Bài tập Thực hành Nhập môn Trí tuệ Nhân tạo tuần 2

Nguyễn Lê Ngọc Duy - 20280023 - 20KDL1

Mục lục

1 Bài toán **2**
1.1 Mô tả bài toán . . . 2
1.2 Phân tích không gian trạng thái. . . . 2

2 Tìm hiểu thuật toán. **3**
2.1 Thuật toán Breadth First Search - BFS. . . . 3
2.2 Thuật toán Depth First Search - DFS. . . . 4
2.3 So sánh ban đầu về hai thuật toán. . . . 4

3 Cài đặt bằng `Python` **5**
3.1 Các thư viện hỗ trợ . . . 5
3.2 Các file . . . 5
3.2.1 File `generate_full_space_tree.py` . . . 5
3.2.2 File `solve.py` . . . 6
3.2.3 File `main.py` . . . 7

4 Kết quả **7**
4.1 Khởi tạo cây không gian trạng thái . . . 7
4.2 Giải bài toán với DFS . . . 8
4.3 Giải bài toán với BFS . . . 9
4.4 Nhận xét . . . 13

5 Tham khảo **13**

1 Bài toán

1.1 Mô tả bài toán

Trong bài báo cáo này, chúng ta sẽ xem xét quy trình tìm ra lời giải cho bài toán sau bằng hai thuật toán BFS và DFS.
Có 3 người truyền giáo và 3 con quỷ ở bờ bên trái của một con sông, cùng với một 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 đó sẽ ăn thịt số người truyền giáo. Tìm cách để đưa tất cả người truyền giáo qua bờ bên phải của sông mà không bị ăn thịt, nghĩa là số người không ít hơn số quỷ ở cùng 1 bờ (bên trái hoặc bên phải).

1.2 Phân tích không gian trạng thái.

Gọi (a, b, k) với $0 \leq a, b \leq 3$; trong đó:

- a là số người
- b là số quỷ ở bờ bên trái
- $k = 1$ nếu thuyền ở bên trái, ngược lại $k = 0$.

Khi đó, không gian trạng thái của bài toán đuọc xác định như sau:

- Trạng thái ban đầu: $(3, 3, 1)$
- Thuyền chở qua sông 1 người, hoặc 1 con quỷ, hoặc 1 người và 1 con quỷ, hoặc 2 người, hoặc 2 con quỷ. Như vậy nếu gọi (x, y) lần lượt là số người và số quỷ di chuyển từ bờ bên trái qua bờ bên phải hay ngược lại thì ta có các phép toán chuyển từ trạng thái này sang trạng thái khác là
  - $(1, 0)$.
  - $(0, 1)$.
  - $(1, 1)$.
  - $(2, 0)$.
  - $(0, 2)$.
- Trạng thái kết thúc là $(0, 0, 0)$.

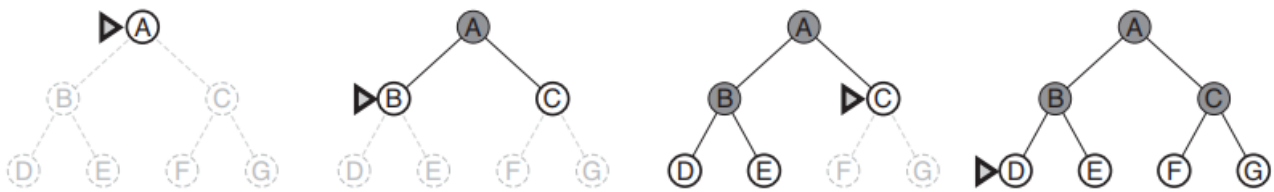
2 Tìm hiểu thuật toán.

2.1 Thuật toán Breadth First Search - BFS.

Thuật toán BFS là thuật toán mà trạng thái gốc được duyệt trước, sau đó đến các bước tiếp theo và cuối cùng đến trạng thái gốc. Tổng quát, tất cả các node ở cùng một mức sẽ được khám phá hết, sau đó đến mức tiếp theo.
Ta có thể dùng một hàng đợi để lưu trữ các node cần khám phá. Các node mới nhất sẽ được thêm vào cuối hàng đợi, và các node cũ sẽ được loại bỏ khỏi đầu hàng đợi.
Dưới đây chính là mã giả cho thuật toán BFS:

Thuật toán 1 Thuật toán Breadth First Search - BFS

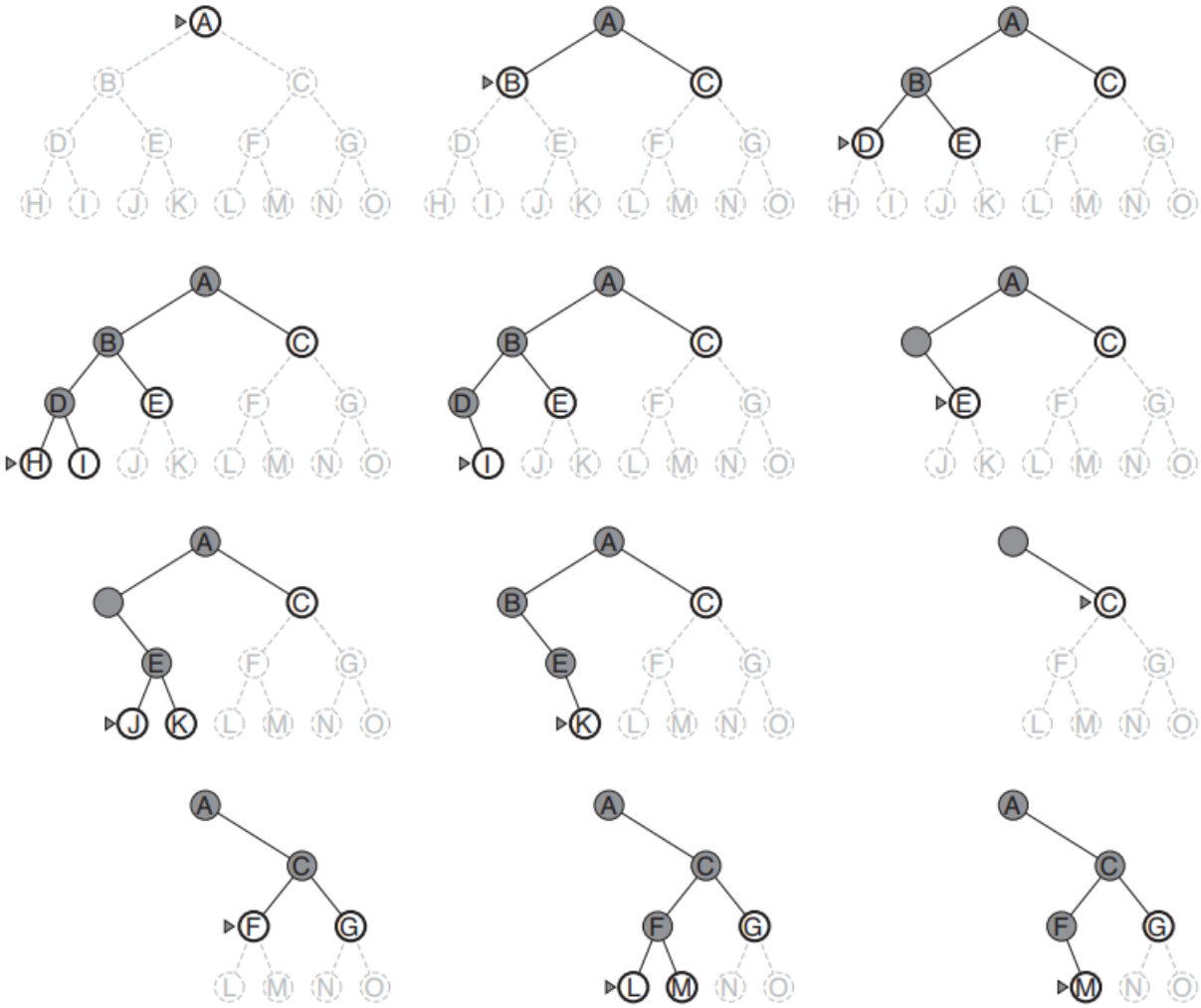
Đầu vào: Bài toán
Đầu ra: Lời giải hoặc thông báo: không tồn tại lời giải.
begin
1: *node* ← một node với STATE = *problem*.INITIAL-STATE, PATH-COST = 0
2: **if** *problem*.GOAL-TEST(*node*.STATE) **then**
3: **return** SOLUTION(*node*)
4: **end if**
5: *frontier* ← một hàng đợi với *node* là phần tử duy nhất.
6: **loop do**
7: **if** EMPTY?(*frontier*) **then**
8: **return** FAILURE
9: **end if**
10: *node* ← POP(*frontier*)
11: Thêm *node*.STATE vào *explored*
12: **for each** action **in** *problem*.ACTIONS(*node*.STATE) **do**
13: *child* ← CHILD-NODE(*problem*, *node*, *action*)
14: **if** *child*.STATE không ở trong *explored* hay *frontier* **then**
15: **if** *problem*.GOAL-TEST(*child*.STATE) **then**
16: **return** SOLUTION(*child*)
17: **end if**
18: *frontier* ← INSERT(*child*, *frontier*)
19: **end if**
20: **end for**
21: **end loop**
end



Hình 1: Minh hoạ thuật toán BFS

2.2 Thuật toán Depth First Search - DFS.

Khác với thuật toán BFS, thuật toán DFS ưu tiên duyệt tìm node sâu nhất ở trạng thái hiện tại của cây. Quá trình tìm kiếm ngay lập tức tiến đến độ sâu tối đa của cây, khi các node không có node con nào thì quay lại node cha và tìm node con tiếp theo.
Ta sử dụng một ngăn xếp để lưu trữ kết quả tìm kiếm, theo đó, các node vừa được thêm vào ngăn xếp được duyệt trước, và các node cũ sẽ được loại bỏ khỏi ngăn xếp.



Hình 2: Minh hoạ thuật toán DFS

2.3 So sánh ban đầu về hai thuật toán.

Bảng sau so sánh hai thuật toán trên dựa trên 4 tiêu chí: độ hoàn thiện, độ phức tạp thời gian, độ phức tạp không gian và độ tối ưu, trong đó:

- b: số node con của một node.
- d: độ sâu của node nông nhất.
- m: độ sâu tối đa của cây.

- [a]: đầy đủ nếu b hữu hạn.
- [c]: tối ưu nếu chi phí ở mọi bước là xác định.

Thuật toán	BFS	DFS
Độ hoàn thiện	Đầy đủ[a]	Không đầy đủ
Độ phức tạp thời gian	$O(b^d)$	$O(b^m)$
Độ phức tạp không gian	$O(b^d)$	$O(bm)$
Độ tối ưu	Tối ưu[c]	Không tối ưu

3 Cài đặt bằng `Python`

3.1 Các thư viện hỗ trợ

- `graphviz`: Là thư viện mã nguồn mở giúp hỗ trợ cho việc minh hoạ các đồ thị với các tính năng hữu dụng như chỉnh font chữ, màu sắc, nét vẽ,...
- `pydot`: Đây là thư viện hỗ trợ cùng thư viện `graphviz` cho `Python`. Thư viện hỗ trợ xuất các đồ thị dưới dạng hình ảnh.
- `pyparsing`: Đây là thư viện hỗ trợ xuất chuỗi dưới các định dạng đặc biệt do người dùng tự tạo ra.
- `emoji`: Thư viện tạo ra các emoji, thường được sử dụng trong xử lý ngôn ngữ.

Hình 3: Thư viện graphviz

3.2 Các file

3.2.1 File `generate_full_space_tree.py`

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

- `is_valid_move(number_missionaries, number_cannibals)`: kiểm tra xem một nước đi nào đó có thoả ràng buộc hay không.
- `draw_edge(number_missionaries, number_cannibals, side, depth_level, node_num)`: vẽ hai trạng thái nào đó và cạnh nối hai trạng thái đó với nhau. Nếu là trạng thái bắt đầu, hàm chỉ vẽ node mà không vẽ cạnh nối.

- `write_image(file_name="state_space")`: xuất ra file ảnh có dạng `{file_name}_{max_depth}.png` (với `max_depth` là độ sâu tối đa của cây) đồ thị đã vẽ được.
- `is_start_state(number_missionaries, number_cannibals, side)`: kiểm tra xem trạng thái nào đó có phải là trạng thái ban đầu không.
- `is_goal_state(number_missionaries, number_cannibals, side)`: kiểm tra xem trạng thái nào đó có phải là trạng thái kết thúc không.
- `number_of_cannibals_exceeds(number_missionaries, number_cannibals)`: kiểm tra xem trạng thái nào đó có đạt đến ngưỡng giới hạn không.
- `generate()`: đây là hàm chính, giúp hình thành lời giải của bài toán. Hàm trả về `True` nếu đi đến được độ sâu sâu nhất của cây, ngược lại trả về `False`.

Ngoài ra, ta tạo thêm tham số dòng lệnh `-d` quyết định độ sâu của cây ta muốn tạo (mặc định là 20).

3.2.2 File `solve.py`

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

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

Các phương thức trong lớp này bao gồm:

- `is_valid_move(self, number_missionaries, number_cannibals)`: kiểm tra xem một nước đi nào đó có thoả ràng buộc hay không.
- `is_start_state(self, number_missionaries, number_cannibals, side)`: kiểm tra xem trạng thái nào đó có phải là trạng thái ban đầu không.
- `is_goal_state(self, number_missionaries, number_cannibals, side)`: kiểm tra xem trạng thái nào đó có phải là trạng thái kết thúc không.
- `number_of_cannibals_exceeds(self, number_missionaries, number_cannibals)`: kiểm tra xem trạng thái nào đó có đạt đến ngưỡng giới hạn không.
- `write_image(self, file_name="state_space.png")`: hàm này dùng để lưu đồ thị ra file ảnh.

- `draw_edge(self, number_missionaries, number_cannibals, side, depth_level)`: vẽ hai trạng thái nào đó và cạnh nối hai trạng thái đó với nhau. Nếu là trạng thái bắt đầu, hàm chỉ vẽ node mà không vẽ cạnh nối.
- `draw_legend(self)`: hàm này dùng để ghi chú thích trên đồ thị.
- `draw(self, *, number_missionaries_left, number_cannibals_left, number_missionaries_right, number_cannibals_right)`: hàm này dùng để hiển thị trình tự các trạng thái trên console.
- `bfs(self)`: hàm này dùng để thực hiện thuật toán BFS.
- `dfs(self, number_missionaries, number_cannibals, side, depth_level)`: hàm này dùng để thực hiện thuật toán DFS. Khác với hàm `bfs(self)`, ta thêm các tham số để có thể gọi đệ quy hàm này nhiều lần.

3.2.3 File `main.py`

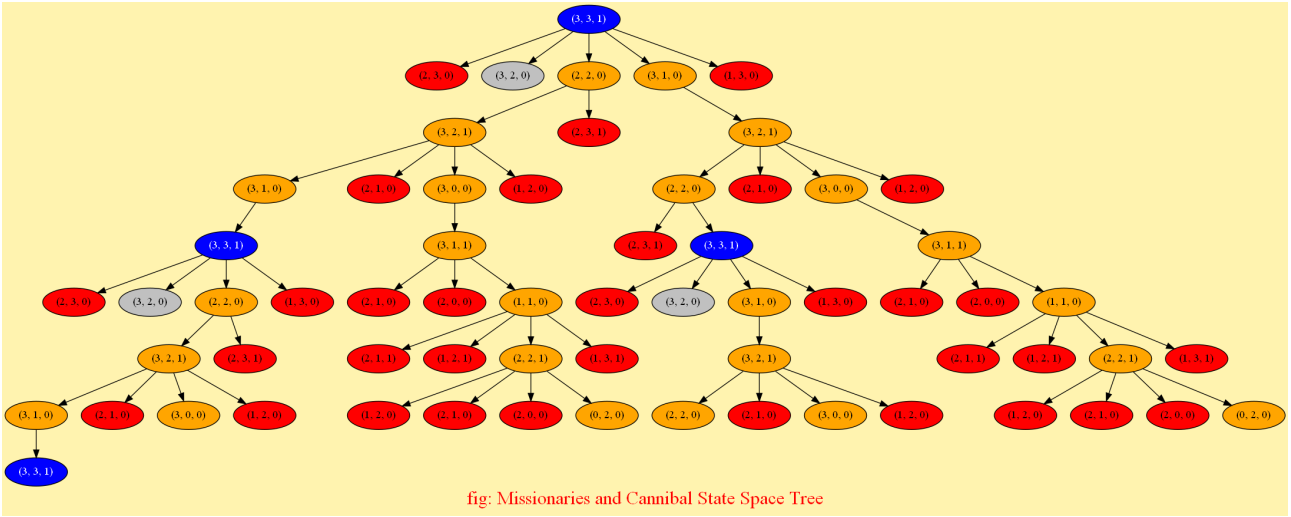
Nhiệm vụ chính của file này là tạo ra một đối tượng của lớp `Solution` và gọi đến các phương thức của lớp đó để giải bài toán. Trong file này, ta còn có thêm một số tham số dòng lệnh đi kèm:

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

4 Kết quả

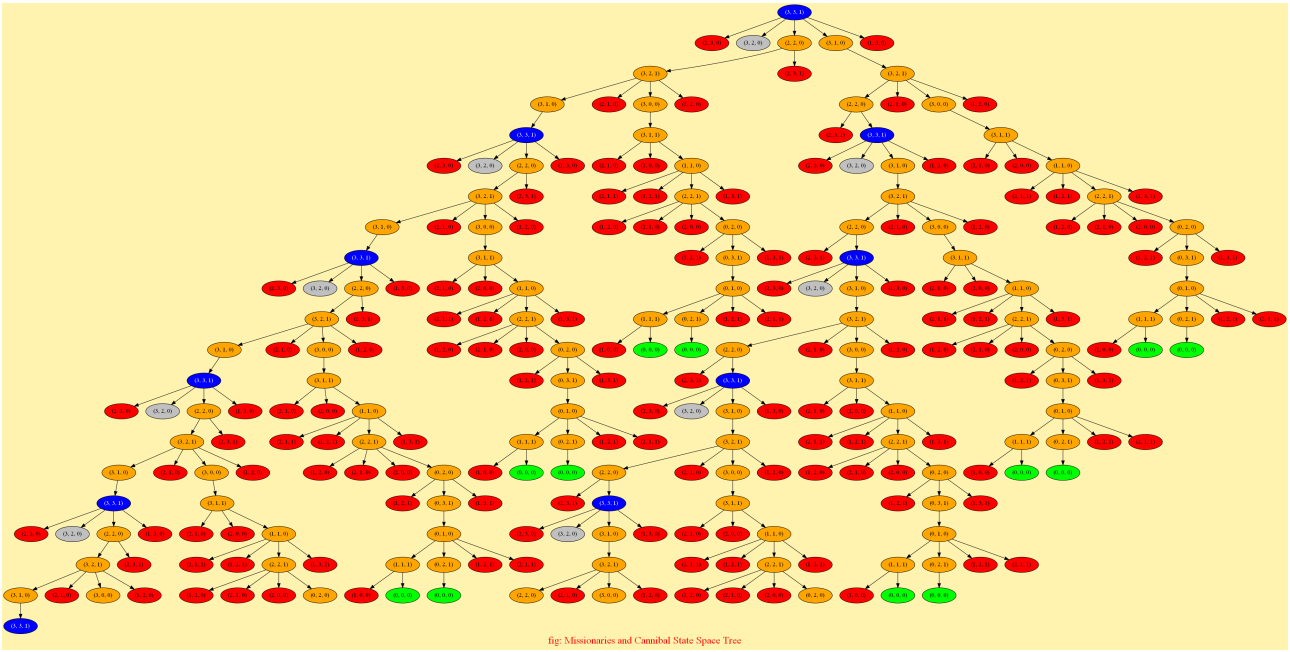
4.1 Khởi tạo cây không gian trạng thái

Gõ lệnh `python generate_full_space_tree.py -d 8` trên terminal, ta thu được kết quả như hình 4:



Hình 4: Cây không gian trạng thái cho bài toán với độ sâu tối đa là 8

Nếu ta gõ lệnh trên terminal `python generate_full_space_tree.py -d 20`, tức là thay đổi độ sâu tối đa của cây tìm kiếm thành 20, ta thu được cây tìm kiếm như hình 5:

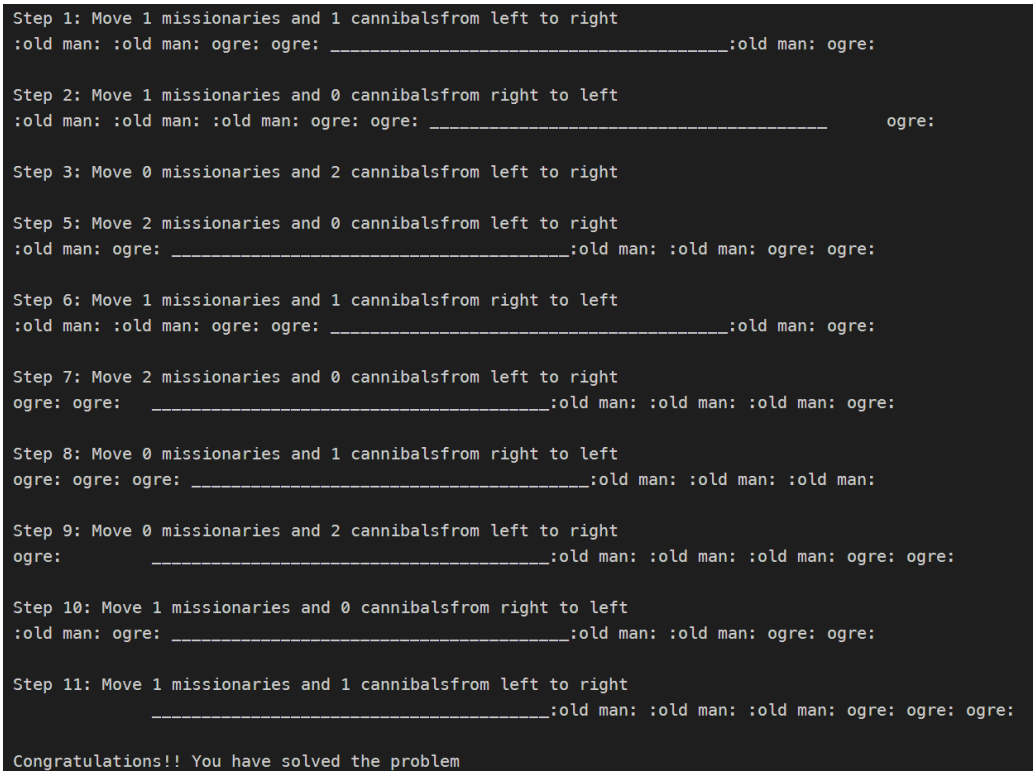


Hình 5: Cây không gian trạng thái cho bài toán với độ sâu tối đa là 20

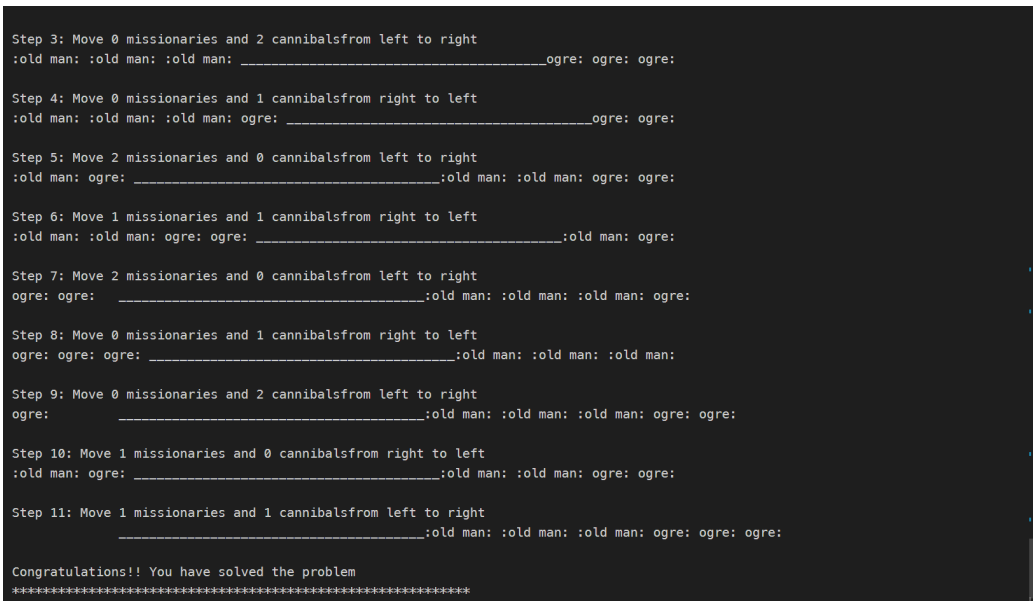
Có thể thấy được rằng khi tăng độ sâu tối đa của cây tìm kiếm lên 20, ta có thể thu được lời giải của bài toán. Ngược lại, nếu độ sâu tối da của cây tìm kiếm là 8, ta không thể tìm được lời giải của bài toán.

4.2 Giải bài toán với DFS

Thực thi lệnh `python main.py -m dfs`, ta thu được kết quả trên màn hình comsole ở hình 6 và file `dfs.png` như hình 8a. Do thuật toán DFS ưu tiên duyệt trạng thái theo chiều sâu, cho nên ta có thể thấy lời giải của bài toán nghiêng hẳn về bên tay phải (các node màu vàng).
Để chú thích cây tìm kiếm, ta thực thi lệnh `python main.py -m dfs -l True`, ta thu được kết quả trên màn hình comsole ở hình 7 và file `dfs_legend.png` như hình 8b.



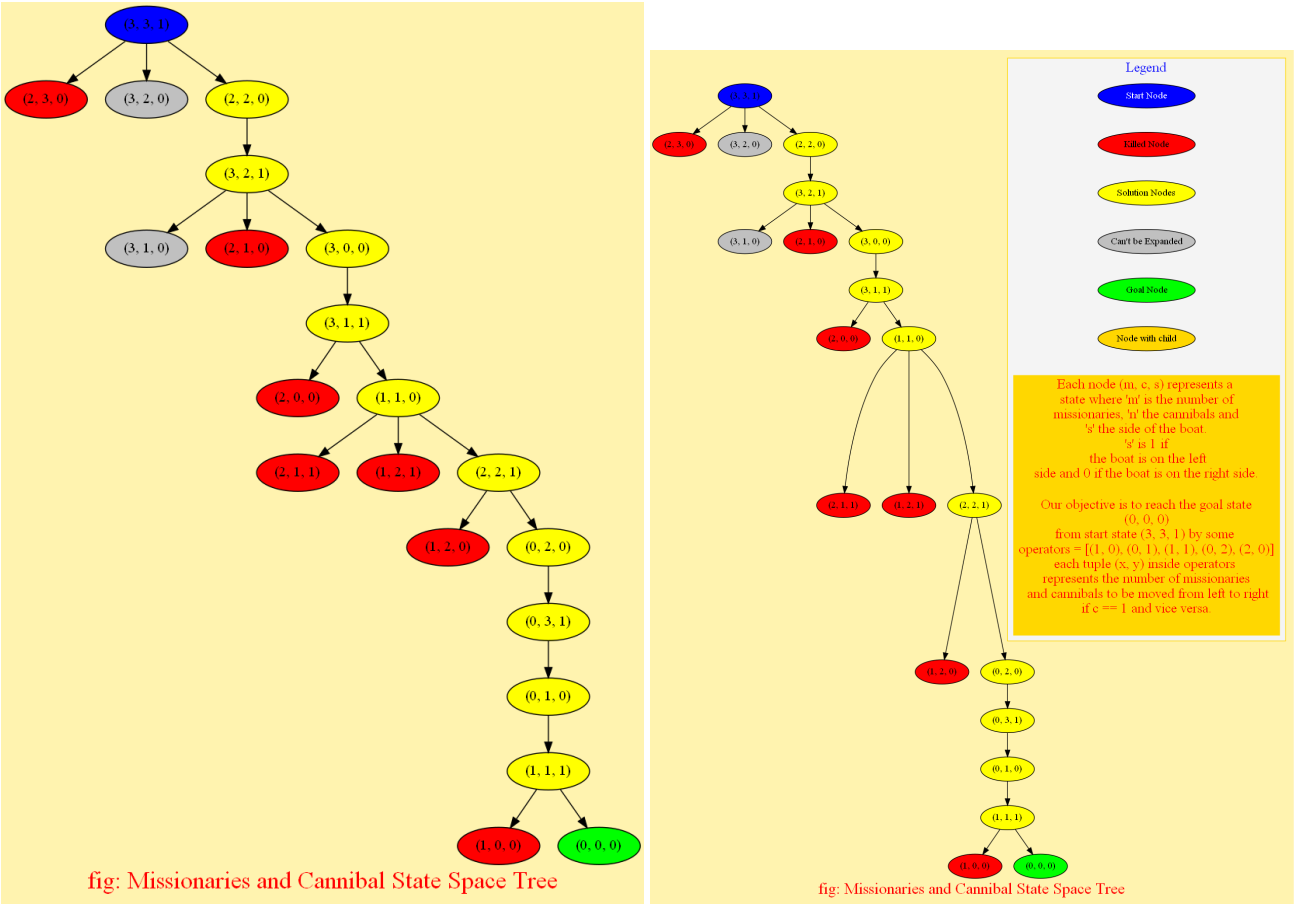
Hình 6: Màn hình console cho thuật toán DFS và không chú thích đồ thị



Hình 7: Màn hình console cho thuật toán DFS và có chú thích đồ thị

4.3 Giải bài toán với BFS

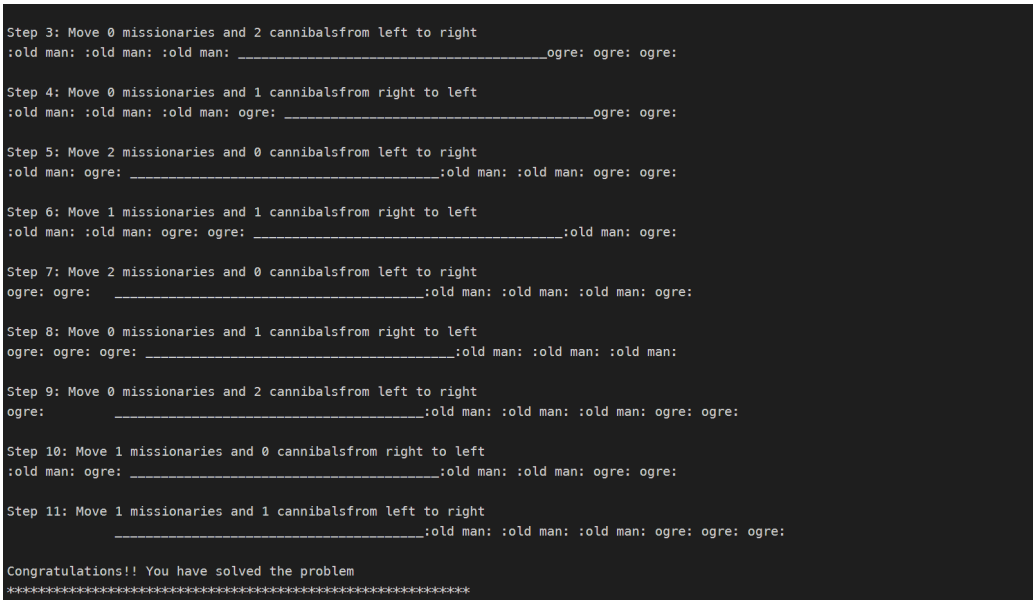
Thực thi lệnh `python main.py -m bfs`, ta thu được kết quả trên màn hình comsole ở hình 9 và file `bfs.png` như hình 11a. Do thuật toán BFS ưu tiên duyệt trạng thái theo chiều rộng, cho nên ta có



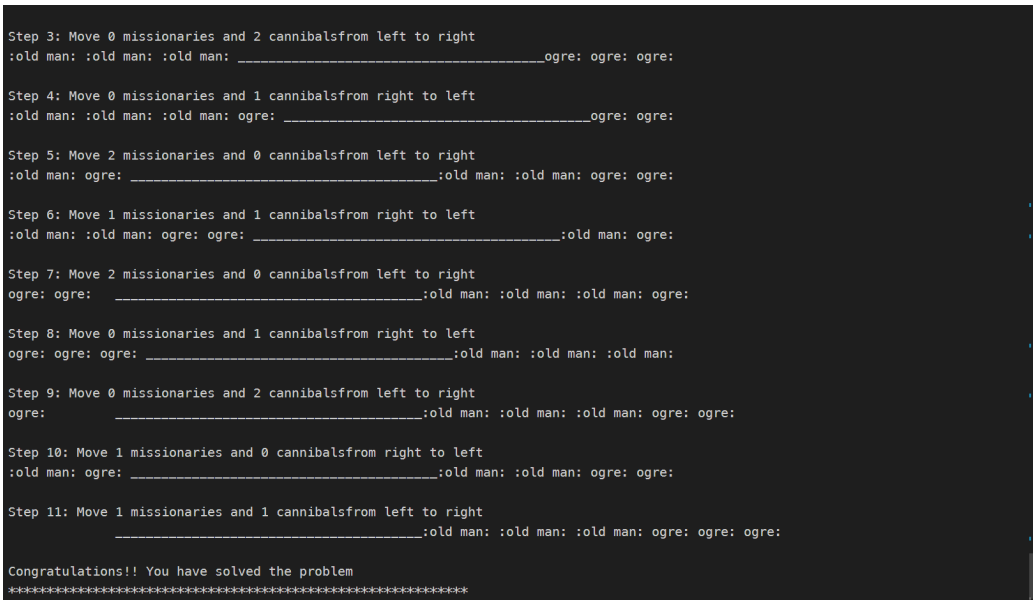
(a) Không chú thích đồ thị (b) Có chú thích đồ thị

Hình 8: Cây tìm kiếm trạng thái bằng thuật toán DFS

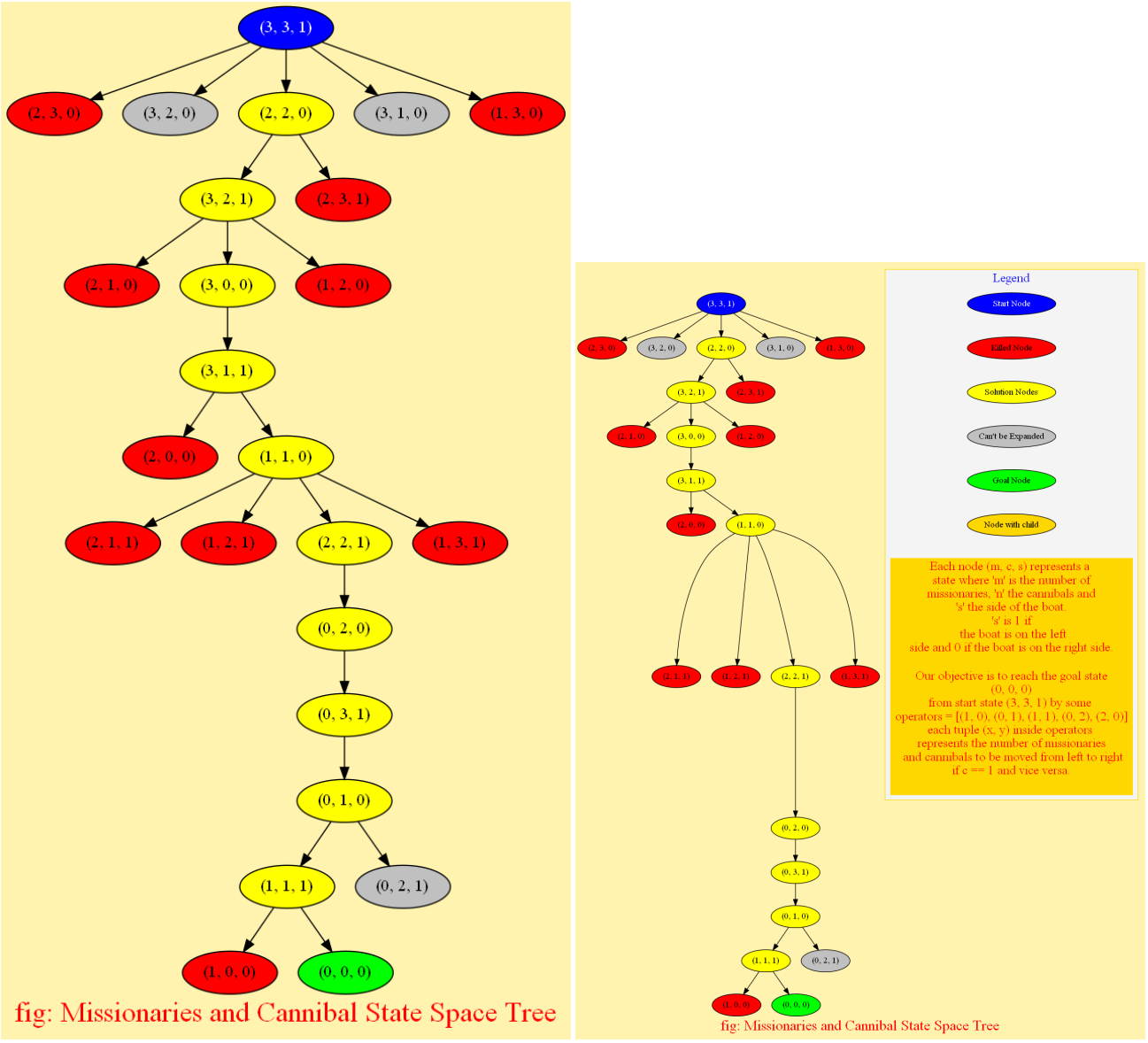
thể thấy các node được mở rộng ra theo nhiều hướng hơn so với thuật toán DFS.
Để chú thích cây tìm kiếm, ta thực thi lệnh `python main.py -m dfs -l True`, ta thu được kết quả trên màn hình comsole ở hình 10 và file `bfs_legend.png` như hình 11b.



Hình 9: Màn hình console cho thuật toán BFS và không chú thích đồ thị



Hình 10: Màn hình console cho thuật toán BFS và có chú thích đồ thị



(a) Không chú thích đồ thị (b) Có chú thích đồ thị

Hình 11: Cây tìm kiếm trạng thái bằng thuật toán BFS

4.4 Nhận xét

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

5 Tham khảo

1. Thư viện `graphviz`: https://www.graphviz.org/
2. Thư viện `pydot`: https://github.com/pydot/pydot
3. Thư viện `pyparsing`: https://github.com/pyparsing/pyparsing/
4. Thư viện `emoji`: https://github.com/carpedm20/emoji/
5. Stuart J. Russell, Peter Norvig (2010), Aritificial Intelligence: A Modern Approach, 3rd Edition, Prentice Hall, ISBN-13: 978-0-13-604259-4, ISBN-10: 0-13-604259-7 (https://zoo.cs.yale.edu/classes/cs470/materials/aima2010.pdf)