```
# Coded By Parsa Yousefi Nejad
# Version 3: tellMove(), DLS_Search() methods added to the Code and some minor improvments were made
# Now River Crossing Problem Completely Solved

# Importing Necessary libraries
from os import system, name  # For clear() method
from copy import copy        # To shallow copy of an object
from time import sleep       # For implementing pause mechanism in pathShow()

# Display: Shows one Record Graphically
def Display(Record):
    listOfChars = ["POLICE", "THIEF", "FATHER", "MOTHER","DAUGHTER_1", "DAUGHTER_2", "SON_1", "SON_2",]
    shore = ('\x1b[0;32;42m'+"⇵"+'\x1b[0m') * 10
    plainText = '\033[2m'+"❖"+'\x1b[0m'+" {} "+'\x1b[4;35;43m'+"|"+'\x1b[0m'+'\x1b[1;33;34m' +
"~~~~~~~~~~~~~~~~"+'\x1b[0m'+'\x1b[4;35;43m'+"|" + '\x1b[0m'+" {} "+('\033[2m'+"❖"+'\x1b[0m')

    print(('\033[2m'+"❋"+'\x1b[0m') * 44)
    print(plainText.format(shore, shore))

    for i in range(0, 8):
        characterName = listOfChars[i] + \
            ('\x1b[1;32;42m'+"⇵"+'\x1b[0m') * (10 - len(listOfChars[i]))
        if Record[i] == 0:
            print(plainText.format(
                '\x1b[7;35;46m'+characterName+'\x1b[0m', shore))
        else:
            print(plainText.format(
                shore, '\x1b[7;35;46m'+characterName+'\x1b[0m'))
        if i == 3:
            if Record[8]:
                print('\033[2m'+"❖"+'\x1b[0m', shore, '\x1b[4;35;43m'+"|"+'\x1b[0m',
'\x1b[1;33;34m'+"~~~~~~~~~~~~~~~~"+'\x1b[0m','\x1b[1;34;41m'+"⛵"+'\x1b[0m', '\x1b[4;35;43m'+"|"+'\x1b[0m',
shore, '\033[2m'+"❖"+'\x1b[0m')
            else:
                print('\033[2m'+"❖"+'\x1b[0m', shore, '\x1b[4;35;43m'+"|"+'\x1b[0m',
'\x1b[1;34;41m'+"⛵"+'\x1b[0m','\x1b[1;33;34m'+"~~~~~~~~~~~~~~~~"+'\x1b[0m', '\x1b[4;35;43m'+"|"+'\x1b[0m',
shore, '\033[2m'+"❖"+'\x1b[0m')

    print(plainText.format(shore, shore))
    print(('\033[2m'+"❋"+'\x1b[0m') * 44)

# pathShow:  Shows Multiple States in order
def pathShow(List_States):
    if List_States == None:
        print("\x1B[41;2;35mThere is Nothing To Show\033[0m")
        exit(-1)
    Counter = 1
    previousState = List_States[0]
    for state in List_States[1:]:
        if Counter != 1:
            sleep(1)
        clear()
        print(f"\033[3;46;35mChild State {Counter}\033[0m")
        Display(state)
        tellMove(previousState, state)
        Counter += 1
        previousState = state

# clear:     Clears Terminal output
def clear():
    if name == 'nt':
        system('cls')
    else:
        system('clear')

# Assigning values to problem members
```

```
POLICE = 0; THIEF = 1; FATHER = 2; MOTHER = 3; DAUGHTER_1 = 4; DAUGHTER_2 = 5; SON_1 = 6; SON_2 = 7;
BOAT_Direction = 8

# Checks whether a state is valid
def isValid(state):
    return ((state[DAUGHTER_1] == state[MOTHER] or state[DAUGHTER_1] != state[FATHER]) and (
             state[DAUGHTER_2] == state[MOTHER] or state[DAUGHTER_2] != state[FATHER])) and ((

             state[SON_1] == state[FATHER] or state[SON_1] != state[MOTHER]) and (
             state[SON_2] == state[FATHER] or state[SON_2] != state[MOTHER])) and (

             state[POLICE] == state[THIEF] or (state[THIEF] != state[FATHER] and
             state[THIEF] != state[MOTHER] and state[THIEF] != state[DAUGHTER_1] and
             state[THIEF] != state[DAUGHTER_2] and state[THIEF] != state[SON_1] and
             state[THIEF] != state[SON_2]))

# checks if the state is the Goal
def isGoal(state):
    return state == [1, 1, 1, 1, 1, 1, 1, 1, 1]

# i t generates all states from a valid state and filters all invalid ones
def generateAllValidStates(state):

    if not isValid(state):
        print(
            '\n'+"\x1B[41;1;35mSorry I can\'t Generate States for an Invalid State\033[0m")
        exit(-1)

    valid_states = []
    for _ in [POLICE, THIEF, FATHER, MOTHER, DAUGHTER_1, DAUGHTER_2, SON_1, SON_2]:
# **//////////////////**
        if state[_] == state[FATHER] == state[BOAT_Direction]:
            new_State = copy(state)

            if new_State[_]:  new_State[_] = new_State[FATHER] = new_State[BOAT_Direction] = 0;
            else:             new_State[_] = new_State[FATHER] = new_State[BOAT_Direction] = 1;

            if isValid(new_State) and new_State not in valid_states:
                valid_states.append(new_State)
# //////////////////
        if state[_] == state[MOTHER] == state[BOAT_Direction]:
            new_State = copy(state)

            if new_State[_]: new_State[_] = new_State[MOTHER] = new_State[BOAT_Direction] = 0;
            else:            new_State[_] = new_State[MOTHER] = new_State[BOAT_Direction] = 1;

            if isValid(new_State) and new_State not in valid_states:
                valid_states.append(new_State)
# //////////////////
        if state[_] == state[POLICE] == state[BOAT_Direction]:
            new_State = copy(state)

            if new_State[_]: new_State[_] = new_State[POLICE] = new_State[BOAT_Direction] = 0;
            else:            new_State[_] = new_State[POLICE] = new_State[BOAT_Direction] = 1;

            if isValid(new_State) and new_State not in valid_states:
                valid_states.append(new_State)
# **//////////////////**
    return valid_states

# Describes State Changes in Context
def tellMove(state, new_state):
    peopleList = ['POLICE', 'THIEF', 'FATHER', 'MOTHER',
                  'DAUGHTER_1', 'DAUGHTER_2', 'SON_1', 'SON_2', 'BOAT_Direction']
```

```
    diff = list()
    for item1, item2 in zip(state, new_state):
        item = item1 - item2
        diff.append(item)

    Direction = 'RIGHT' if diff[8] == -1 else 'LEFT'

    movedPeople = list()
    for i in range(8):
        if diff[i] == -1 or diff[i] == 1:
            movedPeople.append(i)
    if len(movedPeople) == 1:

        print("\n" + f"\033[4;43;35m{peopleList[movedPeople[0]]}\033[0m" +
              ' moved to the ' f"\033[3;44;30m{Direction}\033[0m")
    else:
        print("\n" + f"\033[4;43;35m{peopleList[movedPeople[0]]}\033[0m"' and ' +
              f"\033[4;43;35m{peopleList[movedPeople[1]]}\033[0m"+' moved to the '
f"\033[3;44;30m{Direction}\033[0m")

# Depth-Limited-Search with list of Pre Expanded States
def DLS_Search(state, Depth_Limit, preExpandedNodesList):
    if not isValid(state):
        print(
            '\n'+"\x1B[41;1;35mSorry I cannot Find a Soution for an Invalid State\033[0m")
        exit(-1)

    # Adds Current Node to not Expanded List
    preExpandedNodesList.append(state)

    if isGoal(state):
        return preExpandedNodesList
    if Depth_Limit <= 0:
        return None

    # now state node has been expanded, list keeps Expanded Nodes
    for new_state in generateAllValidStates(state):
        # Display(new_state)     #Toggle Comment Block to Display All Explored Nodes
        # tellMove(state,new_state)
        if new_state not in preExpandedNodesList:
            result = DLS_Search(new_state, Depth_Limit-1, preExpandedNodesList)
            if result is not None:
                return result
            preExpandedNodesList.pop()
    return

# main part of the Code, Calling DLS_Search on begin state=[0..0]
# ////////////////MAIN/////////////////
final_States_List = DLS_Search([0, 0, 0, 0, 0, 0, 0, 0, 0], 20, [])
# DLS_Search([0,0,0,0,0,0,0,0,0],20 ,[])
# Also we can iterate this list to get every desired State
pathShow(final_States_List)

# By Parsa Yousefi Nejad
```