```
from collections import deque
from queue import PriorityQueue
from socket import *

class State_0:
    road = None
    def __init__(self,br,it,turn = 0,spin_count = 0,pack = '1234',dire = None):
        self.br,self.it,self.turn,self.spin_count,self.pack,self.dire = br,it,turn,spin_count,pack,dire

class State_1:
    br,start_pos,goal_pos = [None] * 3
    tier = {'0':1,'1':2,'2':2,'3':2,'4':2}
    def __init__(self,pos):
        self.pos,self.pack = pos,self.br[pos]
    def __lt__(self,other):
        return self.tier[self.pack] <  self.tier[other.pack]

class State_2:
    start_pos,goal_pos = [None] * 2
    def __init__(self,br,pack_dire,move_dire=None):
        self.br,self.pack_dire,self.move_dire = br,pack_dire,move_dire

def exchange(state, now, new, it=None, turn=None, spin_count=None, dire=None, B1orB2=None):
    def exchanging(br,now,new):
        br = list(br)
        br[now],br[new] = br[new],'0'
        return ''.join(br)
    if not G_Type:
        br = exchanging(state.br,now,new)
        pack =  (('34' if state.br[new] in '12' else '12') if state.pack == '1234' else
                 ('34' if state.pack == '12' else '12'))
        return State_0(br,it,turn + 1,spin_count,pack,dire)
    else:
        new_br,new_pack_dire = [None]*2,[None]*2
        B = [B1orB2,not(B1orB2)]
        for i in range(2):
            n = B[i]
            size = G_size[n]
            init_dires = deque([-size[1],1,size[1],-1])
            dires_idxs = deque([0,1,2,3])
            dires = init_dires.copy()
            dires.rotate(state.pack_dire[n])
            dires_idxs.rotate(state.pack_dire[n])
            if not i:
                pack_move_dire = dires_idxs[init_dires.index(now - new)]
                move_dire = pack_move_dire
                br = exchanging(state.br[n],now,new)
                pack_dire = init_dires.index(now - new)
            else:
                pack_move_dire = init_dires.index(init_dires[dires_idxs.index(pack_move_dire)])
                new = state.br[n].index('1')
                y,x = new // size[1], new % size[1]
                ny,nx = y + G_dy[pack_move_dire],x + G_dx[pack_move_dire]
                nz = ny * size[1] + nx
                if  0 <= ny < size[0] and 0 <= nx < size[1] and state.br[n][nz] == '0':
                    br = exchanging(state.br[n],nz,new)
                    pack_dire = init_dires.index(nz - new)
                else:
                    br,pack_dire = state.br[n],state.pack_dire[n]
            new_br[n],new_pack_dire[n] = br,pack_dire
        return State_2(new_br,new_pack_dire,move_dire)

def expand_sorting(state):
    result = []
    def rules(state,ny,nx,nz,size,it=None,dire_idx=None,B1orB2=None):
        br = state.br[B1orB2] if G_Type else state.br
        if  not(0 <= ny < size[0] and 0 <= nx < size[1]) or \
            br[nz] in ('x','0'):
                return False
        if not G_Type:
            if  br[nz] not in state.pack or \
            it[(dire_idx + 4) % 8] not in state.pack:
                    return False
        return True
    if not G_Type:
```

```
        n_p = [i for i in range(len(state.br)) if state.br[i] == '0']
        size = G_size
        it_states = list({0,state.turn,state.turn*-1})
        init_it = state.it.copy()
        for i in it_states:
            it,turn = init_it.copy(),0 if i else state.turn
            it.rotate(i)
            for pos in n_p:
                y,x = pos // size[1],pos % size[1]
                for j in range(8):
                 ay,ax = y + -G_dy[j],x + -G_dx[j]
                    az = ay * size[1] + ax
                    for Len in range(1,6):
                        ny,nx = y + (G_dy[j] * Len),x + (G_dx[j] * Len)
                        nz = ny * size[1] + nx
                        if Len == 1:
                            if  rules(state,ny,nx,nz,size,it,j) and \
                                state.br[nz] in '13':
                                    result.append(exchange(state,pos,nz,it,turn,i,j))
                        if  (not(0 <= ay < size[0] and 0 <= ax < size[1]) or state.br[az] != '0') and \
                            rules(state,ny,nx,nz,size,it,j) and \
                            state.br[nz] in '24':
                                result.append(exchange(state,pos,nz,it,turn,i,j))
                        else:
                            if  not(0 <= ny < size[0] and 0 <= nx < size[1]) or \
                                state.br[nz] != '0':
                                    break
    else:
        for i in range(2):
            n_p = [j for j in range(len(state.br[i])) if state.br[i][j] == '0']
            size = G_size[i]
            for pos in n_p:
                y,x = pos // size[1],pos % size[1]
                for j in range(4):
                    ny,nx = y + G_dy[j], x + G_dx[j]
                    nz = ny * size[1] + nx
                    if rules(state,ny,nx,nz,size,None,None,i):
                        result.append(exchange(state,pos,nz,None,None,None,None,i))
    return result

def expand_getdire(state):
    result = []
    dy,dx = (-1,0,1,0),(0,1,0,-1)
    y,x = state.pos // G_size[1],state.pos % G_size[1]
    for i in range(4):
        ny,nx = y + dy[i], x + dx[i]
        nz = ny * G_size[1] + nx
        if  0 <= ny < G_size[0] and 0 <= nx < G_size[1] and \
            state.br[nz] != 'x':
                result.append(State_1(nz))
    return result

def bfs_sorting(root):
    def get_br(n):
        if not G_Type:
            return n.br + str(''.join(n.it)) + str(n.turn) + str(n.pack)
        else:
            return str(n.br) + str(n.pack_dire) + str(n.move_dire)
    que = deque()
    que.append(root)
    marked = {root:'root'}
    br_marked = {get_br(root)}
    state = root
    def isleaf(br):
        if not G_Type:
            road_packs = [br[i] for i in state.road if br[i] != '0']
            return bool(road_packs)
        else:
            pack_pos = [br[0].index('1'),br[1].index('1')]
            return pack_pos != state.goal_pos
    while isleaf(state.br):
        state = que.popleft()
        for new_state in expand_sorting(state):
            new_br = get_br(new_state)
            if new_br not in br_marked:
```

```
                marked[new_state] = state
                br_marked.add(new_br)
                que.append(new_state)
    marked['leaf'] = state
    return marked

def bfs_getdire(root):
    que = PriorityQueue()
    que.put(root)
    marked = {root.pos:'root'}
    state = root
    while state.pos != state.goal_pos:
        state = que.get()
        for new_state in expand_getdire(state):
            if new_state.pos not in marked:
                marked[new_state.pos] = state.pos
                que.put(new_state)
    marked['leaf'] = state.pos
    return marked

def path(marked):
    state = marked['leaf']
    path = [state]
    while marked[state] != 'root':
            state = marked[state]
            path.append(state)
    return path[::-1]

def convert(result):
    path = []
    first = result[0]
    for second in result[1:]:
        step = ([[None]*2,[None]*2,second.pack_dire,second.move_dire] if G_Type else
                [None,None,second.turn,second.spin_count,second.pack,second.dire])
        for i in range(2):
            first_br = first.br[i] if G_Type else first.br
            second_br = second.br[i] if G_Type else second.br
            STEP = step[i] if G_Type else step
            for j in range(len(second_br)):
                if first_br[j] != second_br[j]:
                    if first_br[j] =='0':
                        STEP[1] = j
                    else:
                        STEP[0] = j
            if not G_Type and not i:
    break
        path.append(step)
        first = second
    return path

def main(n):
    global G_Type,G_size,G_dy,G_dx
    G_Type = n[0]
    G_size = [(6,4),((4,3),(3,4))][G_Type]
    G_dy,G_dx = [[(-1,-1,0,1,1,1,0,-1),(0,1,1,1,0,-1,-1,-1)],
                 [(-1,0,1,0),(0,1,0,-1)]][G_Type]
    if not G_Type:
        br,it,start,end = n[1:]
        State_1.br,State_1.goal_pos = br,end
        getdire_result = path(bfs_getdire(State_1(start)))

        State_0.road = getdire_result
        sorting_result = path(bfs_sorting(State_0(br,deque(it))))
        result = [getdire_result,convert(sorting_result)]

    else:
        br,pack_dire,start,end = n[1:]
        State_2.start_pos,State_2.goal_pos = start,end
        sorting_result = path(bfs_sorting(State_2(br,pack_dire)))
        result = convert(sorting_result)
    return result

while 1:
    Sock = socket(AF_INET, SOCK_STREAM)
    try:
```

```
        Sock.connect(('192.168.137.100',20001))
    except:
        print('.')
        continue
    else:
        try:
            n = eval(Sock.recv(1024).decode('utf-8'))
            result = main(n)
            Sock.send(bytes(str(result),'utf-8'))
        except:
            continue
```