**Implementation of a SAT Solver In Java using Backtracking**

Develop a solver for the Satisfiability problem using backtracking.

**Preliminaries**

Let’s begin with some definitions. (Don’t get psyched by the formality; you’re familiar with most of this already!) This project deals with Boolean formulae. A Boolean formula is comprised of Boolean variables, the negation operator (Boolean **not**), the disjunction operator (Boolean **or**), and the conjunction operator (Boolean **and**). We use **x1**, **x2,** etc., to denote Boolean variables.

**Definition of *literal***

A literal is a Boolean variable or the negation of a Boolean variable.

**Definition of *clause***

A clause is a disjunction (or) of literals, or a single literal.

**Definition of *formula* in conjunctive normal form**

A formula is a conjunction (and) of clauses, or a single clause.

An example formula is: (**x1** or **x2**) and (not-**x2** or **x3** or **x4**) and (not-**x1** or not-**x3** or not-**x4**).

This formula is comprised of Boolean variables **x1**, **x2**, **x3** and **x4**, and has three clauses.

**Definition of *assignment***

An assignment for a formula is a setting of truth values (either **true** or **false**) for the variables in the formula. An example assignment for the example formula given above is: **x1**, **x2** and **x4** are set to **true,** and **x3** is set to **false**. We will write this assignment as (**x1, x2, x3, x4**) = (**true**, **true**, **false**, **true**).

**Definition of *satisfying assignment* for a formula**

An assignment that makes a formula evaluate to true is a satisfying assignment. Observe that the assignment (**x1, x2, x3, x4**) = (**true**, **true**, **false**, **true**) makes each clause in the example formula evaluate to true, and thus makes the example formula evaluate to true and is a satisfying assignment. On the other hand, the assignment (**x1, x2, x3, x4**) = (**false**, **false**, **true**, **true**) makes the above formula evaluate to **false** since it does not satisfy the formula’s first clause, and therefore it is not a satisfying assignment.

**Definition of *partial assignment* for a formula**

A partial assignment is a setting of truth values for 0 or more variables in a formula. A partial assignment for the example formula given above is: **x1** and **x4** are set to **true,** and **x3** is set to **false**. We will write this satisfying partial assignment as (**x1, x2, x3, x4**) = (**true**, **\_**, **false**, **true**).

**Definition of *satisfiable formula***

A Boolean formula is satisfiable when it has a satisfying assignment.

**Definition of *the Satisfiability Problem (SAT)***

Given a Boolean formula, determine whether or not it is satisfiable.

**Your programming project**

You will develop a solver for the Satisfiability problem, a program that receives a Boolean formula as input and computes a (any one) satisfying assignment for it. If no satisfying assignment exists, your program will report that the input formula is unsatisfiable. If a satisfying assignment exists, your program will report that assignment.

Your program will read from the command line the name of a file, and then read the input formula from that file. All input files will have the following format:

* Optional comment lines at the beginning of the file start with the character ‘c’.
* The first data line has the format “p cnf number-of-variables number-of clauses”. Note that variables are numbered starting at 1.
* The specification of the clauses follows, one clause per line. A clause line ends with a ‘0’. A clause is specified by writing the numbers of the literals that occur in it. Note that the negated variable number **i** is specified as **–i**; thus, the clause (not-**x12** or **x3** or **x41**) is specified in this file format as –12 3 41 0.

The content of a file that specifies the example formula (**x1** or **x2**) and (not-**x2** or **x3** or **x4**) and (not-**x1** or not-**x3** or not-**x4**) follows:

**c This file illustrates the file format.**

**p cnf 4 3**

**1 2 0**

**-2 3 4 0**

**-1 -3 -4 0**

For this project, you will develop a solver that uses the Davis-Putnam (DP) algorithm. At any given point, a DP solver has:

* A partial assignment (an assignment of values to some of the variables in the input formula); and
* An unsatisfied formula (a set of clauses, that may be different from the input formula). Two things to note about this formula:
  + it contains only clauses that are unsatisfied by the current partial assignment, and
  + each of these clauses only contains unassigned literals (assigned literals that do not satisfy a clause are "removed" from the clause, either literally removed, or marked and ignored). For example, suppose that the algorithm starts out with a formula with a single clause (-1 2 3), and the completely unasssigned partial assignment of 3 variables (-, -, -). Now suppose that the algorithm assigns the value **true** to **x1**. The clause cannot be satisfied on account of its first term, which is assigned but to a value that is not helpful to the clause, so this term will be (temporarily) removed from the clause.

The DP algorithm proceeds as follows (this is a recursive process):

* Any variable that is currently unassigned is chosen to be the next "branch variable," var.
* The branch variable var is assigned an initial (arbitrary) value of **true**. This assignment of a value to var has two consequences, that preserve the properties of the formula that we discussed above:
  + any clauses in the formula that contain var are now satisfied, thus they are "removed" from the formula; and
  + any occurrences of the literal **-**var in other clauses are "removed."
* The algorithm has extended the partial assignment by assigning a value to the branch variable and it calls recursively to attempt to extend the current partial assignment even more. It may encounter the following scenarios:
  + If the formula is empty (because all its clauses have been satisfied and removed) then the current assignment is a satisfying assignment (the first base case in the DPSolver method).
  + If one of the clauses in the formula is empty (because its last literal was removed and thus no refinement of the current partial assignment could possibly satisfy this clause or the formula), the algorithm backtracks: The algorithm will change the value of var from true to false and call recursively to try to extend this new partial assignment; if this also results in an empty clause, var will be unassigned (backtracking to an earlier partial assignment) and some earlier branch variable assignment will be changed. Notice that unassigning or changing the value assigned to a variable may cause changes in the formula; this preserves the properties of the formula that we discussed above.

After your own testing convinces you that your solver is correct, run and time your solver on the required input set. Preserve and submit the runtimes.

**Submission instructions**

Follow the following schedule:

Assignment 1 – Part 1: due on or before - section 1: Jan 29; section 2: Jan 30

Assignment 1 – Part 2: due on or before - section 1: Jan 29; section 2: Jan 30

Assignment 1 – Part 3: due on or before - section 1: Feb 5; section 2: Feb 6

Assignment 1 – Part 4: due on or before - section 1: Feb 5; section 2: Feb 6

Feb 8 is the hard deadline. The rest are soft deadlines. The difference between the hard and soft deadlines is only in what you will submit. Other than that all deadlines must be adhered to. For the soft deadlines, it is enough if you submit the hard copy of the source code of the Part that is due. Please do submit anything else such input/output files, object code, or an execution of the code. Each such submission will be treated as the final submission of the part that is due, and graded.

By the hard deadline, submit a flash drive containing the source code, input and output files plus the object (or executable) code.

Sometimes I may ask you to resubmit after making some changes/improvements to your code. In those situations, submit both the original and the improved versions of the code.

**\*\*\* Submit your assignments in my office. Please do not hand it to me in class. \*\*\*\***

See the syllabus for additional details on how to submit a project. Must run your final code on at least two test files – one containing a satisfiable and the other containing an unsatisfiable formula.