Individual Class Project

---couple river-crossing problem

i. Zhang Caiqi 18085481D

ii. Problem description

Three couples are on vacation and they need to cross the river to reach their hotel. The boat can only carry two people. The husband cannot let her wife with another man, without presence. The boat cannot run on itself. How can we solve the problem?

iii. Data abstraction

1. The location of each entity: 'E/W'.
2. There are totally 7 entities which are three couples (blue, green and red) and the last one is the boat's location. (E/W, E/W, E/W, E/W, E/W, E/W, E/W)
   Therefore, there are 128 possible states.
3. Due to the problem constraints, there are only 44 legal states. 10 of them are legal but useless which means it will not appear in graph, such as, 'EEEEEWW'（in this situation a wife moves to the west side but she needs to come back to continue the next step, so it is useless.）
4. The relationship of all 44 legal steps are shown in graph. (There are some empty next states in the graph but it will not influence the result because they will not appear in the path.)
5. Data types: (without viewing to any programing language)
   a) Boolean datum: for each state's each person
   b) A set of 7 boolean data: for each state
   c) A graph (consisting of a set of nodes and links)
   d) A sequence: for the shortest path

iv. The algorithm needed to solve the graph problem

Since the problem can be abstracted to a shortest-path problem for a graph, what we should do now is to implement an algorithm to find a shortest path from a source node to a destination node on a path. In the program, we use a recursion algorithm which is more concise.

v. A modular design of the program

def solver():

Input: None

Output: None

(The main program to print out the path that solves the problem)

S←genStates()

G←genGraph(S)

P←genshortestpath(G,s,d)

genTrip(p)

return

	Input	Output
def ganStates():	None	Return a set of all possible states
def genGraph(S):	A set of all possible states	Return a graph based on a set of all possible states and

		given
def isAStateLegal(s):	A state	If a state is legal, return true; else, false
def nextStates (aState, allStates):	aState is a state and allStates is a set of states.	Return a set of states that aState can go to (i.e., neighboring states).
def classifyManPos(S):	A set of states	Return the two sets of states, first the east-side states and the west.
def getPos(astate):	astate	how many people in the west side and how many people in east side.
def neighborNode(n1, n2):	n1, n2	True or False
def findShortestPath(graph, start, end, path=[]):	A graph, a starting node, an end node and an empty path.	Return the shortest path in the form of a list.
def find_all_paths (graph, start, end, path = []):	A graph, a starting node, an end node and an empty path	Return all the shortest path in the form of a list.
def printPath(path)	A shortest path	None
def printMore(G, start, end)	G(genGraph), start, end	None

vi. Python implementation of the data types

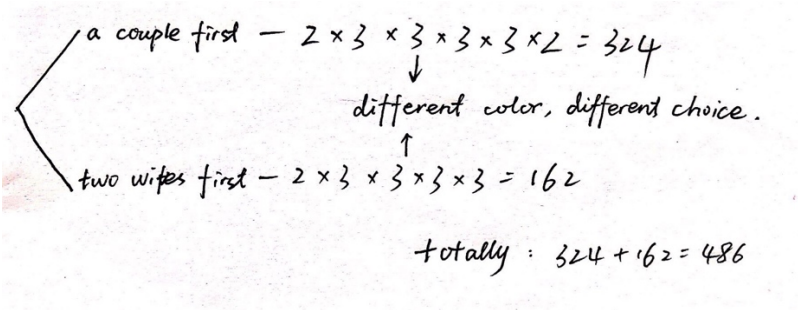
In my code:

Boolean datum: for each state	'E/W'
A set of 7 boolean data: for each state	string
A graph	A dictionary
A sequence: for the shortest path	A list of a set of 7 boolean data

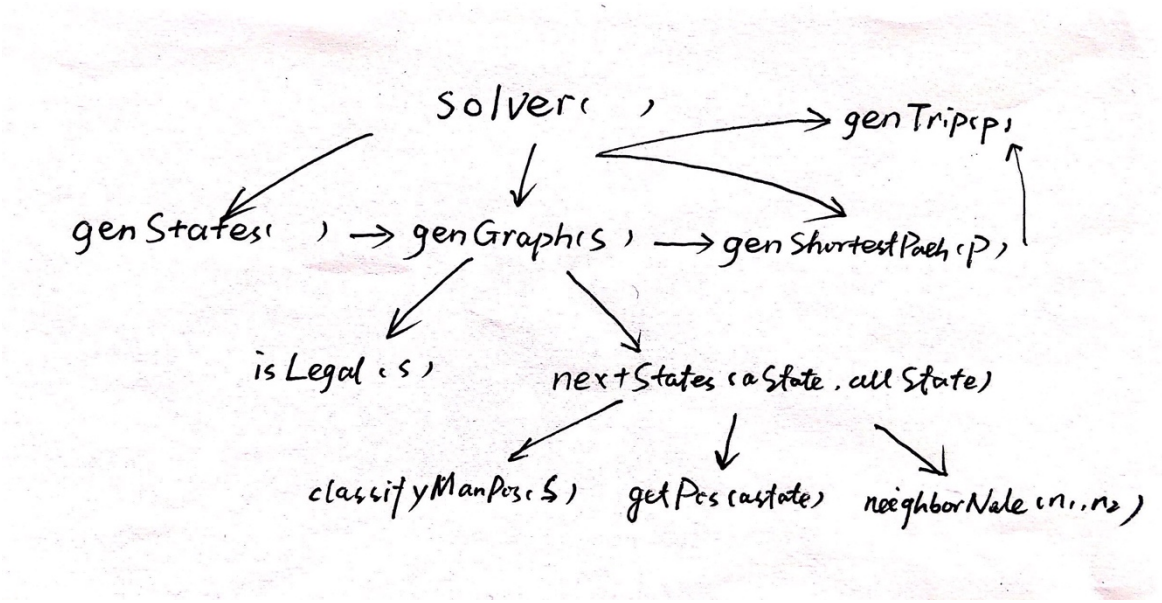
a) We can use 'E/W' or '0/1' or any Python Boolean variable for each state.
b) We can use strings, lists or tuples to implement a set of 7 boolean data but using strings is the easiest.
c) We can use a list of lists to implement the graph, but the time complexity is higher. Therefore, we use the dictionary.

Tips & Graph:

a) When running my program, it will ask you whether you want to get all the answer. You can print 'y' to get all the methods.
b) There are 486 methods mainly due to there are many combinations of different colors. Besides, what the first step is can have different number of methods. (a couple first or two women first)



c) **The algorithm design in function level**:



d) The genGraph:

```
EEEEEEE ['EEEEEWW', 'EEEEWWW', 'EEEWEEW', 'EEEWEWW', 'EEWWEEW', 'EWEEEEW', 'EWEEEWW', 'EWEWEEW', 'WWEEEEW']
EEEEEEW []
EEEEEWE ['EEEEWWW', 'EEEWEWW', 'EWEEEWW', 'EWEWEWW']
EEEEEWW ['EEEEEEE']
EEEEWWE ['EEWWWWW', 'WEWEWWW', 'WWEEWWW']
EEEEWWW ['EEEEEEE', 'EEEEEWE']
EEEWEEE ['EEEWEWW', 'EEWWEEW', 'EWEWEEW', 'EWEWEWW']
EEEWEEW ['EEEEEEE']
EEEWEWE ['EEWWWWW', 'EWEWEWW']
EEEWEWW ['EEEEEEE', 'EEEEEWE', 'EEEWEEE']
EEWWEEE ['EEWWWWW', 'WEWWWEW', 'WWWWEEW']
EEWWEEW ['EEEEEEE', 'EEEWEEE']
EEWWWWE ['WEWWWWW', 'WWWWWWW']
EEWWWWW ['EEEEWWE', 'EEEWEWE', 'EEWWEEE']
EWEEEEE ['EWEEEWW', 'EWEWEEW', 'EWEWEWW', 'WWEEEEW']
EWEEEEW ['EEEEEEE']
EWEEEWE ['EWEWEWW', 'WWEEWWW']
EWEEEWW ['EEEEEEE', 'EEEEEWE', 'EWEEEEE']
EWEWEEE ['EWEWEWW', 'WWWWEEW']
EWEWEEW ['EEEEEEE', 'EEEWEEE', 'EWEEEEE']
EWEWEWE []
EWEWEWW ['EEEEEWE', 'EEEWEEE', 'EEEWEWE', 'EWEEEEE', 'EWEEEWE', 'EWEWEEE']
WEWEWEE ['WEWEWWW', 'WEWWWEW', 'WEWWWWW', 'WWWEWEW', 'WWWEWWW', 'WWWWWEW']
WEWEWEW []
WEWEWWE ['WEWWWWW', 'WWWEWWW', 'WWWWWWW']
WEWEWWW ['EEEEWWE', 'WEWEWEE']
WEWWWEE ['WEWWWWW', 'WWWWWEW', 'WWWWWWW']
WEWWWEW ['EEWWEEE', 'WEWEWEE']
WEWWWWE ['WWWWWWW']
WEWWWWW ['EEWWWWE', 'WEWEWEE', 'WEWEWWE', 'WEWWWEE']
WWEEEEE ['WWEEEWW', 'WWWEWEW', 'WWWWEEW']
WWEEEEW ['EEEEEEE', 'EWEEEEE']
WWEEWWE ['WWWEWWW', 'WWWWWWW']
WWEEWWW ['EEEEWWE', 'EWEEEWE', 'WWEEEEE']
WWWEWEE ['WWWEWWW', 'WWWWWEW', 'WWWWWWW']
WWWEWEW ['WEWEWEE', 'WWEEEEE']
WWWEWWE ['WWWWWWW']
WWWEWWW ['WEWEWEE', 'WEWEWWE', 'WWEEWWE', 'WWWEWEE']
WWWWEEE ['WWWWWEW', 'WWWWWWW']
WWWWEEW ['EEWWEEE', 'EWEWEEE', 'WWEEEEE']
WWWWWEE ['WWWWWWW']
WWWWWEW ['WEWEWEE', 'WEWWWEE', 'WWWEWEE', 'WWWWEEE']
WWWWWWE []
WWWWWWW ['EEWWWWE', 'WEWEWWE', 'WEWWWEE', 'WEWWWWE', 'WWEEWWE', 'WWWEWEE', 'WWWEWWE', 'WWWWEEE', 'WWWWWEE']
```