Algorithms in C++: Assignment 4

1. Objective

Your goal is to write a program to solve the famous water jug puzzle using breadth-first search in C++.

2. Problem

Sim'eon Denis Poisson (1781–1840), a famous French mathematician and physicist, is said to have become interested in mathematics after encountering some version of the following old puzzle: Given an 8-gallon jug full of water and two empty jugs of 5- and 3-gallon capacity, get exactly 4 gallon of water in one of the jugs by completely filling up and/or emptying jugs into others. Solve this puzzle by using breadth-first search.

We will generalize this problem to work with 3 jugs of varying capacity. Let's say that we have 3 jugs, namely A, B, and C. Jug C will always start out completely full. Furthermore, we will define a goal state where jug A will contain *a* gallons, B will contain *b* gallons, and C will contain *c* gallons upon completion of the search. You need to use breadth-first search to find the minimum number of moves to go from the initial state to the goal state. The order in which you choose to pour the water from one jug to another will affect the order in your final solution. Therefore, to ensure your program produces the same result as that expected by the autograder, you must traverse the search space in the following order:

1) Pour from C to A
2) Pour from B to A
3) Pour from C to B
4) Pour from A to B
5) Pour from B to C
6) Pour from A to C

Your program should take in 6 command line arguments, as follows:

```
$ ./waterjugpuzzle
Usage: ./waterjugpuzzle <cap A> <cap B> <cap C> <goal A> <goal B> <goal C>
```

You should perform error checking on each of the inputs. If you are clever, you can write one function to handle this!

```
$ ./waterjugpuzzle 3 5 X 0 4 4
Error: Invalid capacity 'X' for jug C.

$ ./waterjugpuzzle 3 5 8 0 R 4
Error: Invalid goal 'R' for jug B.
```

Be wary of bad user input. The goal for a given jug can never exceed its capacity.

```
$ ./waterjugpuzzle 3 5 8 4 0 4
Error: Goal cannot exceed capacity of jug A.
```

In the example below, jug C has a capacity of 8 gallons, but the goal state is asking for 9, which is impossible. The same is true for goal states of less than 8 gallons for this particular example.

```
$ ./waterjugpuzzle 3 5 8 0 4 5
Error: Total gallons in goal state must be equal to the capacity of jug C.
```

If the program finds a solution, it should display it as follows:

```
$ ./waterjugpuzzle 3 5 8 0 2 6
```

```
Initial state. (0, 0, 8)
Pour 5 gallons from C to B. (0, 5, 3)
Pour 3 gallons from B to A. (3, 2, 3)
Pour 3 gallons from A to C. (0, 2, 6)
```

Sometimes, however, the puzzle has no solution. In that case, the program should notify the user as follows:

```
$ ./waterjugpuzzle 5 7 10 3 3 4
No solution.
```

3. Advice

This project is challenging; therefore, you are allowed to work in pairs. No template will be given, since every team will have different ideas about how to structure the code. The easiest way to pick up a lot of points is to start with parsing the command line arguments. Once your program accepts user data, start working on the breadth-first search.