Program name-Problem Solving Methodology in IT **(COMP1001)**

Name- Sweta Das

Student ID-18080395D

Report- Class project

Problem description:

- There are three couples on a river bank. They have to reach the hotel which is on the other side of the bank, using only one boat.
- The conditions are:
    1. Maximum two people can sit on the boat.
    2. A woman cannot be with other men without the presence of her husband.
- Our goal is to find the shortest path for this problem out of the possible paths.

Data abstraction:

- 'E' means east, i.e., the river bank. 'W' means west, i.e., the hotel.
- There are total 64 possible states, but the position for the boat can be either in eat or west, so the total legal states saved are 128.
- Legal states are only 22, but the position for the boat can be either in eat or west, so the total legal states saved are 44.
- The states of people are represented as 'E' or 'W'.
- The positions of people and boat are as follows:
    1. First position for 'Green wife'.
    2. Second position for 'Green husband'.
    3. Third position for 'Blue wife'.
    4. Fourth position for 'Blue husband'.
    5. Fifth position for 'Red wife'.
    6. Sixth position for 'Red husband'.
    7. Seventh position for 'Boat'.
- The problem is to bring all people to 'W'.
- Each state can be represented as 'ABCDEFG', where A,B,C,D,E,F,G can be 'E' or 'W'. Each state is a node.
- As per the restrictions, we mention the legal states.

Algorithm:

- Start from 'EEEEEEE' and end at 'WWWWWWW'.
- Each state is a node. Each state can go to other possible states which are legal.
- But there are conditions to which legal states one state can proceed.
- Example: 'EEEEWEW' can only go to 'EEEEEEE' because 'Red' wife is on the hotel side with the boat alone, she can come alone only.
- We will write a function that finds the shortest path from one node to the other.
- To create the graph, we use dictionary and set. In between the process, we have to use list, string and tuple. These are sequence datatypes.

Modular design of the program:

- genStates() function is used to generate all possible 64 states.
- isAStateLegal(state) function takes a single argument, which is here one state. It checks whether the state is legal or not.
- nextStates(current,legal) function accepts two arguments, one is the current state which is being worked upon, the other is a set of legal states. It gives a set of those states to which the current state can go to, i.e., neighbour states.

- genGraph(S) function takes all possible states as input and produces a graph. It uses the above functions inside it to generate the graph.
- findShortestPath(graph, start, end, path=[]) function gives the shortest path after taking the starting node, the ending node and the whole graph as inputs.
- printPath(path) function takes the shortest path function as input and shows it to the reader in a readable format.
- solver() function manages the input and output of functions and is responsible to show the final result.

Python implementation of the data types:

- A Boolean value: Either ‘E’ or ‘W’.
- A set of seven Boolean values: ‘ABCDEFG’, where each of A,B,C,D,E,F,G can be either ‘E’ or ’W’.
- Graph: We use dictionary and set to create our graph.
- Sequence: For the shortest path, list is used. Also, to create the possible states list is used.

Graph:

{'EEEEEEE': {'WEEEWEW', 'EEWWEEW', 'EEWEEEW', 'EEEEWEW', 'EEWEWEW', 'EEEEWWW', 'WEWEEEW', 'WEEEEEW', 'WWEEEEW'}, 'EEEEEEW': set(), 'EEEEWEE': {'EEWEWEW', 'EEEEWWW', 'WEEEWEW', 'WEWEWEW'}, 'EEEEWEW': {'EEEEEEE'}, 'EEEEWWE': {'WWEEWWW', 'EEWWWWW', 'EWEWWWW'}, 'EEEEWWW': {'EEEEEEE', 'EEEEWEE'}, 'EEWEEEE': {'WEWEEEW', 'WEWEWEW', 'EEWEWEW', 'EEWWEEW'}, 'EEWEEEW': {'EEEEEEE'}, 'EEWEWEE': {'EEWWWWW', 'WEWEWEW'}, 'EEWEWEW': {'EEEEEEE', 'EEWEEEE', 'EEEEWEE'}, 'EEWWEEE': {'EEWWWWW', 'EWWWEWW', 'WWWWEEW'}, 'EEWWEEW': {'EEEEEEE', 'EEWEEEE'}, 'EEWWWWE': {'WWWWWWW', 'EWWWWWW'}, 'EEWWWWW': {'EEEEWWE', 'EEWWEEE', 'EEWEWEE'}, 'EWEWEWE': {'WWEWWWW', 'EWWWEWW', 'WWWWEWW', 'WWEWEWW', 'EWEWWWW', 'EWWWWWW'}, 'EWEWEWW': set(), 'EWEWWWE': {'WWWWWWW', 'WWEWWWW', 'EWWWWWW'}, 'EWEWWWW': {'EEEEWWE', 'EWEWEWE'}, 'EWWWEWE': {'WWWWWWW', 'WWWWEWW', 'EWWWWWW'}, 'EWWWEWW': {'EEWWEEE', 'EWEWEWE'}, 'EWWWWWE': {'WWWWWWW'}, 'EWWWWWW': {'EWEWWWE', 'EEWWWWE', 'EWWWEWE', 'EWEWEWE'}, 'WEEEEEE': {'WEWEEEW', 'WWEEEEW', 'WEEEWEW', 'WEWEWEW'}, 'WEEEEEW': {'EEEEEEE'}, 'WEEEWEE': {'WWEEWWW', 'WEWEWEW'}, 'WEEEWEW': {'EEEEEEE', 'EEEEWEE', 'WEEEEEE'}, 'WEWEEEE': {'WWWWEEW', 'WEWEWEW'}, 'WEWEEEW': {'EEEEEEE', 'EEWEEEE', 'WEEEEEE'}, 'WEWEWEE': set(), 'WEWEWEW': {'WEEEWEE', 'EEEEWEE', 'WEEEEEE', 'EEWEEEE', 'EEWEWEE', 'WEWEEEE'}, 'WWEEEEE': {'WWEEWWW', 'WWWWEEW', 'WWEWEWW'}, 'WWEEEEW': {'EEEEEEE', 'WEEEEEE'}, 'WWEEWWE': {'WWWWWWW', 'WWEWWWW'}, 'WWEEWWW': {'EEEEWWE', 'WEEEWEE', 'WWEEEEE'}, 'WWEWEWE': {'WWWWWWW', 'WWEWWWW', 'WWWWEWW'}, 'WWEWEWW': {'EWEWEWE', 'WWEEEEE'}, 'WWEWWWE': {'WWWWWWW'}, 'WWEWWWW': {'EWEWWWE', 'WWEWEWE', 'EWEWEWE', 'WWEEWWE'}, 'WWWWEEE': {'WWWWWWW', 'WWWWEWW'}, 'WWWWEEW': {'WEWEEEE', 'EEWWEEE', 'WWEEEEE'}, 'WWWWEWE': {'WWWWWWW'}, 'WWWWEWW': {'WWWWEEE', 'WWEWEWE', 'EWWWEWE', 'EWEWEWE'}, 'WWWWWWE': set(), 'WWWWWWW': set()}

- genGraph() funtion takes all possible states as input.
- The legal states are stored using the function isAStateLegal(state).
- nextStates(current,legal) function stores the possible neighbour states in a set for each of the legal state. This done using dictionary and set.
- Each legal state is a node, for which the respective legal neighbour states are stored.
- What the graph shows: It shows the possible states to which each state can go and the possible paths that can be created from the start node to the end node.

References:

[1] For the names of functions, solver() and genGraph(S) functions:

http://www2.comp.polyu.edu.hk/~comp1001/MCGWv10.py

[2] For findShortestPath(graph, start, end, path=[]) function:

https://www.python.org/doc/essays/graphs/

[3] For the video that contains the question:

https://www.youtube.com/watch?v=dgDnwD4ieWM