Project 1 Readme Team jrelling

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: jrelling</td></tr>
<tr><td>2</td><td>Team members names and netids: Jack Rellinger (jrelling)</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: DPLL algorithm 2-SAT polynomial time solver.</td></tr>
<tr><td>4</td><td>Overall success of the project: The project was largely successful, I learned a lot and reinforced my knowledge about backtracking algorithms. In my testing runs, my dpll solver correctly identified all of the simple satisfiable problems as satisfiable and the unsatisfiable problems as unsatisfiable. However, it was difficult to tell if it was truly polynomial time, and would most likely need to test and plot with a larger dataset.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: approx. 7 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/jackrell/toc-project01-jrelling-2_sat_dpll</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><td>File/folder Name</td><td>File Contents and Use</td></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>dpll_jrelling.py</td><td>This file contains all of the functional code, including the dpll backtracking algorithm (and associated functions), handling of the csv input and output, as well as the plotting.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>check_2SATcnf_S_jrelling.csv
check_2SATcnf_U_jrelling.csv</td><td>These 2 files contain 3 simple problems each that are satisfiable and unsatisfiable</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>respectively, in a format that mirrors the given testing file.</td></tr>
<tr><td>data_2SATcnf_jrelling.csv</td><td>This was the file I used as input for my timing runs, given by the instructor in the specified cnf csv format.</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>output_2SATcnf_jrelling.csv</td><td>This was the csv file generated by running my code program, which was then used to plot the data. It has 3 cols: satisfiability, num variables, and time in seconds.</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>plot_jrelling.png</td><td>This was the plot generated by my code, with unsatisfiable problems in red, satisfiable in green, and a linear best fit line.</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
The project was written entirely in python, using the csv, time, matplotlib.pyplot, pandas, and numpy libraries.</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
The primary data structures used in this project were:
2-D lists to represent the cnf form of the problem at hand, as well as its simplified version while running the algorithm.
A dictionary to store the current assignment of the variables.
CSV reader/writer structures to handle input and output.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject)
The code first parses the CSV file into each individual cnf problem. Then, it uses the dpll algorithm to assign each variable and recursively simplify the clauses. Then it checks if the clauses are all satisfied given the current assignment, or if any are unsatisfied. It follows a depth-first tree structure in assigning each variable True or False, checking if the problem is satisfied by each assignment of True or False for each variable, and backtracks if no solution is found on that branch. After the algorithm runs, the main function then creates a new csv file for the output (satisfiable/unsatisfiable), the number</td></tr>
</table>

	of variables in the problem, and the time it took to complete. Then a graphing function is called to create the scatter plot.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. The test cases I added were the check_2SATcnf_S_jrelling.csv and check_2SATcnf_U_jrelling.csv files, which held short satisfiable and unsatisfiable problems. These helped me first to ensure my csv parsing was working to split the file into problems, then it helped me to know, after the algorithm ran, whether the output matched the problem's satisfiability.
12	How you managed the code development I managed the code development through the use of several online references for DPLL algorithms, namely this presentation that helped me visually understand the algorithm: https://homepage.cs.uiowa.edu/%7Etinelli/classes/196/Fall09/notes/dpll.pdf I also used generative AI to help create pseudocode for the solver, which went a long way in helping me through the process of creating the algorithm and debugging. I used git for the command line to upload all of my files.
13	Detailed discussion of results: The scatter plot and my output csv data demonstrate that the DPLL algorithm I implemented generally increases in solving time as the number of variables grows, with notable differences between satisfiable and unsatisfiable problems. Satisfiable problems tended to be solved faster, as the simplification part of the algorithm allowed for solutions to be found faster, and solutions could be found without much backtracking. However, for unsatisfiable problems, regardless of variable size, they tended to take longer due to the need for repeated backtracking in many cases. It is also worth noting that there is large variation in the data, especially as the number of variables grows, which demonstrates the structure of the clauses impacts the complexity of the problem, and there are more possible complex structures with more variables. Overall, while the algorithm was extremely efficient and consistent for smaller problems (16 or less variables), as the number grew, solving time became much more unpredictable especially for the unsatisfiable cases.
14	How team was organized I worked alone, so organized through individual effort.
15	What you might do differently if you did the project again I would want to test with a larger dataset if I did the problem again, as well as use different plotting software to better demonstrate the variation in unsatisfiable and satisfiable problems, especially for the smaller number of variable problems. A larger dataset would allow me to better determine if the algorithm is running in polynomial time and if the data points I had with extremely long time to completion for 26 and 28 variable count were outliers, or if the algorithm is truly closer to exponential time. If I had more time, I would also want to make the program more interactive, allowing the user to input the file they wanted to test on as well as specify the output file, or if they want to generate a graph or not.
16	Any additional material: N/A