BÀI 9: THỰC NGHIỆM BIỂU DIỄN KHÔNG GIAN TRẠNG THÁI

Yêu cầu:

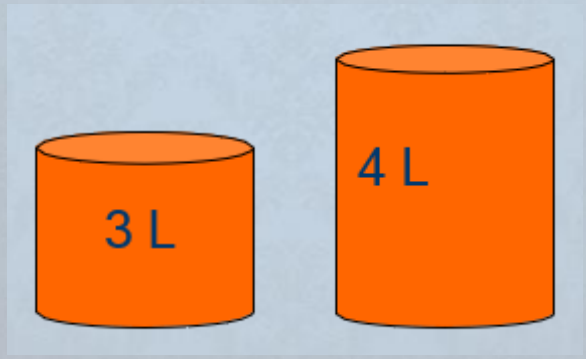
Bài 1: Biểu diễn không gian trạng thái bài toán đong nước

Bài 2: Biểu diễn không gian trạng thái bài toán người lái đò

Bài 1: Bài toán đong nước

Cho hai bình đong nước, một bình có dung tích 4 lít và 1 bình khác có dung tích 3 lít, cả 2 bình không có dấu dung tích.

- Trạng thái ban đầu của 2 bình là rỗng, dùng 1 bơm nước để lấy nước vào hai bình.
- Làm cách nào để có chính xác 2 lít nước trong bình 4 lít ?



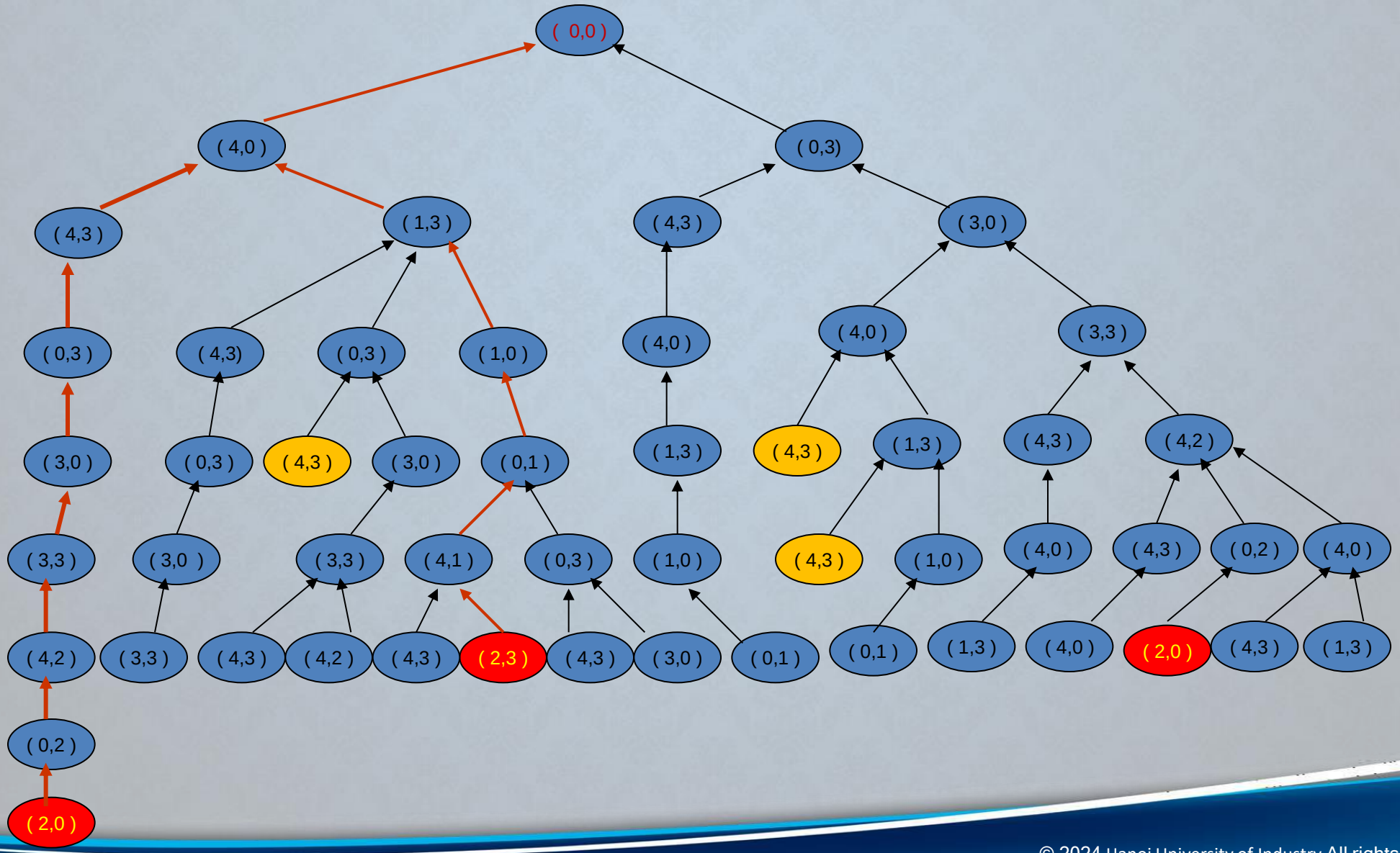
Bài 1: Bài toán đong nước

- Không gian trạng thái của bài toán đong nước được mô hình hóa như sau:
  - Trạng thái bài toán: nước mô tả bằng tập của các cặp số nguyên (x,y), trong đó:
    - x = 0, 1, 2, 3, hoặc 4 biểu diễn số lít nước trong bình 4 lít,
    - y = 0, 1, 2, hoặc 3 biểu diễn số lít nước trong bình 3 lít.
  - Trạng thái ban đầu: hai bình rỗng với cặp số nguyên là (0, 0).
  - Trạng thái đích: Cần chính xác 2 lít nước trong bình 4 lít (2,n) trong đó n là số không xác

Bài 1: Bài toán đong nước

➢Trạng thái chuyển tiếp : thông qua tập các luật If – Then mô tả tổng quát cách giải bài toán bình đong nước được thiết lập là:

- Luật 1 : $(x,y/\ x < 4) \rightarrow (4,y)$.
- Luật 2 : $(x,y/\ y < 3) \rightarrow (x,3)$.
- Luật 3 : $(x,y/\ x > 0) \rightarrow (0,y)$.
- Luật 4 : $(x,y/\ y > 0) \rightarrow (x,0)$.
- Luật 5 : $(x,y/\ x + y >= 4$ và $y > 0) \rightarrow (4,y – (4 – x))$.
- Luật 6 : $(x,y/\ x + y >= 3$ và $x > 0) \rightarrow (x – (3 –y),3)$.
- Luật 7 : $(x,y/\ x + y < 4$ và $y > 0) \rightarrow (x + y,0)$.
- Luật 8 : $(x,y/x + y < 3$ và $x > 0) \rightarrow (0,x + y)$



Bài 2: Bài toán người lái đò

Tại bến sông nọ có bắp cải, sói và dê muốn bác lái đò chở qua sông. Biết rằng tại một thời điểm thuyền của bác lái đò chỉ chở tối đa được 2 khách. Nếu sói và dê đứng riêng với nhau (không có mặt bác lái đò và bắp cải) thì sói sẽ ăn thịt dê. Nếu dê và bắp cải đứng riêng với nhau (không có mặt bác lái đò và sói) thì dê sẽ ăn bắp cải. Làm thế nào để có thể chở tất cả sang bên kia sông?

Bài 2: Bài toán người lái đò

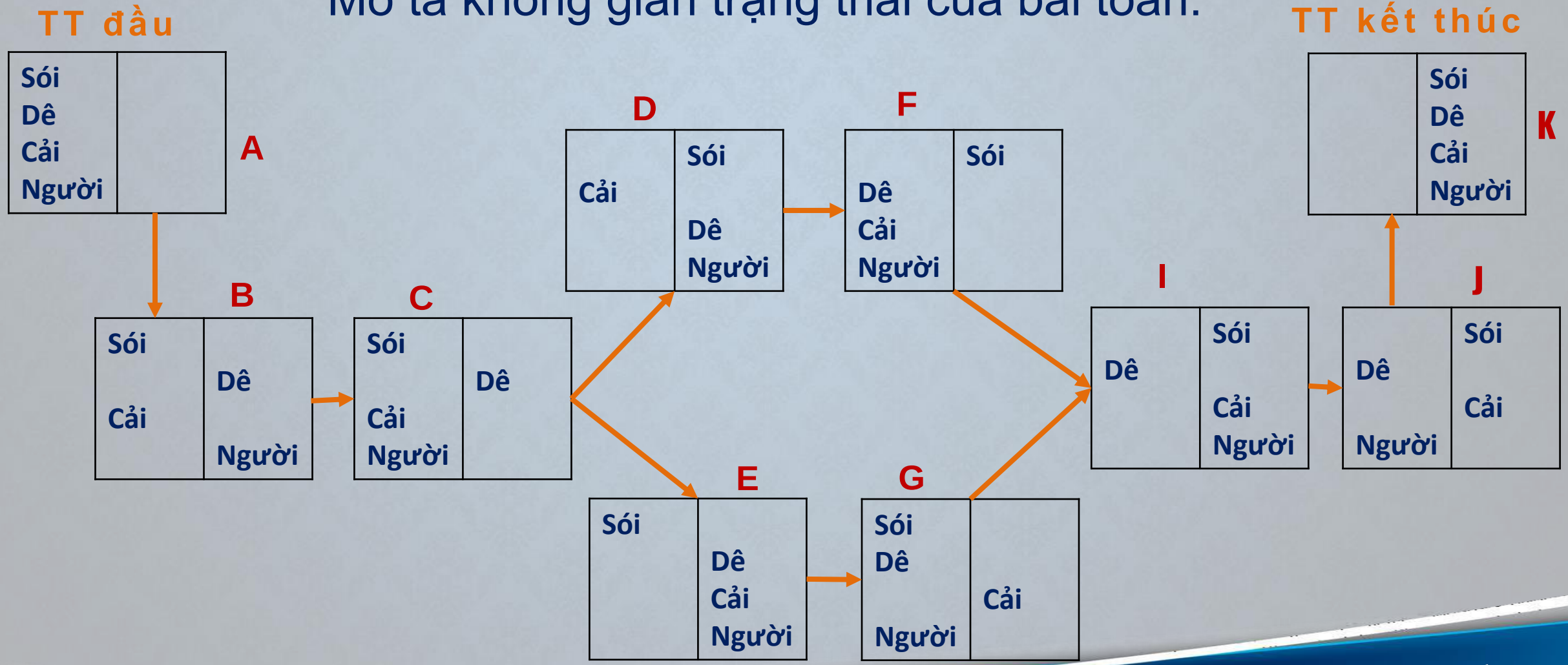
Mô tả trạng thái theo thứ tự sau (Sói, Dê, Cải, Người)

Trạng thái bắt đầu: (0, 0, 0, 0) cả sói, dê, cải và người đang ở bờ A.

Trạng thái kết thúc: (1, 1, 1, 1) cả sói, dê, cải và người đã sang bờ B.

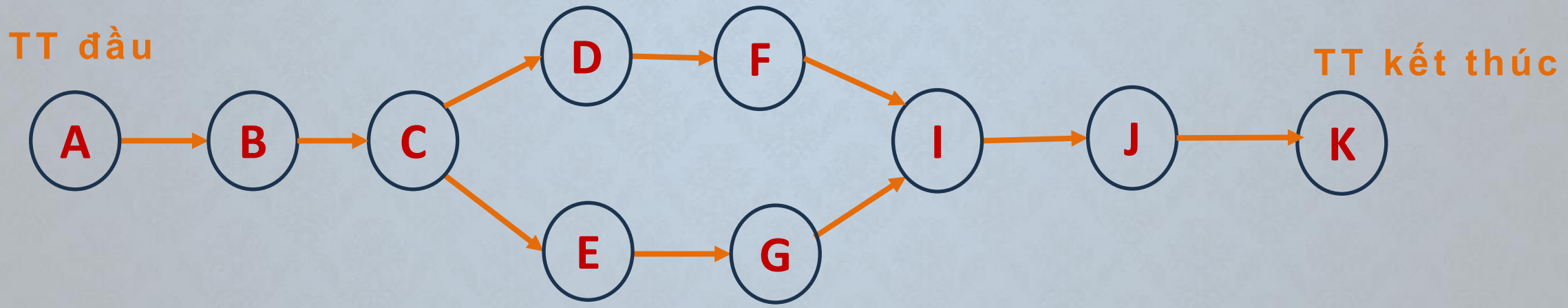
Bài 2: Bài toán người lái đò

Mô tả không gian trạng thái của bài toán:



Bài 2: Bài toán người lái đò

Không gian trạng thái của bài toán được mô tả dạng đồ thị:



Bài 2: Bài toán người lái đò

Định nghĩa vị trí, trạng thái và hàm kiểm tra trạng thái hợp lệ

```python
from enum import Enum
from collections import namedtuple

# Định nghĩa các vị trí
Location = Enum('Location', ['A', 'B'])

# Định nghĩa trạng thái
State = namedtuple('State', ['man', 'cabbage', 'goat', 'wolf'])

# Kiểm tra trạng thái hợp lệ
def is_valid(state):
    goat_eats_cabbage = (state.goat == state.cabbage and state.man != state.goat)
    wolf_eats_goat = (state.wolf == state.goat and state.man != state.wolf)
    return not (goat_eats_cabbage or wolf_eats_goat)
```

Bài 2: Bài toán người lái đò

```
# Tìm kiếm theo chiều sâu
def depth_first_search(start, is_goal, get_neighbors):
    parent = dict() # tạo một từ điển để lưu trữ đỉnh cha của mỗi đỉnh
    to_visit = [start] # khởi tạo ngăn xếp stack
    discovered = set([start]) # tạo một tập thể để lưu trữ các đỉnh đã được khám phá

    while to_visit:
        vertex = to_visit.pop() # lấy ra đỉnh của ngăn xếp
        if is_goal(vertex): # nếu là đỉnh đích
            path = [] # khởi tạo một danh sách để lưu trữ đường đi
            while vertex is not None:# cho qua các đỉnh
                path.insert(0, vertex) # thêm đỉnh hiện tại vào danh sách đường đi
                vertex = parent.get(vertex) # cập nhập đỉnh hiện tại thành đỉnh cha của nó
            return path # trả về đường đi của đỉnh đỉnh bắt đầu đến đích

        for neighbor in get_neighbors(vertex): # lặp qua các đỉnh con đỉnh hiện tại
            if neighbor not in discovered and is_valid(neighbor): # nêu đỉnh chưa xét thì hợp lệ
                discovered.add(neighbor) # thêm đỉnh vào tập hợp đỉnh đã xét
                parent[neighbor] = vertex # đặt hiện tại làm đỉnh cha
                to_visit.append(neighbor) # thêm đỉnh con vào ngăn xếp
```

Bài 2: Bài toán người lái đò

```
# Định nghĩa trạng thái ban đầu và trạng thái mục tiêu
start_state = State(man=Location.A, cabbage=Location.A, goat=Location.A, wolf=Location.A)
goal_state = State(man=Location.B, cabbage=Location.B, goat=Location.B, wolf=Location.B)

# Định nghĩa hàm lấy các trạng thái kế tiếp
def get_neighbors(state):
    neighbors = []
    for obj in ['man', 'cabbage', 'goat', 'wolf']:
        if getattr(state, obj) == state.man:  # Nếu đối tượng đang ở cùng vị trí với người đàn ông
            new_location = Location.A if state.man == Location.B else Location.B # xác định vị trí người đàn ông và đối tượng
            new_state = State(**{k: new_location if k == obj or k == 'man' else v for k, v in state._asdict().items()})
            #Tạo một trạng thái mới với người đàn ông và đối tượng ở vị trí mới, còn lại giữ nguyên.
            neighbors.append(new_state)
    return neighbors
```

Bài 2: Bài toán người lái đò

```
# Tìm đường đi
path = depth_first_search(start=start_state, is_goal=goal_state.__eq__, get_neighbors=get_neighbors)

# In đường đi
for state in path:
    print(state)
```

Bài 2: Bài toán người lái đò

Kết quả

```
State(man=<Location.A: 1>, cabbage=<Location.A: 1>, goat=<Location.A: 1>, wolf=<Location.A: 1>)
State(man=<Location.B: 2>, cabbage=<Location.A: 1>, goat=<Location.B: 2>, wolf=<Location.A: 1>)
State(man=<Location.A: 1>, cabbage=<Location.A: 1>, goat=<Location.B: 2>, wolf=<Location.A: 1>)
State(man=<Location.B: 2>, cabbage=<Location.A: 1>, goat=<Location.B: 2>, wolf=<Location.B: 2>)
State(man=<Location.A: 1>, cabbage=<Location.A: 1>, goat=<Location.A: 1>, wolf=<Location.B: 2>)
State(man=<Location.B: 2>, cabbage=<Location.B: 2>, goat=<Location.A: 1>, wolf=<Location.B: 2>)
State(man=<Location.A: 1>, cabbage=<Location.B: 2>, goat=<Location.A: 1>, wolf=<Location.B: 2>)
State(man=<Location.B: 2>, cabbage=<Location.B: 2>, goat=<Location.B: 2>, wolf=<Location.B: 2>)
```