```
man = 0
lion = 1
goat = 2
grass = 3

def validate_problem_constraints(problem_configuration):
  if problem_configuration[lion] == problem_configuration[goat] and
problem_configuration[man] != problem_configuration[lion]:
    return False

  if problem_configuration[goat] == problem_configuration[grass] and
problem_configuration[man] != problem_configuration[goat]:
    return False

  return True

def display_current_problem_layout(problem_configuration):
  print("Starting Side: {}{}{}{}".format("Man " if not problem_configuration[man] else "",
"Lion " if not problem_configuration[lion] else "", "Goat " if not problem_configuration[goat]
else "", "Grass " if not problem_configuration[grass] else ""))
  print("Target Side: {}{}{}{}".format("Man " if problem_configuration[man] else "", "Lion " if
problem_configuration[lion] else "", "Goat " if problem_configuration[goat] else "", "Grass " if
problem_configuration[grass] else ""))
```

```
    print()


sample_case_1 = (False, False, False, False)
sample_case_2 = (False, True, True, False)
sample_case_3 = (True, True, False, False)

print("Valid: Everyone is on the left side:
{}".format(validate_problem_constraints(sample_case_1)))
display_current_problem_layout(sample_case_1)
print()

print("Invalid: The lion and goat are left without the farmer:
{}".format(validate_problem_constraints(sample_case_2)))
display_current_problem_layout(sample_case_2)
print()

print("Invalid: The goat and grass are left without the farmer:
{}".format(validate_problem_constraints(sample_case_3)))
display_current_problem_layout(sample_case_3)

initial_configuration = (False, False, False, False)
target_configuration = (True, True, True, True)
```

```
def generate_potential_children_configurations(problem_configuration):
    potential_children_configurations = []

    potential_configuration = (not problem_configuration[man], problem_configuration[lion],
problem_configuration[goat], problem_configuration[grass])
    if validate_problem_constraints(potential_configuration):
        potential_children_configurations.append(potential_configuration)

    if problem_configuration[man] == problem_configuration[lion]:
        potential_configuration = (not problem_configuration[man], not
problem_configuration[lion], problem_configuration[goat], problem_configuration[grass])
        if validate_problem_constraints(potential_configuration):
            potential_children_configurations.append(potential_configuration)

    if problem_configuration[man] == problem_configuration[goat]:
        potential_configuration = (not problem_configuration[man], problem_configuration[lion],
not problem_configuration[goat], problem_configuration[grass])
        if validate_problem_constraints(potential_configuration):
            potential_children_configurations.append(potential_configuration)

    if problem_configuration[man] == problem_configuration[grass]:
        potential_configuration = (not problem_configuration[man], problem_configuration[lion],
problem_configuration[goat], not problem_configuration[grass])
```

```
        if validate_problem_constraints(potential_configuration):
            potential_children_configurations.append(potential_configuration)

    return potential_children_configurations

def depth_first_search(initial_configuration, target_configuration):
    frontier_stack = [[initial_configuration]]
    explored_configurations = set([initial_configuration])

    while frontier_stack:
        configuration_sequence = frontier_stack.pop()
        problem_configuration = configuration_sequence[-1]

        if problem_configuration == target_configuration:
            return configuration_sequence

        for next_problem_configuration in
generate_potential_children_configurations(problem_configuration):
            if next_problem_configuration not in explored_configurations:
                explored_configurations.add(next_problem_configuration)
                next_configuration_sequence = configuration_sequence.copy()
                next_configuration_sequence.append(next_problem_configuration)
                frontier_stack.append(next_configuration_sequence)
```

```
    return None

final_configuration_sequence = depth_first_search(initial_configuration, target_configuration)

if final_configuration_sequence:
    for problem_configuration in final_configuration_sequence:
        display_current_problem_layout(problem_configuration)
else:
    print("No solution!")
```