```
'''
Goal Stacking Problem
Conditions -> ON, ONTABLE, CLEAR, HOLDING, ARMEMPTY
Operations -> Stack, Unstack, Pickup, Putdown
Algo ->
1) Push original goal in the stack
2) While stack is not empty
    a) If the top is compound goal then
        i) pop it
    b) If the top is a single unsatisfied condition
        i) Replace it by action and push action's precondition
    c) If the top is action then
        i) pop it and perform the action
    d) If the top in empty
        i) Success
'''


class Predicate:
    def __str__(self):
        pass
    def __repr__(self):
        pass
    def __eq__(self, other):
        pass
    def __hash__(self):
        pass
    def get_action(self, world_state):
        pass

class Operation:
    def __str__(self):
        pass
    def __repr__(self):
        pass
    def __eq__(self, other):
        pass
    def precondition(self):
        pass
    def delete(self):
        pass
    def add(self):
        pass

# Conditions

class ON(Predicate):
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y
```

```
    def __str__(self):
        return f'ON({self.X}, {self.Y})'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        return StackOp(self.X, self.Y)

class ONTABLE(Predicate):
    def __init__(self, X):
        self.X = X
    def __str__(self):
        return f'ONTABLE({self.X})'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        return PutdownOp(self.X)

class CLEAR(Predicate):
    def __init__(self, X):
        self.X = X
    def __str__(self):
        return f'CLEAR({self.X})'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        for predicate in world_state:
            if isinstance(predicate, ON) and predicate.Y == self.X:
                return UnstackOp(predicate.X, predicate.Y)
        return None

class HOLDING(Predicate):
    def __init__(self, X):
        self.X = X
    def __str__(self):
        return f'HOLDING({self.X})'
```

```
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        if ONTABLE(self.X) in world_state:
            return PickupOp(self.X)
        for predicate in world_state:
            if isinstance(predicate, ON) and predicate.X == self.X:
                return UnstackOp(predicate.X, predicate.Y)
        return None

class ARMEMPTY(Predicate):
    def __init__(self):
        pass
    def __str__(self):
        return 'ARMEMPTY()'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def __hash__(self):
        return hash(str(self))
    def get_action(self, world_state):
        for predicate in world_state:
            if isinstance(predicate, HOLDING):
                return PutdownOp(predicate.X)
        return None

# Operations

class StackOp(Operation):
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y
    def __str__(self):
        return f'StackOp({self.X}, {self.Y})'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def precondition(self):
        return [CLEAR(self.Y), HOLDING(self.X)]
    def delete(self):
        return [CLEAR(self.Y), HOLDING(self.X)]
```

```
    def add(self):
        return [ ARMEMPTY(), ON(self.X, self.Y)]


class UnstackOp(Operation):
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y
    def __str__(self):
        return f'UnstackOp({self.X}, {self.Y})'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def precondition(self):
        return [ARMEMPTY(), ON(self.X, self.Y), CLEAR(self.X)]
    def delete(self):
        return [ARMEMPTY(), ON(self.X, self.Y)]
    def add(self):
        return [CLEAR(self.Y) , HOLDING(self.X)]

class PickupOp(Operation):
    def __init__(self, X):
        self.X = X
    def __str__(self):
        return f'PickupOp({self.X})'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def precondition(self):
        return [ARMEMPTY(), ONTABLE(self.X), CLEAR(self.X)]
    def delete(self):
        return [ARMEMPTY(), ONTABLE(self.X)]
    def add(self):
        return [HOLDING(self.X)]

class PutdownOp(Operation):
    def __init__(self, X):
        self.X = X
    def __str__(self):
        return f'PutdownOp({self.X})'
    def __repr__(self):
        return self.__str__()
    def __eq__(self, other):
        return self.__dict__ == other.__dict__ and self.__class__ ==
other.__class__
    def precondition(self):
```

```
        return [HOLDING(self.X)]
    def delete(self):
        return [HOLDING(self.X)]
    def add(self):
        return [ARMEMPTY(), ONTABLE(self.X)]

def isPredicate(obj):
    predicates = [ON, ONTABLE, CLEAR, HOLDING, ARMEMPTY]
    for predicate in predicates:
        if isinstance(obj,predicate):
            return True
    return False

def isOperation(obj):
    operations = [StackOp, UnstackOp, PickupOp, PutdownOp]
    for operation in operations:
        if isinstance(obj,operation):
            return True
    return False

def arm_status(world_state):
    for predicate in world_state:
        if isinstance(predicate, HOLDING):
            return predicate
    return ARMEMPTY()

class GoalStackPlanner:
    def __init__(self, initial_state, goal_state):
        self.initial_state = initial_state
        self.goal_state = goal_state
    def get_steps(self):
        steps = []
        stack = []

        world_state = self.initial_state
        stack.append(self.goal_state)

        while len(stack) != 0:
            stack_top = stack[-1]

            #if stack top in compound goal
            if type(stack_top) is list:
                compound_goal = stack.pop()
                for goal in compound_goal:
                    if goal not in world_state:
                        stack.append(goal)

            #if stack top is action
            elif isOperation(stack_top):
                operation = stack[-1]
```

```
                all_preconditions_satisfied = True

                for predicate in operation.delete():
                    if predicate not in world_state:
                        all_preconditions_satisfied = False
                        stack.append(predicate)

                if all_preconditions_satisfied:
                    stack.pop()
                    steps.append(operation)

                    for predicate in operation.delete():
                        world_state.remove(predicate)
                    for predicate in operation.add():
                        world_state.append(predicate)

            #if stack top is satisfied goal
            elif stack_top in world_state:
                stack.pop()

            #if stack is unsatisfied goal
            else:
                unsatisfied_goal = stack.pop()

                #Replace Unsatisfied Goal with an action that can
complete it
                action = unsatisfied_goal.get_action(world_state)

                stack.append(action)
                #Push Precondition on the stack
                for predicate in action.precondition():
                    if predicate not in world_state:
                        stack.append(predicate)

        return steps

if __name__ == '__main__':
    initial_state = [
        ON('B','A'),
        ONTABLE('A'),ONTABLE('C'),ONTABLE('D'),
        CLEAR('B'),CLEAR('C'),CLEAR('D'),
        ARMEMPTY()
    ]

    goal_state = [
        ON('B','D'),ON('C','A'),
        ONTABLE('D'),ONTABLE('A'),
        CLEAR('B'),CLEAR('C'),
        ARMEMPTY()
    ]
```

```
    goal_stack = GoalStackPlanner(initial_state=initial_state,
goal_state=goal_state)
    steps = goal_stack.get_steps()
    print(steps)

[PickupOp(C), PutdownOp(C), UnstackOp(B, A), PutdownOp(B),
PickupOp(C), StackOp(C, A), PickupOp(B), StackOp(B, D)]
```