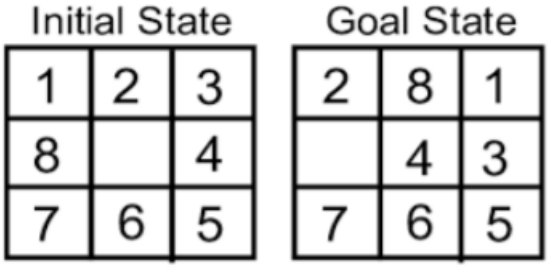
UCS411: ARTIFICIAL INTELLIGENCE

Assignment-2 (Solution)

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



CODE:

```
import sys
import copy

q = []


def compare(s,g):
    if s==g:
        return(1)
    else:
        return(0)

def find_pos(s):

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


def up(s,pos):

    i = pos[0]
    j = pos[1]

    if i > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i-1][j]
        temp[i-1][j] = 0
```

```
        return (temp)
    else:
        return (s)


def down(s,pos):

    i = pos[0]
    j = pos[1]

    if i < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i+1][j]
        temp[i+1][j] = 0
        return (temp)
    else:
        return (s)


def right(s,pos):

    i = pos[0]
    j = pos[1]

    if j < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j+1]
        temp[i][j+1] = 0
        return (temp)
    else:
        return (s)


def left(s,pos):

    i = pos[0]
    j = pos[1]

    if j > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j-1]
        temp[i][j-1] = 0
        return (temp)
    else:
        return (s)

def enqueue(s):
```

```
    global q
    q = q + [s]

def dequeue():
    global q

    # find the state having minimum mis matches with the goal state
    elem = q[0]
    del q[0]
    return (elem)

def search(s,g):

    curr_state = copy.deepcopy(s)
    if s == g:
        return

    c = 0
    while(1):

        pos = find_pos(curr_state)
        new = up(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("found")
                return
            else:
                enqueue(new)


        new = down(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("found")
                return
            else:
                enqueue(new)

        new = right(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("Found")
                return
            else:
```

```
            enqueue(new)

        new = left(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("Found")
                return
            else:
                enqueue(new)

        if len(q) > 0:
            curr_state = dequeue()
        else:
            print ("not found")
            return


def main():
    s = [[1,2,3],[8,0,4],[7,6,5]]

    g = [[2,8,1],[0,4,3],[7,6,5]]

    pos = find_pos(s)

    search(s,g)

if __name__ == "__main__":
    main()
```

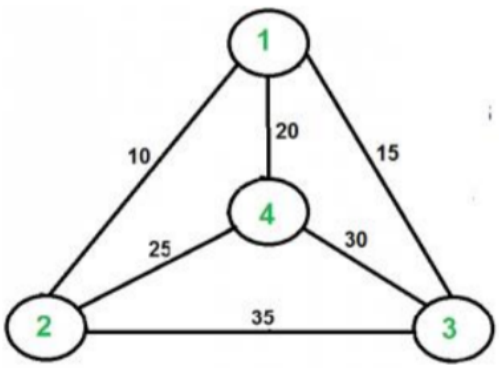
2. Given two jugs- a 4 liter and 3 liter capacity. Neither has any measurable markers on it. There is a pump which can be used to fill the jugs with water. Simulate the procedure in Python to get exactly 2 liter of water into 4-liter jug
CODE:

```
from collections import defaultdict

jug1, jug2, aim = 4, 3, 2

visited = defaultdict(lambda: False)

def waterJugSolver(amt1, amt2):

    if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
        print(amt1, amt2)
        return True
```

```
    if visited[(amt1, amt2)] == False:
        print(amt1, amt2)


        visited[(amt1, amt2)] = True


        return (waterJugSolver(0, amt2) or
                waterJugSolver(amt1, 0) or
                waterJugSolver(jug1, amt2) or
                waterJugSolver(amt1, jug2) or
                waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                amt2 - min(amt2, (jug1-amt1))) or
                waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                amt2 + min(amt1, (jug2-amt2))))


    else:
        return False

print("Steps: ")
waterJugSolver(0, 0)
```

```
In [10]: runfile('D:/python programs/jug_water_problem.py', wdir='D:/
python programs')
Steps:
0 0
4 0
4 3
0 3
3 0
3 3
4 2
0 2
```

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



CODE:

```
dst=[]
def travel(g, v, pos, n, count, cost):
```

```
    if(count==n and g[pos][s]):
        cost+=g[pos][s]
        dst.append(cost)
        return
    for i in range(0,n):
        if(v[i]==False and g[pos][i]):
            v[i]=True
            travel(g,v,i,n,count+1,cost+g[pos][i])
            v[i]=False
n=4
g=[[0, 10, 15, 20],[10, 0, 35, 25],[15, 35, 0, 30],[20, 25, 30, 0]]
s=int(input("Enter a number between 1 and 4: "))
v=[False for i in range(0,n)]
s-=1
v[s]=True
travel(g,v,s,n,1,0)
print(dst)
print(min(dst))
```