A Star Algorithm - 8 Puzzle Problem

1) Code:

In [6]:

```
import copy
from heapq import heappush, heappop

n = 3
row = [ 1, 0, -1, 0 ]
col = [ 0, -1, 0, 1 ]

class priorityQueue:
    def __init__(self):
        self.heap = []

    def push(self, k):
        heappush(self.heap, k)

    def pop(self):
        return heappop(self.heap)

    def empty(self):
        if not self.heap:
            return True
        else:
            return False

class node:
    def __init__(self, parent, mat, empty_tile_pos, cost, level):
        self.parent = parent
        self.mat = mat
        self.empty_tile_pos = empty_tile_pos
        self.cost = cost
        self.level = level

    def __lt__(self, nxt):
        return self.cost < nxt.cost

def calculateCost(mat, final) -> int:
    count = 0
    for i in range(n):
        for j in range(n):
            if ((mat[i][j]) and
                (mat[i][j] != final[i][j])):
                count += 1

    return count

def newNode(mat, empty_tile_pos, new_empty_tile_pos, level, parent, final) -> node:
    new_mat = copy.deepcopy(mat)
    x1 = empty_tile_pos[0]
    y1 = empty_tile_pos[1]
    x2 = new_empty_tile_pos[0]
    y2 = new_empty_tile_pos[1]
    new_mat[x1][y1], new_mat[x2][y2] = new_mat[x2][y2], new_mat[x1][y1]
    cost = calculateCost(new_mat, final)
    new_node = node(parent, new_mat, new_empty_tile_pos, cost, level)
    return new_node

def printMatrix(mat):
    for i in range(n):
        for j in range(n):
            print(mat[i][j], end = " ")
        print()

def isSafe(x, y):
    return x >= 0 and x < n and y >= 0 and y < n

def printPath(root):
    if root == None:
        return

    printPath(root.parent)
    printMatrix(root.mat)
    print()

def solve(initial, empty_tile_pos, final):
    pq = priorityQueue()
    cost = calculateCost(initial, final)
    root = node(None, initial,
                empty_tile_pos, cost, 0)

    pq.push(root)

    while not pq.empty():
        minimum = pq.pop()
        if minimum.cost == 0:
            printPath(minimum)
            return

        for i in range(4):
            new_tile_pos = [
                minimum.empty_tile_pos[0] + row[i],
                minimum.empty_tile_pos[1] + col[i], ]

            if isSafe(new_tile_pos[0], new_tile_pos[1]):
                child = newNode(minimum.mat,
                                minimum.empty_tile_pos,
                                new_tile_pos,
                                minimum.level + 1,
                                minimum, final,)

                pq.push(child)

initial = [[ 1, 2, 3 ],
           [ 5, 6, '_' ],
           [ 7, 8, 4 ]]

final = [[ 1, 2, 3 ],
         [ 5, 8, 6 ],
         [ '_', 7, 4 ]]

empty_tile_pos = [ 1, 2 ]
solve(initial, empty_tile_pos, final)


# Output
```

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

1 2 3
5 _ 6
7 8 4

1 2 3
5 8 6
7 _ 4

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

2) Code:

In [5]:

```
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.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 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):
        start = [[ 1, 2, 3 ],
                [ 5, 6, '_' ],
                [ 7, 8, 4 ]]
        goal = [[ 1, 2, 3 ],
                [ 5, 8, 6 ],
                [ '_', 7, 4 ]]

        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        self.open.append(start)
        while True:
            cur = self.open[0]
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            print("\n")
            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()

#Output
```

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



1 2 3
5 _ 6
7 8 4



1 2 3
5 8 6
7 _ 4



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