COMPUTER SCIENCE AND DATA ANALYTICS

Course: **Artificial Intelligence**

Assignment 1
Informed Search

Student: **Shikhaliyev Anar**

Instructor: **Amrinder Arora**

Baku 2023

1. Problem Description

In this project, water pitcher problem will be implemented to find the shortest path to obtain the target volume of water in the infinite capacity pitcher. Additionally, any type of water movement (from one pitcher to another) counts as one step.
Since, I have also provided comments on the code about what each line stands for, the code will not be included in this report. Instead, the logic and the algorithmic approach will be explained right below.

2. Code analysis

This code is implementing the A* algorithm to solve the water jug problem. The goal of the problem is to find the minimum number of steps required to obtain a target quantity of water using jugs of different sizes. The code is structured into several functions that work together to solve the problem.

- **2.1 processFile()**
  is the driver function that takes the input data from a file and calls the AStarAlgorithm() function to find the shortest path to reach the target water level. It then prints the path and the number of steps taken to reach the target state.

- **2.2 takeInputs(fName)**
  takes a file name as an argument and reads the first two lines of the file. First line holding a variable number of integers, comma separated and second line holding an integer representing the target value. It returns a tuple consisting of a list of integers representing the capacities of the jugs and an integer representing the target water level. This function is used to read the input data from a file.

- **2.3 heuristic(jugs, target)**
  calculates the heuristic value for a state of the jugs. The function takes a list of integers representing the current state of the jugs and an integer representing the target water level as arguments. It returns a float representing the heuristic value.

The heuristic function estimates the remaining cost to reach the target state based on the difference between the current state and the target state.

The heuristic function first calculates the amount of water that is still needed to reach the target quantity by subtracting the current amount of water in the infinite capacity jug, from the target quantity. If the current amount of water in the infinite capacity jug is already greater than or equal to the target quantity, the heuristic function returns float('inf') to indicate that it's impossible to reach the target state from the current state.

If the remaining amount of water needed to reach the target quantity is greater than 0, the

heuristic function estimates the remaining cost as the sum of 1 and the ratio of the remaining amount of water to the number of jugs. This essentially means that the function estimates the remaining cost as the number of steps required to transfer the remaining amount of water to the last jug, assuming that each jug is used equally in the process.

This heuristic function is admissible because it never overestimates the actual remaining cost. The function returns float('inf') if it's impossible to reach the target state from the current state. Otherwise, it estimates the remaining cost as the number of steps required to transfer the remaining amount of water to the last jug, assuming that each jug is used equally in the process.

Since this is a **lower bound** on the actual remaining cost, the heuristic function is admissible.

- ## 2.4 AStarAlgorithm(capacities, target)

  is the main function that implements the A* algorithm to find the shortest path to reach the target water level. The function takes two parameters: a list of integers representing the capacities of the jugs and an integer representing the target water level as arguments. The function first initializes a heap data structure with
  - an initial state of the jugs
  - the heuristic value
  - the number of steps taken to reach the current state.

The function then initializes an empty set to keep track of the states that have already been visited during the search. It then enters into a loop that continues as long as the heap data structure is not empty.

To begin with, the A* algorithm is a search algorithm that uses heuristics to guide the search towards the goal. In this implementation, the function starts by appending a large number to the list of capacities, which represents the infinite capacity of an additional jug that is not included in the original capacities list. The heap data structure is then initialized with a tuple that represents the initial state of the problem. The tuple consists of a heuristic value of 0, which is the sum of the absolute difference between 0 and the target volume and the number of steps taken so far, which is initialized as 0. The third element is a list of 0's, representing the initial state where all the jugs are empty.

A set passed is created to keep track of the visited states during the search process. The loop continues until the heap is empty or the desired volume is reached in one of the jugs. Inside the loop, the **smallest** element is popped from the heap and its contents are assigned to three variables, _, steps, jugs. The steps variable is the number of steps taken so far to reach the current state and jugs is a list that represents the current state of the jugs.

If the last jug(infinite) contains the target amount of water, the function returns the number of steps taken to reach this state and the list of jugs. If the last jug contains more water than the target amount, the function returns -1, which indicates that the problem cannot be solved.

If the current state is not already in the set of visited states, it is added to the set, and nested loops are used to iterate over every pair of jugs in the list capacities, excluding any pair of jugs that are the same.

For each pair of jugs, the code creates a new copy of the jugs list and modifies it based on the capacities of the jugs in the pair.
First we check if the loop is iterating over the same jug. If they are, the loop moves on to the next iteration without executing the code that follows. Then we create a copy of the current jugs list so that we can modify it without affecting the original list.

In the filling process, first we check if it's not the last jug and we set the amount of water in the jug i to the maximum capacity of the jug capacity.
Otherwise, If jug **i** is the last jug in the capacities list, the code checks if jug **j** is not the last jug. If it's not the last jug, the code calculates how much water to pour from jug **j** to jug **i** **by taking the minimum of the amount of water in jug j and the amount of space left in jug i.** The code then updates the amounts of water in both jugs i and j accordingly by adding and subtracting the pour_amount value.

Finally, if the new state of jugs has not been visited before by checking if its tuple representation is not in the set passed. If the state is new, the code calculates the heuristic value of the new state and adds it to the heap data structure for later exploration.

3. Unit tests

In order to make sure the propgram works, I have written a Python unittest module that tests two functions, heuristic and AStarAlgorithm, implemented in the main module.

The test file consists of two test classes TestHeuristic and TestAStarAlgorithm that test the functions heuristic and AStarAlgorithm, respectively. The tests are implemented using Python's built-in unittest module.

The TestHeuristic class tests the heuristic function. It contains two test cases, each of which checks if the heuristic function is **admissible** and **consistent**. The test_admissible method tests the admissibility of the heuristic function by using different jug states and targets. It checks if the heuristic function is returning a value less than or equal to the actual cost of the optimal solution from the current state to the goal state. The test_consistent method tests the **consistency** of the heuristic function. It applies the A* algorithm to the given jug state and target and ensures that the estimated cost to reach the goal from a current state **is not greater than** the cost of reaching that current state plus the estimated cost to reach the goal from that current state.

The TestAStarAlgorithm class tests the AStarAlgorithm function. It contains two test cases, each of which tests if the A* algorithm is finding the optimal solution for different jug states and targets. The test cases provide the input capacities and target amounts to the A* algorithm and verify if the output matches the expected result.

Overall, the tests cover a wide range of jug states and target amounts and check for both admissibility and consistency of the heuristic function as well as the optimal solutions for the A* algorithm.

4. Source Code

https://github.com/anar-sixeliyev/AI-assignment1

5. Sample execution

```
PS C:\Users\Anar\Desktop\GWU-AI\assignment> python .\main.py sample01.txt
Path: [0, 0, 0, 0, 0, 181]
Steps: 20
PS C:\Users\Anar\Desktop\GWU-AI\assignment> python .\main.py sample02.txt
Path: [0, 0, 0, 17]
Steps: 10
PS C:\Users\Anar\Desktop\GWU-AI\assignment> python .\unit-tests.py
Usage: python main.py <filename>
....
----------------------------------------------------------------------
Ran 4 tests in 0.003s

OK
PS C:\Users\Anar\Desktop\GWU-AI\assignment>
```