Project 1 Readme Team WSULLI22

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 Name"

<table>
<tr><td>1</td><td>Team Name: wsulli22</td></tr>
<tr><td>2</td><td>Team members names and netids: wsulli22</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: 2-SAT Solver (DPLL Algorithm)</td></tr>
<tr><td>4</td><td>Overall success of the project: Completed full/accurately (...i think :))</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 8 HOURS</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/wsulli22/toc-project-1</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_algorithm_wsulli22.py</td><td>Actual DPLL algorithm that takes in wff and returns S/U</td></tr>
<tr><td>dpll_implement_wsulli22.py</td><td>Extracts data from input file and calls above function/file on each problem and generates output file</td></tr>
<tr><td>check_algorithm_wsulli22.py</td><td>Compares two output files (generated to expected) to see accuracy of algorithm</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>check-kSATu.cnf-wsulli22.csv</td><td>Given sample test data (input data)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>check-kSATu-results-wsulli22.csv</td><td>Given sample test data expected output</td></tr>
<tr><td></td><td></td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>output_generatd_output_from_check-kSATu_wsulli22.csv</td><td>File created given the test file (input) above</td></tr>
<tr><td>data_generatd_output_from_check-kSATu_wsulli22.csv</td><td>[same as above file] included b/c needed to have “data” in file name. This was used to get times for plot</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>plots_wsulli22.png</td><td>Created based off the (output) file above</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
Python with networkx, matplotlib.pyplot, csv, sys, numpy, scipy.optimize, time, shutil</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project): [only 1 project, so no sub-projects]

dicts, lists, sets, csv data, command-line arguments, time (measurements)</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject) [only 1 project, so no sub-projects]

My program consists of three main files: one that runs an implementation of the DPLL 2SAT solver, one that processes/calls/“solves” the SAT problems given an input (+ creates output files) by calling the DPLL algorithm in a separate file, and a checker which verifies the correctness given a pair of generated and expected satisfiability results. You “run” my full program by first running the implementation file and feeding in an input csv data source via the command line. That file will create a data output file. You can then separately run the check file program that will compare that file to one where you know the outcomes. You specify what files to compare via the command line. After that, that data can/was used to create a plot showing how the performance changes as the number of variables increases (see plot).</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the correctness of your code.

I used the provided “kSATu.cnf-wsulli22.csv” with known results in “kSATu-results.cnf.csv”</td></tr>
</table>

	I wrote a program called check_algorithm_wsulli22.py that compares the output that my implementation of the DPLL algorithm generates to the known/expected output of the input described in the previous sentence. The check algorithm works by comparing the problem ID and outcome (S or U) value of the expected and generated output files. All solutions that were included in the expected results file matched with those generated by my algorithm/solver. This suggests that my code is correctly solving the 2SAT problems.
12	How you managed the code development: I started by understanding the desired outcomes/final product by reviewing the instruction documents and speaking with TAs during office hours. After selecting the project, I designed my approach to solving the problem. My approach involved building a program that reads and processes the data, then calls a separate DPLL algorithm for each of the problems. I used the provided test files (described above) to verify whether the algorithm worked and determined its accuracy/correctness by writing and running a check program. All files were organized in the same folder, and GitHub was only used to deploy the final product.
13	Detailed discussion of results: *[the results discussed will be based on the final graph/plot]* The graph included in the GitHub repository represents the relationship between the number of variables in a given 2-SAT problem and the time it took that given problem to be solved/computed - the program's execution time. The goal of the SAT problem project was to learn how to implement a SAT problem solver and - equally as important - to better understand why some SAT problems are known to be unsolvable in polynomial time because of enormously large execution times that are required to compute those larger problems. The results of my project, testing, and algorithm show this relationship. Specifically, they show how as the number of variables involved in a given 2-SAT problem increases, the execution solve time increases exponentially (as seen in the graph). This is likely due to the following related reasons: For every new variable added, (1) the search space increases, which makes (2) the backtracking more complex, leading to more branching decisions and paths to explore, as well as more memory operations that need to be computed. Speaking generally, these factors together are what cause the execution time to increase rapidly. The data set represented in the graph is from the 2-SAT problems of the provided "kSATu.cnf.csv" test data. The execution times associated with each of the problems were recorded based on the solve times when run with my implementation of the DPLL algorithm. The bounding curve, represented by the equation f(x) = 2.99 * e^(0.40 * x) + 2830.93, models the exponential relationship between the "kSATu.cnf.csv" dataset and the

	execution times for the problems in the sample test dataset. The graph features 8 semi-regularly-occurring vertical “stacks” of points representing the execution times of different 2SAT problems with the number of variables ranging from 4 to 24. ***Error Analysis:*** While the majority of the data follows the exponential trend, the execution times of some of the 4-variable problems on the left do not, which may indicate an anomaly. This issue could potentially be an error or an efficiency problem in the implementation of my program or its related files. However, while I’m bringing attention to those atypical-data points, the dataset as a whole follows the expected trend.
14	How team was organized: N/A - Worked Alone
15	What you might do differently if you did the project again: I would ask TAs for guidance earlier in the project process, spend more time planning out my approach before actually coding, and better handle edge cases to make my program testing/debugging process more efficient.
16	Any additional material: N/A