1:-

```
sour=[]
dest=[]
d={}
print("Enter Source Matrix")
for i in range(3):
    a,b,c=map(int,input().split())
    sour.extend([a,b,c])
print("Enter Destination Matrix")
for i in range(3):
    a,b,c=map(int,input().split())
    dest.extend([a,b,c])
d={}
sour=tuple(sour)
dest=tuple(dest)
d[sour]=sour
q=[sour]
f=0
while q:
    cur=q[0]
    q.pop(0)
    if cur==dest:
        f=1
        break
    for i in range(9):
        if cur[i]==0:
            idx=i
            break
    temp=list(cur)
    if idx>=3:
        a=temp[:]
        a[idx],a[idx-3]=a[idx-3],a[idx]
        a=tuple(a)
        if a not in d:
            d[a]=cur
            q.append(a)
    if idx<=5:
        a=temp[:]
        a[idx],a[idx+3]=a[idx+3],a[idx]
        a=tuple(a)
        if a not in d:
            d[a]=cur
            q.append(a)
    if idx%3!=2:
        a=temp[:]
        a[idx],a[idx+1]=a[idx+1],a[idx]
        a=tuple(a)
        if a not in d:
            d[a]=cur
            q.append(a)
    if idx%3!=0:
        a=temp[:]
        a[idx],a[idx-1]=a[idx-1],a[idx]
        a=tuple(a)
        if a not in d:
            d[a]=cur
            q.append(a)

if f==0:
    print("There are no solutions")
else:
    print("Solution")
    ans=[]
    while d[cur]!=cur:
        ans.append(cur)
        cur=d[cur]
    ans.append(sour)
    ans.reverse()
    for i in ans:
        c=0
        for j in range(3):
            for k in range(3):
                print(i[k+c*3],end=' ')
            print()
            c+=1
        print()
```

2:-

Bfs:-

```
n=int(input("Enter number of vertices: "))
m=int(input("Enter number of edges: "))
l={}
print("Enter edges (source, destination)")
for i in range(m):
    u,v=map(int,input().split())
    if u in l:
        l[u].append(v)
    else:
        l[u]=[v]
    if v in l:
        l[v].append(u)
    else:
        l[v]=[u]
start=int(input("Enter starting node: "))
print("Breadth first order is")
```

```
vist={i:False for i in l}
q=[start]
while q:
    v=q[0]
    q.pop(0)
    if vist[v]:
        continue
    vist[v]=True
    print(v)
    for i in l[v]:
        q.append(i)
```

3:-

DFS:-

```
n=int(input("Enter number of vertices: "))
m=int(input("Enter number of edges: "))
l={}
print("Enter edges (source, destination)")
for i in range(m):
    u,v=map(int,input().split())
    if u in l:
        l[u].append(v)
    else:
        l[u]=[v]
    if v in l:
        l[v].append(u)
    else:
        l[v]=[u]
start=int(input("Enter starting node: "))
print("Breadth first order is")
vist={i:False for i in l}
q=[start]
while q:
    v=q[-1]
    q.pop()
    if vist[v]:
        continue
    vist[v]=True
    print(v)
    for i in l[v]:
        q.append(i)
```

4:-

Mc.py:

```
good = {(0, 0), (3, 0), (0, 3), (3, 1), (3, 2), (2, 2), (1, 1), (0, 2), (0, 1)}
```

```
count = 0

def printsolution(ans, d):
    print(f"Solution - {count}:")
    print("(M, C, B)")
    cur = ans
    while d[cur] != cur:
        print(cur)
        cur = d[cur]
    print(cur)

def solve(v, d):
    global count
    if v[0] == v[1] == 0:
        count += 1
        printsolution(v, d)
        return
    pos = [(-1, 0), (-1, -1), (-2, 0), (0, -2), (0, -1)]
    a, b, c = v[0], v[1], v[2]
    if c == 0:
        mul = -1
    else:
        mul = 1
    for i in pos:
        x, y = i
        na = a+x*mul
        nb = b+y*mul
        t = (na, nb, c ^ 1)
        if (na, nb) in good and t not in d:
            d[t] = v
            solve(t, d)
            del d[t]

s = (3, 3, 1)
d = {s: s}
solve(s, d)
```

5:-

Monkey.py:

```
def solve(banana,box,height,monkey,hold):
    if monkey==banana and height==1:
        ans.append("Monkey took banana")
        return True
    if (banana,box,height,monkey,hold) in d:
        return False
    found=0
    d[(banana,box,height,monkey,hold)]=1
    options={1:"Move to box", 2:"Move to banana", 3:"Climb onto the box", 4:"Hold box to move"}
    for option in options:
        if option==1 and hold==0 and height==0 and solve(banana,box,height,box,hold):
            ans.append(options[option])
            found=1
            break
        elif option==2 and height==0 and ((hold==1 and solve(banana,banana,height,banana,hold)) or
(hold==0 and solve(banana,box,height,banana,hold))):
            ans.append(options[option])
            found=1
            break
        elif option==3 and height==0 and monkey==box and solve(banana,box,height+1,monkey,0):
            ans.append(options[option])
            found=1
            break
        elif option==4 and height==0 and monkey==box and solve(banana,box,height,monkey,1):
            ans.append(options[option])
            found=1
            break
    return found

n=int(input("Enter the size of the world: "))
world=[[0]*n for i in range(n)]
x,y=map(int,input("Enter tree position: ").split())
world[x][y]=1
x,y=map(int,input("Enter box position: ").split())
world[x][y]=-1
x,y=map(int,input("Enter monkey position: ").split())
for i in range(n):
    for j in range(n):
        if world[i][j]==1:
            print(f"Monkey found the banana tree at ({i},{j})")
            banana=(i,j)
        if world[i][j]==-1:
            print(f"Box found at ({i},{j})")
            box=(i,j)
d={}
ans=[]
solve(banana,box,0,(x,y),0)
ans.reverse()
print(*ans,sep="\n")

'''
5
2 2
4 4
```

```
0 0
'''
```

6:-

N-q.py:-

```
ans=0
def printBoard(x):
    print(f"Solution - {ans}")
    for i in range(n):
        for j in range(n):
            if x[i]==j:
                print("Q",end=' ')
            else:
                print(".",end=' ')
        print()

def solve(x,cur,n):
    global ans
    if cur==n:
        ans+=1
        printBoard(x)
    for i in range(n):
        f=0
        for j in range(cur):
            if x[j]==i or abs(cur-j)==abs(i-x[j]):
                f=1
                break
        if f==0:
            x[cur]=i
            solve(x,cur+1,n)

n=int(input("Enter the size of the board: "))
x=[0]*n
solve(x,0,n)
if ans==0:
    print("There are no solutions")
```

7:-

Tic.t.t:-

```
# This program uses min-max algorithm

def printBoard(board):
    for i in range(3):
        print(" "+board[i][0]+"|"+board[i][1]+"|"+board[i][2])
        if i<2:
            print("-------")

board=[[" ", " ", " "] for i in range(3)]

def isFull(board):
    for i in range(3):
        for j in range(3):
            if board[i][j]==" ":
                return False
    return True

def winner(player,board):
    if board[0][0]==board[1][1]==board[2][2]==player:
        return True
    if board[0][2]==board[1][1]==board[2][0]==player:
        return True
    for i in range(3):
        if board[i][0]==board[i][1]==board[i][2]==player:
            return True
        if board[0][i]==board[1][i]==board[2][i]==player:
            return True
    return False

def player():
    print("Player's Turn")
    v=int(input())
    v-=1
    x=v//3
    y=v%3
    if board[x][y]==" ":
        board[x][y]="X"
        printBoard(board)
    else:
        print("Invalid tile already selected")
        player()

def avail():
    l=[]
    for i in range(3):
        for j in range(3):
            if board[i][j]==" ":
                l.append((i,j))
    return l

def getScore():
    if winner("X",board):
        return -1
    if winner("O",board):
        return 1
    return 0

players=["O","X"]
def minimax(depth, is_max):
    score = getScore()
    if score==1:
        return score
    if score==-1:
        return score
    if isFull(board):
        return 0
    if is_max:
        best=-100
        l=avail()
        for i in l:
            board[i[0]][i[1]]=players[0]
            best=max(best,minimax(depth+1,not is_max))
            board[i[0]][i[1]]=" "
    else:
        best=100
        l=avail()
        for i in l:
            board[i[0]][i[1]]=players[1]
            best=min(best,minimax(depth+1,not is_max))
            board[i[0]][i[1]]=" "
    return best

def getBestMove():
    l=avail()
    best=-1
    score=-100
    for i in l:
        board[i[0]][i[1]]=players[0]
        t=minimax(0,False)
        board[i[0]][i[1]]=" "
        if t>score:
            best=i
            score=t
    return best

choice=1
human=True
while choice==1:
    if human:
        printBoard(board)
        while not winner("X",board) and not winner("O",board) and not isFull(board):
            player()
            if winner("X",board):
                print("Well done, You won")
                break
            if isFull(board):
                print("Game is Tie!")
                break
            move = getBestMove()
            board[move[0]][move[1]]='O'
            printBoard(board)
            if winner("O",board):
                print("Sorry, You loose")
                break
    else:
        while not winner("X",board) and not winner("O",board) and not isFull(board):
            move = getBestMove()
            board[move[0]][move[1]]='O'
            printBoard(board)
            if winner("O",board):
                print("Sorry, You loose")
                break
            if isFull(board):
                print("Game is Tie!")
                break
            player()
            if winner("X",board):
                print("Well done, You won")
                break
    choice=int(input("Do you want play again(0/1):"))
    human = not human
    board=[[" ", " ", " "] for i in range(3)]
```

8:-

Tower.py:-

```
def towers(n,sour,des,aux):
    if n==1:
        print(f"Moving disk 1 from {sour} to {des}")
        return
    towers(n-1,sour,aux,des)
    print(f"Moving disk {n} from {sour} to {des}")
    towers(n-1,aux,des,sour)

n=int(input('Enter no.of disks: '))
towers(n,'A','B','C')
```

9:-

Travelling -sales.py:-

```
def solve(v,c,d,dis):
    if v==x and c==n:
        return dis[v]
    ans=float('inf')
    for i in l[v]:
        u,w=i
        if u==x and c!=n-1:
            continue
        if u not in d:
            d[u]=v
            dis[u]=w+dis[v]
            ans=min(ans,solve(u,c+1,d,dis))
            del d[u]
    return ans

n = int(input("Enter number of vertices: "))
m = int(input("Enter number of edges: "))
l={i:[] for i in range(1,n+1)}
print("Enter edges (source, destination, weight)")
for i in range(m):
    u, v, w = map(int, input().split())
    l[u].append([v,w])
x = int(input("Enter starting node: "))
dis={x:0}
d={}
print("Shortest distance:",solve(x,0,d,dis))



'''

Sample input:-

1 2 10
2 1 5
1 3 15
3 1 6
1 4 20
4 1 8
2 4 10
4 2 8
2 3 9
3 2 13
3 4 12
4 3 9
```

'''

10:-

Water-jug.py:-

```
x=int(input("Size of first jar: "))
y=int(input("Size of second jar: "))
z=int(input("Required Amount: "))
q=[(0,0)]
d={(0,0):(0,0)}
f=0
sol=[]
while q:
    a,b=q[0]
    q.pop(0)
    if(a==z or y==z):
        f=1
        sol.append((a,b))
        continue
    pos=[(x,b),(a,y),(0,b),(a,0),(min(a+b,x),max(b-(x-a),0)),(max(a-(y-b),0),min(b+a,y))]
    for i in pos:
        if i in d:
            continue
        d[(i[0],i[1])]=(a,b)
        q.append((i[0],i[1]))
if f==0:
    print("There is no solution")
else:
    for j in range(len(sol)):
        print("Solution - {}".format(j+1))
        ans=[]
        cur=sol[j]
        while d[cur]!=cur:
            ans.append(cur)
            cur=d[cur]
        ans.append((0,0))
        ans.reverse()
        for i in ans:
            print(i)
```