4 → 8 puzzle problem using A* algorith:

(1) ~~A~~ def Node (data, level) : initialize node with puzzle state & level. + path

(2) def puzzle ( ) :
- accept( ) : ← Accept start & goal state.
- f(start, goal) ← calculate f : = h + g
- process( ) ← process A* algorithm.

(3) Heuristic
h(start, goal) : Manhattan dist. b/w current & goal state

(4) A* Algorithm :
- initialize start node, & add to open list.
- loop until goal state come.
  - → select node have lowest f from ~~Fo~~ openlist.
  - → Display state & check goal.
  - → generate child node & calculate f
  - → And current node to closed list & remove from open list
  - → explore list by f
- Display find move.

(5) ~~Execution~~ : Create puzzle instance.
- call process for the execute.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 0 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 0 | 7 | 8 |

h = 3 h = 3 h = 0

h = 4

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 0 | 5 | 6 |
| 4 | 7 | 8 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 0 | 8 |

h = 2

Prasad 8/12

22/12

→ Code of ~~AO*~~ 8 puzzle using A* algorithm:

```
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 children.

def shuffle (self, puz, x1, y1, x2, y2):

    if x2 >= 0 x2 < = len (self.data) and y2 >= 0 and
                                                   y2 < len (
```

```
        temp-puz = []
        temp-puz = self.copy(puz)
        temp = temp-puz[x2][y2]
        temp-puz [x2] [y2] = temppuz[x1][x1]
        temp-puz [x1] [y1] = temp
        return temp-puz
    Else
        return None

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 __init__(self, size):
        self.n = size
        self.open = []
        self.closed = []

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

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 = 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 :
        cur = self.open[0]
        print ("")
        print (" || / \n")
        for i in cur.data :
            for j in i :
                print (j, end = "")

        self.open.sort (key = lambda x: x.fval, rev

puz = puzzle (3)
```

Bafna Gold
Date: Page:

Output

Enter the start Matrix

1 2 3
4 5 6
_ 7 8

Enter the goal state Matrix.

1 2 3
4 5 6
7 8 _

| | | | | |
|---|---|---|---|---|
| 1 2 3 | $\longrightarrow$ | 1 2 3 | $\longrightarrow$ | 1 2 3 |
| 4 5 6 | | 4 5 6 | | 4 5 6 |
| _ 7 8 | | 7 _ 8 | | 7 8 _ |

22/12

False)

```
Enter the start state matrix

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

1 2 3
4 5 6
7 8 _



  |
  |
 \'/

1 2 3
4 5 6
_ 7 8

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |
  |
 \'/

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