Readme_teamname

Version 1 8/22/24

A copy of this file should be included in the github repository with your project. Change the teamname above to your own

1. Team name: mjim
2. Names of all team members: Mariya Jim
3. Link to github repository: https://github.com/mjim18/mjim
4. Which project options were attempted: Implementing a polynomial time 2-SAT solver
5. Approximately total time spent on project: 10 hours
6. The language you used, and a list of libraries you invoked.
   Language: Python
   Python libraries: (time, matplotlib.pyplot, csv)
7. How would a TA run your program (did you provide a script to run a test case?)
   I included a script in main to be able to run the solver so as long as the TA has the matching csv file in the same folder as the script, it should be able to run.
8. A brief description of the key data structures you used, and how the program functioned.
   I used a list data structure for the CNF clauses. It holds lists of integers, where each inner list represents a clause in conjunctive normal form (CNF). Each integer in the inner list corresponds to a literal, with positive integers indicating the variable and negative integers indicating its negation. I also used lists for both problem sizes and execution times that were stored for each problem. I first read in the input using a csv read function and for each line that starts with p, it extracts the number of variables and clauses, and prepares to read the corresponding CNF clauses. Each line that doesn't start with c or p is interpreted as a clause, so when the next p is encountered, it processes the current cnf using the process_cnf function which runs the DPLL algorithm and measures execution time. The DPLL algorithm is implemented with functions for propagating literals, performing pure literal elimination, and selecting literals for branching. It recursively checks for satisfiability by attempting to satisfy the CNF through chosen literals. After processing all CNF formulas, it prints the results (satisfiable or not) and plots the execution time against the number of clauses. It also prints a plot with the execution time (y-axis) vs the number of clauses (x-axis).

9. A discussion as to what test cases you added and why you decided to add them (what did they tell you about the correctness of your code). Where did the data come from? (course website, handcrafted, a data generator, other)
   I incorporated a couple different test cases with the first checking whether it can find satisfiability with two variables and then I did the same with an unsatisfiable CNF. The next case was just a longer CNF to check the algorithm's performance and the last one I checked was whether the program handles a CNF with no clauses appropriately. I used the dataset provided on the course website to verify my code.

10. An analysis of the results, such as if timings were called for, which plots showed what? What was the approximate complexity of your program?

The scatter plot generated from the problem sizes and execution times data illustrates the relationship between the number of clauses in the CNF formula and the time taken to determine satisfiability. As the number of clauses increased, the execution time grew exponentially, which indicates that the complexity of the problem affects the time required to solve the problem. I also observed that it was common for unsatisfiable cases to have longer execution times. The worst case time complexity for my program would be O(2n).

11. A description of how you managed the code development and testing.

I researched the DPLL algorithm to understand its structure and implementation. I broke down the different components of the algorithm into functions to simplify the main function and keep it organized. I incrementally tested simple CNFs as I went along to ensure that the algorithm was running properly. Then I made the function to read in the csv file and started using the dataset.

12. Did you do any extra programs, or attempted any extra test cases

I made mini test cases as I went along to ensure that my code was functioning properly before using the large data set.