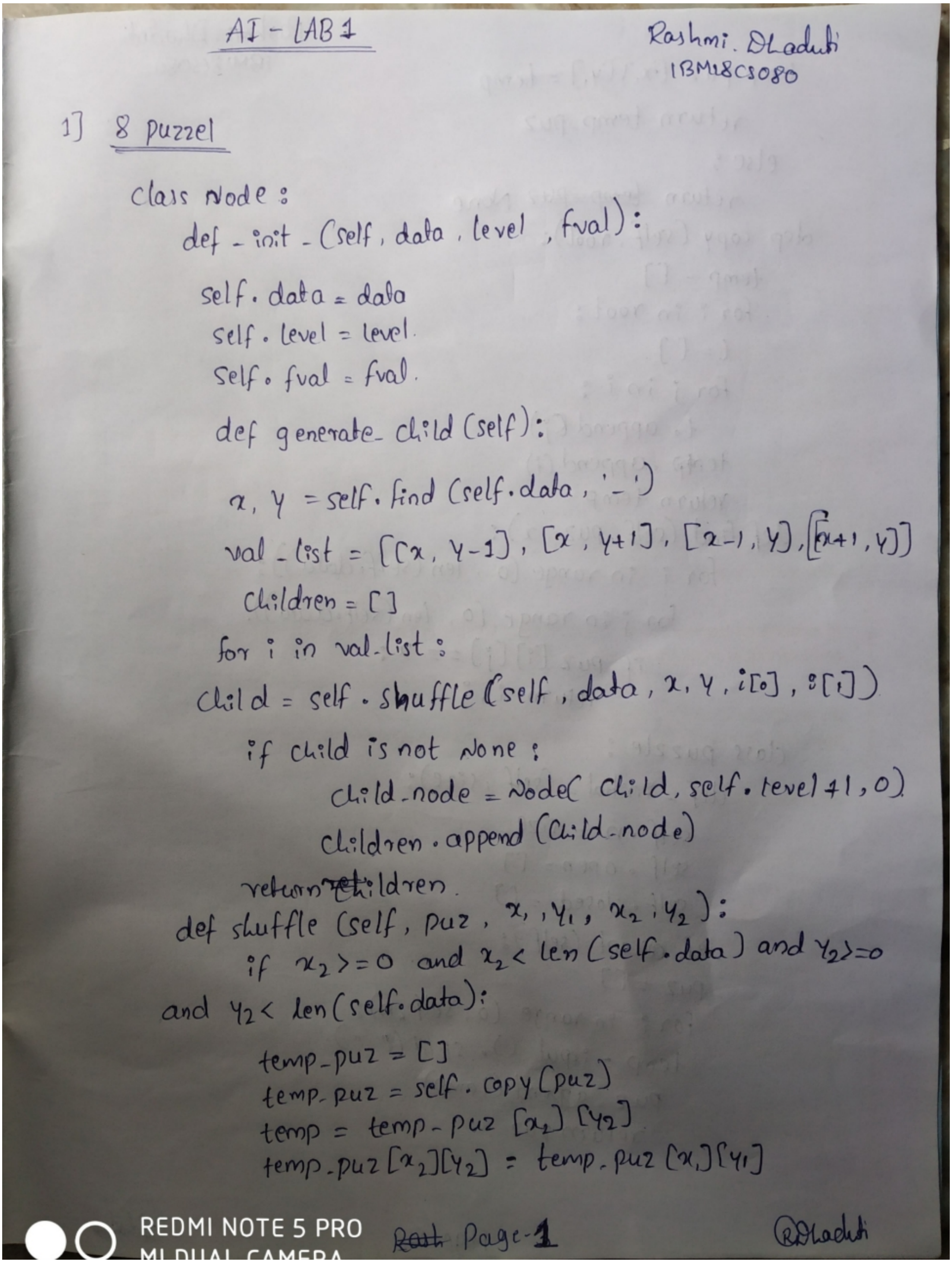


AI - LAB 1

Rashmi. DLaduti
1BM18CS080

1] 8 puzzel

```
class Node :
    def _init_ (self, data, level , fval):
      self. data = data
      self. level = level.
      self. fval = fval.
      def generate_ child (self):
       x, y = self. find (self.data, '-')
       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 ~~ret~~children.
    def shuffle (self, puz , x1 , y1 , x2 , y2 ):
        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]
```

~~Rash~~ Page-1 RDhaduti

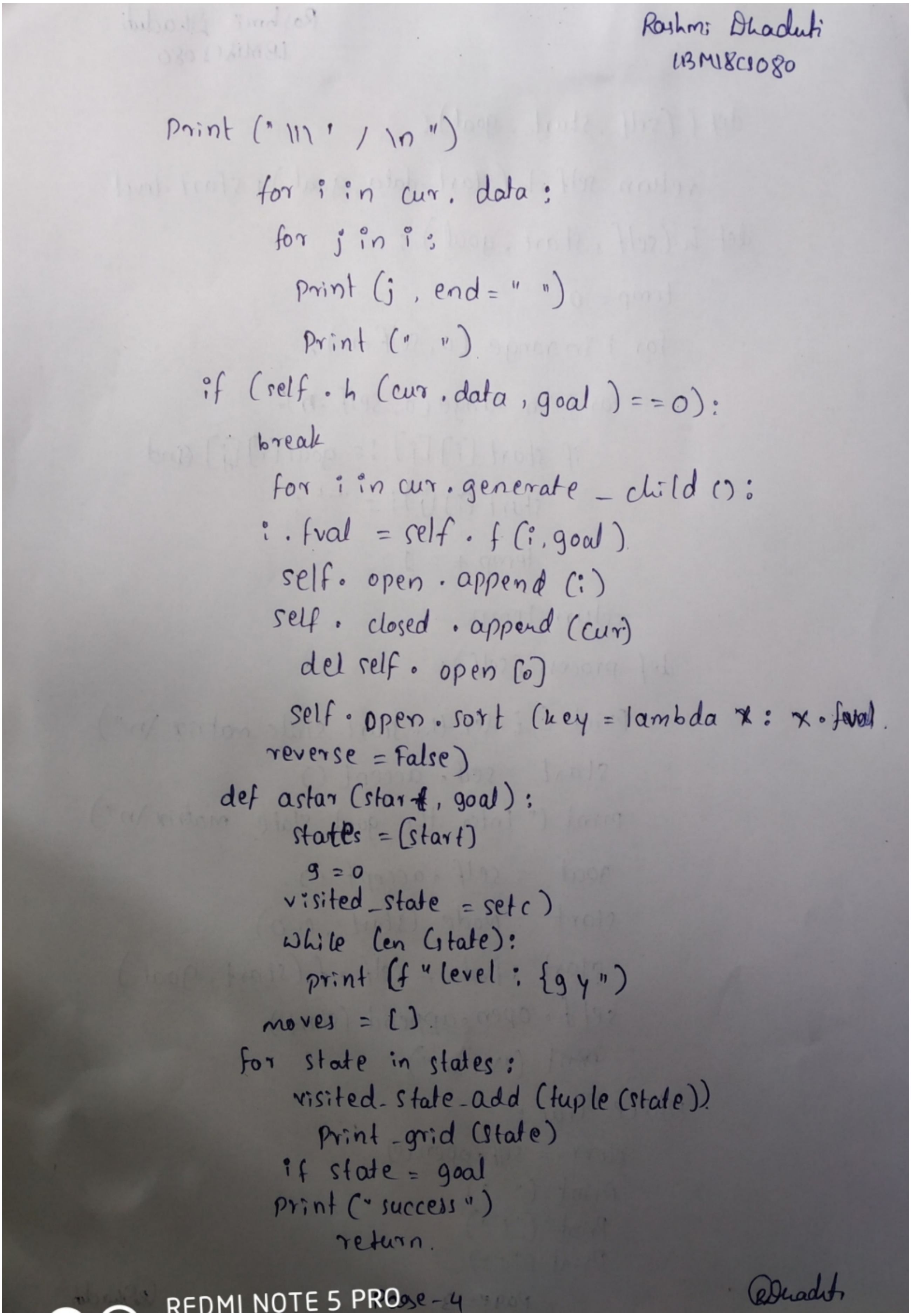
```
temp-puz [x,] [y,] = temp
    return temp-puz
  else :
    return ~~temp-puz~~ None.
dep copy (self, root):
    temp = []
    for i in root :
      t = []
      for j in i :
        t. append (j)
      temp .append (t)
      return temp
  def find (self, puz, x) :
      for i  in range (0 , len (self. data)):
          for j in range (0, len (self.data)):
             if puz [i] [j] = = x :
                return i, j

  class puzzle :
      def _initial_ (self , size);
         self . n = size
         self . open = []
         self . closed = []
      def accept (self):
        puz = []
         for i in range (0, self .n):
         temp = input (). split (" ")
           puz. append (temp)
            return puz .
```

RDhaduti

```
def f (self , start . goal):
    return self.h (start.data, goal) + start.level
def h (self , start , goal) :
      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);
          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)
           self. open.append (start).
            print ("\n\n ").
      while True :
           curr = self.open [0]
           Print ("  ")
           Print ("| ")
           Print ("| ")
```

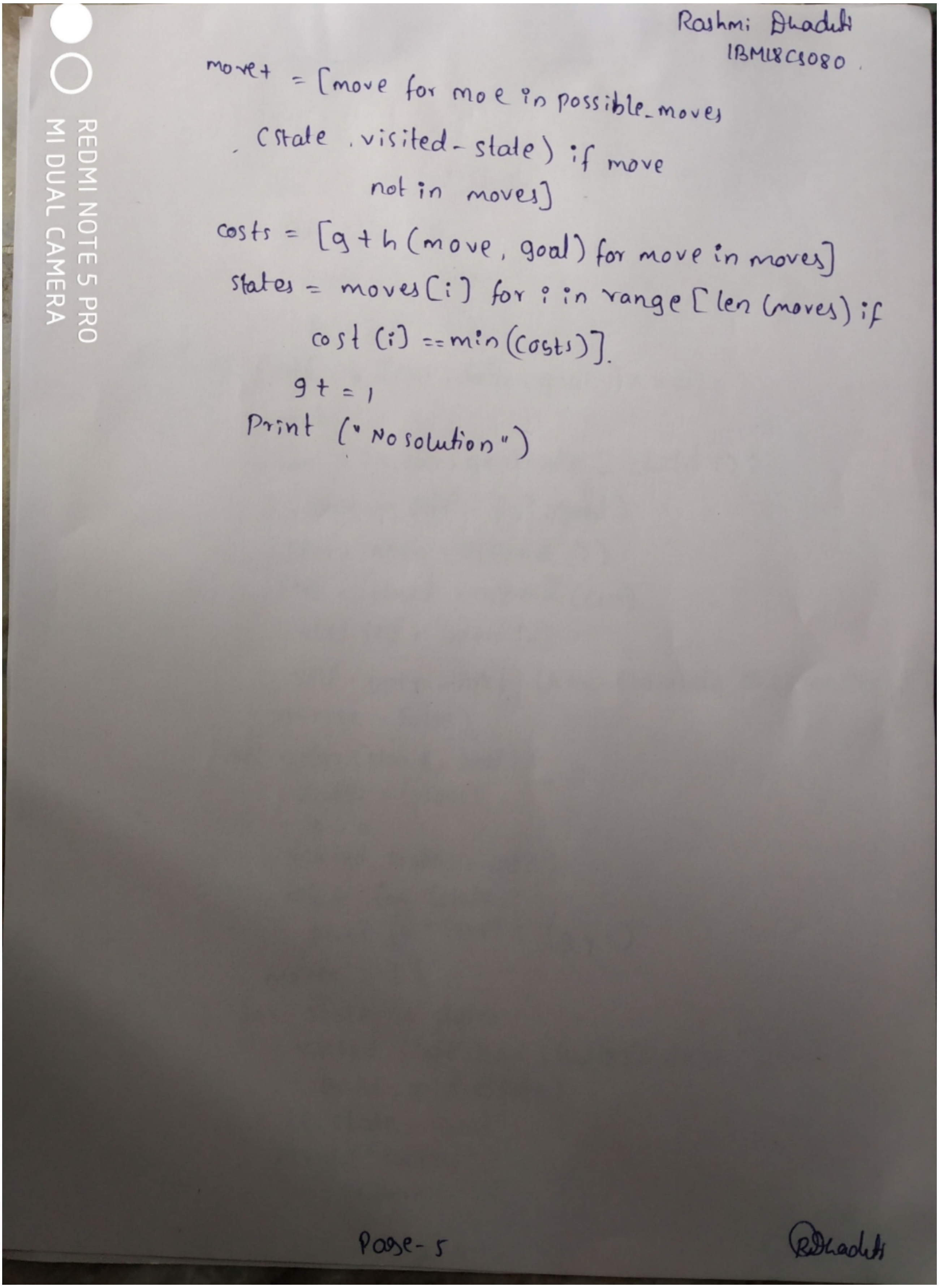
Dhaduti



Rashmi Dhaduti
1BM18CS080

```
Print (" \\\' / \n")
        for i in cur. data :
          for j in i :
            print (j , end = " ")
            Print (" ")
    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]
          Self . open . sort (key = lambda x : x . fval .
        reverse = False)
    def astar (start, goal) :
          states = [start]
            g = 0
          visited_state = set()
          while len (state):
            print (f " level : {g y")
        moves = [].
    for state in states :
          visited_ state_add (tuple (state))
            Print _grid (state)
          if state = goal
        print (" success ")
              return.
```

Dhaduti



Rashmi Dhaditi
1BM18CS080.

```
movet = [move for moe in possible_moves
    (state . visited - state) if move
         not in moves]
costs = [g + h (move, goal) for move in moves]
 states = moves[i] for i in range [len (moves) if
       cost (i] == min ((costs)].
    g + = 1
  Print ("No solution")
```

Page-5

RDhaditi