Greedy_Best_first_Search

June 18, 2021

[1]:
```python
import numpy as np
from queue import PriorityQueue as pq
import pprint as pp

np.random.seed(0)
class Node:
    '''Node
        arg : Val is the actual value inside the Node
        heuristic : Heuristic is the value from current node to goal Node
        parent : parent is parent of current Node '''
    def __init__(self,val):
        self.val = val
        self.parent = None
        # self.child = None

    def printTree(self):
        print(self.parent.val)

# A simple implementation of Priority Queue
# using Queue.
class PriorityQueue(object):
    def __init__(self):
        self.queue = []

    def __str__(self):
            return ' '.join([str(i) for i in self.queue])

        # for checking if the queue is empty
    def isEmpty(self):
        if  (len(self.queue) == 0):
            return True
        else:
            return False


        # 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)):
                # print("Queue {} Queue {}".format(i,self.queue[i][1]))  ##␣
↪remove according to heuristic min
                if self.queue[i][1] < self.queue[max][1]:
                    max = i
            item = self.queue[max]
            # print("Delete item ",item)
            del self.queue[max]
            return item
        except IndexError:
            print()
            exit()


class Problem_Environment:
    '''
    Problem Environment is the Environment of problem which contain the spefic
    Dim of Array in which have one Gaol State and Start state

    Matrix : Specfic of DIM Array
    start_nonde : is simpel start node anywhere in the Matrix
    goal_node  : is simple Gaol Node Anywhere in the Matrix
    '''

    def __init__(self,matrix,start_node , goal_node):
        self.matrix = matrix
        self.start_node = start_node
        self.goal_node = goal_node
        self.heuristic_table = {}
        self.pathCost = 0


    def  Euclidean_distance(self,from_node ,to_node):
        # intializing points in
        # numpy arrays
        point1 = np.array((from_node[0], from_node[1]))
        point2 = np.array((to_node[0],to_node[1]))

        # calculating Euclidean distance
        # using linalg.norm()
```

```python
        dist = np.linalg.norm(point1 - point2)

        # printing Euclidean distance
        return dist

    def find_heuristic(self):

        for row in range(0,len(self.matrix)):

            for col in range(0,len(self.matrix)):
                current_node = (row,col)
                dist = self.Euclidean_distance(current_node,self.goal_node)
                # point1 = np.array((current_node[0], current_node[1]))
                # point2 = np.array((goal_node[0],goal_node[1]))

                # # calculating Euclidean distance
                # # using linalg.norm()
                # dist = np.linalg.norm(point1 - point2)

                self.heuristic_table[current_node] = dist
                # # printing Euclidean distance

        # pp.pprint(self.heuristic_table)




    def Validation_Moves(self,action_dict):
        '''
        take dictionary with Wrong Move and return update dict with valid moves
        '''
        # print("Action : {} ".format(action_dict))

        action = ["right","left","up","down"]
        for key in action:
            if action_dict.get(key)[0] >=0 and  action_dict.get(key)[0] < 5 and␣
↪action_dict.get(key)[1] >=0 and  action_dict.get(key)[1] < 5:
                pass
            else:
                action_dict[key] = 0
        # print("Updated Action {} ".format(action_dict))

        return action_dict


    def Possible_action(self,index_node):
```

```
        '''
        Possible_action()
            index Node Four Possible action according to the current index Node
            Can Move Right
            Can Move left
            Can Move Down
            Can Move Up
            '''

        # initialize by 0
        action = {
            "right" : 0,
            'left' : 0,
            'up':0,
            'down':0
        }

        # print("Current Node {} have index {} ".format(self.
↪matrix[index_node[0],index_node[1]],index_node))

        if index_node[0] >=0 and  index_node[0] < 5 and index_node[1] >=0 and ␣
↪index_node[1] < 5 :
            ## jus to check coming Index node is valid or not
            # print("Valid State {} ".format(index_node))

            # Moves
            right_move = (index_node[0],index_node[1]+1)
            left_move = (index_node[0],index_node[1]-1)
            up_move =  (index_node[0]-1,index_node[1])
            down_move =  (index_node[0]+1,index_node[1])

            ## append in the dict
            action["right"] = right_move
            action["left"] = left_move
            action["up"] = up_move
            action["down"] = down_move

            action = self.Validation_Moves(action)
            return action

        else:
            return "Current index is not Valid"

    def GreedyBestFirstSearch(self, pQ):

        # print("Current Node {} have index {}".format(self.matrix[self.
↪start_node[0],self.start_node[0]] , self.start_node))
```

```
        self.find_heuristic()

        priorityQueue = pQ.insert( (self.start_node,
                                    self.heuristic_table[self.start_node]
                                    )
                                )
        cont=0
        while (pQ.isEmpty() is  False):

            current_node = pQ.delete()
            print("Current Node {} has heuristic {} ".
↪format(current_node[0],current_node[1]))
            self.pathCost +=1
            if current_node[0] == self.goal_node:
                return "Reached goal state {} and Cost is {} ".
↪format(current_node[0],self.pathCost)

            successor = self.Possible_action(current_node[0]) ## sending only␣
↪node not heuristic

            pp.pprint("Successor {} ".format(successor)) ## dict

            ## append into the Queue
            for key in successor.keys():

                if successor[key] !=0:
                    # print("Child {}".format(successor[key]))
                    # print("Current Node loc : ",)
                    # print(" successor location {} and heuristic {} ".
↪format(current_node[0],self.heuristic_table[current_node[0]])

                    pQ.insert( (successor[key],
                                self.heuristic_table[successor[key]]
                                )
                            )

            # break
            cont+=1

            # if cont>3:
            #     break


            print("Queue :",pQ.__str__())
```

```python
matrix = np.arange(25).reshape(5,5)
# print(matrix)
p = Problem_Environment(matrix,start_node =(0,0),goal_node=(4,4))
pQ = PriorityQueue()

# p.Possible_action(index_node=(0,0))
p.GreedyBestFirstSearch(pQ)
```

```
Current Node (0, 0) has heuristic 5.656854249492381
"Successor {'right': (0, 1), 'left': 0, 'up': 0, 'down': (1, 0)} "
Queue : ((0, 1), 5.0) ((1, 0), 5.0)
Current Node (0, 1) has heuristic 5.0
"Successor {'right': (0, 2), 'left': (0, 0), 'up': 0, 'down': (1, 1)} "
Queue : ((1, 0), 5.0) ((0, 2), 4.47213595499958) ((0, 0), 5.656854249492381)
((1, 1), 4.242640687119285)
Current Node (1, 1) has heuristic 4.242640687119285
"Successor {'right': (1, 2), 'left': (1, 0), 'up': (0, 1), 'down': (2, 1)} "
Queue : ((1, 0), 5.0) ((0, 2), 4.47213595499958) ((0, 0), 5.656854249492381)
((1, 2), 3.605551275463989) ((1, 0), 5.0) ((0, 1), 5.0) ((2, 1),
3.605551275463989)
Current Node (1, 2) has heuristic 3.605551275463989
"Successor {'right': (1, 3), 'left': (1, 1), 'up': (0, 2), 'down': (2, 2)} "
Queue : ((1, 0), 5.0) ((0, 2), 4.47213595499958) ((0, 0), 5.656854249492381)
((1, 0), 5.0) ((0, 1), 5.0) ((2, 1), 3.605551275463989) ((1, 3),
3.1622776601683795) ((1, 1), 4.242640687119285) ((0, 2), 4.47213595499958) ((2,
2), 2.8284271247461903)
Current Node (2, 2) has heuristic 2.8284271247461903
"Successor {'right': (2, 3), 'left': (2, 1), 'up': (1, 2), 'down': (3, 2)} "
Queue : ((1, 0), 5.0) ((0, 2), 4.47213595499958) ((0, 0), 5.656854249492381)
((1, 0), 5.0) ((0, 1), 5.0) ((2, 1), 3.605551275463989) ((1, 3),
3.1622776601683795) ((1, 1), 4.242640687119285) ((0, 2), 4.47213595499958) ((2,
```

```
3), 2.23606797749979) ((2, 1), 3.605551275463989) ((1, 2), 3.605551275463989)
((3, 2), 2.23606797749979)
Current Node (2, 3) has heuristic 2.23606797749979
"Successor {'right': (2, 4), 'left': (2, 2), 'up': (1, 3), 'down': (3, 3)} "
Queue : ((1, 0), 5.0) ((0, 2), 4.47213595499958) ((0, 0), 5.656854249492381)
((1, 0), 5.0) ((0, 1), 5.0) ((2, 1), 3.605551275463989) ((1, 3),
3.1622776601683795) ((1, 1), 4.242640687119285) ((0, 2), 4.47213595499958) ((2,
1), 3.605551275463989) ((1, 2), 3.605551275463989) ((3, 2), 2.23606797749979)
((2, 4), 2.0) ((2, 2), 2.8284271247461903) ((1, 3), 3.1622776601683795) ((3, 3),
1.4142135623730951)
Current Node (3, 3) has heuristic 1.4142135623730951
"Successor {'right': (3, 4), 'left': (3, 2), 'up': (2, 3), 'down': (4, 3)} "
Queue : ((1, 0), 5.0) ((0, 2), 4.47213595499958) ((0, 0), 5.656854249492381)
((1, 0), 5.0) ((0, 1), 5.0) ((2, 1), 3.605551275463989) ((1, 3),
3.1622776601683795) ((1, 1), 4.242640687119285) ((0, 2), 4.47213595499958) ((2,
1), 3.605551275463989) ((1, 2), 3.605551275463989) ((3, 2), 2.23606797749979)
((2, 4), 2.0) ((2, 2), 2.8284271247461903) ((1, 3), 3.1622776601683795) ((3, 4),
1.0) ((3, 2), 2.23606797749979) ((2, 3), 2.23606797749979) ((4, 3), 1.0)
Current Node (3, 4) has heuristic 1.0
"Successor {'right': 0, 'left': (3, 3), 'up': (2, 4), 'down': (4, 4)} "
Queue : ((1, 0), 5.0) ((0, 2), 4.47213595499958) ((0, 0), 5.656854249492381)
((1, 0), 5.0) ((0, 1), 5.0) ((2, 1), 3.605551275463989) ((1, 3),
3.1622776601683795) ((1, 1), 4.242640687119285) ((0, 2), 4.47213595499958) ((2,
1), 3.605551275463989) ((1, 2), 3.605551275463989) ((3, 2), 2.23606797749979)
((2, 4), 2.0) ((2, 2), 2.8284271247461903) ((1, 3), 3.1622776601683795) ((3, 2),
2.23606797749979) ((2, 3), 2.23606797749979) ((4, 3), 1.0) ((3, 3),
1.4142135623730951) ((2, 4), 2.0) ((4, 4), 0.0)
Current Node (4, 4) has heuristic 0.0
```

[1]: 'Reached goal state (4, 4) and Cost is 9 '

[ ]:

[ ]:

[ ]: