CSCI 561: Foundations of Artificial Intelligence
Instructor: Prof. Laurent Itti
Homework #3: Logic and SAT Problem
Due on April 9 at 11:59pm, Los Angeles time, 2014

Problem Description

Suppose you have a wedding to plan, and want to arrange the wedding seating for a certain number of guests in a hall. The hall has a certain number tables for seating. Some pairs of guests are couples or close Friends (F), and want to sit together at the same table. Some other pairs of guests are Enemies (E), and must be separated into different tables. The rest pairs are Indifferent (I) with each other, and do not mind sitting together or not. However, each pair of guests can ***only*** have one relationship, (F), (E) or (I). You need to find a seating arrangement that satisfies all the constraints.

Suppose there are <*M*> guests in total, and there are <*N*> tables in the hall. The number of pairs of Friends is <*F*>, and the number of pairs of Enemies is <*E*>. Given relationships of the wedding guests, here we use a matrix *R* with elements R_{ij} = 1, -1 or 0 to represent whether guest *i* and *j* are Friends (F), Enemies (E) or Indifferent (I). The table arrangement task can be represented as First-order Logic (FOL) and further encoded as a Boolean Satisfaction problem (SAT). We introduce Boolean variables X_{mn} to represent whether each guest *m* will be seated at a specific table *n*. You are asked to write FOL to represent the constraints using variables X_{mn}. Then you need to construct clauses and transform the FOL into CNF sentence. To decompose the arrangement task, there are three constraints you have to satisfy:
(a) Each guest *i* should be seated at one and ***only*** one table.
(b) Any two guests *i* and *j* who are Friends (F) should be seated at the same table.
(c) Any two guests *i* and *j* who are Enemies (E) should be seated at different tables.
Note that, for simplicity, you do **NOT** need to consider the capacity constraint of a table. This means the size of each table is assumed to be large enough to seat all the guests.

Before you begin, it may help to think about this question. How many CNF clauses in total are there to encode a wedding seating arrangement in terms of <*M*>, <*N*>, <*E*> and <*F*>?

The input will be formatted as follows:

N M
<Relationship matrix>

For example,

```
2 3
0 1 -1
1 0 0
-1 0 0
```

This input tells us that there are 2 tables (N) and 3 people (M). Also, we can see that the first person has 1 friend and 1 enemy. The 2nd and 3rd person are indifferent to each other. All relationships are mutual, so the matrix is symmetrical. You may assume that the input will always be valid, with each number separated by a single space character. The input file name will be provided as a command-line argument to your program (see Grading).

You are asked to implement a SAT solver to find a satisfying assignment for any given input. You need to implement a modified version of the **PL-Resolution** algorithm (AIMA Figure 7.12). Modifications are necessary because we are using the algorithm for a slightly different purpose than is explained in AIMA. Here, we are not looking to prove entailment of a particular query. Rather, we hope to prove satisfiability. Thus, there is no need to add negated query clauses to the input clauses. In other words, the only input to the algorithm is the set of clauses that comprise a randomly generated sentence. As an additional consequence of our purpose, the outputs will be reversed compared to the outputs listed in AIMA's pseudo code. That is to say, if the empty clause is derived at any point from the clauses of the input sentence, then the sentence is unsatisfiable. In this case, the function should return ***false*** and not ***true*** as the book specifies for this situation. In the opposite situation where the empty clause is never derived, the algorithm should return ***true***, indicating that the sentence is satisfiable.

PL-Resolution is a sound and complete algorithm that it can be used to determine satisfiability and unsatisfiability with certainty. Your program's output will be very simple: 0 if the input is unsatisfiable, or 1 if it is satisfiable. The output file path will be given as a parameter (see grading).

Grading:

The grader will compile your program using the following command, for examples:

C++: g++ *.cpp -o hw3

Java: javac *.java

And execute your program using, for examples for each experiment:

C++: ./hw3 input.txt output.txt

Java: java hw3 input.txt output.txt

Deliverables:

You are required to hand in all code that you wrote to complete this assignment; implementation language not important. However, if you code in C++ or Java, the TA will be better able to assist you ☺. Also, we require that your code be able to run from the command line on the USC aludra.usc.edu Linux servers. For information on accessing aludra, please see https://itservices.usc.edu/web/hosting/students/. Please include your source code, any additional files needed to compile it (makefiles, etc.), and a readme.txt to describe how to run your code, including the command(s) to run it.