

# BÁO CÁO BÀI THỰC HÀNH 2

## I. Yêu cầu:

Bài toán đưa 3 người và 3 quỷ từ bờ trái sang bờ phải của sông.

Trạng thái:  $[a,b,c]$

a, b: Số người và quỷ ở bờ trái. Có giá trị 1, 2 hoặc 3

c: Vị trí của thuyền, 1 khi ở bờ trái, 0 khi ở bờ phải

Mục tiêu: Đưa trạng thái từ  $[3,3,1]$  thành  $[0,0,0]$

Ràng buộc:

- Mỗi lần, thuyền chỉ có thể chở tối đa 2 đối tượng bất kì.
- Không được trong bất kì trường hợp nào, số lượng người nhỏ hơn số lượng quỷ, áp dụng cho cả bờ trái và bờ phải.
- Ta chỉ có thể đưa người và quỷ từ bờ trái sang bờ phải, không đưa ngược lại do hành động này là thừa thãi. Do đó, khi qua đến bờ trái, ta sẽ phải tốn một bước quay lại bờ phải mà không chở gì cả. (Lưu ý: Đây là ràng buộc do người làm đặt ra, đề bài không đề cập đến, tức là, với việc không đề cập đến ràng buộc này, ta có thể đưa người và quỷ từ bờ phải sang lại bờ trái. Một lần nữa, đây là hành động thừa thãi)

## II. Class Node:

### 1) Giải thích:

```
1 class Node:
2     goal_state = [0,0,0]
3     def __init__(self, state, parent, action, depth):
4         self.parent = parent
5         self.state = state
6         self.action = action
7         self.depth = depth
```

state: Trạng thái như đã đề cập

parent: Trạng thái sinh ra state

action: Phép chuyển parent thành state. Ví dụ, [3,3,1] thành [3,1,0], action sẽ là [0,-2,-1]

depth: Độ sâu của node trên cây tìm kiếm.

```
12 def goal_test(self):
13     if self.state == self.goal_state:
14         return True
15     return False
```

Hàm goal\_test dùng để kiểm tra xem node (trạng thái) đang xét có phải node kết thúc hay không, với trạng thái kết thúc được gán cố định goal\_state = [0,0,0]

```
17 def is_valid(self):
18     missionaries = self.state[0]
19     cannibals = self.state[1]
20     boat = self.state[2]
21     if missionaries < 0 or missionaries > 3:
22         return False
23     if cannibals < 0 or cannibals > 3:
24         return False
25     if boat < 0 or boat > 1:
26         return False
27     return True
```

Hàm is\_valid dùng để kiểm tra xem node đang xét có hợp lệ với bài toán không, tức với [a,b,c], 3 tham số phải có giá trị đúng với các giá trị đã nêu ở phần đầu file báo cáo.

Ví dụ trạng thái sai: [10,100,35], [-30,-60,-20], [4,3,1], [3,4,-1]

```
29 def is_killed(self):
30     missionaries = self.state[0]
31     cannibals = self.state[1]
32     if missionaries < cannibals and missionaries > 0:
33         return True
34     # Check for other side
35     if missionaries > cannibals and missionaries < 3:
36         return True
```

Hàm `is_killed` dùng để kiểm tra xem số lượng người có nhỏ hơn số lượng quỷ không ở cả hai bên bờ.

```
41 def generate_child(self):
42     children = []
43     depth = self.depth + 1
44     if self.state[2] == 0:
45         new_state = self.state.copy()
46         new_state[2] = new_state[2] + 1
47         action = [0,0,1]
48         new_node = Node(new_state, self, action, depth)
49         children.append(new_node)
50     if self.state[2] == 1:
51         for x in range(3):
52             for y in range(3):
53                 new_state = self.state.copy()
54                 new_state[0] = new_state[0] - x
55                 new_state[1] = new_state[1] - y
56                 new_state[2] = new_state[2] - 1
57                 action = [-x,-y,-1]
58                 new_node = Node(new_state, self, action, depth)
59                 if (new_node.is_valid() and not new_node.is_killed()):
60                     if x + y >= 1 and x + y <= 2:
61                         children.append(new_node)
62     return children
```

Hàm `generate_child` dùng để mở rộng node đang xét, trả về `children` là danh sách các node con mà thỏa điều kiện trong `if`. Nếu node đang xét là thuyền ở bờ phải, tức `self.state[2] == 0`, ta chỉ có một cách mở rộng duy nhất là thay `self.state[2] = 1`.

```

64     def find_solution(self):
65         path = []
66         solution = []
67         node = self
68         while node.parent != None:
69             path.append(node.state)
70             solution.append(node.action)
71             node = node.parent
72
73         # The while loop above will miss the last node, which is the starting node,
74         # we need to manually add its state and action into path and solution
75         path.append(node.state)
76         solution.append(node.action)
77
78         ## The last node, which is the starting node, whose action value is None,
79         ## the following code is to remove the last value from list.
80         ## However, it is okay not to remove it
81         #solution = solution[:-1]
82
83         path.reverse()
84         solution.reverse()
85         return path, solution

```

Hàm `find_solution` dùng để trả về danh sách `path` và `solution`. Trong đó, `path` chứa các trạng thái từ đầu đến kết thúc, `solution` chứa các action từ đầu đến kết thúc.

## 2) Class Node đầy đủ

```
1 class Node:
2     goal_state = [0,0,0]
3     def __init__(self, state, parent, action, depth):
4         self.parent = parent
5         self.state = state
6         self.action = action
7         self.depth = depth
8
9     def __str__(self):
10        return str(self.state)
11
12    def goal_test(self):
13        if self.state == self.goal_state:
14            return True
15        return False
16
17    def is_valid(self):
18        missionaries = self.state[0]
19        cannibals = self.state[1]
20        boat = self.state[2]
21        if missionaries < 0 or missionaries > 3:
22            return False
23        if cannibals < 0 or cannibals > 3:
24            return False
25        if boat < 0 or boat > 1:
26            return False
27        return True
28
29    def is_killed(self):
30        missionaries = self.state[0]
31        cannibals = self.state[1]
32        if missionaries < cannibals and missionaries > 0:
33            return True
34        # Check for other side
35        if missionaries > cannibals and missionaries < 3:
36            return True
37
38    def generate_child(self):
39        children = []
40        depth = self.depth + 1
41        if self.state[2] == 0:
42            new_state = self.state.copy()
43            new_state[2] = new_state[2] + 1
44            action = [0,0,1]
45            new_node = Node(new_state, self, action, depth)
46            children.append(new_node)
```

```

47         if self.state[2] == 1:
48             for x in range(3):
49                 for y in range(3):
50                     new_state = self.state.copy()
51                     new_state[0] = new_state[0] - x
52                     new_state[1] = new_state[1] - y
53                     new_state[2] = new_state[2] - 1
54                     action = [-x, -y, -1]
55                     new_node = Node(new_state, self, action, depth)
56                     if (new_node.is_valid() and not new_node.is_killed()):
57                         if x + y >= 1 and x + y <= 2:
58                             children.append(new_node)
59
60         return children
61
62     def find_solution(self):
63         path = []
64         solution = []
65         node = self
66         while node.parent != None:
67             path.append(node.state)
68             solution.append(node.action)
69             node = node.parent
70
71         # The while loop above will miss the last node, which is the starting node,
72         # we need to manually add its state and action into path and solution
73         path.append(node.state)
74         solution.append(node.action)
75
76         ## The last node, which is the starting node, whose action value is None,
77         ## the following code is to remove the last value from list.
78         ## However, it is okay not to remove it
79         #solution = solution[:-1]
80
81         path.reverse()
82         solution.reverse()
83         return path, solution

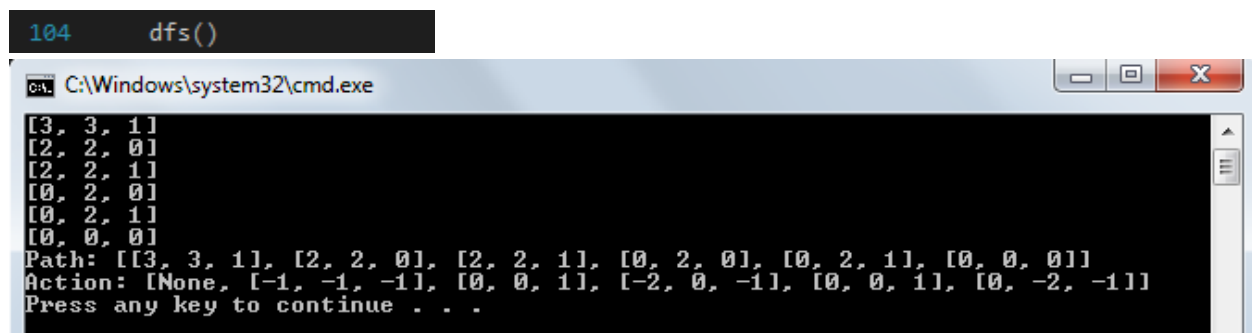
```

### III. Thuật toán tìm kiếm:

```
87     from queue import LifoQueue
88     def dfs():
89         start = Node([3,3,1], None, None, 0)
90         frontier = LifoQueue()
91         frontier.put(start)
92         while True:
93             if frontier.empty():
94                 raise Exception("No solution found")
95
96             current_node = frontier.get()
97             print(current_node)
98             if (current_node.goal_test()):
99                 print("Path:", current_node.find_solution()[0])
100                print("Action:", current_node.find_solution()[1])
101                return
102             else:
103                 num = len(current_node.generate_child())
104                 for i in range(num):
105                     frontier.put(current_node.generate_child()[i])
```

Ở đây, người làm chọn thuật toán Depth-first Search. Phần tử đầu tiên trong hàng đợi frontier được gán trực tiếp là node có state là [3,3,1].

### IV. Chạy Code:



```
104     dfs()

C:\Windows\system32\cmd.exe
[3, 3, 1]
[2, 2, 0]
[2, 2, 1]
[0, 2, 0]
[0, 2, 1]
[0, 0, 0]
Path: [[3, 3, 1], [2, 2, 0], [2, 2, 1], [0, 2, 0], [0, 2, 1], [0, 0, 0]]
Action: [None, [-1, -1, -1], [0, 0, 1], [-2, 0, -1], [0, 0, 1], [0, -2, -1]]
Press any key to continue . . .
```

### V. Nhận xét:

Bài làm này không có tính linh động, chỉ có thể giải quyết duy nhất cho bài toán đưa 3 người và 3 quỷ qua sông. Không thể giải quyết bài toán với số lượng người và quỷ khác. Để giải quyết được, ta cần chỉnh sửa lại các hàm kiểm tra.