LP I

Artificial Intelligence and Robotics
Assignment B1

Date of Completion :- 9.10.2020

Title:- Puzzle.

Problem Statement:- Solve 8-puzzle problem using A* algorithm. Assume any initial configuration and define goal configuration clearly.

OR

- Solve following 6-tile problem stepwise using A* algorithm,

Initial Configuration

B	W	B	W	B	W

Final Configuration

B	B	B	W	W	W

Constraints : Tiles can be shifted left or right 1 or 2 positions with cost 1 and 2 respectively.

Objectives :- Understand 8 puzzle problem
Understand A* algorithm

Outcome :- Students will be able to implement 8 puzzle problem using A* algorithm.

Requirements:- Ubuntu OS, python.

Theory:-

- It is a heuristic search algorithm for finding paths.

1) Consider a square grid having many obstacles and we are given a starting cell and target cell.
2) We want to search target cell from the starting cell as quickly as possible.
3) At start each step, A* algorithm picks the node according to a value 'f' which is equal to sum of 'g' and 'h'.
4) At each step it picks the node cell having least 'f' and process that node.

$$f = g + h.$$

g → movement cost to move from the starting point to a given grid following the path generated to get there.
h → movement cost (estimated) to move from that given grid square on the grid to the final destination. This is often referred to as the heuristic which is nothing but a kind of smart guess.

Algorithm

1. Initialize the open list.
2. Initialize the closed list.
3. put the starting node on the open list
4. while the open list is not empty
   1) find the node with the leaf f on the open list. Call it 'q'
   2) pop 'q' off the open list.
   3) generate 'q''s successors.
   4) for each successor
      a) if successor is the goal, stop search successor.
         g = q.g + distance (successor. q)
         successor.h = distance from goal to successor
         successor.f = successor.g + successor.h

b) if a node with the same position as successor is in the open list which has a lower 'f' than successor, skip the successor.

c) if a node with the same position as successor is in the CLOSED list which has a lower 'f' than successor, skip this successor otherwise and the node to the open list.

5) end for

6) ~~end~~ push q on the closed list.

4 end while

Test Case.

1	2	x
4	5	3
7	8	6

initial

1	2	3
4	5	6
7	8	x

final

solved in 18 moves.

1	2	3
x	4	6
7	5	8

initial

1	2	3
4	5	6
7	8	x

Final

1	2	3		1	2	3		1	2	3		1	2	3
x	4	6	→	4	x	6	→	4	5	6	→	4	5	6
7	5	8		7	5	8		7	x	8		7	8	x

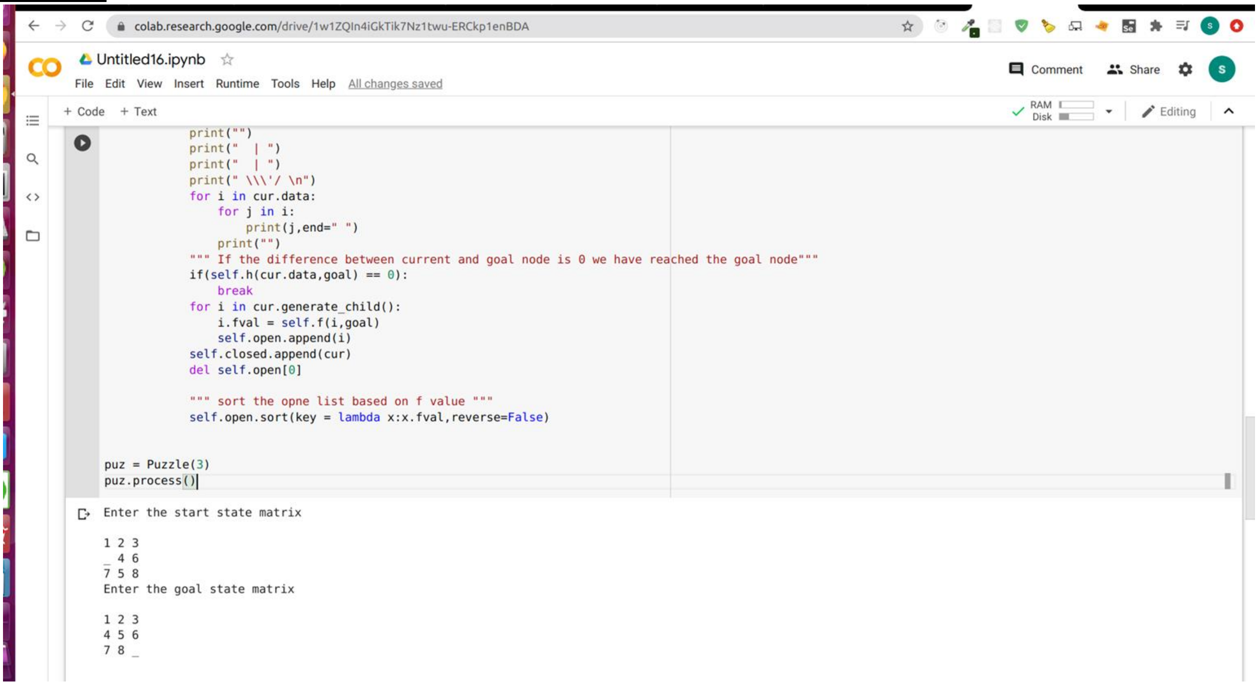
Conclusion:- Thus I understood and implement the 8 puzzle problem ~~and~~ using ~~8~~ A* algorithm.

CODE:

```
class Node:
def __init__(self,data,level,fval):
""" Initialize the node with the data, level of the node and the calculated fvalue """
self.data = data
self.level = level
self.fval = fval

def generate_child(self):
""" Generate child nodes from the given node by moving the blank space
either in the four directions {up,down,left,right} """
x,y = self.find(self.data,'_')
""" val_list contains position values for moving the blank space in either of
the 4 directions [up,down,left,right] respectively. """
val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
children = []
for i in val_list:
child = self.shuffle(self.data,x,y,i[0],i[1])
if child is not None:
child_node = Node(child,self.level+1,0)
children.append(child_node)
return children
def shuffle(self,puz,x1,y1,x2,y2):
""" Move the blank space in the given direction and if the position value are out
of limits the return None """
if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
temp_puz = []
temp_puz = self.copy(puz)
temp = temp_puz[x2][y2]
temp_puz[x2][y2] = temp_puz[x1][y1]
temp_puz[x1][y1] = temp
return temp_puz
else:
return None

def copy(self,root):
""" Copy function to create a similar matrix of the given node"""
temp = []
for i in root:
t = []
for j in i:
t.append(j)
temp.append(t)
return temp
def find(self,puz,x):
""" Specifically used to find the position of the blank space """
for i in range(0,len(self.data)):
for j in range(0,len(self.data)):
if puz[i][j] == x:
return i,j



class Puzzle:
def __init__(self,size):
""" Initialize the puzzle size by the specified size,open and closed lists to empty """
self.n = size
self.open = []
self.closed = []

def accept(self):
""" Accepts the puzzle from the user """
```

```
puz = []
for i in range(0,self.n):
temp = input().split(" ")
puz.append(temp)
return puz

def f(self,start,goal):
""" Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """
return self.h(start.data,goal)+start.level

def h(self,start,goal):
""" Calculates the different between the given puzzles """
temp = 0
for i in range(0,self.n):
for j in range(0,self.n):
if start[i][j] != goal[i][j] and start[i][j] != '_':
temp += 1
return temp

def process(self):
""" Accept Start and Goal Puzzle state"""
print("Enter the start state matrix \n")
start = self.accept()
print("Enter the goal state matrix \n")
goal = self.accept()

start = Node(start,0,0)
start.fval = self.f(start,goal)
""" Put the start node in the open list"""
self.open.append(start)
print("\n\n")
while True:
cur = self.open[0]
print("")
print(" | ")
print(" | ")
print(" \\\'/ \n")
for i in cur.data:
for j in i:
print(j,end=" ")
print("")
""" If the difference between current and goal node is 0 we have reached the goal node"""
if(self.h(cur.data,goal) == 0):
break
for i in cur.generate_child():
i.fval = self.f(i,goal)
self.open.append(i)
self.closed.append(cur)
del self.open[0]

""" sort the opne list based on f value """
self.open.sort(key = lambda x:x.fval,reverse=False)



puz = Puzzle(3)
puz.process()
```

OUTPUT:



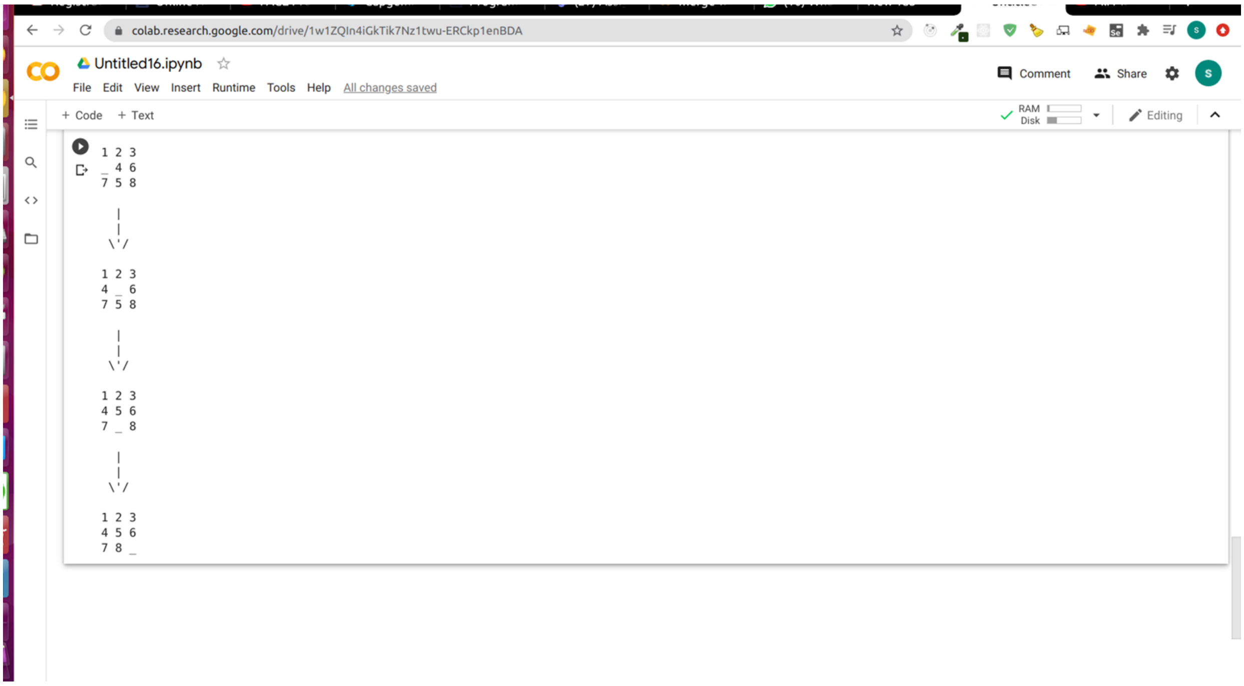
colab.research.google.com/drive/1w1ZQIn4iGkTik7Nz1twu-ERCkp1enBDA

Untitled16.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            """ If the difference between current and goal node is 0 we have reached the goal node"""
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]

            """ sort the opne list based on f value """
            self.open.sort(key = lambda x:x.fval,reverse=False)


puz = Puzzle(3)
puz.process()
```

```
Enter the start state matrix

1 2 3
_ 4 6
7 5 8
Enter the goal state matrix

1 2 3
4 5 6
7 8 _
```



colab.research.google.com/drive/1w1ZQIn4iGkTik7Nz1twu-ERCkp1enBDA

Untitled16.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
1 2 3
_ 4 6
7 5 8

  |
  |
 \'/

1 2 3
4 _ 6
7 5 8

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |
  |
 \'/

1 2 3
4 5 6
7 8 _
```