1 Overview

In this lab, you will continue evaluating satisfiability for CNF propositions. The goal is to improve last lab’s algorithm using the "stop early" optimization, making your implementation run faster with fewer recursive calls.

2 The Benchmark

Provided to you in this week’s Lab05Tester is a benchmark which runs your satisfiability algorithm against a series of propositions of increasingly complex ratios (see below). Also provided is PropGenerator, which is used by the benchmark to randomly generate CNF propositions.

This benchmark reports back the algorithm’s resulting:
- cost, meaning the number of recursive calls
- average time, the time (in milliseconds) to determine the satisfiability of the propositions of a given ratio

Below is an example output of the benchmark – the before uses the algorithm from last week’s lab, and the after implements the "stop early" optimization on the following page. Bear in mind – your results won’t match this example exactly (everyone’s computer and implementation will be a little different), but you will see a significant improvement:

Before:

<table>
<thead>
<tr>
<th>Ratio</th>
<th>r=1.0</th>
<th>fraction</th>
<th>sat=1.0</th>
<th>avg. cost=1352.9</th>
<th>avg. time=11.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio</td>
<td>r=2.0</td>
<td>fraction</td>
<td>sat=1.0</td>
<td>avg. cost=30709.8</td>
<td>avg. time=86.4</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=3.0</td>
<td>fraction</td>
<td>sat=1.0</td>
<td>avg. cost=195743.8</td>
<td>avg. time=644.1</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=4.0</td>
<td>fraction</td>
<td>sat=1.0</td>
<td>avg. cost=658794.6</td>
<td>avg. time=2574.2</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=5.0</td>
<td>fraction</td>
<td>sat=0.2</td>
<td>avg. cost=1741937.8</td>
<td>avg. time=9661.1</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=6.0</td>
<td>fraction</td>
<td>sat=0.0</td>
<td>avg. cost=2097151.0</td>
<td>avg. time=13017.1</td>
</tr>
</tbody>
</table>

Implementation timed out!

After:

<table>
<thead>
<tr>
<th>Ratio</th>
<th>r=1.0</th>
<th>fraction</th>
<th>sat=1.0</th>
<th>avg. cost=48.9</th>
<th>avg. time=0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio</td>
<td>r=2.0</td>
<td>fraction</td>
<td>sat=1.0</td>
<td>avg. cost=408.0</td>
<td>avg. time=3.0</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=3.0</td>
<td>fraction</td>
<td>sat=1.0</td>
<td>avg. cost=972.5</td>
<td>avg. time=5.8</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=4.0</td>
<td>fraction</td>
<td>sat=1.0</td>
<td>avg. cost=2186.8</td>
<td>avg. time=12.7</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=5.0</td>
<td>fraction</td>
<td>sat=0.2</td>
<td>avg. cost=3035.4</td>
<td>avg. time=25.9</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=6.0</td>
<td>fraction</td>
<td>sat=0.0</td>
<td>avg. cost=2789.8</td>
<td>avg. time=27.2</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=7.0</td>
<td>fraction</td>
<td>sat=0.0</td>
<td>avg. cost=2236.4</td>
<td>avg. time=19.7</td>
</tr>
<tr>
<td>Ratio</td>
<td>r=8.0</td>
<td>fraction</td>
<td>sat=0.0</td>
<td>avg. cost=1507.8</td>
<td>avg. time=17.7</td>
</tr>
</tbody>
</table>

Your implementation took 1126 milliseconds to complete.
Your implementation made 139856 recursive calls.
matt@colgateofficecininml 3_solution %

In 1994, researchers learned two important properties about CNF propositions (reported in Science). The likelihood of satisfiability depends on the ratio of clauses to variables. If the ratio is < 4, almost all propositions are satisfiable; if the ratio is > 5, almost all are unsatisfiable. The hardest propositions to check have ratios between 4 and 5.

3 Your Task

Your task is to improve your satisfiability algorithm by implementing the "stop early" optimization on the next page:
3.1 "Stop Early" Optimization Overview

As you are building a variable assignment via recursion, it may not be necessary to assign every variable a value to determine whether the variable assignment leads to satisfiable assignment.

For example, suppose $\varphi := (p \lor q) \land (p \lor r) \land (p \lor s)$ and variable assignment $M$ has assigned $p$ to True. In this case, we can already tell $\varphi$ is satisfiable regardless of how $q, r,$ or $s$ are assigned.

Another example, suppose $\varphi := (p) \land (q \lor r) \land (s \lor \neg q) \land (t \lor r) \land (\neg q \lor t)$ and the variable assignment $M$ has assigned $p$ to False. In this case, $\varphi$ is unsatisfiable under $M$ regardless of how $q, r, s,$ and $t$ are assigned.

One final example, suppose $\varphi := (\neg p \lor q) \land (\neg p \lor r)$ and variable assignment $M$ has assigned $p$ to True. In this case, we do not know yet whether $\varphi$ is satisfiable or not – it may or may not be depending on how $q$ and $r$ are assigned.

3.2 Implementing the Optimization

You will implement the "stop early" optimization by checking the following on each step of the recursive algorithm, regardless of if every variable in the proposition has an assigned value in the variable assignment:

- if you have sufficient variable assignments to determine every clause evaluates True, stop early and return True.
- if you have sufficient variable assignments to determine at least one clause evaluates False, stop early and return False.
- if neither of the above scenarios are realized, the proposition is inconclusive given this (partially incomplete) variable assignment. Therefore, you follow the same recursive case as in last lab’s Part 1:
  1. select an unassigned variable from the variable assignment and assign it true.
  2. recursively check if the proposition is satisfiable with this new variable assignment. If not, assign the same variable false and recursively check the satisfiability again.
  3. if the proposition is not satisfiable with the selected variable assigned true or false, return false.

In CNFProposition.java, implement the satisfiabilitySEHelper method with an enhanced version of the satisfiability algorithm you wrote last week, incorporating the stop early optimization.

4 Pre-Lab Questions

After reading this document and provided code, answer the following before we meet (we will discuss in-lab):

1. Which of the below Java conditionals could potentially throw a NullPointerException?:

(a) if (x.equals(y) || x == null)
(b) if (x == null || x.equals(y))
(c) if (x == null && x.equals(y))
(d) if (x != null && x.equals(y))

2. Suppose we had a static helper function, evaluateClause, which attempts to evaluate a single clause given a (potentially incomplete) variable assignment. What would be an ideal return type for this function?

5 Submission

See the top of this document for your lab section’s due date/time. When uploading your submission to Moodle submit only the files listed below:

- CNFProposition.java
- Lab05Tester.java