**Design and Analysis of Algorithms**

**Fall 2015**

***Programming Assignment* 2**

Due: Upload program 2 to Blackboard by 10/13/2015, 11:59PM.

**The CNF-SAT problem**

Before defining the problem we need to define logical expressions in conjunctive normal form. Note the similarity to Boolean algebra.

Conjunctive normal form (CNF) of a logical expression

A *logical (Boolean) variable* is a variable that may be assigned the value *true* or *false* (for example p is true, while q is false). A *literal* is a logical variable or the negation of a logical variable (For example p and ¬q are literals). A *clause* is a disjunction of literals (for example (p ∨ q ∨ s) and (¬q ∨ r) are clauses). A logical expression is in Conjunctive Normal Form if it is a conjunction of *clauses. F*orexample the expression:

(p ∨q ∨ s) ∧(¬q ∨ r)∧ (¬p ∨ r) ∧ (¬r ∨ s)∧ (¬p ∨ ¬s ∨ ¬q)

The CNF-SAT problem is: Given a logical expression in Conjunctive Normal Form, is there a truth assignment for the variables which causes the CNF expression to be true?

Answer is yes, if there is an assignment that makes all the clauses evaluate to T (true). For example, p = T, q = F, r = T and s = T is a truth assignment for

(p ∨q ∨ s) ∧(¬q ∨ r)∧ (¬p ∨ r) ∧ (¬r ∨ s)∧ (¬p ∨ ¬s ∨ ¬q)

Note that if q=F then ¬q=T

The answer is no, if there is no such an assignment. This means that for each assignment some clauses will evaluate to false.

The CNF-SAT problem is an important problem in logic. It is also the first problem shown to be NP-Complete by Cook

**The program**

This program will read from the command line the following two parameters: <input file name> <output file name>.

**The input file** has the following structure:

Line 1: n m sizeC// where n is the number of variables, and m the number of clauses and sizeC the number of literals in the clauses

Line 2 to m+1: a list of sizeC positive and negative numbers in the range 1 to n. If the number is negative the literal is negated.

**The output file:**

Case 1: There was no satisfying assignment:

Line 1: No satisfying assignment

Line 2: Run time is <x> milliseconds.

Case 2: n<=5 and there are solutions

Print all solutions

Last line: Run time is <x> milliseconds.

Case 3: n>5 and m>30 and there are solutions

Line 1: There are satisfying assignments

Line 2: Run time is <x> milliseconds.

Case 4; n>5, m<=30 and there are solutions:

Print the first solution

Last line: Run time is <x> milliseconds

**The Code**

Use a depth first search algorithm for solving the problem. After you assign values to all n variables check all the clauses to see if the assignment solves the problem. If it does check whether you have case 1, 2 or 3 and decide whether to print the solution. Note that the call tree for the algorithm is the same as the state space tree.

Pseudo code for this problem

dfsCompute(depth) {

**if** (depth==n){

if (all clauses are satisfied with current assignment of variables)

write to disk the appropriate output

return;

}//end if depth ==n

x[depth]=1;

dfsCompute(depth+1);

x[depth]=0;

dfsCompute(depth +1);

}//end dfsCompute

Initial call with depth = 0.

Example 1

(¬x1 ∨x2) ∧(¬x2 ∨ ¬x3)∧ (¬x1 ∨ x3)

The input file:

Line 1: 3 3 2

Line 2: -1 2

Line 3: -2 -3

Line 4: -1 3

The output file:

The solution is

x[1]= 0

x[2]= 1

x[3]= 0

The solution is

x[1]= 0

x[2]= 0

x[3]= 1

The solution is

x[1]= 0

x[2]= 0

x[3]= 0

Run time is 1 milliseconds

Example 2

(¬x1 ∨¬x2) ∧(¬x1 ∨ x2)∧ (x1 ∨ ¬x2) ∧ (x1 ∨ x2)

The input is:

2 4 2

-1 -2

-1 2

1 -2

1. 2

The output is:

No satisfying assignment

Run time is <x> milliseconds.

Please run your code with the provided input files:

**Submission Instructions**

You may write the code using C, C++, or Java. **Your program should compile on Bingsuns. No exceptions.** It should be purely a command line program. NO GUI will be accepted.

Please drop a *.tar.gz* file to blackboard.

The zip file should be named as follows:

<userID>\_P2

When the file is unzipped it should contain a directory with the same name as the zip file.

The directory should contain the following files:

1. The output files generated for the provided input files.
2. The output file largeOutput.txt generated for the file largeInput.txt.
3. File(s) with the source code for the program and possibly a makefile.
4. A read-me file named *readme.txt* which should contain:

Line 1: C, or C++, or Java

Line 2: Either a comma-delimited list of files, which will be compiled and linked with the appropriate compiler (gcc, g++, or javac) and then executed (make sure that for Java the first file has the main method) or the single word “make” which will execute the makefile in the unzipped directory.  The makefile will need to produce an executable called “submission” or a “Submission” class in the java case, which will be interpreted/executed.

Line 3: Your name and email address

**Plagiarism Policy**

All your code will be checked for similarity to other submissions using Moss. Programmers have an uncanny knack of reproducing the same code that they have seen before. So you are advised not to look at each other's code.

Please recall that, upon detection of plagiarism, 0 will be given to all theory and programming assignments to involved students with no exceptions.