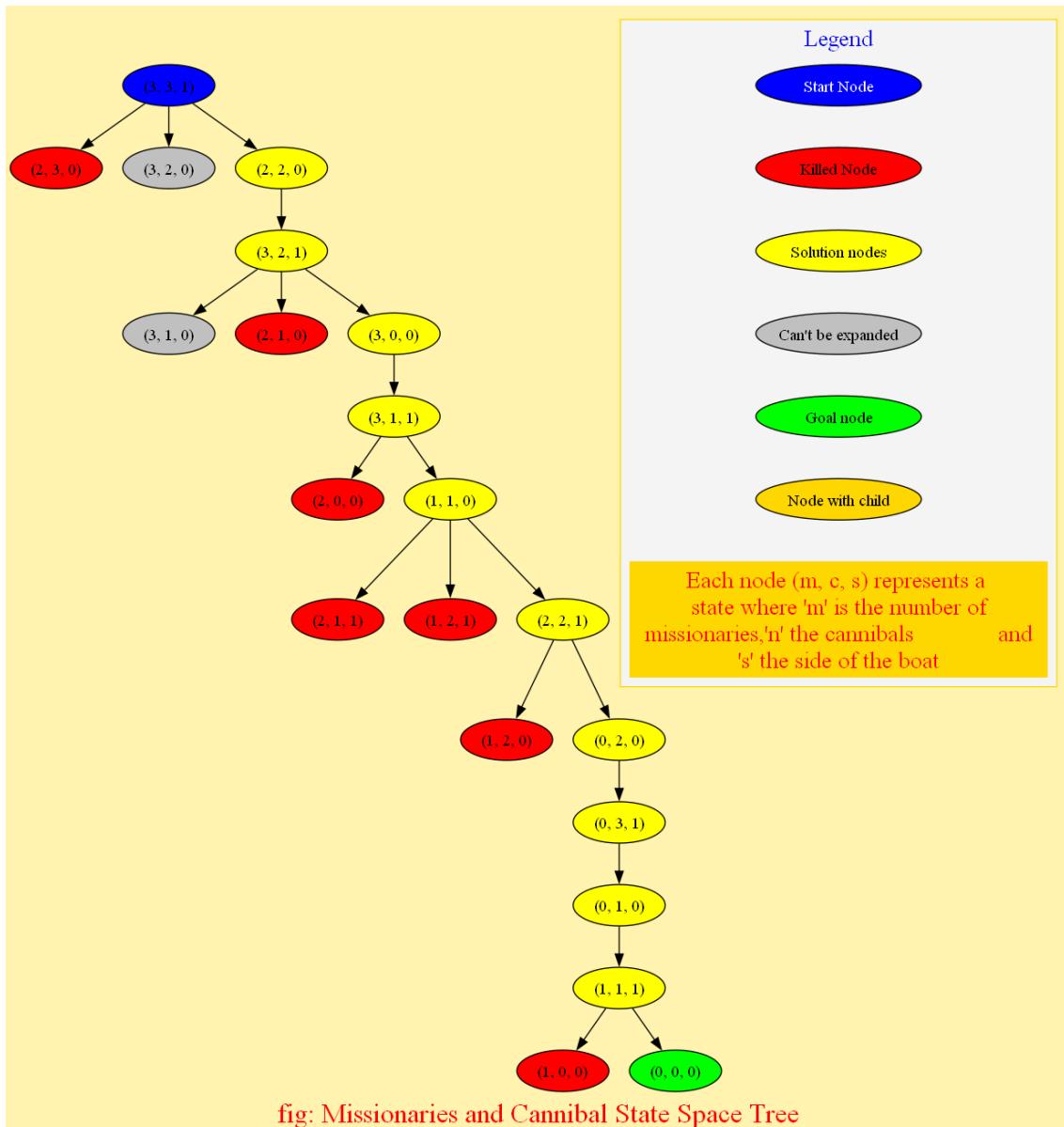


fig: Missionaries and Cannibal State Space Tree

- DFS với legend: python main.py -m dfs -l True

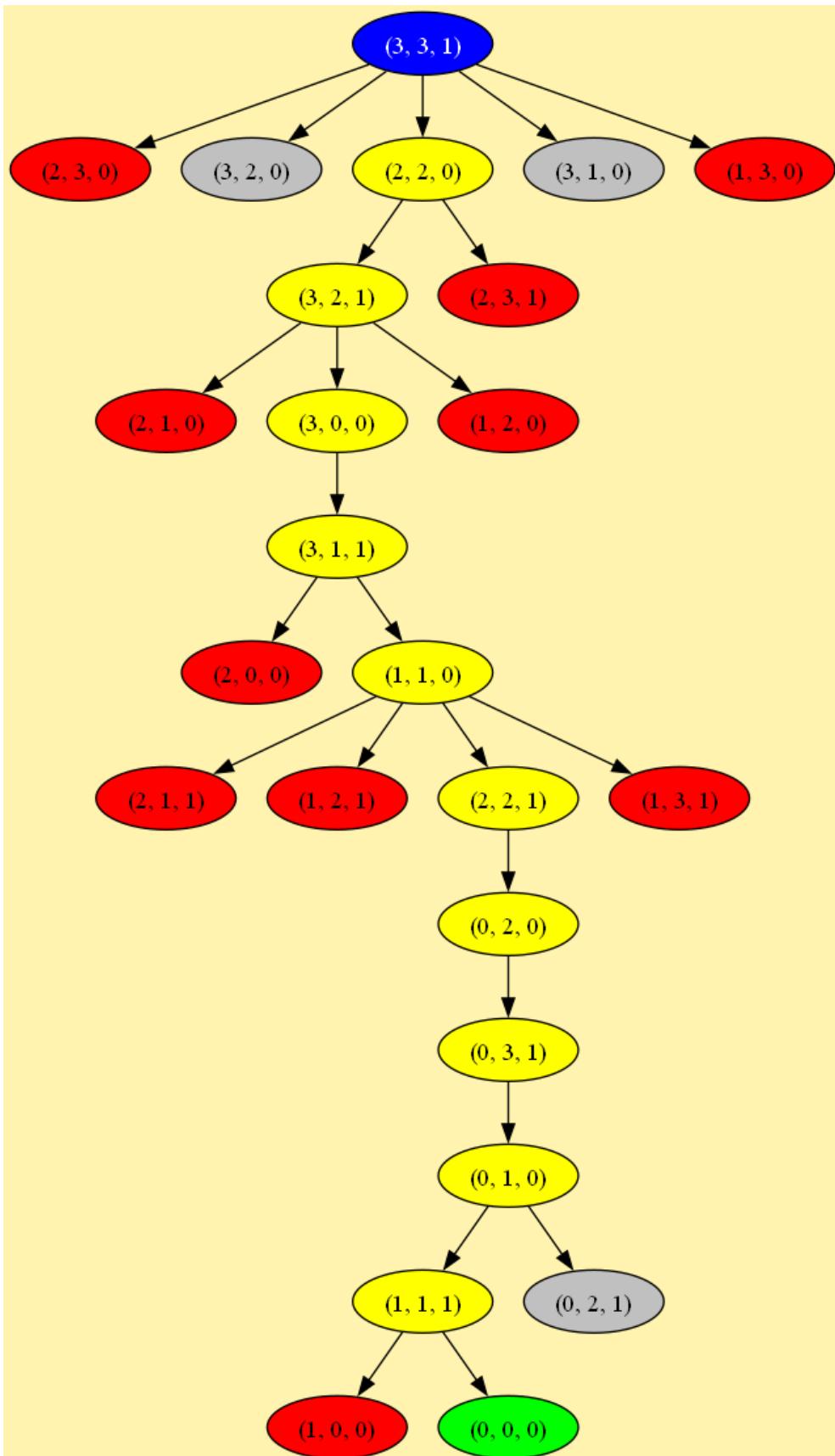
```
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2> python main.py -m dfs -l True
*****
ΘΘΘΘΘ-----ΘΘΘΘΘ
Step 1: Move 1 missionaries and 1 cannibals from left to right.
ΘΘΘΘΘ-----ΘΘΘΘΘ
Step 2: Move 1 missionaries and 0 cannibals from right to left.
ΘΘΘΘΘ-----ΘΘΘΘΘ
Step 3: Move 0 missionaries and 2 cannibals from left to right.
ΘΘΘΘΘ-----ΘΘΘΘΘ
Step 4: Move 0 missionaries and 1 cannibals from right to left.
ΘΘΘΘΘ-----ΘΘΘΘΘ
Step 5: Move 2 missionaries and 0 cannibals from left to right.
ΘΘΘΘ-----ΘΘΘΘΘΘΘΘΘ
Step 6: Move 1 missionaries and 1 cannibals from right to left.
ΘΘΘΘΘ-----ΘΘΘΘΘ
Step 7: Move 2 missionaries and 0 cannibals from left to right.
ΘΘΘΘ-----ΘΘΘΘΘΘΘΘΘ
Step 8: Move 0 missionaries and 1 cannibals from right to left.
ΘΘΘΘΘ-----ΘΘΘΘΘ
Step 9: Move 0 missionaries and 2 cannibals from left to right.
ΘΘΘΘΘ-----ΘΘΘΘΘΘΘΘΘ
Step 10: Move 1 missionaries and 0 cannibals from right to left.
ΘΘΘΘΘ-----ΘΘΘΘΘΘΘΘΘ
Step 11: Move 1 missionaries and 1 cannibals from left to right.
-----
Congratulations!!! you have solved the problem
*****
File dfs_legend.png successfully written.
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2>
```



### 3. Cây BFS.

- Cây BFS: - BFS: python main.py -m bfs

```
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2> python main.py -m bfs
*****
      🧑  🧑  🧑  🧑  🧑  🧑 -----  
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  
*****  
File bfs.png successfully written.  
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2>
```



**fig: Missionaries and Cannibals State Space Tree**

- BFS với legend: python main.py -m bfs -l True

```
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2> python main.py -m bfs -l True
*****
      🧑  🧑  🧑  🧑  🧑  🧑      -----
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
*****
File bfs_legend.png successfully written.
(venv) PS C:\Users\PC-LENOVO\Downloads\test_code_lab2>
```

