Solving a River-Crossing Problem

Class A

賴彥廷**(11120310)** 陳筠涵**(11120324)** 温芊惠**(11120334)**

1 January 2023

I. THE PROBLEM

The goal is to bring a man, a cabbage, a goat, and a wolf from the east bank to the west bank in the smallest number of crossing times. The following are the rules that must be complied with :

1 Only the man can operate the boat.

2. The boat can only carry two items each trip, and the man is counted as one item.

3. The cabbage can only stay with the goat with the man.

4. The goat cannot stay with the wolf without the man.

II. DATA ABSTRACTION

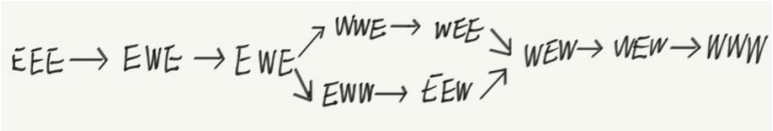
System States The goal of this data abstraction step is to transform all machine-readable forms into human language expressions. First, we use the three items' positions (cabbage, goat, wolf) to figure out which has changed with the man. In the total of fate, without including the man's location, eight possible states are(EEE, EWE, WWE, WEE, WEW, and WWW). Then we use the man's state to decide at which the state will be legal and decide the items moving in the direction it does move; for example, when the state is EEE, it will be legal if and only if the man is currently at E. The table is listed below:

The legal state if and only if the man is located at:	East(E)	West(W)	East(E) and West(W) are both legal
Item's state (the order of items is CGW)	EEE, WEE, EEW	WWE , EWW	EWE, WEW, WWW

Nodes and links We model each state by a node and compare the nodes to each other if the nodes are different from only one of the locations, it means the direction in which the items move, and the detail of the steps are listed below.

1. Firstly, we know that man will change every time until they reach the final goal.
2. Secondly, we know that the items would move in the same direction as the man if they did move.
3. Determine if the state is legal by considering the man's position.
4. If all the above conditions are perfectly met, they are neighbor nodes.

A shortest-path problem By solving this question, we come up with a path that can help us to find the solution more quickly.



There are two equally good paths for this question, and to fit the required output, we changed our item order from CGW to WGC

Abstract data types From the discussions above, we identify the following abstract data types (ADTs) in our data model.

1. A list of three items' locations.

2. A dictionary of a graph that keeps all legal states by 3 items and the list of next_states will keep the state before the current state and next_state.

3. Append the node that the "findshortespath" lost.

4. A list of the final path that shows the whole route(solution)

III. Algorithm Design

We find out the whole legal graph and input it into the given function "findShortestPath," find out the unfinished path and input it into the "ifnopath" function, and the output will be the absolute path. Then, finally, input the "printPath" function, and it will do all the formatting for us to print out the final goal.

IV. Program Design

The table listed below is to show how each function work:

Functions	Inputs	Outputs
main()	None	Print out the solution to the MCGW problem
genStates()	None	Return a set of all possible states (3 letters)
genLegalStates()	A list of current state, all state, and next position and current position.	A dictionary of current legal graphs and neighbor states.
genLegalGraph()	A list of all possible states, starting position and ending point	Return a graph whole graph
findShortestPath()	A graph, a source node and a destination node	Return a list of nodes that is a shortest path from the source node to the destination node
ifnopath()	A list of path from findShortestPath	Return the final path
printPath()	A list of nodes	Print out the solution to the MCGW problem

V. A PYTHON IMPLEMENTATION

• We use a list to keep the state of items(without man's position) 'E' means the East 'W' means the west. Moreover, why we use E and W is to have higher readability.

• We use a dictionary of legal states

• We use lists to keep the neighborhood. Since we need to append new states, it is used to help the genLegalGraph to get the legal neighbor node.

• We use a Python dictionary in which the keys are the states (implemented as 3-character), and the values are the neighboring states. The key is the middle node of two neighbor nodes which also implies the value. We can concatenate every function (genState, genLegalState, genLegalGraph), and it will be the solution for the findShortestPath then put the resulting path into ifnopath, and it will be the absolute path. Furthermore, input it to the printPath, and it will output as human language as shown below:

By the way, why did we implement ifnopath into our program? We noticed that the path that findShortestpath found would lose the state when the man returns by himself. Hence, we need this to help it decide when even the man goes back by himself, which implies the goat is not with the cabbage alone. So the goat is not with the wolf alone, it will append the same node to the final path.

The final code and output can be verify by G08.py