ICS 46 Fall 2022 Assignment 1: Combinatorial Puzzles

Due Monday, October 10 at 7:30 AM. This project is eligible for late submissions.

Introduction

In the first three lectures, we saw that recursion can make some seemingly laborious problem solving into a straight-forward exercise. In project 0, we saw that we can check if a solution is valid to summation puzzles. In this assignment, we're going to solve summation puzzles.

In a summation puzzle, you are given three strings of the form POT + PAN = BIB. Typically each is a word, often with a theme to the three chosen. Your goal is to assign a *distinct* digit to each letter in the equation in order to make the resulting true. For example, if the puzzle is POT + PAN = BIB, the mapping P:2, O:3, T:1, A:7, N:4, B:5, I:0 will solve this, as 231 + 274 = 505.

Reviewing related material

If you are using the Goodrich/Tamassia textbook, in the second edition, section 3.5 deals with recursion. This book is good at getting to the point, so this should not be a long read. Furthermore, you should look at your notes from the lecture when we discussed the *n* Queens problem and solved it via recursion.

Requirements

You are required to implement the function `puzzleSolver` in `proj1.cpp`. This function should return true if, and only if, the puzzle is solvable: that is, if there is a mapping of the letters that appear in the three strings to *distinct* digits such that the sum of the first two is the third. No string will have a value larger than 4,294,967,295 in its correct substitution, nor will the addition have any integer-overflow to check for. If you do not know what integer overflow is, you do not need to check during this assignment (although it's worth knowing in general).

For this project, you have a few requirements:

- You must implement the function `puzzleSolver` in `proj1.cpp`**.** You may assume it is called with three valid non-empty strings as parameters and with an otherwise empty map. The strings will always consist only of all-capital letters.
- The puzzle solution *may* have a leading zero for a string.
    - For example, in the provided test cases, we see that "UCI + ALEX = MIKE" has a solution. This corresponds to 572 + 8631 = 9203. The puzzle "KUCI + ALEX = MIKE" also has a solution with the same mapping.
- Your solution must explicitly use recursion in a *meaningful* way towards solving the problem. You may **not** solve this by using a function like `std::next_permutation` (from `<algorithm>`) to enumerate possibilities.
    - The function `puzzleSolver` itself need not be recursive if you would prefer to have a helper function that is.

- The function must return a boolean indicating whether or not the puzzle *has* a solution.
    - If the puzzle *does not* have a solution, your function should return false.
    - If the puzzle *does* have a solution your function should return true **and** have the `unordered_map<char, unsigned>` parameter contain said solution. That is, the four parameters to the `puzzleSolver` function need to be such that a correct solution to project 0 would return `true` with those parameters.
    - If there are multiple solutions, returning any of them is fine. You can think of my grading code as this:
        - I know if the test case has or hasn't a solution. I check that you return the right bool value.
        - If it has a solution, I also run a (correct) solution to proj0's related function on the three strings + the map's status at the end of your function.
    - If the previous point means you need to modify the given gtest code, feel free to do so -- my real grading code will run a verifier instead of checking for an explicitly expected mapping.
- For writing test cases, you may make use of the `gradeYesAnswer` function (whose code is provided as pre-compiled). You ***may not*** use this function when writing the app portions of your code.
- Your program must run in under three minutes on a reasonably modern computer. Test cases that take longer than this to run may be deemed to be incorrect. Note that this means you will need to think a little about efficiency in your program. You aren't expected to be an expert on efficiency at this point in your career. Many students in previous quarters were able to get theirs to run in under 90 seconds, even for the "difficult" (large) test cases.

You *may* use standard libraries as appropriate, unless there is one that makes solving this problem trivial. I am unaware of any such part of the library.

You are **explicitly permitted** to use std::unordered_set, std::list, std::queue, and std::stack if you so choose. You are pretty much required to use std::unordered_map. You are welcome to ask anything you want about these libraries, or to look up material about them online. Information about how to use an explicitly-permitted library may always be shared among classmates, but refrain from telling one another how you solved a problem in the assignment with them. For example, answering "how do I check if an element is in a std::unordered_set?" is great and encouraged, while answering "what did you use std::unordered_set for in your project?" is not.

A good reference for the STL container classes (such as those listed above, including std::unordered_map) http://www.cplusplus.com/reference/unordered_map/unordered_map/

If you would like to reuse some or all of *your code* (not that of someone else) from project 0 for part of this project, you are welcome to do so.