2nd Quiz of DAA-F Report

Arranged By:

M. Amir Fauzan Al-Machdi

Bagas Juwono Priambodo

Wisnu

Department Of Informatics

Batch 2017

By the name of Allah (God) Almighty, herewith we pledge and truly declare that we have solved quiz 2 by ourself, didn’t do any cheating by any means, didn’t do any plagiarism, and didn’t accept anybody’s help by any means. We are going to accept all of the consequences by any means if it has proven that we have been done any cheating and/or plagiarism.

Surabaya, 24 March 2019

Wishu	Bagas Juwono Priambodo	M. Amir Fauzan
05111740000170	05111740000163	05111740000157

Solving 8-Puzzle using A* Algorithms

(Disclaimer: This is not a game)

N-Puzzle or sliding puzzle is a popular puzzle that consists of N tiles where N can be 8, 15, 24 and so on. In our example N = 8. The puzzle is divided into sqrt(N+1) rows and sqrt(N+1) columns. Eg. 15-Puzzle will have 4 rows and 4 columns and an 8-Puzzle will have 3 rows and 3 columns. The puzzle consists of N tiles and one empty space where the tiles can be moved. Start and Goal configurations (also called state) of the puzzle are provided. The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal configuration.

1	2	3
	4	6
7	5	8

initial state

1	2	3
4	5	6
7	8	

goal state

The tiles in the initial(start) state can be moved in the empty space in a particular order and thus achieve the goal state.

Rules for solving the puzzle.

Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile, basically swapping the tile with the empty space. The empty space can only move in four directions viz.,

1. Up

2.Down

3. Right or

4. Left

The empty space cannot move diagonally and can take only one step at a time (i.e. move the empty space one position at a time).

Before we talk about A* algorithm, we will explain about Heuristic Search in short.

1. Uninformed Search

The Linear Search, Binary Search, Depth-First Search or the Breadth-First Search algorithms fallsq into the category of uninformed search techniques i.e. these algorithms do not know anything about what they are searching and where they should search for it. That's why the name "uninformed" search. Uninformed searching takes a lot of time to search as it doesn't know where to head and where the best chances of finding the element are.

2. Informed Search

Informed search is exactly opposite to the uninformed search. In this, the algorithm is aware of where the best chances of finding the elements are and the algorithm heads that way. Heuristic search is an informed search technique. A heuristic value tells the algorithm which path will provide the solution as early as possible. The heuristic function is used to generate this heuristic value. Different heuristic functions can be designed depending on the searching problem. So we can conclude that Heuristic search is a technique that uses heuristic value for optimizing the search.

A* Algorithm

A* is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes. Noted for its performance and accuracy, it is a widespread use.

The key feature of the A* algorithm is that it keeps a track of each visited node which helps in ignoring the nodes that are already visited, saving a huge amount of time. It also has a list that holds all the nodes that are left to be explored and it chooses the most optimal node from this list, thus saving time not exploring unnecessary or less optimal nodes.

So we use two lists namely 'open list' and 'closed list'. The open list contains all the nodes that are being generated and are not existing in the closed list and each node explored after it's neighboring nodes are discovered is put in the closed list and the neighbors are put in the open list. This is how the nodes expand. Each node has a pointer to its parent so that at any given point it can retrace the path to the parent. Initially, the open list holds the start(Initial) node. The next node chosen from the open list is based on its f score, the node with the least f score is picked up and explored.

f-score = h-score + g-score

A* uses a combination of heuristic value (h-score: how far the goal node is) as well as the g-score (i.e. the number of nodes traversed from the start node to current node).

In our 8-Puzzle problem, we can define the h-score as the number of misplaced tiles by comparing the current state and the goal state or summation of the Manhattan distance between misplaced nodes.

g-score will remain as the number of nodes traversed from start node to get to the current node.

From Fig 1, we can calculate the h-score by comparing the initial(current) state and goal state and counting the number of misplaced tiles.

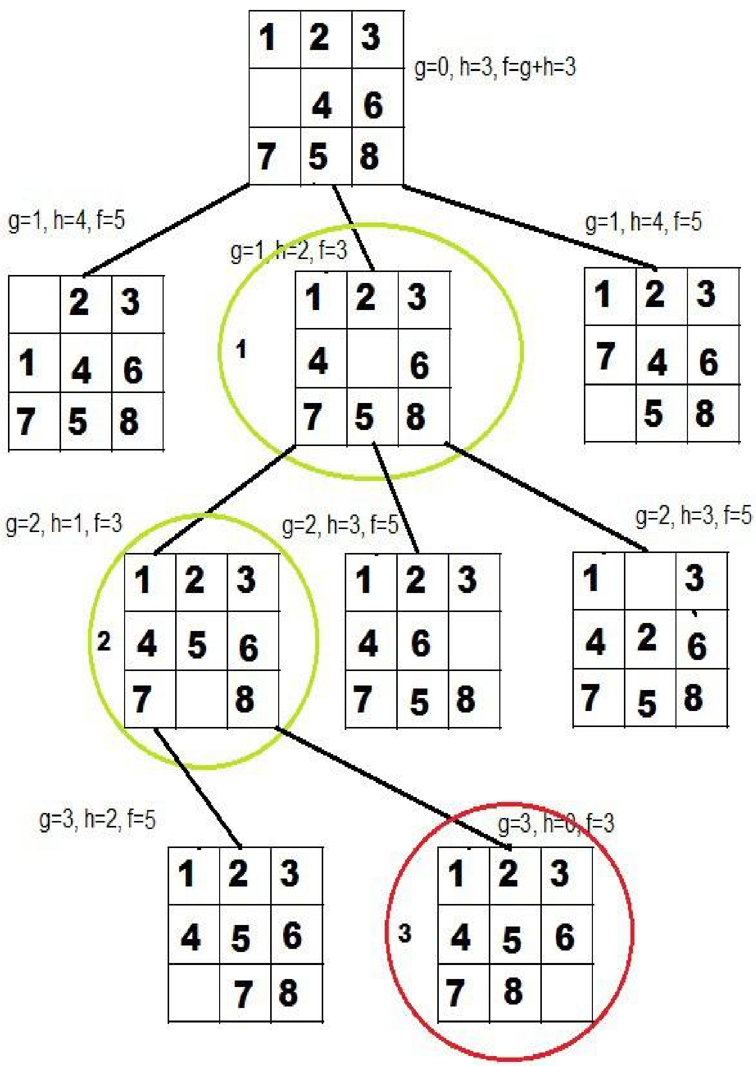
Thus, h-score = 5 and g-score = 0 as the number of nodes traversed from the start node to the current node is 0.

How A* solves the 8-Puzzle problem.

We first move the empty space in all the possible directions in the start state and calculate f-score for each state. This is called expanding the current state.

After expanding the current state, it is pushed into the closed list and the newly generated states are pushed into the open list. A state with the least f-score is selected and expanded again. This process continues until the goal state occurs as the current state. Basically, here we are providing the algorithm a measure to choose its actions. The algorithm chooses the best possible action and proceeds in that path.

This solves the issue of generating redundant child states, as the algorithm will expand the node with the least f-score.



Source Codes (Explainations are in the codes comment).

AStar8Puzzyle.py

```
from copy import deepcopy

case = [ [1 , 2 , 3],
         [0 , 4 , 6],
         [7 , 5 , 8]]

goal = [ [1 , 2 , 3],
     [4 , 5 , 6],
     [7 , 8 , 0]]

def getHammingValue(case):
    global goal
    hammingValue = 0
    for i in range(len(case)):                              #Here, we are using a hamming value to define / state the position
        for j in range(len(case[i])):                       #of the nodes in the array (puzzle board)
            if case[i][j] != goal[i][j] and goal[i][j] != 0:
                hammingValue += 1

    return hammingValue

def getManhattanDistance(case):
    manhattanSum = 0
    for i in range(len(case)):                              #In this manhattan Distance function, we will find the shortest distance to the goal
        for j in range(len(case[i])):
            value = case[i][j]
            if value != 0:
                targetX = (value - 1) / 3                   #In this X and Y, we are matching the value of the goal and the value of the states
                targetY = (value - 1) % 3                   #and check wether it is in the same array position, or not.
                deviationX = i - targetX                    #if not, move the blank nodes again
                deviationY = j - targetY
                if deviationX < 0:
                    deviationX *= -1
                elif deviationY < 0:
                    deviationY *= -1

                manhattanSum += (deviationX + deviationY)

    return manhattanSum
```

```
class PriorityQueue(object):
    def __init__(self):
        self.queue = []

    # for checking if the queue is empty
    def isEmpty(self):
        return len(self.queue) == []

    # for inserting an element in the queue
    def insert(self, data):
        self.queue.append(data)

    # for popping an element based on Priority
    def delete(self):
        try:
            max = 0
            for i in range(len(self.queue)):
                if self.queue[i].getPriorityValue() < self.queue[max].getPriorityValue():
                    max = i
            item = deepcopy(self.queue[max])
            del self.queue[max]
            return item
        except IndexError:
            print()
            exit()
```

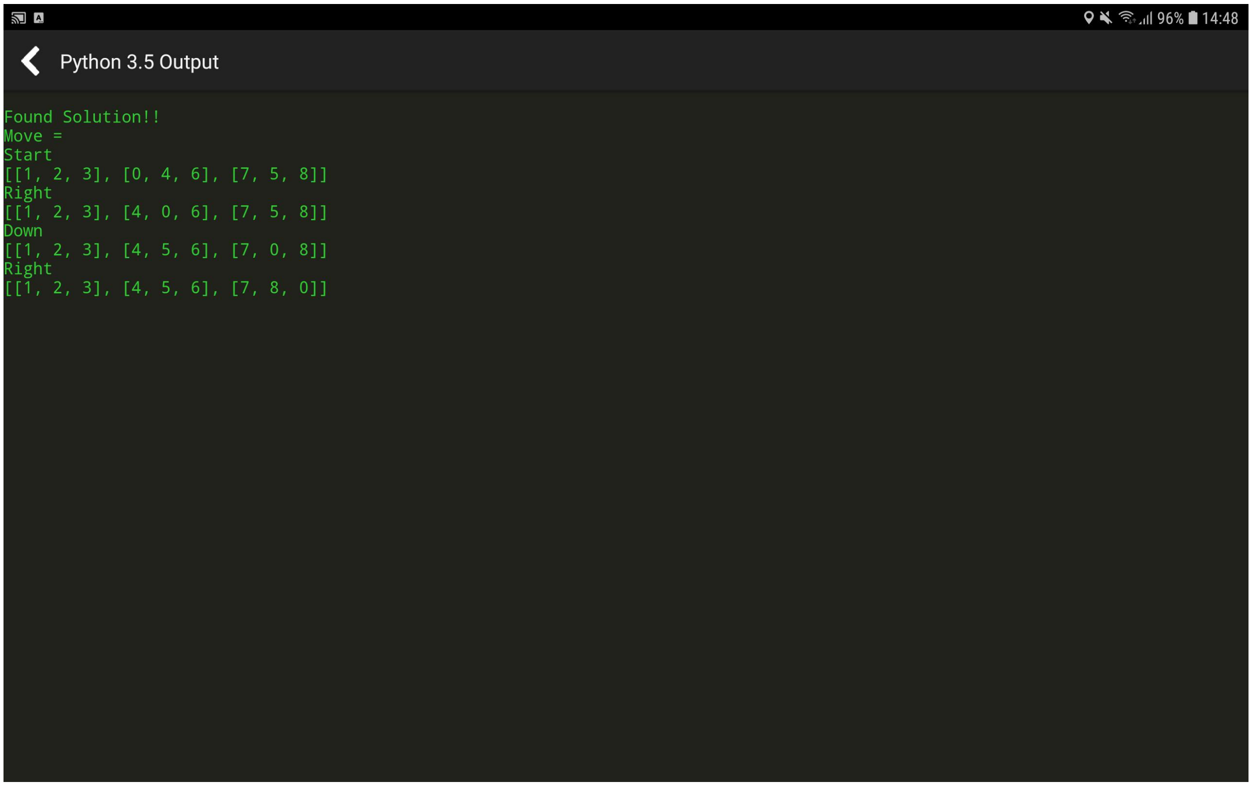
```python
class States:

    def __init__(self,state,heuristic,dist,priorityValue,directionPath,lastIndex):
        self.heuristic = heuristic
        self.dist = dist
        self.priorityValue = priorityValue
        self.state = state
        self.directionPath = directionPath
        self.lastIndex = lastIndex
        self.parent = None

    def getPriorityValue(self):
        return self.priorityValue

    def getState(self):
        return self.state

    def getDist(self):
        return self.dist

    def getParent(self):
        return self.parent

    def setParent(self,parentState):
        self.parent = parentState

    def getPath(self):
        return self.directionPath

    def getLastIndex(self):
        return self.lastIndex

    def printList(self):
        print ("dist  : "),self.dist
        print ("state : "),self.state
```

```python
def find_index_0(case):
    for i in range(len(case)):
        for j in range(len(case[i])):
            if case[i][j] == 0:
                return i,j

def swap(state,firstIndex,secondIndex):
    temp = state[firstIndex[0]][firstIndex[1]]
    state[firstIndex[0]][firstIndex[1]] = state[secondIndex[0]][secondIndex[1]]
    state[secondIndex[0]][secondIndex[1]] = temp
    return state

def printPath(goalState):

    if(goalState != None):
        printPath(goalState.getParent())
        print (goalState.getPath())
        print (goalState.getState())
```

```python
if __name__ == '__main__':
    moves = [[-1,0],[1,0],[0,1],[0,-1]]
    currentHeuristic = getHammingValue(case) + getManhattanDistance(case)
    priorityValue = currentHeuristic
    lastZeroIndex = [-1,-1]
    currentState = States(case,currentHeuristic,0,priorityValue,"Start",lastZeroIndex)
    zero_index = find_index_0(currentState.getState())
    openList = PriorityQueue()
    closeList = []
```

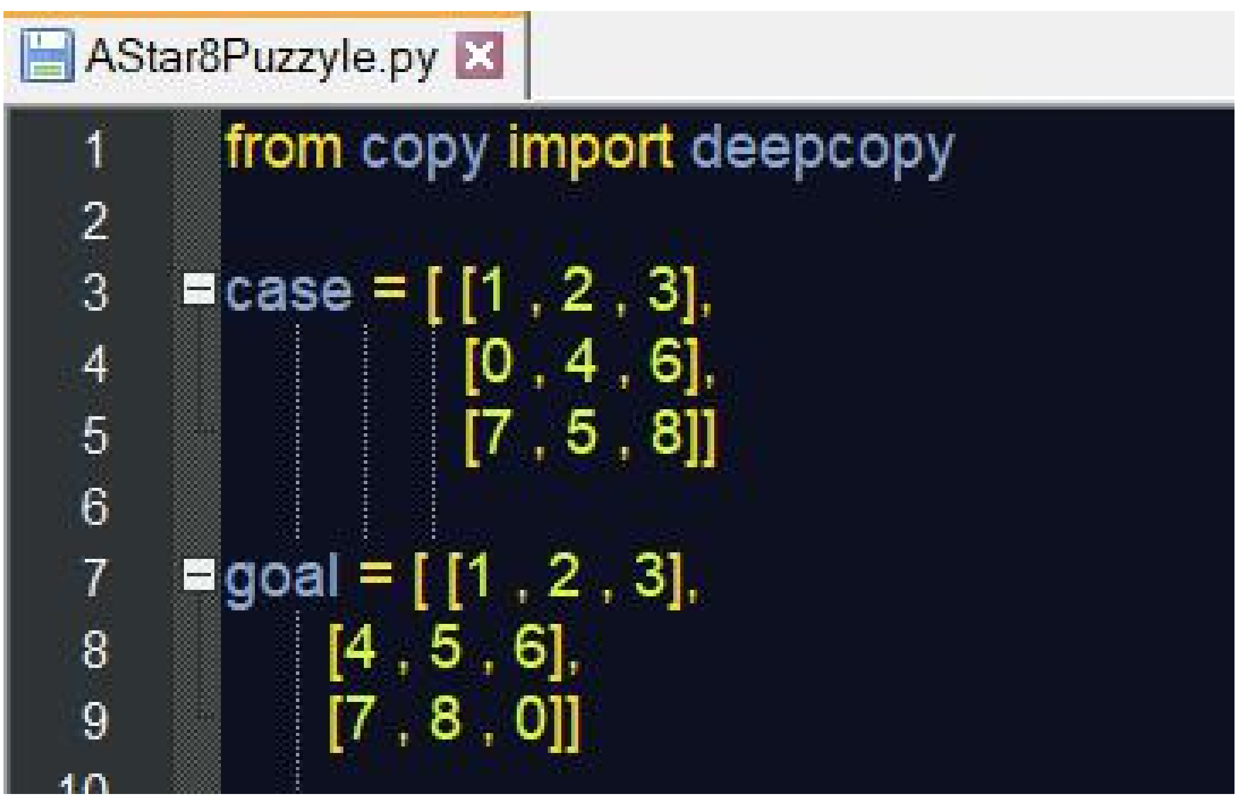
```python
    while True:


        if currentState.getState() == goal:
            print ("Found Solution!!")
            print ("Move = "),currentState.getDist()
            break

        for move in moves:
            predictedMove = zero_index[0] + move[0] , zero_index[1] + move[1]
            if predictedMove[0] >=0 and predictedMove[0] < 3 and predictedMove[1] >= 0 and predictedMove[1] < 3 and predictedMove != currentState.getLastIndex():

                originalState = deepcopy(currentState.getState())
                nextState = swap(originalState,predictedMove,zero_index)
                nextHeuristic = getHammingValue(nextState) + getManhattanDistance(nextState)
                nextDist = currentState.getDist() + 1
                nextPriority = nextHeuristic + nextDist
                lastZeroIndex = deepcopy(zero_index)

                if move == [-1,0]:
                    tracePath = "Up"
                elif move == [1,0]:
                    tracePath = "Down"
                elif move == [0,1]:
                    tracePath = "Right"
                elif move == [0,-1]:
                    tracePath = "Left"

                expandState = States(nextState,nextHeuristic,nextDist,nextPriority,tracePath,lastZeroIndex)
                expandState.setParent(currentState)
                openList.insert(expandState)

        currentState = openList.delete()
        zero_index = find_index_0(currentState.getState())

        closeList.append(currentState)


    printPath(closeList.pop())
```

Outputs:


96% 14:48
Python 3.5 Output
Found Solution!!
Move =
Start
[[1, 2, 3], [0, 4, 6], [7, 5, 8]]
Right
[[1, 2, 3], [4, 0, 6], [7, 5, 8]]
Down
[[1, 2, 3], [4, 5, 6], [7, 0, 8]]
Right
[[1, 2, 3], [4, 5, 6], [7, 8, 0]]

Guidance on how to use this console application:

Disclaimer: This is not a game. We are trying to explain on how A* Algorithms that works on most Artificial Intelligence helps you solve this board game called 8 Puzzle.




```
from copy import deepcopy

case = [ [1 , 2 , 3],
         [0 , 4 , 6],
         [7 , 5 , 8]]

goal = [ [1 , 2 , 3],
    [4 , 5 , 6],
    [7 , 8 , 0]]
```

1. If you are struggling on finishing a 3x3 8Puzzle, put your most current number positions in the “Case” arrays, and put the Goal state that you want in the “Goal” Array.

2. Run the program.

3. Voila, the output will show you the moves that you need to reach the goal state.

Github:

https://github.com/wisnugroho28/Using-A-Star-algorithm-to-Solve-8-Puzzle.

References:

https://blog.goodaudience.com