Project 1 (Water Pitcher)

Instructor: *Dr. Amrinder Arora*
Course: Artificial Intelligence
Course-Code: 202501 - CSCI_6511_ADA
GWID: G33160048

Overview

The Water Pitcher problem is:

- You have several finite-capacity pitchers (3 liters, 5 liters, etc.) plus one **infinite** pitcher.
- Your goal is to measure out a specific target quantity of water into the infinite pitcher.
- Each move (pouring or filling) has a cost of 1 step.
- You want the **shortest** sequence of steps that results in the infinite pitcher holding the target amount.

File Structure

- **main.py**
  - `parse_file`: Reads capacities and target from file.
  - `can_measure`: Checks if target is possible (via GCD).
  - `heuristic`: **Lower bound** for A* = `ceil(remaining / smallest_pitcher)` (never overestimates).
  - `fill` / `pour`: Generate next states.
  - `a_star`: Runs A* to find shortest solution or -1 if impossible.
- **test.py**
  - Uses `unittest` to validate parsing, measuring, heuristic correctness, pouring/filling operations, and final A* results.

Tests ensure that:

- The input file is correctly parsed.
- Feasibility of measuring the target with given containers is accurately determined.
- The heuristic function provides reasonable estimates.
- Filling and pouring operations produce correct new states.
- The overall A* search algorithm correctly identifies both unsolvable and trivial cases.

A* Overview

A* expands states in order of their priority `f = g + h`, where:

- `g` = the number of steps taken so far.

- **h** = the heuristic (our **lower bound** on how many more steps are needed).

Why the Lower Bound Is Crucial

```
def heuristic(state, goal, sizes):
    needed = goal - state[-1]
    if not needed:
        return 0
    finite = sizes[:-1]
    return math.inf if not finite else ceil(abs(needed) / min(finite))
```

- The heuristic here in the code snippet, doesn't *overestimate* the remaining cost, so it is **admissible**.
- An admissible heuristic guarantees that A* always finds the optimal (shortest) path. Without a valid lower bound, it would just be a uniform-cost search, losing A*'s efficiency advantage.

Run program:

```
python main.py input1.txt
```

Run test:

```
python -m unittest test.py
```