BÀI 2: CÁC THUẬT TOÁN TÌM KIẾM: BFS, DFS VÀ UCS(tiếp theo)

I. MỤC TIÊU:

Sau khi thực hành xong, sinh viên nắm được:

- Thuật toán BFS, DFS trên cây tìm kiếm.

- Cài đặt được các thuật toán này trên máy tính.

II. TÓM TẮT LÝ THUYẾT:

1. Cấu trúc dữ liệu của các node trong cây tìm kiếm:

- State: trạng thái trong không gian trạng thái.

- Node: chứa 1 trạng thái, con trỏ tới predecessor, độ sâu, và chi phí đường đi, hành động.

- Depth: số bước dọc theo đường đi từ trạng thái ban đầu.

- Path Cost: chi phí đường đi từ trạng thái ban đầu tới node.

- Fringe: bộ nhớ lưu trữ các node mở rộng. Ví dụ, s là stack hoặc hàng đợi.

2. Các hàm thực thi:

- Make-Node(state): khởi tạo 1 node từ 1 trạng thái (state).

- Goal-Test(state): trả về true nếu state là trạng thái kết thúc.

- Successor-Fn(state): thực thi các hàm successor (mở rộng một tập các node mới với tất cả các hành động có thể áp dụng trong trạng thái).

- Cost(state, action): trả về chi phí thực thi hành động trong trạng thái.

- Insert(node, fringe): thêm 1 node mới vào fringe.

- Remove-First(fringe): trả về node đầu tiên từ fringe.

3. Tìm kiếm trên cấu trúc cây:

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure

    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?(fringe) then return failure
        node ← REMOVE-FIRST(fringe)
        if GOAL-TEST[problem] applied to STATE[node] succeeds
            then return SOLUTION(node)
        fringe ← INSERT-ALL(EXPAND(node, problem), fringe)

function EXPAND(node, problem) returns a set of nodes

    successors ← the empty set
    for each ⟨action, result⟩ in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        STATE[s] ← result
        PARENT-NODE[s] ← node
Make-    ACTION[s] ← action
Node     PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

### 4. Thuật toán BFS:

5. Thuật toán DFS:

III. NỘI DUNG THỰC HÀNH:

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 $0 \leqslant a, b \leqslant 3$, 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).

2. Cây tìm kiếm(state_space_20.png):
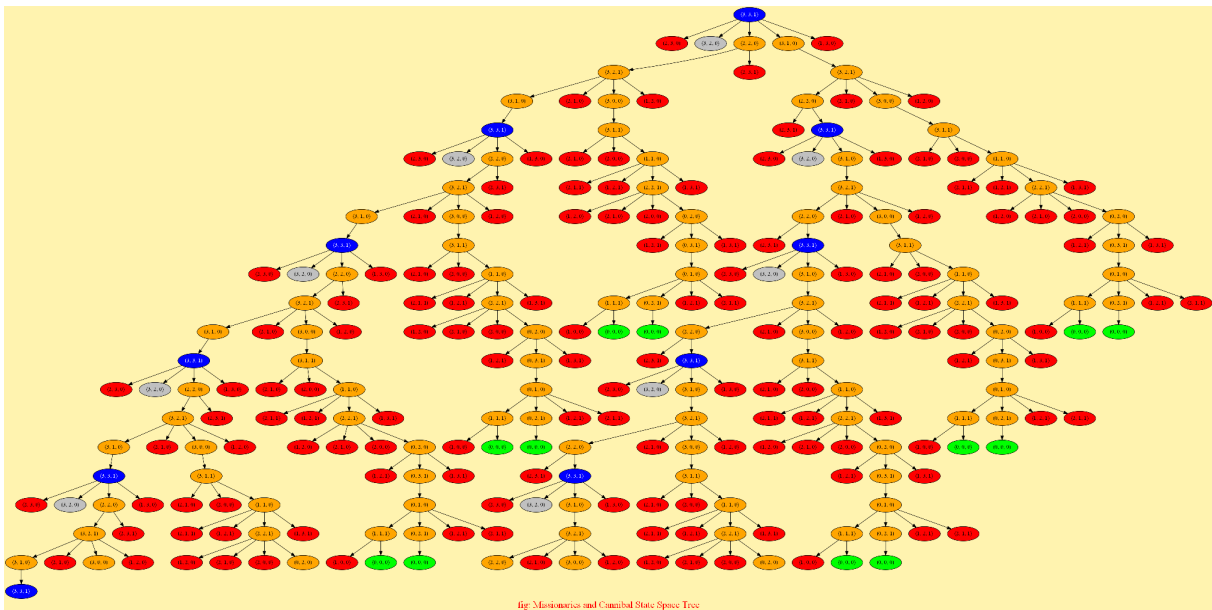
fig: Missionaries and Cannibal State Space Tree

3. Cài đặt:


```
      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()
     16    graph = pydot.Dot(graph_type='graph',strict=False, bgcolor="#fff3af",
     17                      label="fig: Missionaries and Cannibal State Space Tree",
     18                      fontcolor="red", fontsize="24", overlap="true")
     19    # To track node
     20    i = 0
     21
     22    arg = argparse.ArgumentParser()
     23    arg.add_argument("-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    max_depth = int(args.get("depth", 20))
     29
     30    def is_valid_move(number_missionaries, number_cannnibals):
     31        """
     32        Checks if number constraints are satisfied
     33        """
     34        return (0 <= number_missionaries <= 3) and (0 <= number_cannnibals <= 3)
     35
     36    def write_image(file_name="state_space"):
     37        try:
     38            graph.write_png(f"{file_name}_{max_depth}.png")
     39        except Exception as e:
     40            print("Error while writing file", e)
     41        print(f"File {file_name}_{max_depth}.png successfully written.")
     42
     43    def draw_edge(number_missionaries, number_cannnibals, side, depth_level, node_num):
     44        u, v = None, None
     45        if Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)] is not None:
     46            u = pydot.Node(str(Parent[(number_missionaries, number_cannnibals,side,depth_level, node_num)]),
     47                           label=str(Parent[(number_missionaries,number_cannnibals, side, depth_level, node_num)][:3]))
     48            graph.add_node(u)
```

```python
                    v = pydot.Node(str((number_missionaries, number_cannnibals, side, depth_level, node_num)),
                                   label=str((number_missionaries, number_cannnibals, side)))
                    graph.add_node(v)

                    edge = pydot.Edge(str(Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)]),
                                      str((number_missionaries, number_cannnibals, side, depth_level, node_num) ), dir='forward')
                    graph.add_edge(edge)
                else:
                    # For start node
                    v = pydot.Node(str((number_missionaries, number_cannnibals, side, depth_level, node_num)),
                                   label=str((number_missionaries, number_cannnibals, side)))
                    graph.add_node(v)
            return u, v


def is_start_state(number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == (3, 3, 1)


def is_goal_state(number_missionaries, number_cannnibals, side):
    return (number_missionaries, number_cannnibals, side) == (0, 0, 0)


def number_of_cannibals_exceeds(number_missionaries, number_cannnibals):
    number_missionaries_right = 3 - number_missionaries
    number_cannnibals_right = 3 - number_cannnibals
    return (number_missionaries > 0 and number_cannnibals > number_missionaries) \
 or (number_missionaries_right > 0 and number_cannnibals_right > number_missionaries_right)


def generate():
        global i
        q = deque()
        node_num = 0
        q.append((3, 3, 1, 0, node_num))

        Parent[(3, 3, 1, 0, node_num)] = None


        while q:

            number_missionaries, number_cannnibals, side, depth_level, node_num = q.popleft()
            # print(number_missionaries, number_cannnibals)
            # Draw Edge from u -> v
            # Where u = Parent[v]
            # and v = (number_missionaries, number_cannnibals, side, depth_level)
            u, v = draw_edge(number_missionaries, number_cannnibals, side, depth_level, node_num)

            if is_start_state(number_missionaries, number_cannnibals, side):
                v.set_fontcolor("white")
            elif is_goal_state(number_missionaries, number_cannnibals, side):
                v.set_style("filled")
                v.set_fillcolor("green")
                continue
                # return True
            elif number_of_cannibals_exceeds(number_missionaries, number_cannnibals):
                v.set_style("filled")
                v.set_fillcolor("red")
                continue
            else:
                v.set_style("filled")
                v.set_fillcolor("orange")

            if depth_level == max_depth:
                return True

            op = -1 if side == 1 else 1

            can_be_expanded = False

            # i = node_num
            for x, y in options:
                next_m, next_c, next_s = number_missionaries + op * x, number_cannnibals + op * y, int(not side)

                if Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)] is None or(next_m, next_c, next_s)\
                        != Parent[(number_missionaries, number_cannnibals, side, depth_level, node_num)][:3]:
                    if is_valid_move(next_m, next_c):
                        can_be_expanded = True
                        i += 1
                        q.append((next_m, next_c, next_s, depth_level + 1, i))
                        # Keep track of parent
                        Parent[(next_m, next_c, next_s, depth_level + 1, i)] =\
                        (number_missionaries, number_cannnibals, side, depth_level, node_num)
            if not can_be_expanded:
                v.set_style("filled")
                v.set_fillcolor("gray")
        return False


if __name__ == "__main__":
    if generate():
        write_image()
```

```python
      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()
     21
     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', bgcolor="#fff3af",
     36  │  │                              label="fig: Missionaries and Cannibal State Space Tree",fontcolor="red", fontsize="24")
     37  │  └─      self.visited = {}
     38  │  └─      self.solved = False
     39  │
     40  │  ┌─  def is_valid_move(self, number_missionaries, number_cannnibals):
     41  │  ┌─      """
     42  │  │       Checks if number constraints are satisfied
     43  │  │       """
     44  │  └─      return (0 <= number_missionaries <= 3) and (0 <= number_cannnibals <= 3)
     45  │
     46  │  ┌─  def is_goal_state(self, number_missionaries, number_cannnibals, side):
     47  │  └─      return (number_missionaries, number_cannnibals, side) == self.goal_state
     48  │
     49  │  ┌─  def is_start_state(self, number_missionaries, number_cannnibals, side):
     50  │  └─      return (number_missionaries, number_cannnibals, side) == self.start_state
     51  │
     52  │  ┌─  def number_of_cannibals_exceeds(self, number_missionaries, number_cannnibals):
     53  │  │       number_missionaries_right = 3 - number_missionaries
     54  │  │       number_cannnibals_right = 3 - number_cannnibals
     55  │  ┌─      return (number_missionaries > 0 and number_cannnibals > number_missionaries) \
     56  │  └─              or (number_missionaries_right > 0 and number_cannnibals_right > number_missionaries_right)
     57  │
     58  │  ┌─  def write_image(self, file_name="state_space.png"):
     59  │  ┌─      try:
     60  │  └─          self.graph.write_png(file_name)
     61  │  ┌─      except Exception as e:
     62  │  └─          print("Error while writing file", e)
     63  │  └─      print(f"File {file_name} successfully written.")
     64
     65  │  ┌─  def solve(self, solve_method="dfs"):
     66  │  │       self.visited = dict()
     67  │  │       Parent[self.start_state] = None
     68  │  │       Move[self.start_state] = None
     69  │  │       node_list[self.start_state] = None
     70  │  │
     71  │  └─      return self.dfs(*self.start_state, 0) if solve_method == "dfs" else self.bfs()
     72  │
     73  │  ┌─  def draw_legend(self):
     74  │  ┌─      """
     75  │  │           Utility method to draw legend on graph if  legend flag is ON
     76  │  └─      """
     77  │  ┌─      graphlegend = pydot.Cluster(graph_name="legend", label="Legend", fontsize="20", color="gold",
     78  │  └─                                  fontcolor="blue", style="filled", fillcolor="#f4f4f4")
     79
     80
     81  │       node1 = pydot.Node("1", style="filled", fillcolor="blue", label="Start Node", fontcolor="white", width="2", fixedsize="true")
     82  │       graphlegend.add_node(node1)
     83
     84  │       node2 = pydot.Node("2", style="filled", fillcolor="red", label="Killed Node", fontcolor="black", width="2", fixedsize="true")
     85  │       graphlegend.add_node(node2)
     86
     87  │       node3 = pydot.Node("3", style="filled", fillcolor="yellow", label="Solution nodes", width="2", fixedsize="true")
     88  │       graphlegend.add_node(node3)
     89
     90  │       node4 = pydot.Node("4", style="filled", fillcolor="gray", label="Can't be expanded",  width="2", fixedsize="true")
     91  │       graphlegend.add_node(node4)
     92
     93  │       node5 = pydot.Node("5", style="filled", fillcolor="green", label="Goal node", width="2", fixedsize="true")
```

```python
            graphlegend.add_node(node5)

            node7 = pydot.Node("7", style="filled", fillcolor="gold", label="Node with child", width="2", fixedsize="true")
            graphlegend.add_node(node7)


            description = "Each node (m, c, s) represents a \nstate where 'm' is the number of\n missionaries,\'n' the cannibals \
                    and \n's' the side of the boat\n"
            " where  '1' represents the left \nside and '0' the right side \n\nOur objective is to reach goal state (0, 0, 0)\
            \nfrom start state (3, 3, 1) by some \noperators = [(0, 1), (0, 2), (1, 0), (1, 1), (2, 0),]\n"\
                    "each tuples (x, y) inside operators \nrepresents the number of missionaries and\
            \ncannibals to be moved from left to right \nif c == 1 and viceversa"

            node6 = pydot.Node("6", style="filled", fillcolor="gold", label= description, shape="plaintext", fontsize="20", fontcolor="red")
            graphlegend.add_node(node6)

            self.graph.add_subgraph(graphlegend)

            self.graph.add_edge(pydot.Edge(node1, node2, style="invis"))
            self.graph.add_edge(pydot.Edge(node2, node3, style="invis"))
            self.graph.add_edge(pydot.Edge(node3, node4, style="invis"))
            self.graph.add_edge(pydot.Edge(node4, node5, style="invis"))
            self.graph.add_edge(pydot.Edge(node5, node7, style="invis"))
            self.graph.add_edge(pydot.Edge(node7, node6, style="invis"))

    def draw(self, *, number_missionaries_left, number_cannnibals_left, number_missionaries_right, number_cannnibals_right):
        """
            Draw state on console using emojis
        """
        left_m = emoji.emojize(f":old_man: " * number_missionaries_left)
        left_c = emoji.emojize(f":ogre: " * number_cannnibals_left)
        right_m = emoji.emojize(f":old_man: " * number_missionaries_right)
        right_c = emoji.emojize(f":ogre: " * number_cannnibals_right)

        print('{}{}{}{}{}'.format(left_m, left_c + " " * (14 - len(left_m) - len(left_c)), "_" * 40, " " * (12 - len(right_m) - len(right
        print("")

    def show_solution(self):
        # Recursively start from Goal State
        # And find parent until start state is reached

        state = self.goal_state
        path, steps, nodes = [] ,[], []
        while state is not None:
            path.append(state)
            steps.append(Move[state])
            nodes.append(node_list[state])

            state = Parent[state]

        steps, nodes = steps[::-1], nodes[::-1]

        number_missionaries_left, number_cannnibals_left = 3, 3
        number_missionaries_right, number_cannnibals_right = 0, 0

        print("*" * 60)
        self.draw(number_missionaries_left=number_missionaries_left, number_cannnibals_left=number_cannnibals_left,
                    number_missionaries_right=number_missionaries_right, number_cannnibals_right=number_cannnibals_right)

        for i, ((number_missionaries, number_cannnibals, side), node) in enumerate(zip(steps[1:], nodes[1:])):

            if node.get_label() != str(self.start_state):
                node.set_style("filled")
                node.set_fillcolor("yellow")

            print(f"Step {i + 1}: Move {number_missionaries} missionaries  and {number_cannnibals} \
                    cannibals from {self.boat_side[side]} to {self.boat_side[int(not side)]}.")

            op = -1 if side == 1 else 1

            number_missionaries_left = number_missionaries_left + op * number_missionaries
            number_cannnibals_left = number_cannnibals_left + op * number_cannnibals

            number_missionaries_right = number_missionaries_right - op * number_missionaries
            number_cannnibals_right = number_cannnibals_right - op * number_cannnibals

            self.draw(number_missionaries_left=number_missionaries_left, number_cannnibals_left=number_cannnibals_left,
                        number_missionaries_right=number_missionaries_right, number_cannnibals_right=number_cannnibals_right)

        print("Congratulations!!! you have solved the problem")
        print("*" * 60)

    def draw_edge(self, number_missionaries, number_cannnibals, side, depth_level):
        u, v = None, None
        if Parent[(number_missionaries, number_cannnibals, side)] is not None:
            u = pydot.Node(str(Parent[(number_missionaries, number_cannnibals, side)] + (depth_level - 1, )),
```

```python
                                 label=str(Parent[((number_missionaries, number_cannnibals, side))]))
                    self.graph.add_node(u)

                    v = pydot.Node(str((number_missionaries, number_cannnibals, side, depth_level)),
                                 label=str((number_missionaries, number_cannnibals, side)))
                    self.graph.add_node(v)

                    edge = pydot.Edge(str(Parent[(number_missionaries, number_cannnibals, side)] + (depth_level - 1, )),
                                 str((number_missionaries, number_cannnibals, side, depth_level) ), dir='forward')
                    self.graph.add_edge(edge)
            else:
                # For start node
                v = pydot.Node(str((number_missionaries, number_cannnibals, side, depth_level)),
                             label=str((number_missionaries, number_cannnibals, side)))
                self.graph.add_node(v)
            return u, v

    def bfs(self):
        q = deque()
        q.append(self.start_state + (0, ))
        self.visited[self.start_state] = True

        while q:
            number_missionaries, number_cannnibals, side, depth_level = q.popleft()
            # Draw Edge from u -> v
            # Where u = Parent[v]
            # and v = (number_missionaries, number_cannnibals, side, depth_level)
            u, v = self.draw_edge(number_missionaries, number_cannnibals, side, depth_level)


            if self.is_start_state(number_missionaries, number_cannnibals, side):
                v.set_style("filled")
                v.set_fillcolor("blue")
                v.set_fontcolor("white")
            elif self.is_goal_state(number_missionaries, number_cannnibals, side):
                v.set_style("filled")
                v.set_fillcolor("green")
                return True
            elif self.number_of_cannibals_exceeds(number_missionaries, number_cannnibals):
                v.set_style("filled")
                v.set_fillcolor("red")
                continue
            else:
                v.set_style("filled")
                v.set_fillcolor("orange")

            op = -1 if side == 1 else 1

            can_be_expanded = False

            for x, y in self.options:
                next_m, next_c, next_s = number_missionaries + op * x, number_cannnibals + op * y, int(not side)
                if (next_m, next_c, next_s) not in self.visited:
                    if self.is_valid_move(next_m, next_c):
                        can_be_expanded = True
                        self.visited[(next_m, next_c, next_s)] = True
                        q.append((next_m, next_c, next_s, depth_level + 1))

                        # Keep track of parent and corresponding move
                        Parent[(next_m, next_c, next_s)] = (number_missionaries, number_cannnibals, side)
                        Move[(next_m, next_c, next_s)] = (x, y, side)
                        node_list[(next_m, next_c, next_s)] = v

            if not can_be_expanded:
                v.set_style("filled")
                v.set_fillcolor("gray")
        return False

    def dfs(self, number_missionaries, number_cannnibals, side, depth_level):
        self.visited[(number_missionaries, number_cannnibals, side)] = True

        # Draw Edge from u -> v
        # Where u = Parent[v]
        u, v = self.draw_edge(number_missionaries, number_cannnibals, side, depth_level)


        if self.is_start_state(number_missionaries, number_cannnibals, side):
            v.set_style("filled")
            v.set_fillcolor("blue")
        elif self.is_goal_state(number_missionaries, number_cannnibals, side):
            v.set_style("filled")
            v.set_fillcolor("green")
            return True
        elif self.number_of_cannibals_exceeds(number_missionaries, number_cannnibals):
            v.set_style("filled")
            v.set_fillcolor("red")
            return False
        else:
```

```
269                    v.set_style("filled")
270                    v.set_fillcolor("orange")
271
272            solution_found = False
273            operation = -1 if side == 1 else 1
274
275            can_be_expanded = False
276
277            for x, y in self.options:
278                next_m, next_c, next_s = number_missionaries + operation * x, number_cannnibals + operation * y, int(not side)
279
280                if (next_m, next_c, next_s) not in self.visited:
281                    if self.is_valid_move(next_m, next_c):
282                        can_be_expanded = True
283                        # Keep track of Parent state and corresponding move
284                        Parent[(next_m, next_c, next_s)] = (number_missionaries, number_cannnibals, side)
285                        Move[(next_m, next_c, next_s)] = (x, y, side)
286                        node_list[(next_m, next_c, next_s)] = v
287
288                        solution_found = (solution_found or self.dfs(next_m, next_c, next_s, depth_level + 1))
289
290                        if solution_found:
291                            return True
292
293            if not can_be_expanded:
294                v.set_style("filled")
295                v.set_fillcolor("gray")
296
297            self.solved = solution_found
298            return solution_found
```

main.py ☒

```
 5      from solve import Solution
 6      import argparse
 7      import itertools
 8
 9      arg = argparse.ArgumentParser()
10      arg.add_argument("-m", "--method", required=False, help="Specify which method to use")
11      arg.add_argument("-l", "--legend", required=False, help="Specify if you want to display legend on graph")
12
13      args = vars(arg.parse_args())
14
15      solve_method = args.get("method", "bfs")
16      legend_flag = args.get("legend", False)
17
18
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")
42
43
44      if __name__ == "__main__":
45          main()
```

a. Cài đặt thư viện graphviz tải về từ link: `https://graphviz.org/download/`
và đặt đường dẫn tới thư mục bin của graphviz đã cài đặt trên máy tính.

```
generate_full_space_tree.py
10
11    # Set it to bin folder of graphviz
12    os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin'
```

```
solve.py
11
12    # Set it to bin folder of graphviz
13    os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin'
```

b. Cài đặt các yêu cầu trong file "requirements.txt":


```
requirements.txt - Notepad
File  Edit  Format  View  Help
emoji==0.5.4
pydot==1.4.1
pyparsing==2.4.5
```

pip install -r requirements.txt


```
C:\WINDOWS\system32\cmd.exe
C:\Users\Huynh>cd C:\Users\Huynh\Desktop\Tuan2

C:\Users\Huynh\Desktop\Tuan2>pip install -r requirements.txt
```

c. Khởi tạo cây không gian trạng thái:

python generate_full_space_tree.py -d 8 (với d là độ sâu (depth=8))


```
C:\Users\Huynh\Desktop\Tuan2>python generate_full_space_tree.py -d 8
File state_space_8.png successfully written.
```

python generate_full_space_tree.py -d 20 (depth = 20)


```
C:\Users\Huynh\Desktop\Tuan2>python generate_full_space_tree.py -d 20
File state_space_20.png successfully written.
```

Làm tương tự với depth = 10, depth = 40, . . .

d. Cây DFS:

- DFS:

python main.py -m dfs

```
C:\Users\Huynh\Desktop\Tuan2>python main.py -m dfs
**********************************************************
▯  ▯  ▯  ▯  ▯  ▯          _____

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 dfs.png successfully written.
```
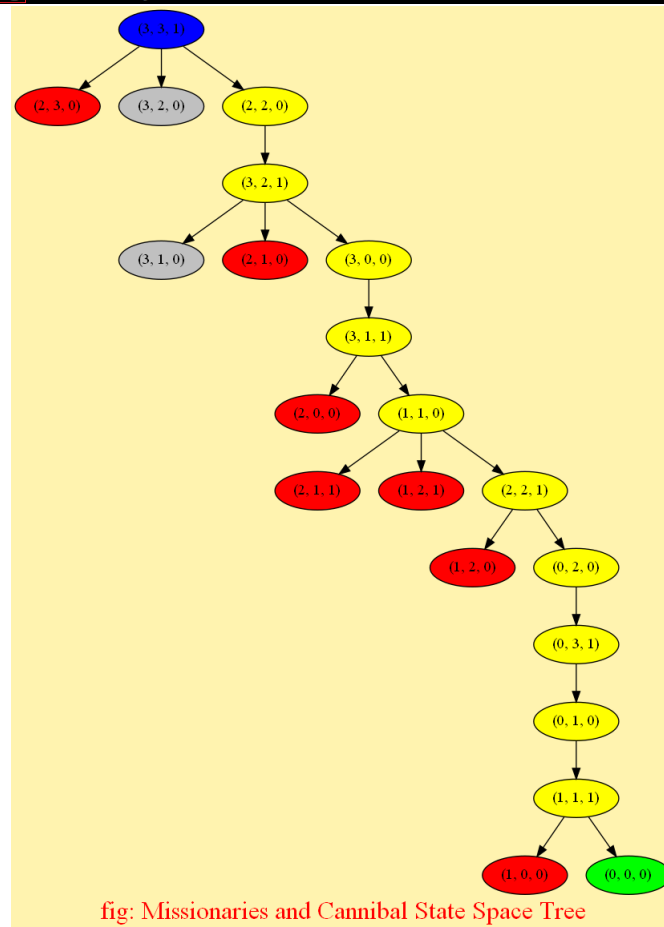

fig: Missionaries and Cannibal State Space Tree

11
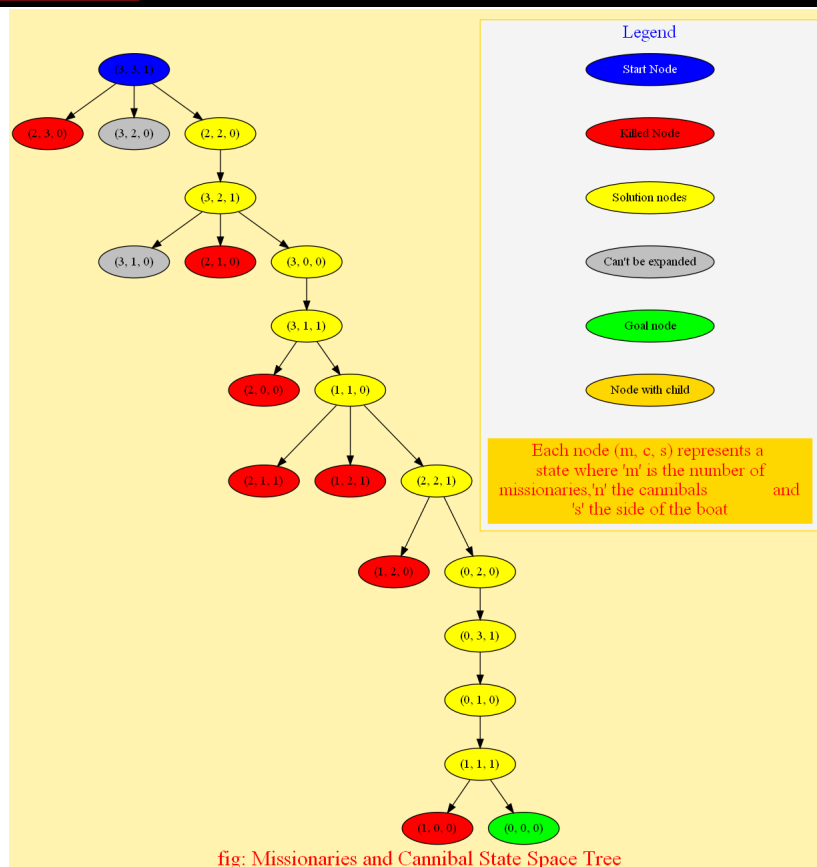
- DFS với legend:

python main.py -m dfs -l True

```
C:\Users\Huynh\Desktop\Tuan2>python main.py -m dfs -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 dfs_legend.png successfully written.
```


fig: Missionaries and Cannibal State Space Tree

e. Cây BFS:

- BFS:
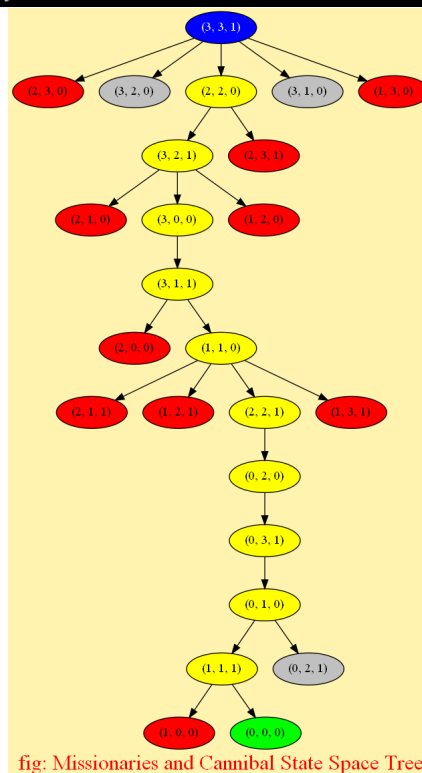
python main.py -m bfs

fig: Missionaries and Cannibal State Space Tree

- BFS với legend:

  python main.py -m bfs -l True


```
C:\Users\Huynh\Desktop\Tuan2>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.
```


fig: Missionaries and Cannibal State Space Tree
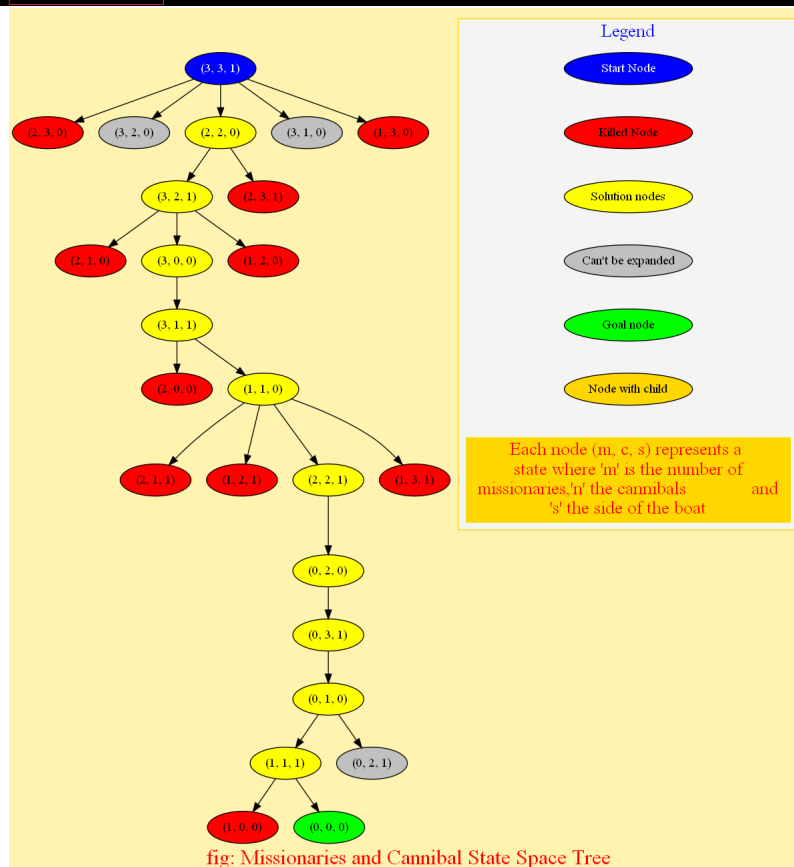
4. **Yêu cầu:**

   - Cài đặt và thực thi chương trình.

   - Viết báo cáo trình bày:

      ❀ Nếu chương trình bị báo lỗi thì lỗi ở dòng nào và sửa lại như thế nào? (Nếu có).

      ❀ Đọc hiểu code đã cho và trình bày lại chi tiết hơn đoạn code cho sẵn dùng để làm gì?