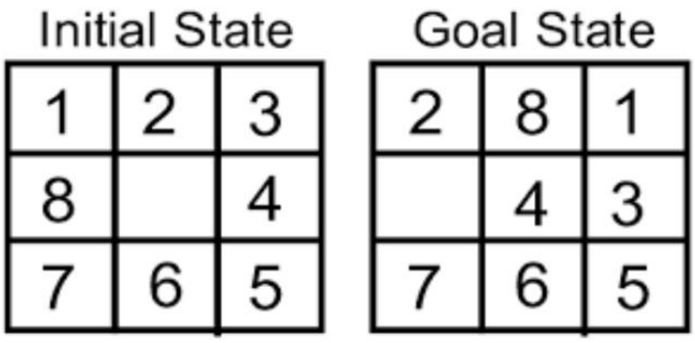
Computer Science and Engineering Department

Artificial Intelligence (UCS-521)

Lab Assignment-2

1. Write a code in python for the 8 puzzle problem by taking the following initial and final states



Initial State

1	2	3
8		4
7	6	5

Goal State

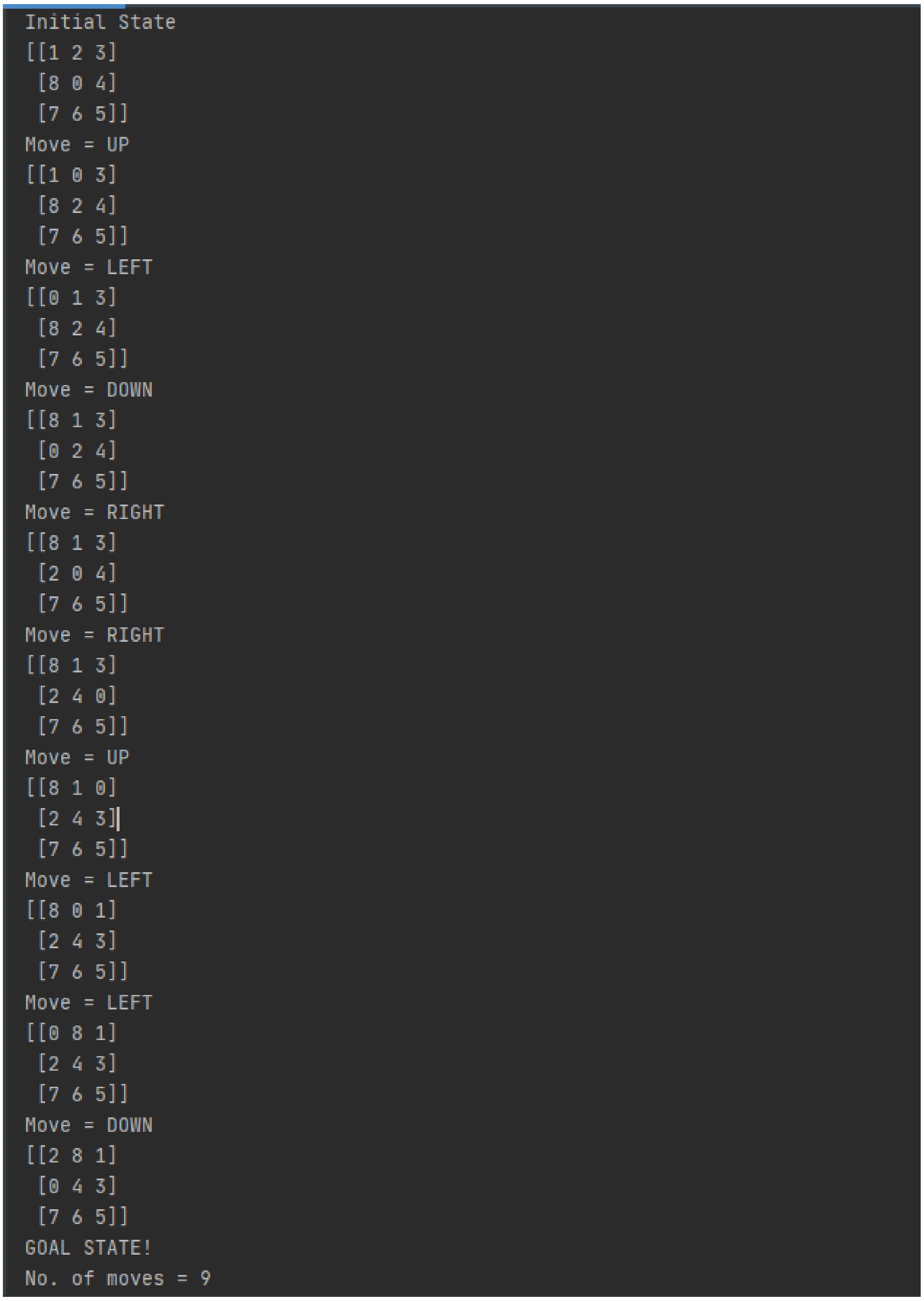
2	8	1
	4	3
7	6	5

CODE:

```
#Uninformed Searching using Breadth First Search

import numpy as np

initial_arr = np.array([[1,2,3],[8,0,4],[7,6,5]])
final_arr = np.array([[2,8,1],[0,4,3],[7,6,5]])

#All possible moves
# up = (-1,0)
# down = (1,0)
# left = (0,-1)
# right = (0,1)

moves = [(-1,0),(1,0),(0,-1),(0,1)]
movesName = ['UP', 'DOWN', 'LEFT', 'RIGHT']


#checking valid moves
def isValidMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    if i<len(initial_arr) and i>=0 and j>=0 and j<len(initial_arr):
        return True
    return False

def performMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    temp_arr = initial_arr.copy()
    temp = temp_arr[i][j]
    temp_arr[i][j] = temp_arr[idx[0]][idx[1]]
    temp_arr[idx[0]][idx[1]] = temp
    return temp_arr
```

```python
def findZeroIndex(initial_arr):

    for i in range(0,len(initial_arr)):
        for j in range(0,len(initial_arr[i])):
            if initial_arr[i][j] == 0:
                return i,j


#Function to print all intermediate states and the moves involved
def printRes(bfs, i):

    if bfs[i][2] == -1:
        print('Initial State')
        print(bfs[i][0])
        return 0

    count = printRes(bfs,bfs[i][2])
    ind = moves.index(bfs[i][1])
    print(f'Move = {movesName[ind]}')
    print(bfs[i][0])
    return count + 1


def findSolBFS(initial_arr, final_arr, prevMove):

    bfs = [(initial_arr,prevMove,-1)]
    flag = 0
    l = len(bfs)
    i = 0

    while i<l:
        currNode = bfs[i]
        idx = findZeroIndex(currNode[0])

        for move in moves:
            if isValidMove(currNode[0],idx,move):
                new_arr = performMove(currNode[0],idx,move)
                bfs.append((new_arr,move, i))

                if np.count_nonzero(np.subtract(new_arr,final_arr)) == 0:
                    l = len(bfs)
                    count = printRes(bfs,l-1)
                    print('GOAL STATE!')
                    print(f'No. of moves = {count}')
                    flag = 1
                    break

        l = len(bfs)
        i = i+1

        if flag == 1:
            break

findSolBFS(initial_arr, final_arr, (0,0))
```

OUTPUT:



```
Initial State
[[1 2 3]
 [8 0 4]
 [7 6 5]]
Move = UP
[[1 0 3]
 [8 2 4]
 [7 6 5]]
Move = LEFT
[[0 1 3]
 [8 2 4]
 [7 6 5]]
Move = DOWN
[[8 1 3]
 [0 2 4]
 [7 6 5]]
Move = RIGHT
[[8 1 3]
 [2 0 4]
 [7 6 5]]
Move = RIGHT
[[8 1 3]
 [2 4 0]
 [7 6 5]]
Move = UP
[[8 1 0]
 [2 4 3]
 [7 6 5]]
Move = LEFT
[[8 0 1]
 [2 4 3]
 [7 6 5]]
Move = LEFT
[[0 8 1]
 [2 4 3]
 [7 6 5]]
Move = DOWN
[[2 8 1]
 [0 4 3]
 [7 6 5]]
GOAL STATE!
No. of moves = 9
```

2. Given two jugs- a 4 liter and 3 liter capacity. Neither has any measurable markers on it. There is a pump which can be used to fill the jugs with water. Simulate the procedure in Python to get exactly 2 liter of water into 4-liter jug

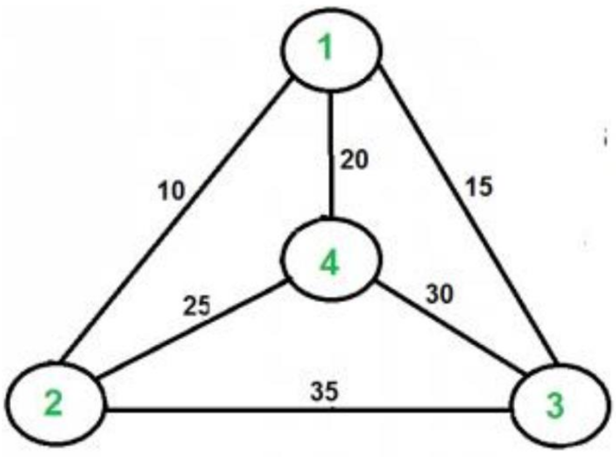
   CODE:

```python
x,y,m,n = 0,0,4,3

print('Initital State = (0,0)')
print('Capacitites = ({0},{1})'.format(m,n))
print('Goal State = (2,0)')

while x!=2 or y!=0:
    r = int(input('Enter Rule: '))

    if r==1:
    #full x
        if x<m:
            x = m

    if r==2:
    #full y
        if y<n:
            y = n

    if r==3:
    #Empty x
        if x>0:
            x = 0

    if r==4:
    #empty y
        if y>0:
            y=0

    if r==5:
    #transfer from y to x when x+y>=m
        if x+y>=m and y>0:
            x,y = m,y-(m-x)

    if r==6:
    # transfer from x to y when x+y>=n
        if x+y>=n and x>0:
            x,y = x-(n-y),n

    if r==7:
    # transfer from y to x when x+y<=m
        if x+y<=m and y>0:
            x,y = x+y,0

    if r==8:
    # transfer from x to y when x+y<=n
        if x+y<=n and x>0:
            x,y = 0,x+y

    print(f'x={x}')
    print(f'y={y}')

    if x==2 and y==0:
        print('Goal State')
```

OUTPUT:

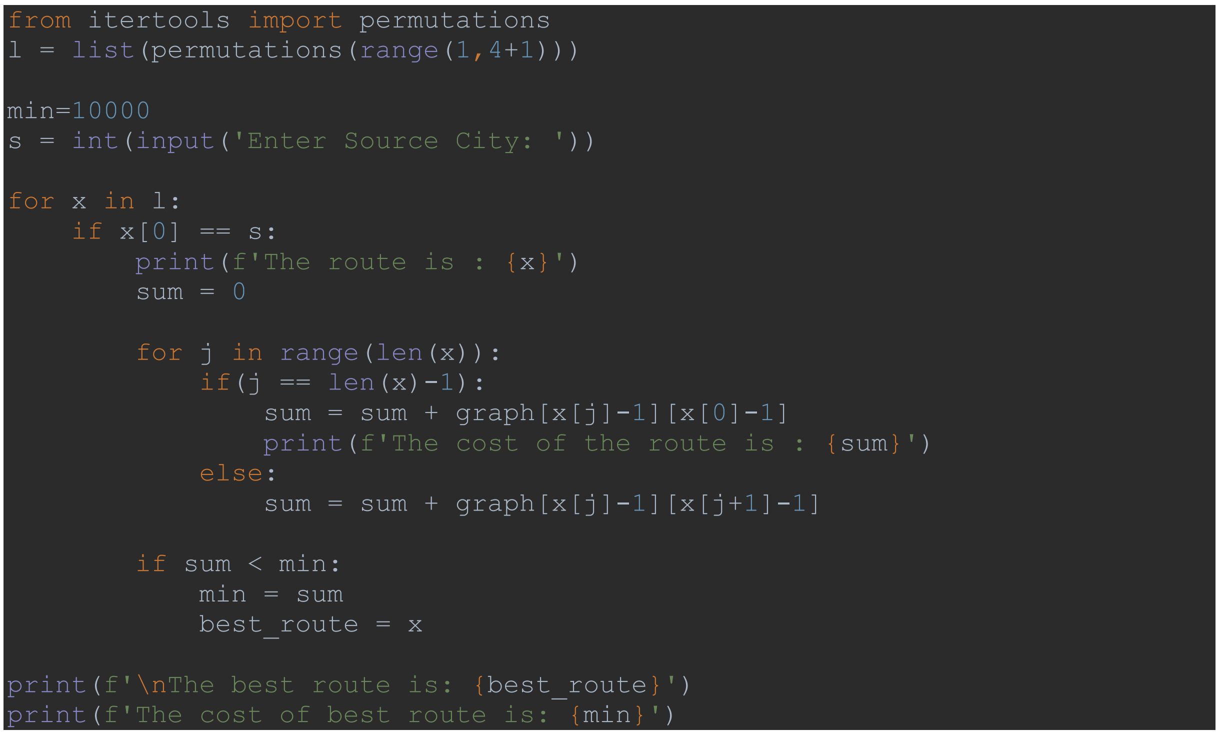
```
Initital State = (0,0)
Capacitites = (4,3)
Goal State = (2,0)
Enter Rule: 2
x=0
y=3
Enter Rule: 7
x=3
y=0
Enter Rule: 2
x=3
y=3
Enter Rule: 5
x=4
y=2
Enter Rule: 3
x=0
y=2
Enter Rule: 7
x=2
y=0
Goal State

Process finished with exit code 0
```

3. Write a Python program to implement Travelling Salesman Problem (TSP). Take the starting node from the user at run time.



CODE:

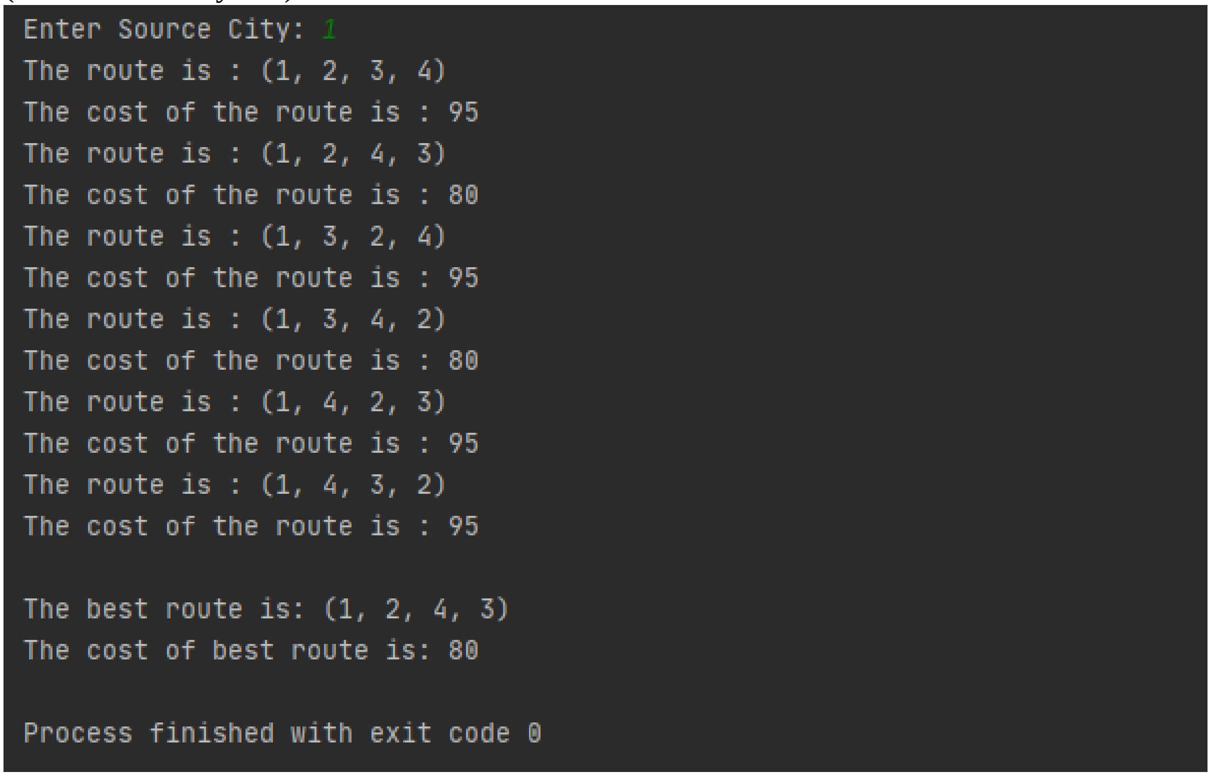
```
graph = [[0,10,15,20],
         [10,0,35,25],
         [15,35,0,30],
         [20,25,30,0]]
```



```python
from itertools import permutations
l = list(permutations(range(1,4+1)))

min=10000
s = int(input('Enter Source City: '))

for x in l:
    if x[0] == s:
        print(f'The route is : {x}')
        sum = 0

        for j in range(len(x)):
            if(j == len(x)-1):
                sum = sum + graph[x[j]-1][x[0]-1]
                print(f'The cost of the route is : {sum}')
            else:
                sum = sum + graph[x[j]-1][x[j+1]-1]

        if sum < min:
            min = sum
            best_route = x

print(f'\nThe best route is: {best_route}')
print(f'The cost of best route is: {min}')
```

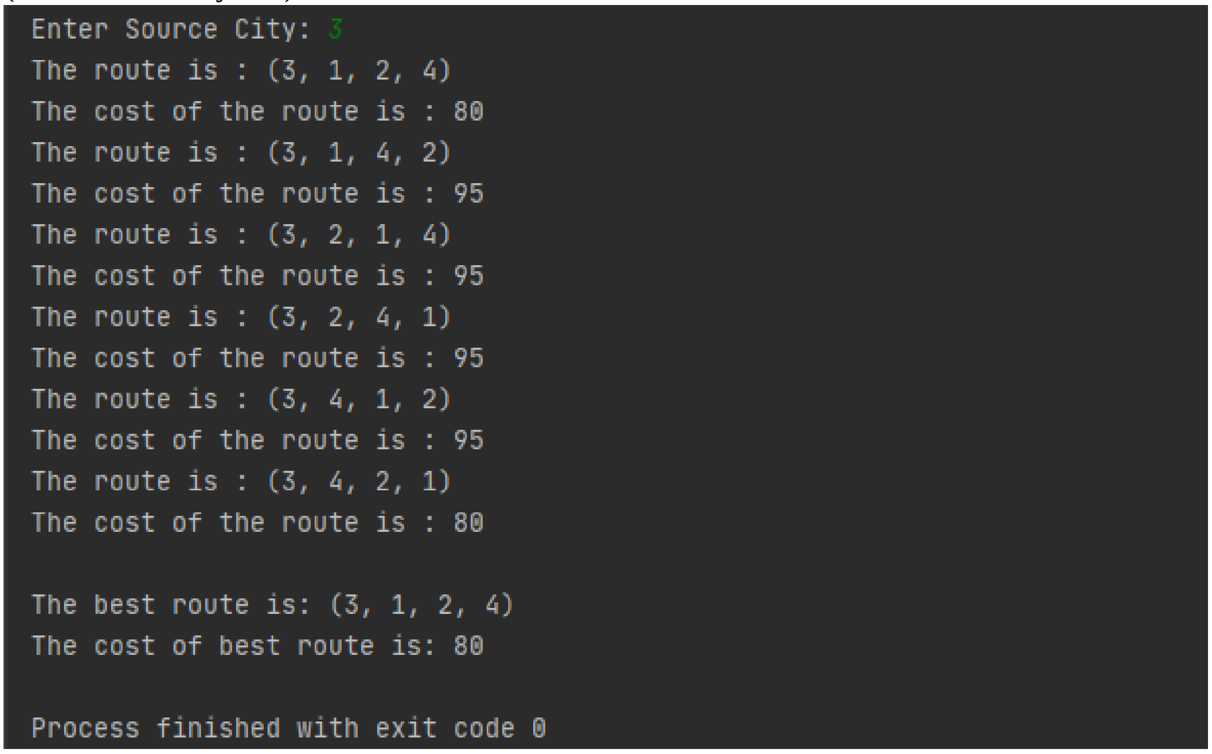
OUTPUT:

(when source city is 1)



```
Enter Source City: 1
The route is : (1, 2, 3, 4)
The cost of the route is : 95
The route is : (1, 2, 4, 3)
The cost of the route is : 80
The route is : (1, 3, 2, 4)
The cost of the route is : 95
The route is : (1, 3, 4, 2)
The cost of the route is : 80
The route is : (1, 4, 2, 3)
The cost of the route is : 95
The route is : (1, 4, 3, 2)
The cost of the route is : 95

The best route is: (1, 2, 4, 3)
The cost of best route is: 80

Process finished with exit code 0
```

(when source city is 3)



```
Enter Source City: 3
The route is : (3, 1, 2, 4)
The cost of the route is : 80
The route is : (3, 1, 4, 2)
The cost of the route is : 95
The route is : (3, 2, 1, 4)
The cost of the route is : 95
The route is : (3, 2, 4, 1)
The cost of the route is : 95
The route is : (3, 4, 1, 2)
The cost of the route is : 95
The route is : (3, 4, 2, 1)
The cost of the route is : 80

The best route is: (3, 1, 2, 4)
The cost of best route is: 80

Process finished with exit code 0
```