COMS 572: Homework #3

September 14, 2018 by 17:00pm

Professor Jin Tian

Le Zhang

Problem 1

(30 pts.) The **missionaries and cannibals** problem is usually stated as follows. Three mission- aries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of mis- sionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

a. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.

Answer: If we use four integers, each from 0 to 3, to represent the number of missionaries and number of cannibals on both sides of the river, we will be able to use four-integer tuples to represent the states. For example, state [2,2,1,1] means there are 2 missionaries and 2 cannibals are on the left side, and 1 missionary and 1 cannibal are on the right side. The state of number of people on boat is not considered as a space state because cannibals can never outnumber missionaries on a boat.

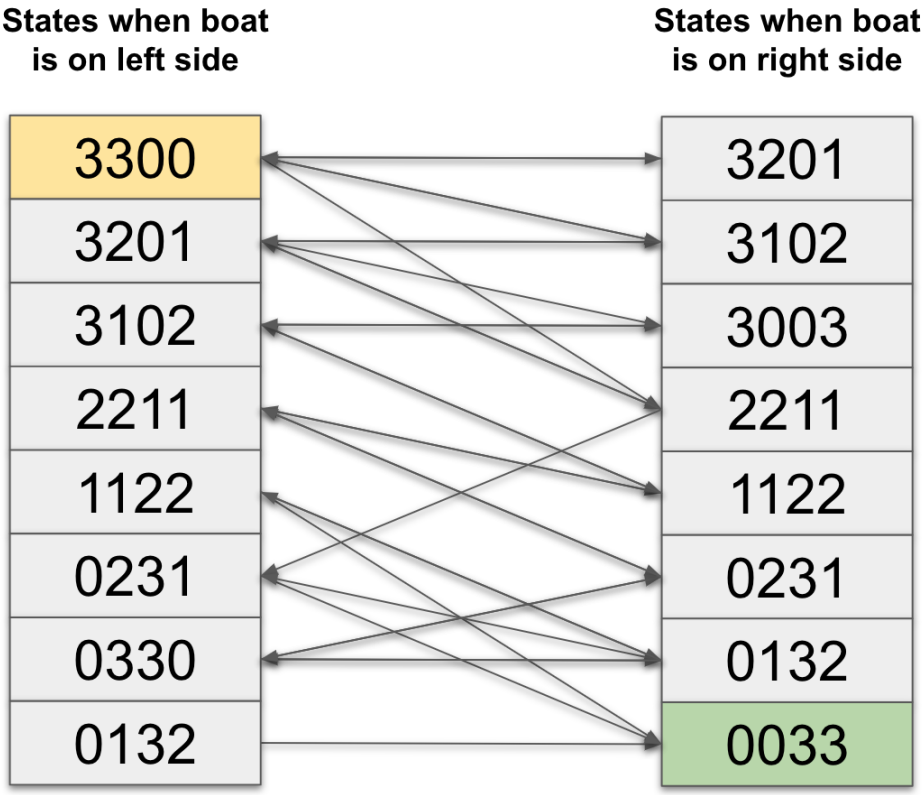
(a) **Initial State**: All people are on the left side, so we get [3,3,0,0].

(b) **States**: Any combinations of numbers. When missionaries are outnumbered by cannibals on any side, it is a bad state.

(c) **Actions**: Move up to 2 people from one side to the other. We have two directions here, from left to right, and from right to left depending on the position of the boat. We cannot move along one direction twice continuously. For instance, [-2,0,2,0] means that we move 2 missionaries from left side to the right. And the next move should be from right to left.

(d) **Goal State**: All the people are moved to the other side of the river, which is state [0,0,3,3].

A diagram of complete state space is shown below:



A table of all possible moves and states is shown below:

Moves\Valid States	3300	2211	1122	3201	3102	3003	0132	0231	0330
L to R									
0,-1,0,1	3201	2112	1023	3102	3003		0033	0132	0231
-1,0,1,0	2310	1221	0132	2211	2112	2013			
-1,-1,1,1	2211	1122	0033	2112	2013				
0,-2,0,2	3102	2013		3003				0033	0132
-2,0,2,0	1320	0231		1221	1122	1023			
R to L									
0,1,0,-1		2310	1221	3300	3201	3102	0231	0330	
1,0,-1,0		3201	2112				1122	1221	1320
1,1,-1,-1		3300	2211				1221	1320	
0,2,0,-2			1320		3300	3201	0330		
2,0,-2,0			3102				2112	2211	2310
Initial State									
Bad State									
Good State									
Termination State									

b. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

Answer:

The search algorithm I choose to use is BFS. First, we start from the initial state [3,3,0,0] with boat on the left side. Then, we apply all possible moves from left to right and check if the state after the action is valid. If it is valid, store that state in a list of states for next step. After we go through all the current states, we change the boat position and update the current state list with the list of next states. Then we apply the right to left actions to find out the next possible valid states. Repeat this process until we reach the goal state [0,0,3,3].

It is critical to check for repeated states, because you dont want to get stuck in a loop and waste computing time and memory on those visited paths. It is even more important if we apply DFS because a loop can lead to infinite computing time.

Down below is a python program I wrote to solve this problem. The source code is attached to the submission. And the solution given by my algorithm is shown below.

Solutions:

```
0. init state [3, 3, 0, 0]
1. action [0, -2, 0, 2] , state [3, 1, 0, 2]
2. action [0, 1, 0, -1] , state [3, 2, 0, 1]
3. action [0, -2, 0, 2] , state [3, 0, 0, 3]
4. action [0, 1, 0, -1] , state [3, 1, 0, 2]
5. action [-2, 0, 2, 0] , state [1, 1, 2, 2]
6. action [1, 1, -1, -1] , state [2, 2, 1, 1]
7. action [-2, 0, 2, 0] , state [0, 2, 3, 1]
8. action [0, 1, 0, -1] , state [0, 3, 3, 0]
9. action [0, -2, 0, 2] , state [0, 1, 3, 2]
10. action [0, 1, 0, -1] , state [0, 2, 3, 1]
11. action [0, -2, 0, 2] , state [0, 0, 3, 3]
```

Problem 1 continued on next page...

Python source code:

```
# This is a problem solver written by Le Zhang, September, 2018
# It solves the Missionaries and Cannibals problem
# The states are in this format:
#   [num_missionaries_left, num_cannibals_left,
#                   num_missionaries_right, num_cannibals_right]
# Each move indicates the number of people changed on both sides within one step
# It uses BFS to give a shortest solution, may not be the only solution

# change the boat position
def change_boat_pos(flag):
    if flag:
        flag = False
    else:
        flag = True
    return flag

# determin the direction of a move
def direct_a_move(flag, move):
    if flag:
        return move
    else:
        ret = []
        for num in move:
            ret.append(-num)
        return ret

# compute next state based on current state and move
def compute_state(state, move):
    ret = []
    for i in xrange(len(state)):
        ret.append(state[i]+move[i])
    return ret

# determin if a state is a valid state
def is_valid_state(state):
    for i in xrange(len(state)):
        if state[i] <0:
            return False
    if state[0] < state[1] and state[0] > 0:
        return False
    if state[2] < state[3] and state[2] > 0:
        return False
    return True

# determin if a state is a goal state
def terminate_state(state, goal_state):
  if len(state) != len(goal_state):
    return False
  for i in xrange(len(goal_state)):
    if state[i] != goal_state[i]:
      return False
  return True
```

```
init_state = [3,3,0,0] # initially everyone on the left side
goal_state = [0,0,3,3] # eventually everyone on the right side
init_boat_pos = True # it begin with a boat on the left side of the river

# positive moves are from left to right, negative moves are from right to left
# controled by the "make_a_move()" function
moves = [[0,-1,0,1],[0,-2,0,2],[-1,0,1,0],[-2,0,2,0],[-1,-1,1,1]]

# initialize current states
# format: [[state_1,[move_sequence_1]],[state_2,[move_sequence_2],...]
curr_states = []
curr_states.append([init_state,[]])

# initialize next states, empty
next_states = []

# initialize boat position
curr_boat_pos = init_boat_pos

# initialize solution
solution = []

# BFS algorithm to find a solution
solve_flag = False # flag indicates if it is solved
while not solve_flag:
  # go through each current state
  for i in xrange(len(curr_states)):
    state = list(curr_states[i][0])
    # try to apply all possible moves
    for j in xrange(len(moves)):
      move = direct_a_move(curr_boat_pos, moves[j])
      new_state = compute_state(state,move)
      # proceed only if the new state is valid
      if is_valid_state(new_state):
        # if it is a valid state, update the sequence of moves
        seq = list(curr_states[i][1])
        seq.append(j)
        # update the next_states list with current info
        next_states.append([new_state, seq])
        # check if this state is the goal state
        if terminate_state(new_state,goal_state):
          # if true, flip the flag
          solve_flag = True
          # save the move sequence as solution
          solution = list(seq)
          #break when solved
          break
    if solve_flag:
      #break when solved
      break
  # update boat position and current states
```

```
    curr_boat_pos=change_boat_pos(curr_boat_pos)
    curr_states = list(next_states)
    next_states = []

# Reproduce the move sequence
state = list(init_state)
print '0. init state',state
curr_boat_pos = True
# for each sequence id in solution, reproduce the move and its corresponding state
for i in xrange(len(solution)):
    move = direct_a_move(curr_boat_pos, moves[solution[i]])
    state = compute_state(state,move)
    print str(i+1)+'. action',move,', state',state
    curr_boat_pos = change_boat_pos(curr_boat_pos)
```

c. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

Answer:

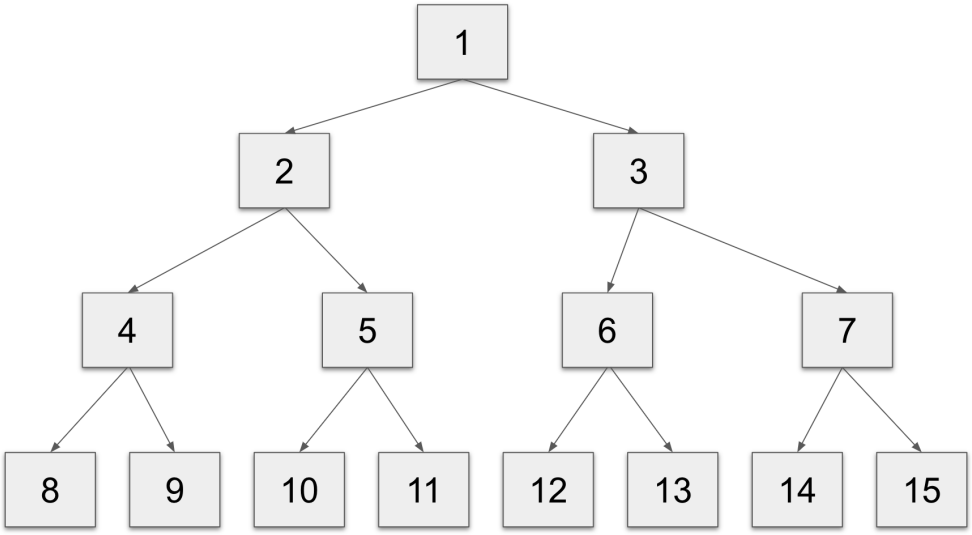
As we can see from the above table, there are 16 possible states on each side, and 10 of them are valid states. If we take the position of boat into consideration, it's 20 possible valid states on both sides. The actions are moving people back and forth on the two sides of a river and the boat position is changing at each step. The searching algorithm for human brain is DFS by default. People feel more comfortable to follow a stream and apply DFS instead of spread out and apply BFS on a specific problem. Unfortunately, DFS is not the best option to this problem because it requires a lot of memory to check repeat states and loops. The 32 possible states and 20 valid states is too much for human being to apply BFS with. Therefore, this problem seems to be easy for computers but hard for human.

Problem 2

(20 pts.) Consider a state space where the start state is number 1 and each state k has two successors: numbers $2k$ and $2k + 1$.

a. Draw the portion of the state space for states 1 to 15.

Answer:



b. Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

Answer:

BFS:

$1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9 \to 10 \to 11$

DLS:

$1 \to 2 \to 4 \to 8 \to 9 \to 5 \to 10 \to 11$

IDS:

- ☐ Iter_1 Depth 0:
  1
- ☐ Iter_2 Depth 1:
  $1 \to 2 \to 3$
- ☐ Iter_3 Depth 2:
  $1 \to 2 \to 4 \to 5 \to 3 \to 6 \to 7$
- ☐ Iter_4 Depth 3:
  $1 \to 2 \to 4 \to 8 \to 9 \to 5 \to 10 \to 11$