30|11|22

Week:4 : A* algorithm for 8 puzzle .

Aim :- To implement A* algorithm for 8 puzzle problem.

```
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 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) :

        temp=[]
        for i in root :
            t = []
            for j in i :
                t.append(j)
            temp.~~self~~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 __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.append(temp)
        return puz

    def f(self, start, goal):
        return self.h(start.data, goal) + start.lev

    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. (start, goal)

        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 (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 )

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

O/P: Enter start state   Enter goal state.

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

   ↓

1  2  3
4  5  6
7  _  8

   ↓

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

Algorithm ::

$f(n) = g(n) + h(n)$.

$g(n)$ = sum of edge costs from start to n.

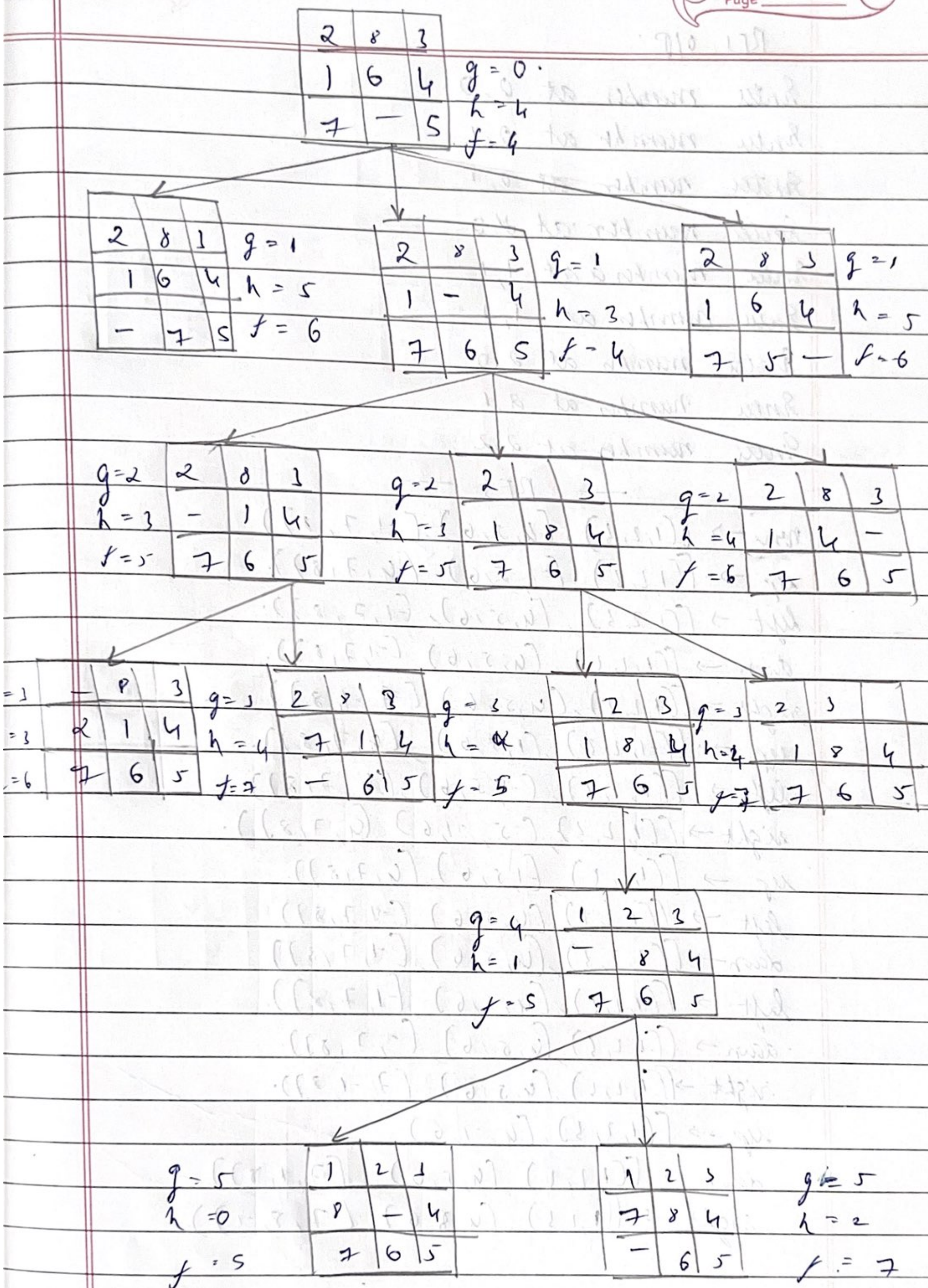
$h(n)$ = estimate of lowest cost path from n to goal.

$f(n)$ = actual distance so far + estimated distance remaining.

```
function A* search (problem) returns a solution or failure
 node ← a node n with n.state = problem.
                                  initial state

 frontier ← a priority Queue ordered by ascending g+h,
            only element n.
 loop do
    if empty ? (frontier) then return failure.
      n ← pop (frontier)
    if problem . goal Test (n.state) then return solution(n)
    for each action a in problem - actions (n.state)
     do
     n' ← child Node (problem, n, a)
     insert (n', g(n') + h(n'), frontier)
```

State space tree :



2 8 3 / 1 6 4 / 7 – 5
g = 0.
h = 4
f = 4
2 8 3 / 1 6 4 / – 7 5
g = 1
h = 5
f = 6
2 8 3 / 1 – 4 / 7 6 5
g = 1
h = 3
f = 4
2 8 3 / 1 6 4 / 7 5 –
g = 1
h = 5
f = 6
g = 2
h = 3
f = 5
2 8 3 / – 1 4 / 7 6 5
g = 2
h = 3
f = 5
2 3 / 1 8 4 / 7 6 5
g = 2
h = 4
f = 6
2 8 3 / 1 4 – / 7 6 5
= 3
= 3
= 6
– 8 3 / 2 1 4 / 7 6 5
g = 3
h = 4
f = 7
2 8 3 / 7 1 4 / – 6 5
g = 3
h = 4
f = 5
2 3 / 1 8 4 / 7 6 5
g = 3
h = 4
f = 7
2 3 / 1 8 4 / 7 6 5
g = 4
h = 1
f = 5
1 2 3 / – 8 4 / 7 6 5
g = 5
h = 0
f = 5
1 2 3 / 8 – 4 / 7 6 5
1 2 3 / 7 8 4 / – 6 5
g = 5
h = 2
f = 7