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: Chicken</td></tr>
<tr><td>2</td><td>Team members names and netids: Luke Zimmermann: lzimmer5, Tim Gunn: tgunn2, Arda Kurama: akurama2</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: SAT (brute force, backtracking, and best case)</td></tr>
<tr><td>4</td><td>Overall success of the project: Very successful, each sub-project functions as intended including the plotting. Overall 10/10.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 15 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/TimGunn60/Project1-TOC</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>sat.py</td><td>Implementation of the 3 sub-projects: solving SAT formulas via brute force, backtracking, and best case</td></tr>
<tr><td>generate_plots_chicken.py</td><td>Creates the plots demonstrating the run time of each algorithm. Run the file by using the following from the command line: uv run generate_plot_chicken.py</td></tr>
<tr><td colspan="2">Test Files</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>N/A</td><td>N/A</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>results</td><td>Folder containing all CSV outputs for the different input files, plots made from the CSV files, and the generate_plots_chicken.py</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>results</td><td>The folder that contains all of the plots. A plot is made for each CSV file that exists within the results folder. The plots show execution time over the number of literals in the formula, as well as if the formula is satisfiable or not.</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
Programming languages: Python
Libraries: pandas, matplotlib, pytest</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
Brute force: Python dictionary (implemented via a hashmap under the hood)
Backtracking: Python list (implemented via a linked list under the hood), Stack (i.e. call stack used for recursive function calls)
Best case: Python dictionary (implemented via a hashmap under the hood)</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):
Brute force: Builds every possible combination of literal assignments, checking each one to see if satisfies the given formula
Backtracking: Simplifies as much as possible by assigning values to single-literal clauses and removing clauses that are satisfied. If the algorithm hasn’t finished, it guesses by picking a variable and setting it to True, then tries False if that fails.
Best case: generates every possible combination of literal assignments, keeping track of the one that results in the largest number of satisfied clauses, as well as if the formula is satisfiable or not</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the</td></tr>
</table>

	correctness of your code: N/A
12	How you managed the code development: Each team member cloned the shared GitHub repository to their own computer. Each team member then did work on separate branches and merged them into the main branch when completed. We worked both together and individually to develop the code.
13	Detailed discussion of results: The results are more or less exactly what we expected. The brute force algorithm takes longer to complete than the backtracking algorithm in general. This is what we expected, as the backtracking algorithm takes advantage of the recursive nature of the problem to generate solutions, which is faster than simply trying every single possible solution. However, the time that both algorithms take to complete both generally increases exponentially as the number of literals in the formula increases. Given that SAT problems fall into the class of NP, this is the exact output that is expected. The best case algorithm uses a brute force implementation to find the best assignments, so executes in the same time as the brute force algorithm.
14	How team was organized: Each team member completed one sub-project (i.e. one member did backtracking, one did brute force, and one did best case). We worked collaboratively and individually to complete the necessary code and tests.
15	What you might do differently if you did the project again: If we could do the project again, we would try to work in person together more. Although we did work together at some times, we feel that we could've figured out how to write the code faster had we all worked with each other to develop the implementations.
16	Any additional material: N/A