Project 1 Readme Team cbowers4

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"

1	Team Name: **cbowers4**
2	Team members names and netids: **Collin Bowers, cbowers4**
3	Overall project attempted, with sub-projects: **2-SAT: Implementing a polynomial time 2-SAT solver (look up the DPLL algorithm)**
4	Overall success of the project: **Success**
5	Approximately total time (in hours) to complete: **9 hours**
6	Link to github repository: **https://github.com/collinb424/cbowers4_theoryproj1**
7	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.

File/folder Name	File Contents and Use
Code Files	
two_SAT_cbowers4.py	Code for solving 2-SAT and generating the plot
Test Files	
check_cbowers4.csv	CSV file for testing 2-SAT
Output Files	
output_cbowers4.txt	Results of running two_SAT_cbowers4.py on check_cbowers4.csv
Plots (as needed)	
plots_cbowers4.png	Plot of Number of variables vs Time

8	Programming languages used, and associated libraries: **Python, matplotlib, csv, numpy, scipy**
9	Key data structures (for each sub-project): - **The cnf data structure is a 2d list used to track the current form of the wff based on where we are in the passed in csv file. Within it are lists, with each one made up of a clause, which is itself a list made up of literals. The literals correspond to the variables and their negation, as specified in the csv file.** - **The assignments data structure is a dictionary keeping track of variables and what assignments they currently have. It is primarily used to detect conflicts in my helper functions. If a variable wants to be assigned to something other than what it is in the assignments dictionary, this shows there is a conflict and we need to backtrack**
10	General operation of code (for each subproject) - **The code begins by reading the specified data file in, which it assumes is a cnf file but in csv form. It defaults to using check_cbowers4.csv. It creates a cnf 2d list after iterating over the current problem in check_cbowers4.csv. After it has created the cnf, it calls the dpll function.** - **The dpll function first calls functions that perform unit propagation and pure literal elimination. These essentially simplify the cnf so that less work needs to be done in the average case. It then checks to see if all clauses or variables are satisfied, in which case it returns true. It also checks if there is an empty clause in the cnf, in which case it returns unsatisfiable. After this, it chooses a random variable to assign and assigns true to the variable. If there is a solution down this path, it will return true. If not, it will try assigning false to the variable. If there is a solution down this path, it will return true. If not, it will have to backtrack and try changing some other assignment, assuming there are still assignments left to change.** - **After all of the problems in the csv file have been exhausted, the data is plotted using matplotlib and a curve is fitted using numpy and matplotlib.**
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. - **I primarily used check_cbowers4.csv, which is a csv that Professor Kogge provided to test 2-SAT. Professor Kogge specifically mentioned in class that it was okay to only use this file for testing. I decided to use this because it had a variety of thorough test cases. However, I also played with some of my own test cases to ensure that specific edge cases were handled. Especially for building up the helper functions, I used many of my own test cases.**
12	How you managed the code development - **I used the recommended wikipedia article and began with coding the main helper functions, unit_propagate and pure_literal_assign. Before coding these, I played with examples on paper to understand how to solve them. I then tested the code with my own examples to ensure they were working. After this, I coded the DPLL function to bring everything together. I used my own examples and Professor Kogge's examples to test that,**

	troubleshooting as needed. After all of this, I turned to focus on creating the plots and used trial and error until they appeared correctly.
13	Detailed discussion of results: - **The plot for the number of variables (on the x-axis) vs time (on the y-axis) is clearly exponential. The line fits the data quite well, and the formula for it is 31.23*exp(0.0593*x). As the number of variables increases, the time begins by increasing slowly, and then smaller increases in the number of variables make the DPLL algorithm take much longer. In my graph you can also see some outliers far above the line of best fit, and essentially all of these are when the wff is unsatisfiable. I tried adding some additional wffs with a greater number of variables, but the time increased too quickly, and I was unable to do enough data points that would have been useful for plotting.**
14	How team was organized - **Since I was an individual team, I handled all of the responsibilities: the algorithm, plot, documentation, GitHub, etc.**
15	What you might do differently if you did the project again - **Split up the project work over multiple days**
16	Any additional material: **N/A**