Project 2 Morally Due April 23 at 12:30PM (Dead Cat April 25 12:30PM)

This project is worth 10% of your grade.

For this project, you will implement the Recursive 7-ALG SAT solving algorithm. As in the first project, there will be three parts.

1 2-SAT

For the first part of this project, your goal will be to implement an algorithm for solving 2-SAT. Specifically, you will be asked to determine whether a given boolean formula of the form

$$C_1 \wedge \cdots \wedge C_m,$$

where each C_i has at most two literals, is satisfiable and to give a satisfying assignment if it is.

The algorithm that you implement should run in polynomial time. Particularly, do not cheat and use some brute force method to determine the appropriate solution. Your code may be inspected to verify that you are implementing this part of the project correctly. *Any submission found to be using a forbidden algorithm for this part of the project will receive ZERO CREDIT for the entire project.* You are allowed to look online for an algorithm on how to solve this problem.

1.1 Input

You will receive one line from standard input as input for this part of the project. This line will encode a boolean formula in conjunctive normal form (CNF). The encoded formula consists of a sequence of one or more encoded clauses each separated by a single semicolon (`;`). Each encoded clause consists of a sequence of one or more encoded literals each separated by a single comma (`,`). Each encoded literal consists of one or more lowercase letters and numbers representing a variable name, optionally preceded by a dash (`-`) representing the negation of the variable.

To illustrate, the input

```
-a1,a2;a2,-b;b
```

encodes the boolean formula

$$(\neg a_1 \vee a_2) \wedge (a_2 \vee \neg b) \wedge (b).$$

Of course, for this part of the project, you may assume that the boolean formula represented by the input contains at most two literals per clause.

1.2 Output

You will output either two or three lines to standard output, depending on the input. For the first line, you will either output `yes` or `no`, where `yes` means that the given formula is satisfiable, and `no` means otherwise.

If the formula is satisfiable, you will also output another line representing a satisying assignment of variables for the formula. You can assume that all relevant variables show up in the formula at least once. This line will contain a sequence of comma-separated substrings of the form *var_name*=*value*, where for some variable v, *var_name* denotes the variable name of v and *value* denotes the boolean value it receives from the assignment (either `T` or `F`). There should be one such substring per variable that appears in the formula. The order in which these substrings are given does not matter. Keep in mind that there may be multiple valid assignments that can be given.

So for the formula

$$(\neg a_1 \vee a_2) \wedge (a_2 \vee \neg b) \wedge (b)$$

given above, one possible assignment would be $(a_1, a_2, b) = (F, T, T)$. A possible output corresponding to this assignment would be formatted as

`b=T,a1=F,a2=T`.

Regardless of whether the formula is satisfiable or not, on a new line, you should output a single dollar sign (`$`). This is required to separate output from this part of the project from output from the next part of the project.

2 STAND

In the second part of this project, your goal will be to implement the *STAND* algorithm from the slides. To recap, the STAND algorithm takes as input a boolean formula and a partial assignment of variables and searches for easy ways to extend the partial assignment, thus simplifying the problem at hand. Refer to the slides for details on how the algorithm works. Keep in mind that you may need to perform multiple reductions / simplifications of the given formula before producing the final output.

2.1 Input

You will receive two lines from standard input as input for this part of the project.

The first line of input consists of an encoded boolean formula that will be used by STAND. This encoded formula will be encoded the same way as the encoded formula given in part one of the project. This time, however, the boolean formula may have up to three (instead of two) literals per clause.

The second line of input consists of an encoded partial assignment that will be used by STAND. This line will be encoded using the same encoding scheme specified in the output section of part one. Keep in mind that this represents a partial assignment, so it will not necessarily determine the meaning of every variable that appears in the given formula. It is possible that no variables will be assigned a value, in which case only an empty line will be provided.

2.2 Output

You will output either two or three lines to standard output, depending on the input.

For the first line, you will either output `yes`, `no`, or `maybe`, where `yes` means that the given formula is satisfiable under the partial assignment, `no` means that it is not, and `maybe` means that not enough information could be determined from STAND to determine satisfiability.

If you outputted `yes` or `maybe` previously, then you will output a new line representing an extended partial assignment that is produced by STAND on the given input. If your previous output was specifically `yes`, then this extended partial assignment will be a total assignment. Use the same formatting conventions specified in part one to encode your output.

Lastly, on a new line, output a single dollar sign to separate the output from this part of the project from output from the last part.

3 Recursive 7-ALG

Your goal for the final part of this project will be to implement the *Recursive 7-ALG* (R7ALG) algorithm as described in the lecture slides. However, in addition to outputting whether or not a given formula is satisfiable, you must also output a satisfying assignment if it is.

Similar to the first part of this project, do not cheat and use some algorithm other than R7ALG to determine the appropriate solution. Your code may be inspected to verify that you are implementing this part of the project correctly. *Any submission found to be using some other algorithm for this part of the project will receive ZERO CREDIT for the entire project.*

3.1 Input

You will receive one line from standard input as input for this part of the project. The line of input consists of an encoded boolean formula that will be used by R7ALG. This encoded formula will be encoded the same way as the encoded formula given in part one of the project. Similarly to the second part of the project, the boolean formula may have up to three literals per clause.

Notice that no partial assignment is given as part of the input for this part of the project. Your first call to R7ALG as described in the slides will begin by passing the given input formula along with an empty partial assignment.

3.2 Output

You will output either one or two lines to standard output, depending on the input. Similar to part one, for the first line, you will either output `yes` or `no`, where `yes` means that the given formula is satisfiable, and `no` means otherwise. If the formula is satisfiable, you will also output another line representing a satisfying assignment of variables for the formula. This line should be formatted using the same rules used in the previous two parts.

For this part of the project, *do not* output any dollar sign as was required in the previous two parts.

4 Miscellaneous

Your code will be graded autonomously by a program that runs your code against several test cases. The output your program produces will be the only means to determine whether or not your program works correctly. With that said, it is *very important* that your output is formatted correctly as specified in the previous sections.

As a reminder, for this project, input will be provided to you via standard input, as opposed to via command-line arguments as was done in the previous project. In total, 4 lines will be provided as input. Your program should output somewhere between 5 to 8 lines (inclusive), depending on the input received.

Examples of how your program should work will be posted on Piazza. Keep in mind that unlike in the previous project, for this project, a given input may have multiple correct outputs. As a result, the output that your program produces on an example input may be different than the output provided in the example. Do not worry if this happens; you will receive full credit in such a case so long as your output matches the given input and is correct. You will, however, still need to manually verify in some cases that the output your code produces is indeed correct.

We highly recommend that you test your code on multiple inputs that *you* generate and that you are carefully verifying that the output your program generates is correct and that the output is properly formatted. Furthermore, this project will probably take you longer to complete than the first project did, so do not start working on this project at the last minute. If something specified in this write-up is unclear, you should ask about the issue on Piazza. We will be much less sympathetic toward excuses for why your project didn't work correctly than we were in the previous project. *You have been warned.*

For the sake of your convenience, you may assume that the testing program treats your output in a case-insensitive manner, and that it ignores any whitespace other than newline characters.

5 Writing and uploading your code

This project is (somewhat) language-agnostic, meaning that we do not require you to write your code in any one specific language. With that said, since we want to be able to run your code directly from the command line, we want your code to be written in a *scripting language*. (We don't want to be burdened with trying to manually compile code from several different languages.) In order for us to determine how to run your code, you will place a shebang on the *very first* line of your code indicating what interpreter to use. Your shebang should take the form

`#!/usr/bin/env` *interpreter*

where *interpreter* is the name of the interpreter you plan to use. For this

project, you are allowed to use any of the following interpreters: `python2`, `python3`, and `ruby`. If there is an interpreter not listed that you would like to use instead, ask about it on Piazza. On the next line of code (below the shebang), put a comment containing your full name and your UID.

Once you have completed the project, name your code file `proj02_`*lastname*. That is, name it "proj02" underscore {your last name in lowercase}. Submit your code to project `proj02` on the Submit Server.