Project Readme Template

Version 1 9/11/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_"teamname"

Also change the title of this template to "Project x Readme Team xxx"

<table>
<tr><td>1</td><td>Team Name: dpll_ianrobinett</td></tr>
<tr><td>2</td><td>Team members names and netids: Ian Robinett, irobinet</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: My project was to create a polynomial time 2-SAT solver using the DPLL algorithm.</td></tr>
<tr><td>4</td><td>Overall success of the project: I successfully implemented the dpll algorithm to solve 2-SAT problems; however, in the worst case the complexity of DPLL for 2-SAT appeared to be still 2^n. Professor Kogge indicated this was acceptable.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 6</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/irobinett3/theory_project1_dpll_ianrobinett</td></tr>
<tr><td>7</td><td>List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.
<table>
<tr><th>File/folder Name</th><th>File Contents and Use</th></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>DPLL_solver_dpll_ianrobinett.ipynb</td><td>This contains my algorithm, reads in SAT Problems, solves them, and outputs results. It also graphs time complexity as well as checking accuracy of results using a python library and displaying the accuracy.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>2SAT.cnf_dpll_ianrobinett.csv</td><td>This is the file provided for the sake of testing our algorithm for 2SAT.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>results_dpll_ianrobinett.csv</td><td>This contains the outputs Test Case number, Number of Variables, Elapsed Time, Whether or not it is satisfiable, and whether or not my answer was correct. This is in csv format</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>Timing_Screenshot_dpll_ianrobinett.png</td><td>This is my plot created using matplotlib in real time to generate a graph of time taken versus number of variables.</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
Languages: Python. Used Jupyter Notebooks for ease of plotting.
Libraries used: matplotlib, python-sat, csv, time</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):

1. Clauses:
○ Type: 2D list
○ Purpose: Each clause represents a disjunction of literals. A clause is satisfied if at least one of its literals is true.
2. Assignment:
○ Type: Dictionary
○ Purpose: This structure keeps track of the current assignment of truth values to each variable. The key is the variable's absolute value, and the value is a boolean indicating whether the variable is assigned true or false.
3. Pure Literals:
○ Type: Set
○ Purpose: This structure is used to store the pure literals found during the literal elimination phase.
4. Unit Clauses:
○ Type: List
○ Purpose: A unit clause is a clause with only one literal.
5. Results:
○ Type: 2D List
○ Purpose: This structure is used to store results for each test case, including the number of variables, elapsed time, and satisfiability status.
6. Timing Information:</td></tr>
</table>

	○ Type: Lists ○ Purpose: Two separate lists (one for satisfiable and one for unsatisfiable cases) track the elapsed time taken to solve each test case. Summary ● The clauses and assignment structures are crucial for the DPLL algorithm's recursive nature. The algorithm modifies these structures as it progresses through variable assignments and simplifications. ● The pure literals and unit clauses help in optimizing the search by reducing the problem size early, which is a significant part of the DPLL approach. ● The results and timing information allow for the tracking of the algorithm's performance across multiple test cases, providing insights into efficiency and accuracy.
10	General operation of code (for each subproject): The project implements the DPLL algorithm for solving 2-SAT problems. The code reads CNF format SAT problems from the given csv file, applies unit propagation and pure literal elimination, and recursively tries to assign values to variables to determine satisfiability. It measures execution time for performance analysis and checks results against a Python-SAT library for accuracy. It then outputs results to a csv file.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I added test cases using the Python-SAT library. This library determines if there is a solution for a specific SAT problem. If I found a solution, I checked to see if the library agreed with my solution. If no solution was found, I verified that the library concurred. The library indicated that my code was 100% accurate.
12	How you managed the code development: I utilized a structured approach, implementing the DPLL algorithm incrementally. I frequently tested components as I built them, ensuring each part functioned correctly before integrating it into the larger project. I maintained version control through Git to track changes and manage code versions.
13	Detailed discussion of results: 1. Accuracy of Results: ○ The DPLL solver achieved a 100% accuracy rate when compared against the results provided by the Python-SAT library. 2. Time Complexity Observations: ○ While the algorithm effectively solved the 2-SAT problems, theworst-case time complexity still approached O(2^n) , where n is the number of variables. Despite this, Professor Kogge indicated that this performance was acceptable for the given project scope. 3. Test Case Diversity: ○ The diversity of test cases sourced from the given file provided a robust framework for evaluating the solver’s performance. Each test case varied

	in size and complexity, allowing for a comprehensive assessment of the algorithm's capability to handle different scenarios. This allowed to test both time complexity and accuracy of the algorithm 4. Graphical Representation of Results: ○ The timing plots generated using Matplotlib visually represented the relationship between the number of variables and the time taken to resolve each test case. The plots revealed a relationship that appeared to be O(2^n) in the worst case. 5. Missed Test Cases Compared with Existing Libraries: ○ The algorithm did not ever miss a single test case, and the value of misses was always 0 compared with the existing Python-SAT library.
14	How team was organized: I worked entirely alone and completed the project by myself.
15	What you might do differently if you did the project again: Overall, I really enjoyed how the project went. However, if I were to change one thing, I would have randomly generated some of my own SAT problems, but the provided ones were definitely sufficient.
16	Any additional material: The complete code, test cases, and results can be found in the linked GitHub repository. Additionally, the plotted results visualize the performance of the algorithm.