I&C SCI 46 Winter 2023 Project 1: Combinatorial Puzzles

Due Friday, January 27 at 7:30 AM. This project is eligible for late submissions.

Note that the due date is also the date of your first exam.

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 result 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

For this project, you are only required to implement one function found in puzzleSolver.cpp:

bool puzzleSolver(const std::string& s1, const std::string& s2, const std::string& s3, std::unordered_map<char, unsigned> & mapping)

- 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 (s1 + s2 = s3).
- 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).
- 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.

• 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 45 seconds 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 under this bound, even for the “difficult” (large) test cases.

• There are three test cases provided in `puzzletests.cpp`. These are not comprehensive tests -- they are meant to get you started. All three are required: when we grade your assignment, the first thing we will do is run these three test cases. If these do not all pass (within the required time bound), your score will be a zero on the assignment, **even if other test cases pass successfully**.

**Restrictions**

• You **may** use standard libraries as appropriate, unless there is one that makes solving this problem trivial.

• You are **explicitly permitted** to use `std::unordered_set`, `std::list`, `std::queue`, and `std::stack` if you so choose. If you want to have a stack, you do **not** need to use your stack from project zero. 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/

- You are explicitly permitted to use some or all of your code (not that of someone else)
  from project 0 for part of this project.