CS29003 ALGORITHMS LABORATORY
ASSIGNMENT 10
Date: 3rd Oct, 2019

Important Instructions

1. **List of Input Files:** input.txt
2. **Output Files to be submitted:** ROLLNO_A10_P1.c/.cpp, ROLLNO_A10_P2.c/.cpp
3. Files must be read using file handling APIs of C/C++. Reading through input redirection isn't allowed.
4. You are to **stick to the file input output formats strictly** as per the instructions.
5. Submission through **.zip files are not allowed**.
6. Write your name and roll number at the beginning of your program.
7. Do not use any global variable unless you are explicitly instructed so.
8. Use proper indentation in your code.
9. Please follow all the guidelines. Failing to do so will cause you to lose marks.
10. **There will be part marking.**

Satisfiability of Frog Formula

Frog Formulas

Frog formulas are a particular form of Boolean logic, and often used in AI systems for logical reasoning. A Boolean variable can only take one of two values $\{true, false\}$. Each Boolean variable represents an event (or possibility). A literal is either a variable x or its negation $\sim x$. In Frog formula, the constraints among the variables are represented by two types of clauses.

- ▷ **Implication clauses**: whose left-hand-side is an **AND** of any number of positive literals, and whose right-hand-side is a single positive literal. For instance, in a murder mystery, suppose the variables z, y and x denote 'the colonel was asleep at 8 pm', 'the murder took place at 8 pm', and 'the murder took place in the kitchen', respectively. Consider the following implication clause:

  $$(z \wedge y) \implies x$$

  It asserts that "if the colonel was asleep at 8 pm, and the murder took place at 8 pm, then the murder took place in the kitchen." A degenerate statement of the type $(\implies x)$ means that x is unconditionally true. For instance, "the murder definitely took place in the kitchen."

- ▷ **Negative clauses**: such a clause consists of an **OR** of any number of negative literals, as in

  $$(\sim u \vee \sim v \vee \sim w).$$

where suppose that u, v, w, resp., mean that constable, colonel, and butler is innocent. This clause asserts that "they can't all be innocent."

- ▷ A Frog formula is a set of implications and negative clauses.
- ▷ A Frog formula must have at least one positive literal/unconditionally true literal.
- ▷ The formula $x \implies y$ is equivalent to $\sim x \vee y$.

Problem Statement

Given a Frog formula, determine if it is satisfiable. That is, is there a *true/false* assignment of variables where all clauses are satisfied. Such an assignment is called a satisfying assignment.

Example 1

You may have seen puzzles like the following.

- ▷ If Mr. Smith has a dog, then Mrs. Brown has a cat.
- ▷ If Mr. Jones has a dog, then he has a cat, too.
- ▷ If Mr. Smith has a dog and Mr. Jones has a cat, then Mrs. Peacock has a dog.
- ▷ If Mrs. Brown and Mr. Jones share a pet of the same species, then Mr. Smith has a cat.
- ▷ All the men have dogs.

Now, what can we say about who has what kind of pet? We can encode this problem using Boolean variables. Let's write s, j, b, p for Boolean variables that are *true* if Mr. Smith, Mr. Jones, Mrs. Brown, or Mrs. Peacock (respectively) have a dog, and write S, J, B, P for Boolean variables that are *true* if they have a cat. (Note that any given person can have both a cat and a dog, or might have no pets whatsoever.) Then the above puzzle turns into the following logical Frog formula:

$$(s \implies B) \wedge (j \implies J) \wedge ((s \wedge J) \implies p) \wedge ((b \wedge j) \implies S) \wedge ((B \wedge J) \implies S) \wedge (\implies s) \wedge (\implies j)$$

Note, as stated earlier, the expression $(\implies x)$ indicates that x is a fact and is always *true*. Which of the Boolean variables must be *true*? Evidently, s, j, p, S, B, J must be true, but the others can be *false*. This corresponds to the conclusion that Mr. Smith and Mr. Jones must own both a cat and a dog, Mrs. Peacock must own a dog (though we don't know whether she owns a cat), and Mrs. Brown must own a cat (though we don't know whether she owns a dog).

Example 2

Consider the following Frog formula

$$(\implies x) \wedge (\implies y) \wedge ((x \wedge u) \implies z) \wedge (\sim x \vee \sim y \vee \sim z)$$

has a satisfying assignment u = *false*, x = *true*, y = *true*, z = *false* since for this assignment of the variables the entire formula is *true*.

Example 3

The following Frog formula

$$(\implies x) \wedge (\implies y) \wedge ((x \wedge y) \implies z) \wedge (\sim x \vee \sim y \vee \sim z)$$

is not satisfiable.

Part I

You have to develop a greedy algorithm to identify if a given Frog formula is satisfiable. Your algorithm GREEDYFROG(ϕ) should run in $O(n^2)$ time where n is the length of ϕ. The length of ϕ refers to the total number of literals present in the formula. *Hint*: Observe if a certain subset of variables is set to *true*, then they must be *true* in any satisfying assignment. Think greedily as to what actions can you take (variables you can set/reset to *true/false*) to test satisfiability.

```
//Example code for identifying if a Frog formula is satisfiable
GreedyFrog(φ)
{
...
...
}
```

Part II

It turns out that if you are slightly clever in how you implement the above algorithm, you can make it run in linear time using some smart data structures.
Hint: Assume that ϕ has been already parsed into a conjunction (i.e., $\wedge$) of clauses, and each literal v will have *a link to all the clauses where it appears on the left-hand side of an implication*. You can store the left-hand side of each such clause as a (doubly) linked list of variables. Also, consider a queue W which would hold the work list, i.e., W is a set of literals that appears on the right-hand side of an implication whose left-hand side has become empty (empty implication). Under this revised settings you can use the same concept that you developed in Part I for testing satisfiability. Each variable enters the work list at most once. The complexity should be $O(n)$, where n denotes the length of ϕ.

Note that the above is just a hint and you are absolutely free to choose some other strategy to bring down the complexity from $O(n^2)$ to $O(n)$.

```
FastGreedyFrog(φ)
{

}
```

Sample Input File

FILE: *input.txt*

```
4
IMP x
IMP y
x AND u IMP z
NEG x OR NEG y OR NEG z
```

Each input file contains one Frog Formula in the following format. The first line contains the number of clauses (4 in this case). Each of the next lines contain one clause each. Thus, there are 4 lines containing a clause each. You have to parse the formula and evaluate for different assignments of the variables. The variable names can be only a single character from the lower case English alphabet (i.e.,

[a-z]). Further, we do not care about the data structure that you use to store your parsed output; you are completely free to use any data structure. Read IMP: $\implies$ (unary or binary), NEG: $\sim$ (unary always), AND: $\wedge$ (binary always), OR: $\vee$ (binary always).

Sample Output File

FILE: *output.txt*

```
u = false, x = true, y = true, z = false
satisfiable
```

If the input is satisfiable, the output prints the assignment of the variables for which the formula is satisfiable and the next line prints "satisfiable". If the input is not satisfiable, the output just prints "not satisfiable".

Make sure to read the input ϕ from the file *"input.txt"*. Create a sample input file like the one shown above to test your code. At the time of evaluation, the input data might be different from the one given here.

You need not submit *"output.txt"*, but your code should write the output in the given format in a file named *"output.txt"*.

Additional hints – General

You can follow the pipeline stated below to complete the assignment. However this is not mandatory, and you can choose your own pipeline.

1. Read an individual clause from the *input.txt* as a `char*`.
2. Parse individual clauses to get the literals and the logical connectives. However you can use your own convenient technique of parsing. Also as indicated earlier, we do not care of the data structure that you use to store your parsed output; you are completely free to use any data structure.
3. You can write mini functions like `unaryimplies ()`, `binaryimplies ()`, `and ()`, `or ()` and `negation ()` that returns the desired outputs of these functions. In-built functions like `atoi ()` and `itoa ()` can be used as and when necessary.
4. Use the the above functions as and when necessary to develop the GREEDYFROG(ϕ) algorithm. Note that your task is to develop the algorithm yourself; we are NOT going to tell you how to get the solution.
5. Once you have developed GREEDYFROG(ϕ), try to think how you can optimise it using certain data structures. We have given some sketches but you are fully free to use your own logic. Only point is that the complexity should go down from $O(n^2)$ to $O(n)$.