CS 411 — Artificial Intelligence I — Spring 2017
Assignment 1
Department of Computer Science, University of Illinois at Chicago

Name : Sarit Adhikari
Net-id : sadhik6

1. [6 pts] How are you going to represent the states and actions? You can write down the class definition in Java/Python, with comments on the meaning of each variable. For each action, explain how it maps an old state into a new one. Try to be concise and use symbols rather than 1,2,3… The mod operator might be useful here (the modulo operation that finds the remainder after division of one number by another).

The state of puzzle is represented as a tuple . The location of symbol on tuple corresponds to location of tile in current state of puzzle .

Meaning of symbols used to represent state

'r' => respresents red tile

'g' => represents green tile

'*' => represents empty tile

For e.g. The following random initial state is represented as ['r','*','g','g','r','r','*','g']

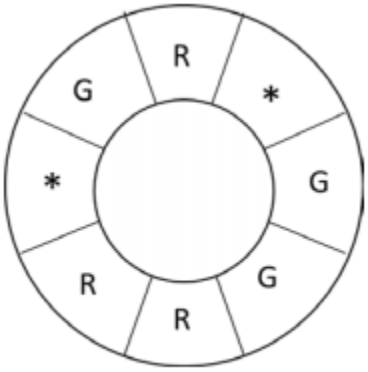


Figure 2: A random initial state for N=3

The circular behavior is emulated with the help of mod operator (% in python)

For example, if 'i' is current index, the index of it's left neighbor and right neighbor is given by

l = (i-1) % (len(state))

r = (i+1) % (len(state))

where 'state' is the tuple representing current state

2. [6 pts] Suppose we want to randomly initialize the configuration. What do you think is a good way to do so? Implement it as a function that takes a seed of random number generator. This will allow random experiments to be replicated.

To randomly initialize the state, an empty list is taken. The symbols ‘g’ and ’r’ are inserted n times while ‘*’ is inserted twice into the list. The list is then shuffled using random.shuffle() method. Before running shuffle, the seed is set so that the experiment can be replicated.

The function is as follows

```
def generateRandomInitialState(n,seed):
    random.seed(seed)
    state = []
    counter = 0
    while(counter< 2*n):
        if (counter < n):
            state.append('g')
        else:
            state.append('r')
        counter = counter+1
    state.append('*')  # since there are always two empty
    state.append('*')
    random.shuffle(state)
    return tuple(state)
```

3. [8 pts] Define an admissible heuristic h(n) that can be used by the A* algorithm to solve this problem. Try to be as smart as possible (what does this mean?). For example, h(n) = 0 for all n is a bad heuristic. For grading, 4 points are given if it is admissible. The rest 4 points are computed by max{0, 5 – x}, where x is the number of different heuristics proposed by your classmates that strictly dominate your heuristic. h1 strictly dominates h2 if and only if $h1(n) \geq h2(n)$ for all n, and $h1(n) > h2(n)$ for at least one n.

I have defined heuristic h(n) as the number of pairs of tiles that are located side by side.

For.e.g.

h(n) value for the state ['r','r','*','g','g','*'] is 2 as there are two pairs of tiles located side by side

h(n) value for the state [‘r’,’r’,’r’,’*’,’g’,’g’,’g’,’*’] is 4 as there are four pairs of tiles located side by side

As it never overestimates the cost to reach goal , it is admissible.

It is defined in code as below:

```
def h(self, node):
        state_dup = list(node.state[:]) #creating duplicate to avoid changing original tuple, converting it to list so that value within list can be changed
        h_val = 0
        for i in range(0, len(state_dup)):
            if(state_dup[i]=='*' or state_dup[i]=='-'):
                continue

            l = (i-1) % (len(state_dup))
            r = (i+1) % (len(state_dup))
            if state_dup[l] == state_dup[i] or state_dup[i]==state_dup[r]:
                h_val = h_val+1
                state_dup[i] = '-' #changing tile to - so for a pair of tile h_val is incremented just once

        return h_val #number of pairs of same-colored tiles together
```

4. [15 pts] Fix N=3. Pick one uninformed search algorithm among the following: breadth-first (graphsearch version), uniform-cost (graph-search version), depth-first (tree-search), iterative-deepening. Randomly generate three initial states and run both A* algorithm (graph-search version) and your picked algorithm on the three problems. In principle, you should use seeds 1, 2, 3, ... and skip a value if and only if it results in a state that is 0~2 steps away from a goal state.

(a) In your PDF report, show all the following results (3*2=6 sets of results), and comment on them.

i. the number of nodes expanded;

ii. the cost of the solution

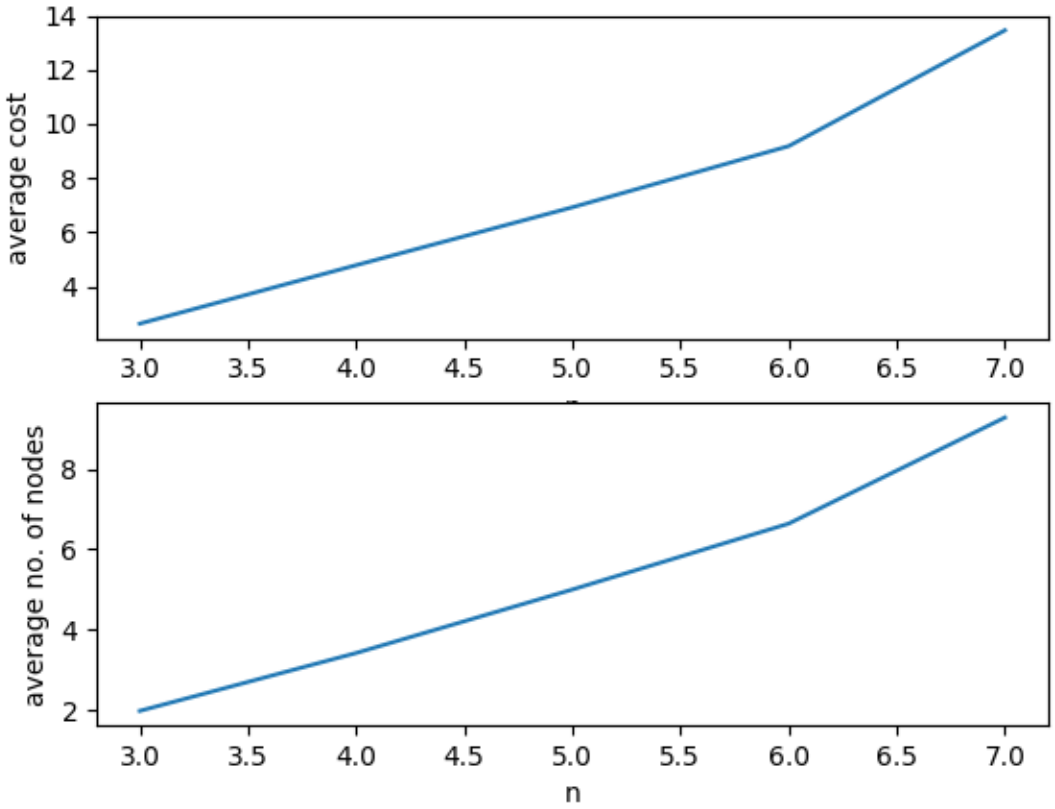
I have picked Breadth first search as the uninformed search . The initial states were generated randomly with seeds 2,3,4,….

Here is the table summarizing the result

Seed	Astar Search		Breadth First Search		Remarks
	Nodes Expanded	Cost	Nodes Expanded	Cost	
2	('r', 'r', 'r', 'g', 'g', '*', '*', 'g'), ('r', 'r', 'r', 'g', '*', 'g', '*', 'g'), ('*', 'r', 'r', 'g', '*', 'g', 'r', 'g'), ('r','*', 'r', 'g', '*', 'g', 'r', 'g')	4	('r', 'r', 'r', 'g', 'g', '*', '*', 'g'), ('r', 'r', 'r', 'g', '*', 'g', '*', 'g'), ('r', 'r', '*', 'g', 'r', 'g', '*', 'g'), ('r','*', 'r', 'g', 'r', 'g', '*', 'g')	4	There's a difference in node expanded in 3rd step
3	('g', 'r', '*', 'g', 'g', '*', 'r', 'r'), ('g', 'r', '*', 'g', 'g', 'r', '*', 'r'), ('g', 'r', 'g', '*', 'g', 'r', '*', 'r')	2	('g', 'r', '*', 'g', 'g', '*', 'r', 'r'), ('g', 'r', 'g', '*', 'g', '*', 'r', 'r'), ('g', 'r', 'g', '*', 'g', 'r', '*', 'r')	2	There's difference in node expanded in 2nd step
4	('g', '*', 'r', 'r', '*', 'g', 'g', 'r'), ('g', '*', 'r', 'r', 'g', '*', 'g', 'r'), ('g', 'r', '*', 'r', 'g', '*', 'g', 'r')	2	('g', '*', 'r', 'r', '*', 'g', 'g', 'r'), ('g', 'r', '*', 'r', '*', 'g', 'g', 'r'), ('g', 'r', '*', 'r', 'g', '*', 'g', 'r')	2	There's difference in node expanded in 2nd step
Note: Seed 1 is not used because it resulted in the goal state while shuffling the tile					

5. [15 pts] Conduct the following for each value of N from 3 to 10. Randomly generate 50 initial states. Run A* algorithm to find an optimal solution for all of them, and record the average number of (averaged over 100 initializations): a) the path cost of the optimal solution (i.e. the smallest number of moves leading to a goal state); b) the number of nodes generated by A* (not just the number of expanded node). This number might not be readily available from the functions of AIMA code base. If necessary, you can approximate it by the number of times that your heuristic function is invoked. Plot your results in two figures like Figure 5 and 6 (curves are made up). The horizontal axis is N, and the vertical axis is the average numbers above. Comment on the result. If it is too costly (in computation or space) to run for some values of N, mention it in your report and say how long you've waited or how many nodes are being kept in memory.

The graph plotted is as follows. Value of N ranges from 3 to 7 and the cost is an average over 50 random initial states.



It took a lot of time (over 1 hour) to run for n=7.