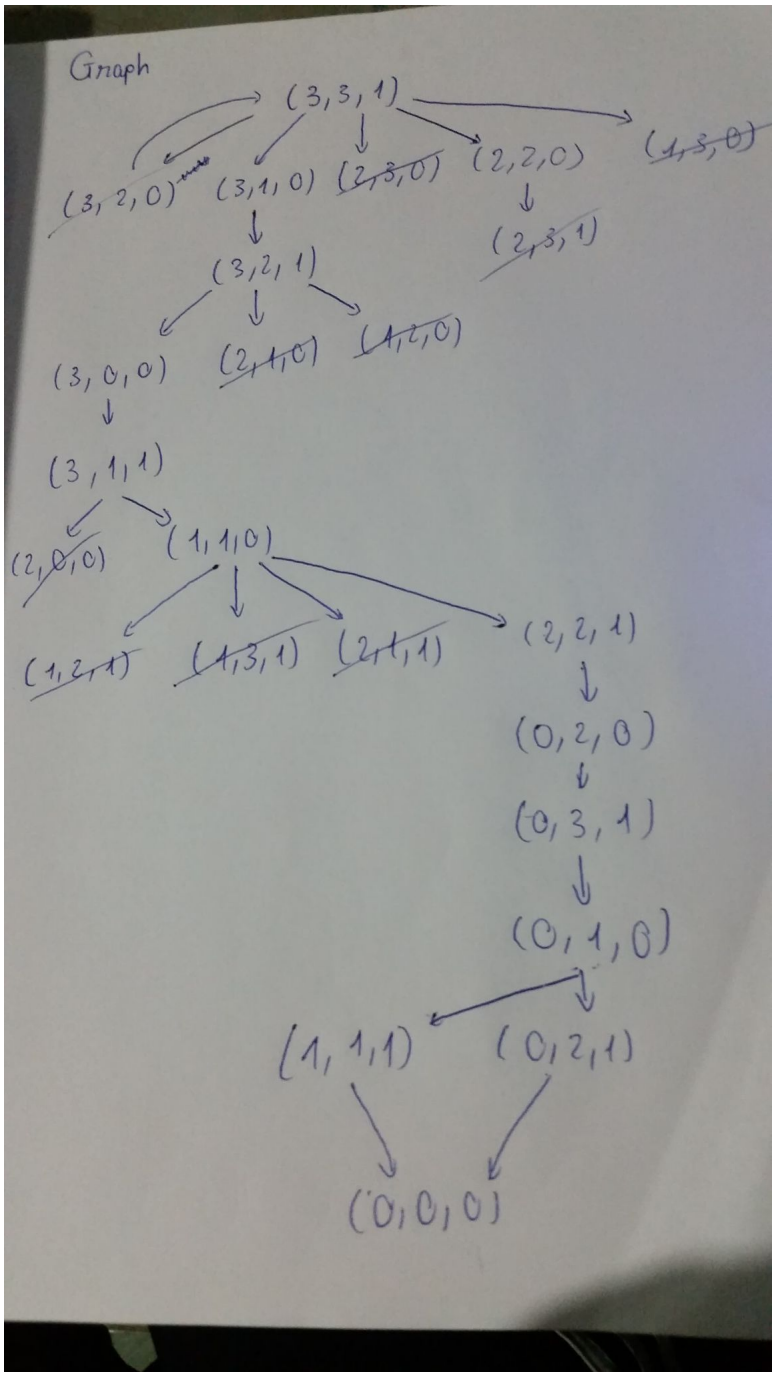
Report AI

Nguyễn Quốc Bảo

18110053

Tuan2

A. Vẽ sơ đồ liệt kê tất cả các khả năng có thể để tìm lời giải của bài toán (search tree)



B. Thuật toán tìm kiếm để đưa ra lời giải tối ưu cho bài toán

1. Xây dựng Node

```python
class Node :
    goal_state = [0,0,0]
    num_of_instances = 0
    def __init__(self,state,parent=None,action=None,depth=0):
        self.parent = parent
        self.state = state
        self.action = action
        self.depth = depth

        Node.num_of_instances +=1
    def __str__(self):
        return str(self.state)

    def goal_test(self):
        if self.state == self.goal_state:
            return True
        return False
```

Node bao gồm state,parent,action,depth. Lưu các trạng thái của từng node khi tạo graph. Và hàm goal-test để kiểm tra trạng thái có phải là điểm kết thúc chưa.

```
def is_valid(self):
    missionaries = self.state[0]
    cannibals = self.state[1]
    boat = self.state[2]
    if missionaries < 0 or missionaries > 3:
        return False
    if cannibals < 0 or cannibals >3 :
        return False
    if boat > 1 or boat < 0 :
        return False
    return True
     is_killed: is_killed
def is_killed(self):
    missionaries = self.state[0]
    cannibals = self.state[1]
    if missionaries < cannibals and missionaries > 0:
        return True
    # Check for the other side
    if missionaries > cannibals and missionaries < 3:
        return True
```

Hai hàm trên dùng để kiểm tra node có hợp lệ hay không và những node nào sẽ bị loại bỏ.

```
def generate_child(self):
    children = []
    depth = self.depth +1
    op = -1 #Subtract
    # boat_move = "from left shore to right"
    if self.state[2] == 0:
        op = 1  #Add
        # boat_move = "from right shore to left"
    for x in range(3):
        for y in range(3):
            #by_move ="Move %s missionaries and %s cannibals %s" % (x,y,boat_move)
            new_state = self.state.copy()
            new_state[0],new_state[1],new_state[2] = new_state[0]+ op*x , new_state[1]+ op*y,  new_state[2]+ op*1
            action = [x,y,op]
            new_node = Node(new_state,self,action,depth)
            if x+y >= 1 and x+y <= 2 :
                children.append(new_node)

    return children
```

Khi chạy qua các node hiện tại đang xét sẽ tạo ra các node con mà nó đi qua được.

```
def find_solution(self):
    solution = []
    solution.append(self.state)
    path = self
    while path.parent != None :
        path = path.parent
        solution.append(path.state)
    # solution = solution[:-1]
    solution.reverse()
    return solution
```

Khi tìm ra được điểm kết thúc ta sử dụng hàm find-solution để tìm ngược lại quảng đường đã đi được từ điểm bắt đầu.

2. Thuật toán BFS

```
def bfs(initial_state):
    start_node = Node(initial_state)
    if start_node.goal_test():
        return start_node.find_solution()

    frontier = [start_node]
    explored = []

    # print("The Starting node is \nDepth = %d" % start_node.depth)
    # print(str(start_node.state))

    while frontier:
        node = frontier.pop(0)
        # print("\nThe node selected to expand is\nDepth=" + str(node.depth) + "\n" + str(node.state) + "\n")
        explored.append(node.state)

        if not node.is_killed():
            children = node.generate_child()
            # print("The children nodes of this node are", end="")
            for child in children:
                if child.state not in explored:
                    # print("\nDepth=%d" % child.depth)
                    # print(str(child.state))
                    if child.goal_test():
                        return child.find_solution()
                    if child.is_valid():
                        frontier.append(child)
                        # explored.append(child.state)
    return 'No way solution'
```

Ta chuyền vào node bắt đầu và duyệt xem node đó có hợp lệ hay không, ta mở rộng node bằng các node con và xét từng node con có hợp lệ hay bị loại bỏ không, cho đến

khi tìm được điểm cuối cùng, ta sẽ duyệt ngược lại điểm ban đầu để ra được đường đi từ điểm đầu tới điểm cuối cùng.

3. Hàm main

```
def main():
    initial_state = [3, 3, 1]
    solution = bfs(initial_state)
    print("Solution: ")
    print(" --> ".join(str(i) for i in solution))
```

4. Kết quả

```
Solution:
[3, 3, 1] --> [3, 1, 0] --> [3, 2, 1] --> [3, 0, 0] --> [3, 1, 1] --> [1, 1, 0] --> [2, 2, 1] --> [0, 2, 0] --> [0, 3, 1] --> [0, 1, 0] --> [0, 2, 1] --> [0, 0, 0]
```