Problem Solving in IT (COMP1001)
Individual Class Project

Tianrui WANG
14109121D
BBA in Accountancy

Project description

This project deals with the couple river-crossing problem. 3 couples has to take turns to ride on a boat to cross the river. There are 3 constrains for the problem:

1. The boat could only carry 2 people a time.
2. The wives couldn't stay with other men without their own husband's presence.
3. There should be at least one person stay on the boat.

Data abstraction

1. The location of each entity: E/W
2. The state of the system could be abstract as a combination of the 7 entities:

Husband 1	Wife 1	Husband 2	Wife 2	Husband 3	Wife 3	Boat
E/W	E/W	E/W	E/W	E/W	E/W	E/W

   In total, there are $2^7 = 128$ states.
3. Due to the problem constraints, there are only 42 legal states.
   - The constraints include:
     a. When wives are staying with their husband, there is no other constraints
     b. When wives are not staying with their husband, they could not stay with other men
     c. If everyone is staying on the same side, the boat could only be on this side
4. Relationship among the 42 legal states .
5. Required data types & python implementations for each:

Required data types	**Python implementations**
Boolean data for each entity E/W.	Strings : "E" & "W".
A set of 7 boolean data for each state, e.g. EEEEEEE.	A string combining 7 letters.
A graph.	A dictionary with each state as a key, and its neighbouring states as corresponding values contained in a list.
A sequence for the shortest path.	A list that stores the sequence from the starting state to the ending state.

Algorithm

def Solver():
    Input: None.
    Output: the shortest path.

def genStates():
    Input: None.
    Output: Return a set of all possible states for the problem, e.g. 128 states.

def genGraph(S):
    Input: S, a set of all possible states.
    Output: a graph connecting all the legal states in S.

    def islegal(s):
        Input: s, a state.

Output: return True if n is legal; otherwise, return False.

def nextStates(n, R):

Input: n, a state; R, the set of legal states.

Output: return a set of n's neighbouring states in R.

def genShortestPath(G, s, d):

Input: G, a graph connecting all the legal states; s, a source node; d, a destination node.

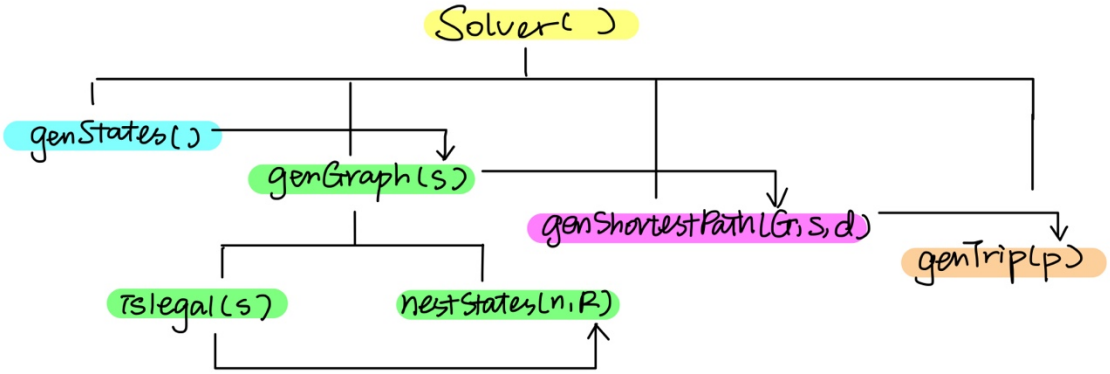
Output: Return a path connecting from s to d with minimum distance.

def genTrip(p):

Input: p, the shortest path from s to d

Output: print out the solution to the problem

Modular design



Graph from data abstraction showcasing all the possible results: