ICS 46 Fall 2021 Assignment 1: Combinatorial Puzzles

Due Wednesday, October 13, 9:59 AM.

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.

Getting Started

Before you begin work on this project, there are a couple of chores you'll need to complete on your ICS 46 VM to get it set up to proceed.

Refreshing your ICS 46 VM environment

Even if you previously downloaded your ICS 46 VM, you will probably need to refresh its environment before proceeding with this project. Log into your VM and issue the command **ics46 version** to see what version of the ICS 46 environment you currently have stored on your VM. Note, in particular, the timestamp; if you see a version with a timestamp older than the one listed below, you'll want to refresh your environment by running the command **ics46 refresh** to download the latest one before you proceed with this project.

If you're unable to get outgoing network access to work on the ICS 46 VM — something that afflicts a handful of students each quarter — then the **ics46 refresh** command won't work, but an alternative approach is to download the latest environment, then to upload the file to your ICS 46 VM using SCP. (See the Project #0 write-up for more details on using SCP.) Once the file is on your VM, you can run the command **ics46 refresh_local NAME_OF_ENVIRONMENT_FILE**, replacing **NAME_OF_ENVIRONMENT_FILE** with the name of the file you uploaded; note that you'd need to be in the same directory where the file is when you run the command.

If you need a link to the environment file directly, it is available at https://www.ics.uci.edu/~mikes/ics46/vm/ics46-shindler-2021fall-environment.tar.gz

Creating your project directory on your ICS 46 VM

A project template has been created specifically for this project, containing a similar structure to the basic template you saw in Project #0.

Decide on a name for your project directory, then issue the command **ics46 start YOUR_CHOSEN_PROJECT_NAME project1** to create your new project directory using the project1 template. (For example, if you wanted to call your project directory proj1, you would issue the command **ics46 start proj1 project1** to create it.) Now you're ready to proceed!

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. A video on this topic is also available on Canvas.

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 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.

Deliverables

After using the gather script in your project directory to gather up your C++ source and header files into a single **project1.tar.gz** file (as you did in Project #0), submit that file (and only that file) to Checkmate. Refer back to Project #0 if you need instructions on how to do that. This time, it should give you the correct file name, insert innocent-looking face emoji here.

You will submit your project via Checkmate. Keep in mind that you're responsible for submitting the version of the project that you want graded. We won't regrade a project simply because you submitted the wrong version accidentally. (It's not a bad idea to look at the contents of your tarball before submitting it; see Project #0 for instructions on how to do that.)

Can I submit after the deadline?

Yes, it is possible, subject to the late work policy for this course, which is described in the section titled Late work in the course reference and syllabus.

Grading

Your “raw score” is based solely on correctness : we will run some number of test cases for your code and, based on how many and which ones you get correct, you will earn some number of points between 0 and 6 (inclusive, and might not be integer-valued). Each is worth some number of points and is graded based on whether or not your code correctly determines if the puzzle has a solution, and if so, what it is. If it is determined that your program does not make an attempt to solve the problem at hand, you will not get these points, regardless of the result from testing. The tests will look a lot like the tests in your Google Test starting directory for this assignment; if you pass those, you’re off to a good start, but it’s not a guarantee.

You will get an email with both your grade and a description of the test cases you missed, if any. We will not provide the test cases missed to you. If you have any questions about your grade, including a request for reconsideration, please *respond to that email* to ask. See the syllabus for the grade reconsideration policy.

If we review your code and find your style to be sufficiently bad, we reserve the right to deduct points based on this, proportional to how bad the style is. If we do so, we will alert you to both the penalty and the reason. Here are some guidelines to follow when submitting your code:

- Include a reasonable level of comments. Do not comment every line, but do not omit comments either. A decent guideline is that if you were asked to explain your code six months from now, your comments should guide you to be able to do so.

- Use meaningful variable names where appropriate. Loop counters need not have a long name, but if you declare a std::unordered_set, give it a name reflecting what is in the set, not “mySet.”

- Use proper scoping for variables. Avoid global variables except in rare circumstances. Pass parameters appropriately.

- Indent appropriately. While C++ does not have Python's indentation requirements for writing usable code, the guidelines of *readable* code are an issue here.

- Avoid vulgarity in your code and comments. Variable names, output statements, and the like should not make reference to topics that are not discussed in polite company.

- Remove debug output as appropriate; rather than commenting it out, delete it if it is unnecessary. If you think you might want to return to those debug statements, enclose them instead. Write something like this as a global variable (this is a case where such is acceptable):

  ```
  const bool DEBUG_OUT = true;
  ```

  Then, when you need a debug statement, enclose it:

  ```
  if( DEBUG_OUT )
  {
        std::cerr << "Set has been successfully declared!" << std::endl;
  }
  ```

  When you want to remove the debug output, you can change the declaration to set the variable to false. You may wish to have multiple variables to control debug in various functions, either globally or having them local to functions. In the past, there have been several instances of students who have otherwise correct code but retained debug output statements -- and failed cases because they went over the time limit. An output statement takes more time than many instructions; I encourage you to set your program to run without output when it is submitted. We will not consider regrade requests that ask us to remove output then run the timed-out cases again.

  Similarly, if you keep your debug output lines in the file you turn in, having meaningful output statements is better than "code is here!" or "aaaaa." This is also true if you are going to ask someone for help with debugging.