Assignment2

May 20, 2019

Student: Marcu Andrei Cristian

Artificial Inteligence

CEN 2.2 A

2nd Year

Problem statements

Problem 1 (50p): The Water Jug Problem: Given two jugs without any measuring markings on it, a 4-litre one and a 3-litre one and an infinite source of water, how can you get exactly 2 gallons of water in the 4-gallon jug? Based on the given Python framework, create a model representation for this problem and apply proper informed/uninformed search strategy.

Problem 2 (50p): Based on the 8-puzzle problem from the provided frame- work build a model representation of the 15-puzzle problem and create two heuristic functions (different from those presented at the laboratory) to solve the problem by applying A* search strategy.

Application design

The project is formed of 5 python files ***main.py P1.py P2.py search.py utils.py***.
main.py contains all the function calls and the initial value and the goal for each problem.
P1.py contains the solution for the WaterJug problem. It is implemented using inheritance of the Problem class in ***search.py*** .
P2.py contains the solution for the 15-Puzzle problem and the heuristics needed. It is implemented using inheritance of the Problem class in ***search.py***.
search.py was already implemented in the framework we used, it contains different search algorithms and virtual classes which help us implement the solutions.
utils.py was already implemented in the framework we used.

The Water Jug Implementation(P1.py)

The function actions implements each possible action for each jug:
1. From full to empty
2. From empty to full
3. From one to another
There are 3 cases for each jug, 6 cases in total

The function result takes each of the actions created before and adds it to the present state.

15-Puzzle Implementation(P2.py)

The function actions excludes each impossible action for the empty tile:
1. If it is on the most left column it can't go to left
2. If it is on the most right column it can't go right
3. If it is on the first row it can't go up
4. If it is on the last row it can't go down

The function result takes all the possible actions and adds those to the present state.
I've implemented two heuristics for this problem :
The heuristic defined in the class Puzzle15MisDist it's a heuristic which makes the sum of the number of misplaced tiles and the distance from the misplaced tile to it's goal.
The heuristic defined in the class Puzzle15MisColRow it's a heuristic which makes the sum of total misplaced tiles on columns and the total misplaced tiles on rows.

Source Code

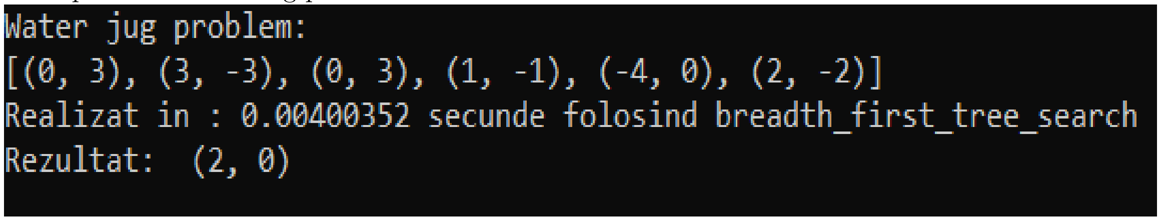
```
#-----------------------------P1.py-------------------------------
from search import Problem

class WaterJug(Problem):

    def __init__(self, initial, goal=(1, 2, 3, 4, 5, 6, 7, 8, 0)):
        """ Define goal state and initialize a problem """

        self.goal = goal
        Problem.__init__(self, initial, goal)
        self.maxJug = (4, 3)

    def actions(self, state):

        new_results = []

        if state[0] == 0:
            new_results.insert(0,(self.maxJug[0], 0))
        if state[1] == 0:
            new_results.insert(0,(0, self.maxJug[1]))
        if state[0] == self.maxJug[0]:
            new_results.insert(0,(-self.maxJug[0], 0))
        if state[1] == self.maxJug[1]:
            new_results.insert(0,(0, -self.maxJug[1]))
        if state[0] != 0 and state[1] != self.maxJug[1]:
            move_ammt = min(state[0], self.maxJug[1] - state[1])
            new_results.insert(0,(-move_ammt, +move_ammt))
        if state[1] != 0 and state[0] != self.maxJug[0]:
            move_ammt = min(state[1], self.maxJug[0] - state[0])
            new_results.insert(0,(+move_ammt, -move_ammt))

        return new_results

    def result(self, state, action):
        """ Given state and action, return a new state that is the
            result of the action.
        Action is assumed to be a valid action in the state """

        # blank is the index of the blank square
        new_state = list(state)

        new_state[0] += action[0]
        new_state[1] += action[1]

        return tuple(new_state)
```

```
#----------------------------P2.py------------------------------
from search import Problem
import math

class Puzzle15(Problem):

    def __init__(self, initial, goal=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
        12, 13, 14, 15, 0)):
        """ Define goal state and initialize a problem """

        self.goal = goal
        Problem.__init__(self, initial, goal)

    def find_blank_square(self, state):
        """Return the index of the blank square in a given state"""

        return state.index(0)

    def actions(self, state):
        """ Return the actions that can be executed in the given state.
        The result would be a list, since there are only four possible
            actions
        in any given state of the environment """
        possible_actions = ['UP', 'DOWN', 'LEFT', 'RIGHT']
        index_blank_square = self.find_blank_square(state)

        if index_blank_square % 4 == 0:
            possible_actions.remove('LEFT')
        if index_blank_square < 4:
            possible_actions.remove('UP')
        if index_blank_square % 4 == 3:
            possible_actions.remove('RIGHT')
        if index_blank_square > 11:
            possible_actions.remove('DOWN')

        return possible_actions

    def result(self, state, action):
        """ Given state and action, return a new state that is the
            result of the action.
        Action is assumed to be a valid action in the state """

        # blank is the index of the blank square
        blank = self.find_blank_square(state)
        new_state = list(state)

        delta = {'UP':-4, 'DOWN':4, 'LEFT':-1, 'RIGHT':1}
        neighbor = blank + delta[action]
        new_state[blank], new_state[neighbor] = new_state[neighbor],
            new_state[blank]

        return tuple(new_state)

    def h(self, node):
        """ Return the heuristic value for a given state. Default
            heuristic function used is
        h(n) = number of misplaced tiles """

        return sum(s != g for (s, g) in zip(node.state, self.goal))


class Puzzle15MisDist(Puzzle15):
    def h(self, node):
        misplaced = 0 # the number of misplaced tiles
        dist = 0 # the distance between the misplaced tiles

        for i in range(15):
            if node.state[i] != self.goal[i]:
                misplaced += 1
        for i in node.state:
            dist += math.fabs(node.state.index(i) - self.goal.index(i))
        total_heurst = dist + misplaced

        return total_heurst


class Puzzle15MisColRow(Puzzle15):
    def h(self, node):
        misplaced_row = 0 # number of misplaced tiles on rows
        misplaced_col = 0 # number of misplaced tiles on columns

        for i in range(15):
            if node.state[i] != self.goal[i]:

                if i % 4 == 0:
                    misplaced_col += 1
                if i % 4 == 1:
                    misplaced_col += 1
                if i % 4 == 2:
                    misplaced_col += 1
                if i % 4 == 3:
                    misplaced_col += 1

                if i // 4 == 0:
                    misplaced_row += 1
                if i // 4 == 1:
                    misplaced_row += 1
                if i // 4 == 2:
                    misplaced_row += 1
                if i // 4 == 3:
                    misplaced_row += 1

        total_heurst = misplaced_col + misplaced_row

        return total_heurst
```

```
#-----------------------------main.py------------------------------
import time
from search import *
import P1
import P2

def main():
    #water jug problem
    print("Water jug problem:")
    water_jug = P1.WaterJug((0,0), (2,0))
    t1 = time.time()
    print(breadth_first_tree_search(water_jug).solution())
    t2 = time.time()
    exe_time = t2-t1
    print("Realizat in : %.8f secunde folosind
        breadth_first_tree_search" % exe_time)
    print("Rezultat: ", breadth_first_tree_search(water_jug).state)

    #15 puzzle problem
    print("\n\n15 Puzzle problem:")

    puzzle_15 = P2.Puzzle15((5,1,7,3,9,2,11,4,13,6,15,8,0,10,14,12),
        (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0))
    puzzle_15_mis_dist =
        P2.Puzzle15MisDist((5,1,7,3,9,2,11,4,13,6,15,8,0,10,14,12),
        (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0))
    puzzle_15_mis_col_row =
        P2.Puzzle15MisColRow((5,1,7,3,9,2,11,4,13,6,15,8,0,10,14,12),
        (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0))

    print("\nInitial : ",puzzle_15_mis_dist.initial)
    t1 = time.time()
    print(astar_search(puzzle_15_mis_dist).solution())
    t2 = time.time()
    exe_time = t2-t1
    print("Realizat in : %.8f secunde folosind A* cu heuristica care
        calculeaza suma elementelor misplaced si distanta dintre
        acestea" % exe_time)
    print("Rezultat : ", astar_search(puzzle_15_mis_dist).state)

    print("\nInitial : ", puzzle_15_mis_col_row.initial)
    t1 = time.time()
    print(astar_search(puzzle_15_mis_col_row).solution())
    t2 = time.time()
    exe_time = t2-t1
    print("Realizat in : %.8f secunde folosind A* cu heuristica care
        calculeaza suma elementelor misplaced de pe fiecare coloana si
        linie" % exe_time)
    print("Rezultat : ", astar_search(puzzle_15_mis_col_row).state)

    print("\nInitial : ",puzzle_15.initial)
    t1 = time.time()
    print(astar_search(puzzle_15).solution())
    t2 = time.time()
    exe_time = t2-t1
    print("Realizat in : %.8f secunde folosind A* cu heuristica
        initiala" % exe_time)
    print("Rezultat : ", astar_search(puzzle_15).state)



if __name__ == "__main__":
    main()
```

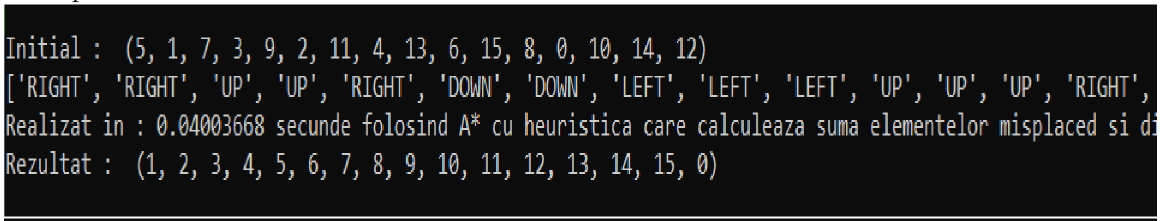
Experiments and results

Example with Water Jug problem

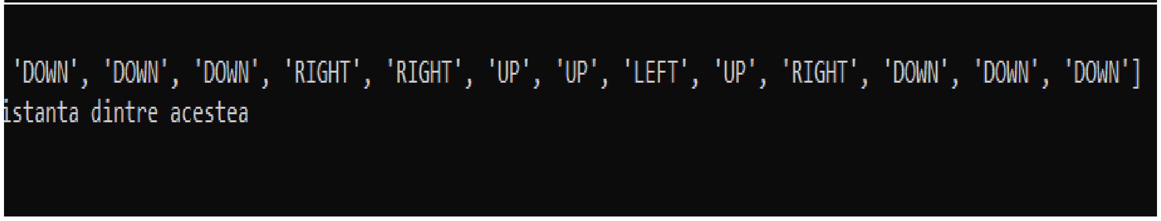


```
Water jug problem:
[(0, 3), (3, -3), (0, 3), (1, -1), (-4, 0), (2, -2)]
Realizat in : 0.00400352 secunde folosind breadth_first_tree_search
Rezultat:  (2, 0)
```

Example with 15-Puzzle method heuristic 1

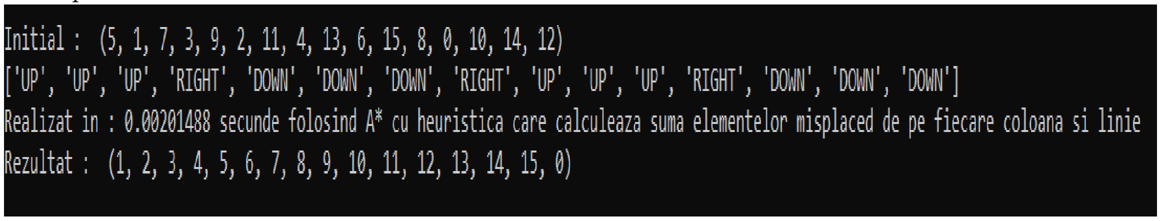


```
Initial :  (5, 1, 7, 3, 9, 2, 11, 4, 13, 6, 15, 8, 0, 10, 14, 12)
['RIGHT', 'RIGHT', 'UP', 'UP', 'RIGHT', 'DOWN', 'DOWN', 'LEFT', 'LEFT', 'LEFT', 'UP', 'UP', 'UP', 'RIGHT',
Realizat in : 0.04003668 secunde folosind A* cu heuristica care calculeaza suma elementelor misplaced si di
Rezultat :  (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)
```



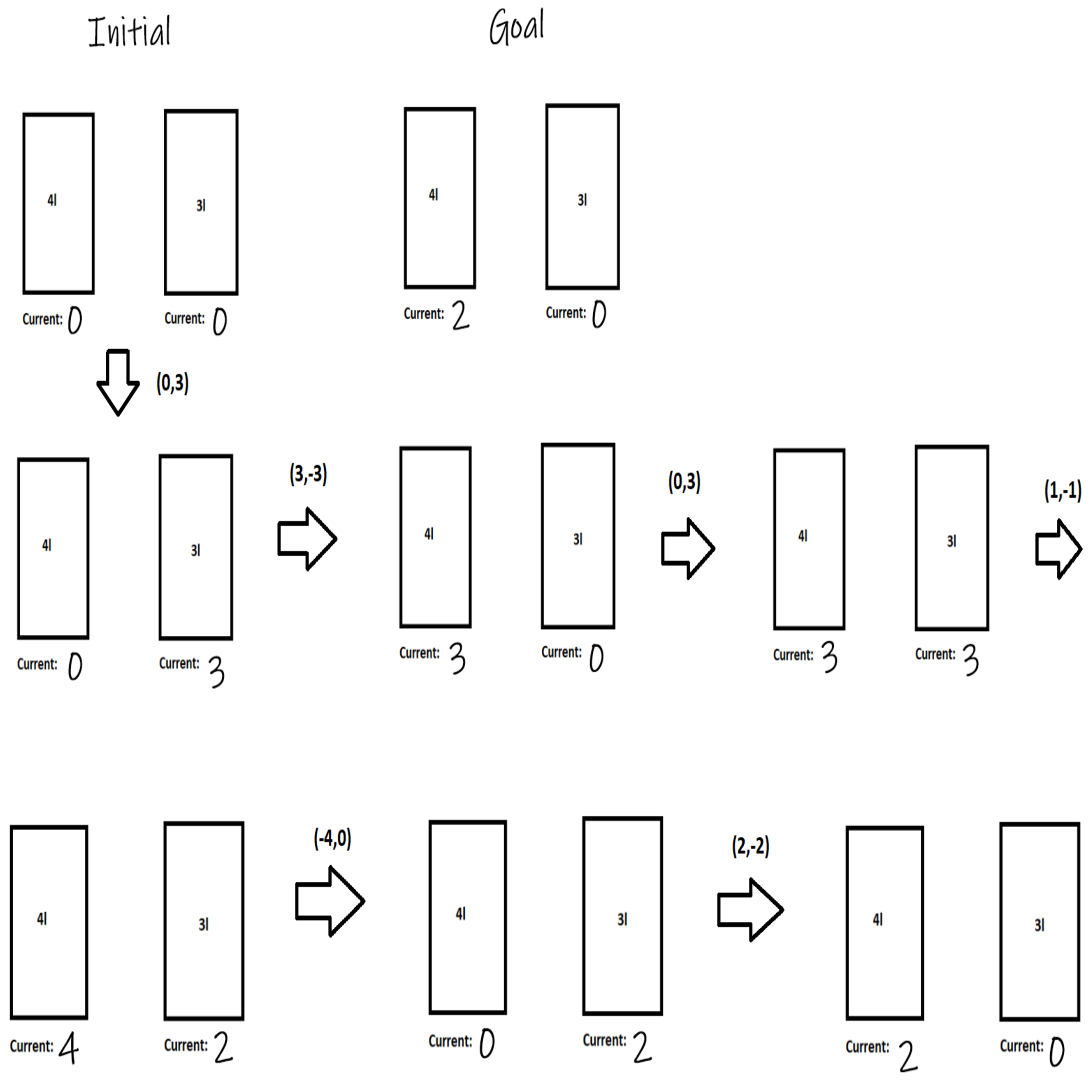
```
 'DOWN', 'DOWN', 'DOWN', 'RIGHT', 'RIGHT', 'UP', 'UP', 'LEFT', 'UP', 'RIGHT', 'DOWN', 'DOWN', 'DOWN']
istanta dintre acestea
```

Example with 15-Puzzle method heuristic 2

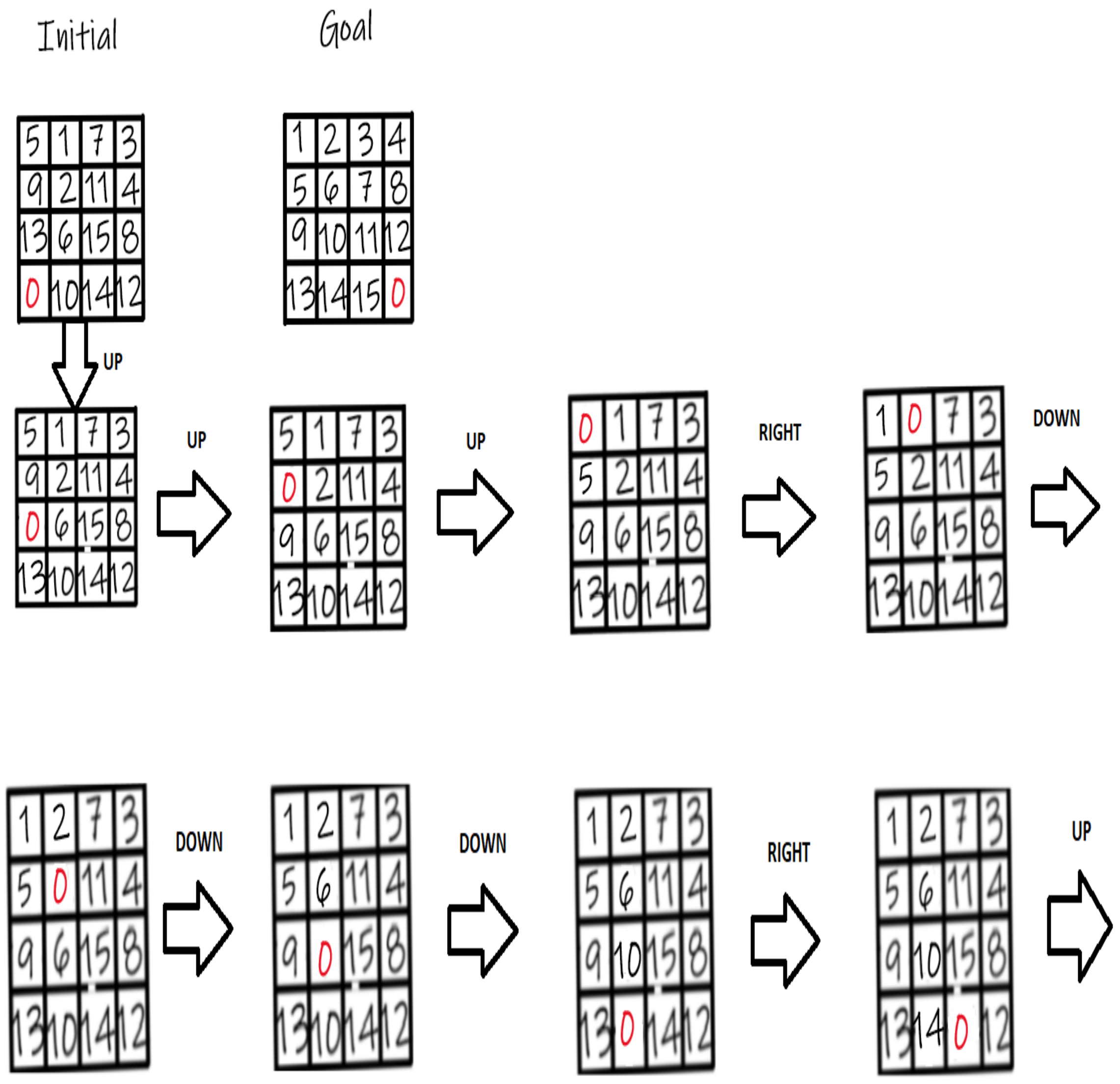


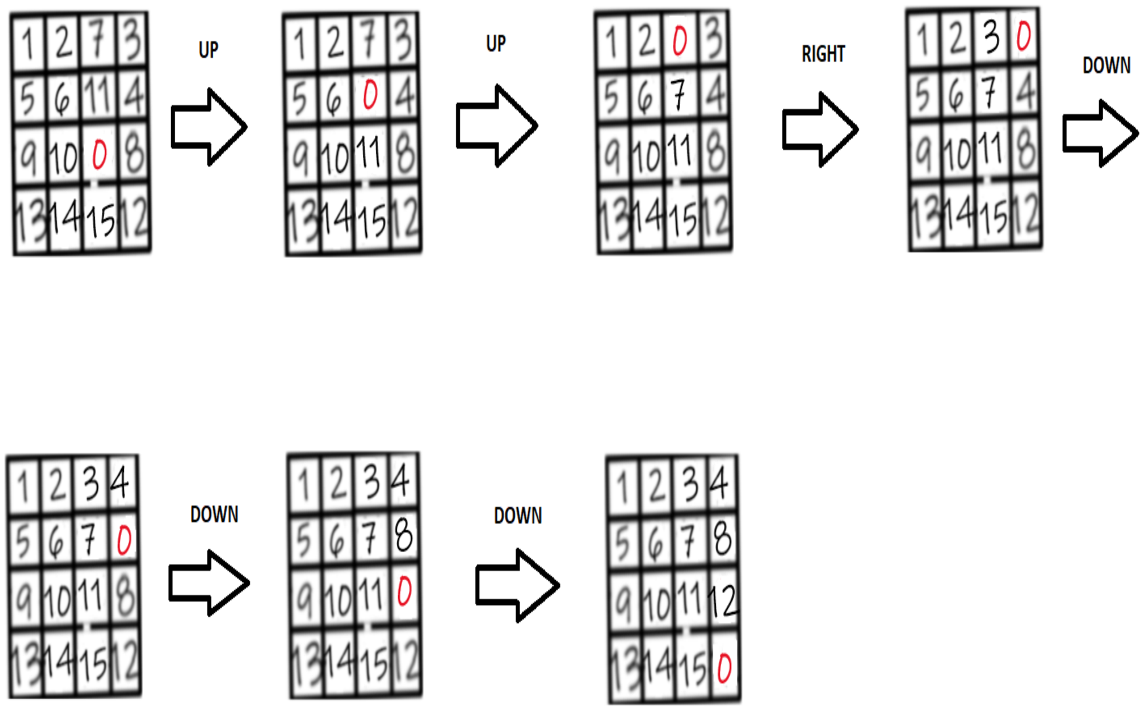
```
Initial :  (5, 1, 7, 3, 9, 2, 11, 4, 13, 6, 15, 8, 0, 10, 14, 12)
['UP', 'UP', 'UP', 'RIGHT', 'DOWN', 'DOWN', 'DOWN', 'RIGHT', 'UP', 'UP', 'UP', 'RIGHT', 'DOWN', 'DOWN', 'DOWN']
Realizat in : 0.00201488 secunde folosind A* cu heuristica care calculeaza suma elementelor misplaced de pe fiecare coloana si linie
Rezultat :  (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)
```

The graphical representation of the WaterJug solution, it's the same example that I showed before in console.



The graphical representation of the 15-Puzzle solution with heuristic 2, it's the same example that I showed before in console.




UP
UP
RIGHT
DOWN
DOWN
DOWN