Computer Science I — CSCI-141
Ball Puzzle — Lab 8

10/25/2023

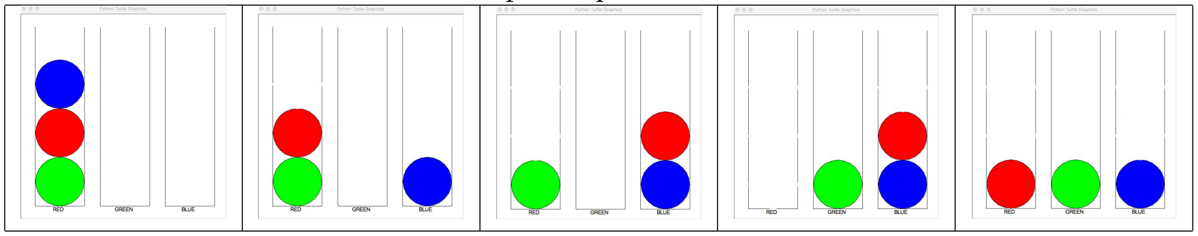
1 Problem

Consider the following puzzle: you are given an arbitrary number of balls of three colors: red, green and blue. You are also given three cans that are identical, except that one can is labelled ‘red’, one ‘green’, and one ‘blue’. Each can is capable of holding all or some of the balls, and open at just one end. The cans are just wide enough that the balls can slide in, but once inside, they can’t move around - they stay in the same order that they were put in. Initially, all the balls are in the ‘red’ can.

The only operation allowed is that you may pour any number of balls out of one can and directly into another. You may repeat this operation as many times as necessary. Your goal is to end up with each can (‘red’, ‘green’, ‘blue’) containing only balls of its respective color. Furthermore, you should try to accomplish this in as few moves as possible. A move is defined as a single ball leaving one can and entering a different can.

The images below show an example puzzle that can be solved in four moves, starting with three balls (‘blue’, ‘red’, ‘green’, from the top) in the ‘red’ can.

Table 1: Example Sequence for Ball Puzzle

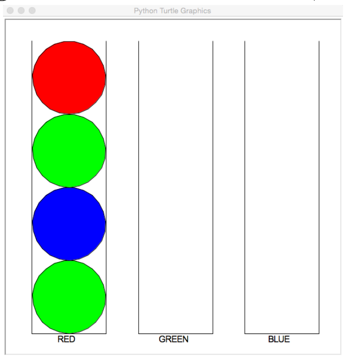


You will implement a program that solves this puzzle automatically (not necessarily in the minimum number of moves), and returns the number of moves it required to solve it.

1.1 Problem-solving Session (15%)

Work together in a team of students as determined by your instructor to complete the activities below.

1. Solve the puzzle whose initial configuration is shown below. Using illustrations as in the example (just labelling the different balls 'R', 'G', 'B'), show each move.

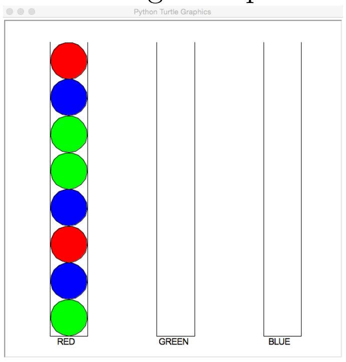


2. What data structure(s) will be useful for your internal representation of the puzzle? As you answer this question, consider how the balls are accessed. Also, consider that there are variations of the puzzle that involve more than 3 cans. Would you want a variable for each can? How can you represent a sequence of cans?

3. Write a function that moves one ball from the top of one can to another can. Your function should be able to use the data structures you chose in the previous question. You might also want to provide the function with parameters indicating which cans are involved in the operation. Assume you have already imported the necessary functions.

4. Describe an approach for automatically solving the puzzle. It does not have to be optimal (that is, it does not have to always solve the puzzle in the minimum possible number of moves). You can answer in a few sentences or with pseudocode.

   Validate your algorithm on the following example:



   Does your algorithm need fewer than 20 moves? Fewer than 16?

When finished, have the instructor or SLI review your work.

1.2 In-lab Session (10%)

Each student will begin to *individually implement* and submit their own solution to the problem as a Python program named `ball_puzzle.py`.

- Read the rest of this document.
- Create a new folder and PyCharm project for this assignment.
- Download the provided code, and install the necessary code from lecture.
- Begin writing the `main` function as described in the requirements.
- Begin writing the function to move the balls.
- Develop the progam enough so that it will at least display the three empty columns and wait for the user to close the display.

Show your SLI or instructor your progress when you leave the lab. Zip your Python files into a file named `lab08.zip` and upload that to the Inlab dropbox by the deadline.

2 Implementation (75%)

2.1 Download Provided Code

You are provided with a file, `ball_puzzle_animate.py`, which will generate Python Turtle graphics animation as your program solves the puzzle. The provided code can be downloaded from `http://www.cs.rit.edu/~csci141/Labs/08/ball_puzzle_animate.py`

(You should place the file in the same folder as your `ball_puzzle.py` file).

You need to exactly follow the requirement specifications described for interfacing with the animation file's functions.

2.2 Requirements

- Your program should have a `main()` function which does the following:
  - Prompt for the input string representing the initial balls in the 'red' can. The string should contain only the capital letters: 'R', 'G', 'B'. You do not have to do any error checking if input does not satisfy this requirement. (It's ok if your program crashes on faulty input. Your program will not be tested using faulty input.)
  - Call the `animate_init()` function to set up the animation.
    (Note: the `animate_init()` function currently only accepts strings with no more than 50 characters.)
  - Call your function to solve the puzzle. Your algorithm does not have to be optimal.
  - Your solving function should return the number of moves required for your algorithm to solve the puzzle.

  - After every move you make, call the `animate_move()` function to generate the animation corresponding to that move.
  - Output the number of moves required to solve the puzzle.
  - Prompt the user to "Close the window to quit".
  - Call the `animate_finish()` function which closes the animation.

- You must use `dataclasses` and the lecture's node and stack code to create your stack data structure. Download these files from the course web page link to lecture material.

- In order to get the animation to display correctly, you must do the following:

  - Use only string values 'R', 'G', or 'B' for the data slot for `Node` objects you create to represent the balls in the puzzle.
  - The function `animate_init()` takes one input argument: the input string representing the initial balls to be placed in the 'red' can.
  - The function `animate_finish()` takes no arguments and closes the turtle window.
  - The function `animate_move(stack_list, from_can, to_can)` takes three arguments.

    * `stack_list` is a list of stack objects. It is assumed that the list has 3 elements, corresponding to the 3 stacks used in the puzzle. The 'red' can is first, the 'green' can is next, and the 'blue' can is last.
    * `from_can` is an integer having value 0, 1, or 2, corresponding to the index of the stack in `stack_list` from which a ball was moved.
    * `to_can` is an integer having value 0, 1, or 2, corresponding to the index of the stack in `stack_list` to which a ball was moved.

  - You must call `animate_move()` ***IMMEDIATELY AFTER*** you actually make the move and update your stacks. The animation function assumes that the move that it is going to animate has just been completed.

  - You must begin with all of the balls on the first stack (index 0) and end with each ball in the stack of the corresponding color label.

- You must follow the true nature of stacks and not write any functions that traverse the stack "to see what's in it".

Below is an example session run which shows the output of moves.

```
$ python3 ball_puzzle.py
Input initial string (R,G,B) of balls in first can: RGBBBGGRR
Puzzle solved in 18 moves!
```

3 Suggestions

- Use a debugger, or print the state of your stacks after every move you make, to see that they are being modified correctly.
- First make sure your program works correctly independent of animation. If you're having trouble interfacing with the animation functions, you can comment out the calls to the animation functions as you develop.
  Do not forget to uncomment the animations calls, however; for full credit, your final submission must work with animation.

4 Implementation Grading

- 20%: The program uses `dataclasses` from `cs_stack.py` and `node_types.py`.
- 30%: The program automatically solves the puzzle.
- 15%: The program interfaces correctly with the provided animation functions.
- 10%: The program correctly prompts for input and displays the number of moves needed to solve the puzzle and pauses before closing the window.
- 10%: *Off the top deduction* if the code fails to follow the style guidelines for layout, author identification and docstrings.

5 Submission

Zip your `ball_puzzle.py` file into `lab08.zip`, and submit it to the **myCourses dropbox** by the due date for this assignment.