**Manisha Therupalli**

**CSCI 6511 Project 1**

In the "Water Jug problem" the objective is to determine the smallest number of actions necessary to take to meet the given target capacity. As a result, I determined four basic probable actions and actions are represented in a class named "Jug". The actions are:

1.FillWater: Completely fill the jug with liquid.

2.DrainWater: emptying the jug to zero.

3.PourWater: Pours the water in jug into the other jug of your choice.

4.Cap: Empties the jug into an infinite jug with the desired capacity.

Also, the Node Class contains the bucket array and the goal capacity's present condition. There will be a parent node and its offspring for every node. The potential and legitimate activities that the node can do are referred to as the children. Each potential state from the current state is consequently represented by the children.

The Water class allows us to execute the algorithm, which is the last step. We may set up the creation of the state space tree with the aid of the Water class. Prior to finding the node with the desired capacity, we first specify our starting node and build the state space tree. The create graph method performs this. If the objective state cannot be reached and there are no further legal actions left to take, this will likewise return -1. Once the function is complete, we can use apply heuristic to apply the heuristic value to each node (). An estimation of the true cost from node to end node is the heuristic value.

Histogram value lower bound: The number of edges between the starting node and the ending node serves as the lower bound. Initially, h(end node) = 0. Then, beginning from the end node, taking its parent while setting the parents cost to 1 and its children, omitting the current node, to 2. This would give me the value that I need. This process would be repeated until we finally arrived at the start node, which would represent the total cost of traveling to the end node.

Run the A search algorithm after applying the heuristic parameters. Employed   a priority queue and double the heuristic values of each child node by 1 to add them all together. We loop until either we have no solution (the priority queue is empty), in which case we return -1, or until we reach our goal state, in which case we return the number of steps. The child nodes are added if they haven't already been visited. As we are utilizing a priority queue, the number of steps is represented as the minimum cost path. So, the process is finished once we return the steps used to obtain the minimum. Then if loop will check where total steps equal to -1 not, if it is -1 then "No solution found " will be printed otherwise numbers of steps will be printed. Added new Python file and imported unittest for the unit testing. Using the help of the test cases provided in the project documentation and some others I came up with, I made a few examples.