```
# Implementation of Brute Force Algorithm with Comments. - Dela Cruz, Gabrielle
class State:
    def __init__(self, islandA, islandB): #Frozensets used for immutability and hashability
        self.islandA = frozenset(islandA)
        self.islandB = frozenset(islandB)

    def safe(self):
        islandA, islandB = self.islandA, self.islandB
        unsafe = [('Wolf', 'Sheep'), ('Sheep', 'Cabbage')] #Tuples in list to pair items u
        return all('Farmer' in islandA or not (a in islandA and b in islandA) for a, b in u
               all('Farmer' in islandB or not (a in islandB and b in islandB) for a, b in u
       #a and b represent the pairs of items in tuples
       #Checks if Farmer is in Island, automatically returns True (safe); otherwise, check

    def __eq__(self, other):
        return self.islandA == other.islandA and self.islandB == other.islandB
        #Other is the comparative parameters. Checks, returns "True", if both islands' cur

    def __hash__(self):
        return hash((self.islandA, self.islandB))
        #Frozensets are hashable. Checks for both the frozensets, generates hash value to

def nextmove(state):
    islandA, islandB = state.islandA, state.islandB #Extract contents of islandA and islan
    nextState = [] #Initializes the state to store new possible next states.
    if 'Farmer' in islandA: #Iterates on this island if "Farmer" is present. Otherwise, che
        for item in islandA:
            if item != 'Farmer': #Checks each item other than the farmer.
                nextState.append(State(islandA - {'Farmer', item}, islandB | {'Farmer', ite
                nextState.append(State(islandA - {'Farmer'}, islandB | {'Farmer'}))
    else:
        for item in islandB:
            if item != 'Farmer':
                nextState.append(State(islandA | {'Farmer', item}, islandB - {'Farmer', ite
                nextState.append(State(islandA | {'Farmer'}, islandB - {'Farmer'}))
    return nextState #Returns the list of appended possible next state towards the empty l

def solve():
    startpoint = State({'Farmer', 'Wolf', 'Sheep', 'Cabbage'}, set()) #Initial instance of
    endgoal = State(set(), {'Farmer', 'Wolf', 'Sheep', 'Cabbage'})    #Ending instanec of '
    return recursion(startpoint, endgoal, set(), [])
    #When solving the problem, recursion between the startpoint and endgoal is instigated,

def recursion(current, endgoal, visited, path): #Current parameter that holds current stat
    if current == endgoal:        #Checks if current state meets goal state.
        return path + [current]   #Return entire path taken including the recent, current
    visited.add(current)          #Add current state to visited state.
    for nextState in nextmove(current): #Loop of possible next states from the current sta
        if nextState not in visited and nextState.safe(): #Next possible state that is not
```

```
            result = recursion(nextState, endgoal, visited, path + [current]) #Recursion li
            if result:
                return result #Return result of recursion.
    visited.remove(current) #Backtracking, removes current state from visited state if no v
    return None #Returns none if no valid solutions are found, current path has no leading

solution = solve() #Object to instantiate the solve(), which also calls the recursion funct

if solution:
    for i, state in enumerate(solution):  #Returns iterable counter value and the state for
        print(f"\nState {i + 1}:")        #Prints the looping +1 for every state that is ma
        print(f"  islandA: {', '.join(state.islandA) if state.islandA else 'No Animal or Ob
        print(f"  islandB: {', '.join(state.islandB) if state.islandB else 'No Animal or Ob
else:
    print("No solution found.")
```

```
State 1:
  islandA: Sheep, Farmer, Wolf, Cabbage
  islandB: No Animal or Object

State 2:
  islandA: Wolf, Cabbage
  islandB: Sheep, Farmer

State 3:
  islandA: Farmer, Wolf, Cabbage
  islandB: Sheep

State 4:
  islandA: Cabbage
  islandB: Sheep, Farmer, Wolf

State 5:
  islandA: Sheep, Farmer, Cabbage
  islandB: Wolf

State 6:
  islandA: Sheep
  islandB: Farmer, Wolf, Cabbage

State 7:
  islandA: Sheep, Farmer
  islandB: Wolf, Cabbage

State 8:
  islandA: No Animal or Object
  islandB: Sheep, Farmer, Wolf, Cabbage
```

```
# Implementation of Brute Force Algorithm without Comments. - Dela Cruz, Gabrielle
class State:
    def __init__(self, islandA, islandB):
        self.islandA = frozenset(islandA)
        self.islandB = frozenset(islandB)

    def safe(self):
        islandA, islandB = self.islandA, self.islandB
        unsafe = [('Wolf', 'Sheep'), ('Sheep', 'Cabbage')]
        return all('Farmer' in islandA or not (a in islandA and b in islandA) for a, b i
              all('Farmer' in islandB or not (a in islandB and b in islandB) for a, b in

    def __eq__(self, other):
        return self.islandA == other.islandA and self.islandB == other.islandB

    def __hash__(self):
        return hash((self.islandA, self.islandB))

def nextmove(state):
    islandA, islandB = state.islandA, state.islandB
    nextState = []
    if 'Farmer' in islandA:
        for item in islandA:
            if item != 'Farmer':
                nextState.append(State(islandA - {'Farmer', item}, islandB | {'Farmer', 
                nextState.append(State(islandA - {'Farmer'}, islandB | {'Farmer'}))
    else:
        for item in islandB:
            if item != 'Farmer':
                nextState.append(State(islandA | {'Farmer', item}, islandB - {'Farmer', 
                nextState.append(State(islandA | {'Farmer'}, islandB - {'Farmer'}))
    return nextState

def solve():
    startpoint = State({'Farmer', 'Wolf', 'Sheep', 'Cabbage'}, set())
    endgoal = State(set(), {'Farmer', 'Wolf', 'Sheep', 'Cabbage'})
    return recursion(startpoint, endgoal, set(), [])

def recursion(current, endgoal, visited, path):
    if current == endgoal:
        return path + [current]
    visited.add(current)
    for nextState in nextmove(current):
        if nextState not in visited and nextState.safe():
            result = recursion(nextState, endgoal, visited, path + [current])
            if result:
                return result
    visited.remove(current)
    return None

solution = solve()

if solution:
    for i, state in enumerate(solution):
          print(f"\nState {i + 1}:")
          print(f"  islandA: {', '.join(state.islandA) if state.islandA else 'No Animal or
          print(f"  islandB: {', '.join(state.islandB) if state.islandB else 'No Animal or
  else:
      print("No solution found.")


# Catulay, Ballesteros
islandA = []
islandA.append('jack')
islandA.append('wolf')
islandA.append('sheep')
islandA.append('cabbage')
print("Island A passengers")
print(islandA)
print("")

islandB = []
print("Island B passengers")
print(islandB)
print("==========================================")
#Nothing animal or object yet across the river

islandA1 = []
islandA1.append('wolf')
islandA1.append('cabbage')
print("Current Island A passengers")
print(islandA1)

print("\nPassengers removed from Island A")
print(islandA.pop(0))
print(islandA.pop(1))

islandB1 = []
islandB1.append('jack')
islandB1.append('sheep')
print("\nCurrent Island B passengers")
print(islandB1)
print("==========================================")
#Jack needs to go back to Island A to get the cabbage

islandB2 = []
islandB2.append('sheep')
print("Current Island B passengers")
print(islandB2)

print("\nPassengers removed from Island B")
print(islandB1.pop(0))

islandA2 = []
islandA2.append('jack')
```

```
islandA2.append('wolf')
islandA2.append('cabbage')
print("\nCurrent Island A passengers")
print(islandA2)
#Jack will across the river with the cabbage and leave the wolf

print("\nPassengers removed from Island A")
print(islandA2.pop(0))
print(islandA2.pop(1))

islandB3 = []
islandB3.append('jack')
islandB3.append('sheep')
islandB3.append('cabbage')
print("\nCurrent Island B passengers")
print(islandB3)
print("==========================================")
#Jack needs to go back to Island A with the sheep to get the wolf

islandB4 = []
islandB4.append('cabbage')
print("Current Island B passengers")
print(islandB4)

print("\nPassengers removed from Island B")
print(islandB3.pop(0))
print(islandB3.pop(0))

islandA3 = []
islandA3.append('jack')
islandA3.append('sheep')
islandA3.append('wolf')
print("\nCurrent Island A passengers")
print(islandA3)

print("\nPassengers removed from Island A")
print(islandA3.pop(0))
print(islandA3.pop(1))

islandB4 = []
islandB4.append('jack')
islandB4.append('wolf')
islandB4.append('cabbage')
print("\nCurrent Island B passengers")
print(islandB4)
print("==========================================")
#Jack gets back the sheep from Island A and leaves the wolf and cabbage

islandB5 = []
islandB5.append('wolf')
islandB5.append('cabbage')
```

```
islandB5.append('cabbage')
print("Current Island B passengers")
print(islandB5)

print("\nPassengers removed from Island B")
print(islandB4.pop(0))

islandA4 = []
islandA4.append('jack')
islandA4.append('sheep')
print("\nCurrent Island A passengers")
print(islandA4)

print("\nPassengers removed from Island A")
print(islandA4.pop(0))
print(islandA4.pop(0))

islandB5 = []
islandB5.append('jack')
islandB5.append('sheep')
islandB5.append('wolf')
islandB5.append('cabbage')
print("\nCurrent Island B passengers")
print(islandB5)

islandA5 = []
print("\nCurrent Island A passengers")
print(islandA5)
#All passengers are safely with the farmer in the other island
```

```
Island A passengers
['jack', 'wolf', 'sheep', 'cabbage']

Island B passengers
[]
==========================================
Current Island A passengers
['wolf', 'cabbage']

Passengers removed from Island A
jack
sheep

Current Island B passengers
['jack', 'sheep']
==========================================
Current Island B passengers
['sheep']

Passengers removed from Island B
jack

Current Island A passengers
```

```
['jack', 'wolf', 'cabbage']

Passengers removed from Island A
jack
cabbage

Current Island B passengers
['jack', 'sheep', 'cabbage']
==========================================
Current Island B passengers
['cabbage']

Passengers removed from Island B
jack
sheep

Current Island A passengers
['jack', 'sheep', 'wolf']

Passengers removed from Island A
jack
wolf

Current Island B passengers
['jack', 'wolf', 'cabbage']
==========================================
Current Island B passengers
['wolf', 'cabbage']

Passengers removed from Island B
jack

Current Island A passengers
['jack', 'sheep']
```