Assignment-03-VacuumCleaner

June 18, 2021

[183]:
```
import random
import numpy as np
import pprint as p

from graphviz import Digraph
# from graphviz import Digraph

def visualize_tree(tree):
    if tree is None:
        return 'Nothing in the tree!'

    def add_nodes_edges(tree, dot=None):

        if dot is None:
            dot = Digraph()
            dot.attr('node', shape='circle')
            dot.node(name=str(tree), label=str(tree.val))

        for child in [tree.left, tree.right]: # do for all children
            if child is not None:
                if child == tree.left: dot.attr('node', shape='circle',␣
↪style='filled',
                fillcolor='lightblue')
                if child == tree.right: dot.attr('node', shape='doublecircle',␣
↪style='filled',
                fillcolor='seashell')
                if child == tree.down: dot.attr('node', shape='doublecircle',␣
↪style='filled',
                fillcolor='seashell')
                if child == tree.up: dot.attr('node', shape='doublecircle',␣
↪style='filled',
                fillcolor='seashell')
                dot.node(name=str(child) ,label=str(child.val))
                dot.edge(str(tree), str(child))
                dot = add_nodes_edges(child, dot=dot) # recursive call
        return dot
    dot = add_nodes_edges(tree)
```

```
    display(dot)

class Node:

    ## Node Classs has left and right option or pointer and val is the node␣
 ↪value
    def __init__(self,val):
        self.val = val
        self.left = None
        self.right=None
        self.down = None
        self.up = None


class TreeNode(Node):

    def __init__(self,val,parent=None):
        super().__init__(val)
        self.parent = parent

    def insert(self,val):

        if val[1] =="R":
            if self.right is None:
                newNode =  TreeNode(val,parent=self)
                self.right = newNode
            else:
                    self.right.insert(val)
        elif val[1] == "L":
            if self.left is None:

                newNode = TreeNode(val,parent=self)
                self.left = newNode
            else:
                self.left.insert(val)
        elif val[1] == "D":
            if self.down is None:

                newNode = TreeNode(val,parent=self)
                self.down = newNode
            else:
                self.down.insert(val)
        elif val[1] == "U":
            if self.up is None:

                newNode = TreeNode(val,parent=self)
```

```
                self.up = newNode
            else:
                self.up.insert(val)


        return

    def bfs(self):

        visited = [self]
        while len(visited)>0:

            current = visited.pop(0)


            print(current.val)

            if self.left :
                visited.append(current.left)


            if self.right:
                visited.append(current.right)

    def BreathFirstSearch(self,goalNode):


        Frontier  = [self]  # Queue # fifo Data struck use
        pathCost = 0
        visited_Nodes = []

        while len(Frontier)>0:

            current = Frontier.pop(0)

            if current.val not in visited_Nodes:  # avoid to repeat the node
                pathCost+=1
                if current.val == goalNode: ## if goal state reach

                    return "Reached Goal "+ str(current.val) +" " +"Path Cost :
↪" +str(pathCost)

                if current.left:
                    Frontier.append(current.left)

                if current.right:
                    Frontier.append(current.right)
```

```
    def DepthFirstSearch(self,goalNode):


        Frontier  = [self]  # Stack used lifo data strucutre
        pathCost = 0
        visited_Nodes = []

        while len(Frontier)>0:

            current = Frontier.pop()

            if current.val not in visited_Nodes:  # avoid to repeat the node
                pathCost+=1
                if current.val == goalNode: ## if goal state reach

                    return "Reached Goal "+ str(current.val) +" " +"Path Cost :␣
 ↪" +str(pathCost)

                if current.left:
                    Frontier.append(current.left)

                if current.right:
                    Frontier.append(current.right)









class Environment(TreeNode):

    def __init__(self,row=3,col=3):
       # self.row = row  # number of row size of matrix
       # self.col = col # number of column size of matrix

        self.matrix = matrix
#         print(self.locationCondition)
        self.initialState = None
        self.GoalState = None
        self.pathCost = 0
```

```
        ## Possible places Neighbour

    def set_initailState(self):
        print("Enter the Initial State of Agent :")
#         x = input("Enter the location of x value for 5X5 matrix (x,y) :")
#         y = input("Enter the location of y value for 5X5 matrix (x,y) :")
        x= 0
        y= 0
        self.initialState = [(x,y),"root"]

#         TreeNode(self.initialState)

    def set_GoalState(self):
        print("Enter the goal State of Agent")
#         x = input("Enter the location of x value for 5X5 matrix (x,y) :")
#         y = input("Enter the location of y value for 5X5 matrix (x,y) :")
        x,y = 3,3
        self.GoalState = (x,y)


    def validAction(self,actions ):
        ## valid actions
        validActionofAgent = []
        for state in actions:
#             print("Current State : ",state)

            if state[0][0]>=0 and state[0][0]<5 and state[0][1]>=0 and␣
 ↪state[0][1]<5:
#                 print("Valid state : ",state)
                validActionofAgent.append(state)
#                 print(state)

        return validActionofAgent

    def PossibleAction(self,state):

        print("You can Move only those location from : ",state)

        PossibleState = [ [(state[0][0],state[0][1]+1),"R"],␣
 ↪[(state[0][0],state[0][1]-1),"L"],[ (state[0][0]+1,state[0][1]),"D"]␣
 ↪,[(state[0][0]-1,state[0][1]) , "U"]]
        ## refine the action into the valid action
#         print("Possible Moves : ",PossibleState)
        validAction = self.validAction(PossibleState)
        print("Valid Moves : ",validAction)
        return validAction
```

```
    def ShowTree(self):

        print(self.root)
        print("Right ",self.right)
        print("down : ",self.down)
        print("left : ",self.left)
        print("up : ",self.up)

    def bfs(self,environment,t1):

        print("Our Environment  : " ,environment)
        print("Our Initial State : ",self.initialState)
        print("Our Goal State : ",self.GoalState)
        frontier = [self.initialState]  ## Queue
        visited = []  ## avoid to re visit the state

        while len(frontier) >0:

            currentState = frontier.pop(0)
            print("current state : ",currentState)

            if currentState[0] == self.GoalState:
                    print("Found it Goal state " +"Path code is "+str(self.
↪pathCost) )
                    return

            if currentState not in visited:
                self.pathCost +=1
                visited.append(currentState)  ## to set this node is visited
                print("Insert Node : ",currentState)
                t1.insert(currentState)

                moves = self.PossibleAction(currentState) ## return possible␣
↪and valid moves

                for state in moves:  ## push the next state in the Queue
                    frontier.append(state)


    ## search algorithm

    def searchAlgorithm(self,algo , environment ):

        if algo == "bfs" or algo == "BFS":
            self.set_GoalState()
            self.set_GoalState()
```

```
            t1 = TreeNode(self.initialState)
            self.bfs(environment,t1)
            t1.bfs()
            visualize_tree(t1)
        else:
            return "Wait"
```

[184]:
```
matrix = [[1,2,3,4,5],
          [1,2,"B",3,5],
          [1,2,3,"f",5],
          [1,"d",3,4,5],
          [1,2,3,"D",5]]

e1 = Environment(matrix)
e1.set_initailState()
e1.set_GoalState()
e1.searchAlgorithm("bfs",matrix)
```

```
Enter the Initial State of Agent :
Enter the goal State of Agent
Enter the goal State of Agent
Enter the goal State of Agent
Our Environment  :  [[1, 2, 3, 4, 5], [1, 2, 'B', 3, 5], [1, 2, 3, 'f', 5], [1,
'd', 3, 4, 5], [1, 2, 3, 'D', 5]]
Our Initial State :  [(0, 0), 'root']
Our Goal State :  (3, 3)
current state :  [(0, 0), 'root']
Insert Node :  [(0, 0), 'root']
You can Move only those location from :  [(0, 0), 'root']
Valid Moves :  [[(0, 1), 'R'], [(1, 0), 'D']]
current state :  [(0, 1), 'R']
Insert Node :  [(0, 1), 'R']
You can Move only those location from :  [(0, 1), 'R']
Valid Moves :  [[(0, 2), 'R'], [(0, 0), 'L'], [(1, 1), 'D']]
current state :  [(1, 0), 'D']
Insert Node :  [(1, 0), 'D']
You can Move only those location from :  [(1, 0), 'D']
Valid Moves :  [[(1, 1), 'R'], [(2, 0), 'D'], [(0, 0), 'U']]
current state :  [(0, 2), 'R']
Insert Node :  [(0, 2), 'R']
You can Move only those location from :  [(0, 2), 'R']
Valid Moves :  [[(0, 3), 'R'], [(0, 1), 'L'], [(1, 2), 'D']]
current state :  [(0, 0), 'L']
Insert Node :  [(0, 0), 'L']
You can Move only those location from :  [(0, 0), 'L']
Valid Moves :  [[(0, 1), 'R'], [(1, 0), 'D']]
```

```
current state :  [(1, 1), 'D']
Insert Node :  [(1, 1), 'D']
You can Move only those location from :  [(1, 1), 'D']
Valid Moves :  [[(1, 2), 'R'], [(1, 0), 'L'], [(2, 1), 'D'], [(0, 1), 'U']]
current state :  [(1, 1), 'R']
Insert Node :  [(1, 1), 'R']
You can Move only those location from :  [(1, 1), 'R']
Valid Moves :  [[(1, 2), 'R'], [(1, 0), 'L'], [(2, 1), 'D'], [(0, 1), 'U']]
current state :  [(2, 0), 'D']
Insert Node :  [(2, 0), 'D']
You can Move only those location from :  [(2, 0), 'D']
Valid Moves :  [[(2, 1), 'R'], [(3, 0), 'D'], [(1, 0), 'U']]
current state :  [(0, 0), 'U']
Insert Node :  [(0, 0), 'U']
You can Move only those location from :  [(0, 0), 'U']
Valid Moves :  [[(0, 1), 'R'], [(1, 0), 'D']]
current state :  [(0, 3), 'R']
Insert Node :  [(0, 3), 'R']
You can Move only those location from :  [(0, 3), 'R']
Valid Moves :  [[(0, 4), 'R'], [(0, 2), 'L'], [(1, 3), 'D']]
current state :  [(0, 1), 'L']
Insert Node :  [(0, 1), 'L']
You can Move only those location from :  [(0, 1), 'L']
Valid Moves :  [[(0, 2), 'R'], [(0, 0), 'L'], [(1, 1), 'D']]
current state :  [(1, 2), 'D']
Insert Node :  [(1, 2), 'D']
You can Move only those location from :  [(1, 2), 'D']
Valid Moves :  [[(1, 3), 'R'], [(1, 1), 'L'], [(2, 2), 'D'], [(0, 2), 'U']]
current state :  [(0, 1), 'R']
current state :  [(1, 0), 'D']
current state :  [(1, 2), 'R']
Insert Node :  [(1, 2), 'R']
You can Move only those location from :  [(1, 2), 'R']
Valid Moves :  [[(1, 3), 'R'], [(1, 1), 'L'], [(2, 2), 'D'], [(0, 2), 'U']]
current state :  [(1, 0), 'L']
Insert Node :  [(1, 0), 'L']
You can Move only those location from :  [(1, 0), 'L']
Valid Moves :  [[(1, 1), 'R'], [(2, 0), 'D'], [(0, 0), 'U']]
current state :  [(2, 1), 'D']
Insert Node :  [(2, 1), 'D']
You can Move only those location from :  [(2, 1), 'D']
Valid Moves :  [[(2, 2), 'R'], [(2, 0), 'L'], [(3, 1), 'D'], [(1, 1), 'U']]
current state :  [(0, 1), 'U']
Insert Node :  [(0, 1), 'U']
You can Move only those location from :  [(0, 1), 'U']
Valid Moves :  [[(0, 2), 'R'], [(0, 0), 'L'], [(1, 1), 'D']]
current state :  [(1, 2), 'R']
current state :  [(1, 0), 'L']
```

```
current state :  [(2, 1), 'D']
current state :  [(0, 1), 'U']
current state :  [(2, 1), 'R']
Insert Node :  [(2, 1), 'R']
You can Move only those location from :  [(2, 1), 'R']
Valid Moves :  [[(2, 2), 'R'], [(2, 0), 'L'], [(3, 1), 'D'], [(1, 1), 'U']]
current state :  [(3, 0), 'D']
Insert Node :  [(3, 0), 'D']
You can Move only those location from :  [(3, 0), 'D']
Valid Moves :  [[(3, 1), 'R'], [(4, 0), 'D'], [(2, 0), 'U']]
current state :  [(1, 0), 'U']
Insert Node :  [(1, 0), 'U']
You can Move only those location from :  [(1, 0), 'U']
Valid Moves :  [[(1, 1), 'R'], [(2, 0), 'D'], [(0, 0), 'U']]
current state :  [(0, 1), 'R']
current state :  [(1, 0), 'D']
current state :  [(0, 4), 'R']
Insert Node :  [(0, 4), 'R']
You can Move only those location from :  [(0, 4), 'R']
Valid Moves :  [[(0, 3), 'L'], [(1, 4), 'D']]
current state :  [(0, 2), 'L']
Insert Node :  [(0, 2), 'L']
You can Move only those location from :  [(0, 2), 'L']
Valid Moves :  [[(0, 3), 'R'], [(0, 1), 'L'], [(1, 2), 'D']]
current state :  [(1, 3), 'D']
Insert Node :  [(1, 3), 'D']
You can Move only those location from :  [(1, 3), 'D']
Valid Moves :  [[(1, 4), 'R'], [(1, 2), 'L'], [(2, 3), 'D'], [(0, 3), 'U']]
current state :  [(0, 2), 'R']
current state :  [(0, 0), 'L']
current state :  [(1, 1), 'D']
current state :  [(1, 3), 'R']
Insert Node :  [(1, 3), 'R']
You can Move only those location from :  [(1, 3), 'R']
Valid Moves :  [[(1, 4), 'R'], [(1, 2), 'L'], [(2, 3), 'D'], [(0, 3), 'U']]
current state :  [(1, 1), 'L']
Insert Node :  [(1, 1), 'L']
You can Move only those location from :  [(1, 1), 'L']
Valid Moves :  [[(1, 2), 'R'], [(1, 0), 'L'], [(2, 1), 'D'], [(0, 1), 'U']]
current state :  [(2, 2), 'D']
Insert Node :  [(2, 2), 'D']
You can Move only those location from :  [(2, 2), 'D']
Valid Moves :  [[(2, 3), 'R'], [(2, 1), 'L'], [(3, 2), 'D'], [(1, 2), 'U']]
current state :  [(0, 2), 'U']
Insert Node :  [(0, 2), 'U']
You can Move only those location from :  [(0, 2), 'U']
Valid Moves :  [[(0, 3), 'R'], [(0, 1), 'L'], [(1, 2), 'D']]
current state :  [(1, 3), 'R']
```

```
current state :  [(1, 1), 'L']
current state :  [(2, 2), 'D']
current state :  [(0, 2), 'U']
current state :  [(1, 1), 'R']
current state :  [(2, 0), 'D']
current state :  [(0, 0), 'U']
current state :  [(2, 2), 'R']
Insert Node :  [(2, 2), 'R']
You can Move only those location from :  [(2, 2), 'R']
Valid Moves :  [[(2, 3), 'R'], [(2, 1), 'L'], [(3, 2), 'D'], [(1, 2), 'U']]
current state :  [(2, 0), 'L']
Insert Node :  [(2, 0), 'L']
You can Move only those location from :  [(2, 0), 'L']
Valid Moves :  [[(2, 1), 'R'], [(3, 0), 'D'], [(1, 0), 'U']]
current state :  [(3, 1), 'D']
Insert Node :  [(3, 1), 'D']
You can Move only those location from :  [(3, 1), 'D']
Valid Moves :  [[(3, 2), 'R'], [(3, 0), 'L'], [(4, 1), 'D'], [(2, 1), 'U']]
current state :  [(1, 1), 'U']
Insert Node :  [(1, 1), 'U']
You can Move only those location from :  [(1, 1), 'U']
Valid Moves :  [[(1, 2), 'R'], [(1, 0), 'L'], [(2, 1), 'D'], [(0, 1), 'U']]
current state :  [(0, 2), 'R']
current state :  [(0, 0), 'L']
current state :  [(1, 1), 'D']
current state :  [(2, 2), 'R']
current state :  [(2, 0), 'L']
current state :  [(3, 1), 'D']
current state :  [(1, 1), 'U']
current state :  [(3, 1), 'R']
Insert Node :  [(3, 1), 'R']
You can Move only those location from :  [(3, 1), 'R']
Valid Moves :  [[(3, 2), 'R'], [(3, 0), 'L'], [(4, 1), 'D'], [(2, 1), 'U']]
current state :  [(4, 0), 'D']
Insert Node :  [(4, 0), 'D']
You can Move only those location from :  [(4, 0), 'D']
Valid Moves :  [[(4, 1), 'R'], [(3, 0), 'U']]
current state :  [(2, 0), 'U']
Insert Node :  [(2, 0), 'U']
You can Move only those location from :  [(2, 0), 'U']
Valid Moves :  [[(2, 1), 'R'], [(3, 0), 'D'], [(1, 0), 'U']]
current state :  [(1, 1), 'R']
current state :  [(2, 0), 'D']
current state :  [(0, 0), 'U']
current state :  [(0, 3), 'L']
Insert Node :  [(0, 3), 'L']
You can Move only those location from :  [(0, 3), 'L']
Valid Moves :  [[(0, 4), 'R'], [(0, 2), 'L'], [(1, 3), 'D']]
```

```
current state :  [(1, 4), 'D']
Insert Node :  [(1, 4), 'D']
You can Move only those location from :  [(1, 4), 'D']
Valid Moves :  [[(1, 3), 'L'], [(2, 4), 'D'], [(0, 4), 'U']]
current state :  [(0, 3), 'R']
current state :  [(0, 1), 'L']
current state :  [(1, 2), 'D']
current state :  [(1, 4), 'R']
Insert Node :  [(1, 4), 'R']
You can Move only those location from :  [(1, 4), 'R']
Valid Moves :  [[(1, 3), 'L'], [(2, 4), 'D'], [(0, 4), 'U']]
current state :  [(1, 2), 'L']
Insert Node :  [(1, 2), 'L']
You can Move only those location from :  [(1, 2), 'L']
Valid Moves :  [[(1, 3), 'R'], [(1, 1), 'L'], [(2, 2), 'D'], [(0, 2), 'U']]
current state :  [(2, 3), 'D']
Insert Node :  [(2, 3), 'D']
You can Move only those location from :  [(2, 3), 'D']
Valid Moves :  [[(2, 4), 'R'], [(2, 2), 'L'], [(3, 3), 'D'], [(1, 3), 'U']]
current state :  [(0, 3), 'U']
Insert Node :  [(0, 3), 'U']
You can Move only those location from :  [(0, 3), 'U']
Valid Moves :  [[(0, 4), 'R'], [(0, 2), 'L'], [(1, 3), 'D']]
current state :  [(1, 4), 'R']
current state :  [(1, 2), 'L']
current state :  [(2, 3), 'D']
current state :  [(0, 3), 'U']
current state :  [(1, 2), 'R']
current state :  [(1, 0), 'L']
current state :  [(2, 1), 'D']
current state :  [(0, 1), 'U']
current state :  [(2, 3), 'R']
Insert Node :  [(2, 3), 'R']
You can Move only those location from :  [(2, 3), 'R']
Valid Moves :  [[(2, 4), 'R'], [(2, 2), 'L'], [(3, 3), 'D'], [(1, 3), 'U']]
current state :  [(2, 1), 'L']
Insert Node :  [(2, 1), 'L']
You can Move only those location from :  [(2, 1), 'L']
Valid Moves :  [[(2, 2), 'R'], [(2, 0), 'L'], [(3, 1), 'D'], [(1, 1), 'U']]
current state :  [(3, 2), 'D']
Insert Node :  [(3, 2), 'D']
You can Move only those location from :  [(3, 2), 'D']
Valid Moves :  [[(3, 3), 'R'], [(3, 1), 'L'], [(4, 2), 'D'], [(2, 2), 'U']]
current state :  [(1, 2), 'U']
Insert Node :  [(1, 2), 'U']
You can Move only those location from :  [(1, 2), 'U']
Valid Moves :  [[(1, 3), 'R'], [(1, 1), 'L'], [(2, 2), 'D'], [(0, 2), 'U']]
current state :  [(0, 3), 'R']
```

```
current state :  [(0, 1), 'L']
current state :  [(1, 2), 'D']
current state :  [(2, 3), 'R']
current state :  [(2, 1), 'L']
current state :  [(3, 2), 'D']
current state :  [(1, 2), 'U']
current state :  [(2, 1), 'R']
current state :  [(3, 0), 'D']
current state :  [(1, 0), 'U']
current state :  [(3, 2), 'R']
Insert Node :  [(3, 2), 'R']
You can Move only those location from :  [(3, 2), 'R']
Valid Moves :  [[(3, 3), 'R'], [(3, 1), 'L'], [(4, 2), 'D'], [(2, 2), 'U']]
current state :  [(3, 0), 'L']
Insert Node :  [(3, 0), 'L']
You can Move only those location from :  [(3, 0), 'L']
Valid Moves :  [[(3, 1), 'R'], [(4, 0), 'D'], [(2, 0), 'U']]
current state :  [(4, 1), 'D']
Insert Node :  [(4, 1), 'D']
You can Move only those location from :  [(4, 1), 'D']
Valid Moves :  [[(4, 2), 'R'], [(4, 0), 'L'], [(3, 1), 'U']]
current state :  [(2, 1), 'U']
Insert Node :  [(2, 1), 'U']
You can Move only those location from :  [(2, 1), 'U']
Valid Moves :  [[(2, 2), 'R'], [(2, 0), 'L'], [(3, 1), 'D'], [(1, 1), 'U']]
current state :  [(1, 2), 'R']
current state :  [(1, 0), 'L']
current state :  [(2, 1), 'D']
current state :  [(0, 1), 'U']
current state :  [(3, 2), 'R']
current state :  [(3, 0), 'L']
current state :  [(4, 1), 'D']
current state :  [(2, 1), 'U']
current state :  [(4, 1), 'R']
Insert Node :  [(4, 1), 'R']
You can Move only those location from :  [(4, 1), 'R']
Valid Moves :  [[(4, 2), 'R'], [(4, 0), 'L'], [(3, 1), 'U']]
current state :  [(3, 0), 'U']
Insert Node :  [(3, 0), 'U']
You can Move only those location from :  [(3, 0), 'U']
Valid Moves :  [[(3, 1), 'R'], [(4, 0), 'D'], [(2, 0), 'U']]
current state :  [(2, 1), 'R']
current state :  [(3, 0), 'D']
current state :  [(1, 0), 'U']
current state :  [(0, 4), 'R']
current state :  [(0, 2), 'L']
current state :  [(1, 3), 'D']
current state :  [(1, 3), 'L']
```

```
Insert Node :  [(1, 3), 'L']
You can Move only those location from :  [(1, 3), 'L']
Valid Moves :  [[(1, 4), 'R'], [(1, 2), 'L'], [(2, 3), 'D'], [(0, 3), 'U']]
current state :  [(2, 4), 'D']
Insert Node :  [(2, 4), 'D']
You can Move only those location from :  [(2, 4), 'D']
Valid Moves :  [[(2, 3), 'L'], [(3, 4), 'D'], [(1, 4), 'U']]
current state :  [(0, 4), 'U']
Insert Node :  [(0, 4), 'U']
You can Move only those location from :  [(0, 4), 'U']
Valid Moves :  [[(0, 3), 'L'], [(1, 4), 'D']]
current state :  [(1, 3), 'L']
current state :  [(2, 4), 'D']
current state :  [(0, 4), 'U']
current state :  [(1, 3), 'R']
current state :  [(1, 1), 'L']
current state :  [(2, 2), 'D']
current state :  [(0, 2), 'U']
current state :  [(2, 4), 'R']
Insert Node :  [(2, 4), 'R']
You can Move only those location from :  [(2, 4), 'R']
Valid Moves :  [[(2, 3), 'L'], [(3, 4), 'D'], [(1, 4), 'U']]
current state :  [(2, 2), 'L']
Insert Node :  [(2, 2), 'L']
You can Move only those location from :  [(2, 2), 'L']
Valid Moves :  [[(2, 3), 'R'], [(2, 1), 'L'], [(3, 2), 'D'], [(1, 2), 'U']]
current state :  [(3, 3), 'D']
Found it Goal state Path code is 54
[(0, 0), 'root']
[(0, 0), 'L']
[(0, 1), 'R']
[(0, 1), 'L']
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-184-2f44fc5db4a5> in <module>
      8 e1.set_initailState()
      9 e1.set_GoalState()
---> 10 e1.searchAlgorithm("bfs",matrix)

<ipython-input-183-6c17eb4d6a70> in searchAlgorithm(self, algo, environment)
    261             t1 = TreeNode(self.initialState)
    262             self.bfs(environment,t1)
--> 263             t1.bfs()
    264             visualize_tree(t1)
    265         else:
```

```
<ipython-input-183-6c17eb4d6a70> in bfs(self)
     93
     94
---> 95             print(current.val)
     96
     97             if self.left :

AttributeError: 'NoneType' object has no attribute 'val'
```

[44]:
```
-1>=0
```

[44]:
```
False
```

[13]:
```
matrix = [[1,2,3,4,5],
          [1,2,"B",3,5],
          [1,2,3,"f",5],
          [1,"d",3,4,5],
          [1,2,3,"D",5]]

p.pprint(matrix)
```

```
[[1, 2, 3, 4, 5],
 [1, 2, 'B', 3, 5],
 [1, 2, 3, 'f', 5],
 [1, 'd', 3, 4, 5],
 [1, 2, 3, 'D', 5]]
```

[151]:
```
def validAction(actions ):
    print("Comes for valid mov : ",actions)
    validAction_moves = []
    for state in actions:

        if state[0][0]>=0 and state[0][0]<5 and state[0][1]>=0 and␣
 ↪state[0][1]<5:
            print("Valid state : ",state)
            validAction_moves.append(state)
        else:
            continue

    return validAction_moves

def PossibleAction(state):

    print("You can Move only those location :")
    # N -> Neighbour
    PossibleState = [ [(state[0],state[1]+1),"right_N"] # Right Move
                     ,
```

```
                    [(state[0],state[1]-1),"left_N"] # left Move
                    ,
                     [ (state[0]+1,state[1]),"down_N"] # Down Move
                     ,
                     [(state[0]-1,state[1]) , "up_N"] # Up move
                    ]
#     ## refine the action into the valid action
    print("Actios (before valid ): ",PossibleState)
    moves = validAction(PossibleState)
    print(moves)
    return moves
```

[153]:
```
PossibleAction([(0,0),"root"])

# action = [[(1, 2), 'right_N'], [(0, -1), 'left_N'], [(2, 1), 'down_N'], [(0,␣
 ↪1), 'up_N']]
# validAction = []
# for state in action:
#     print("State : ",state)
#     print("do : ",state[0][0])
#     if state[0][0]>=0 and state[0][0]<5 and state[0][1]>=0 and state[0][1]<5:
#         print("Valid state : ",state)
#         validAction.append(state)
# print(validAction)
```

You can Move only those location :

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-153-ef4daa02c3bf> in <module>
----> 1 PossibleAction([(0,0),"root"])
      2
      3 # action = [[(1, 2), 'right_N'], [(0, -1), 'left_N'], [(2, 1),␣
 ↪'down_N'], [(0, 1), 'up_N']]
      4 # validAction = []
      5 # for state in action:

<ipython-input-151-b94f6e47735b> in PossibleAction(state)
     16     print("You can Move only those location :")
     17     # N -> Neighbour
---> 18     PossibleState = [ [(state[0],state[1]+1),"right_N"] # Right Move
     19                      ,
     20                     [(state[0],state[1]-1),"left_N"] # left Move
```

```
TypeError: can only concatenate str (not "int") to str
```

[161]:
```
state = [(1,1),"right"]
state[1]
```

[161]:
```
'right'
```

[135]:
```
list_ = [[(0, 1), 'right_N'], [(0, -1), 'left_N'], [(1, 0), 'down_N'], [(-1,␣
↪0), 'up_N']]
print([(-1, 0), 'up_N'] in list_)
```

```
True
```

[149]:
```
def PossibleAction(state):

    print("You can Move only those location from : ",state)

    PossibleState = [ [(state[0],state[1]+1),"right_N"],␣
↪[(state[0],state[1]-1),"left_N"],[ (state[0]+1,state[1]),"down_N"]␣
↪,[(state[0]-1,state[1]) , "up_N"]]
    ## refine the action into the valid action
    print("Possible Moves : ",PossibleState)
    validAction = self.validAction(PossibleState)
    print("Valid Moves : ",validAction)
    return validAction
```

[150]:
```
PossibleAction([(0,0),"Root"])
```

```
You can Move only those location from :  [(0, 0), 'Root']
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-150-68ead43242af> in <module>
----> 1 PossibleAction([(0,0),"Root"])

<ipython-input-149-d48e1560611b> in PossibleAction(state)
      3     print("You can Move only those location from : ",state)
      4
----> 5     PossibleState = [ [(state[0],state[1]+1),"right_N"],␣
↪[(state[0],state[1]-1),"left_N"],[ (state[0]+1,state[1]),"down_N"]␣
↪,[(state[0]-1,state[1]) , "up_N"]]
      6     ## refine the action into the valid action
      7     print("Possible Moves : ",PossibleState)

TypeError: can only concatenate str (not "int") to str
```

[ ]: