Readme_asolis4

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:
   asolis4

2. Names of all team members:
   Anaregina Solis

3. Link to github repository:
   https://github.com/asolis5405/asolis4_TOCProject1.git

4. Which project options were attempted:
   2-SAT Solver using DPLL algorithm

5. Approximately total time spent on project:
   15hrs

6. The language you used, and a list of libraries you invoked.:
   Language: Python;
   Libraries: csv, time, typing: List

7. How would a TA run your program (did you provide a script to run a test case?):
   To run on terminal after making executable: ./2-Sat_Solver_asolis4.py
   Currently the script reads in the file of test cases provided in the Resources page (test_cases_asolis4) and outputs the results in the output file (output_asolis4).

8. A brief description of the key data structures you used, and how the program functioned.:

   Key data structures used were lists, dictionaries, and sets. I used lists to hold the x and y values containing number of literals and execution time for each case. The most important list of my program is the list containing the clauses of each case. This is the list that gets passed to the dpll function, the unit propagation function, and the pure literal elimination function. Throughout these functions, the clauses list gets modified with the purpose of finding the truth assignments that will cause each clause to return True. Another important list I used was the variables list which also gets passed to all three functions and contains the assignment (truth value) for each variable. The assignments get modified with the goal of satisfying a clause (making it True). Sets are introduced in the pure_literal function with the literal_count set containing the occurrences of each literal as well as the pure_literals set which later becomes a dictionary containing the literals as keys and their assigned values as values. This dictionary is used to find pure literals in order to eliminate clauses.

In the end, the dpll algorithm will return True (satisfiable) if all clauses have been satisfied (clauses list returns empty). If however, one of the clauses is not satisfied (a clause has no literals left) then the dpll algorithm will return False (unsatisfiable).

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 added the test cases found on the course's Resources page called 2SAT.cnf.csv (saved as test_cases_asolis4 in my repository). I decided to add them since they would evaluate whether my code was capable of reading in multiple entries at a time. Furthermore, the large amount of test cases in this file ensured that my program would be tested with a variety of different cases that could present challenges I had not considered beforehand. I also knew that these test cases contained 50 satisfiable cases and 50 unsatisfiable cases, therefore, I would be able to test the correctness of my code by testing if it identified those exact numbers of satisfiable and unsatisfiable test cases, which it did.

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?:

    To analyze my results, I created a scatter plot comparing the number of literals to the execution time for each case. The number of literals was computed my multiplying the number of clauses by 2 since there are 2 literals per clause. Execution time (s) was recorded by using .time() from the time library. The plot of this data created an exponential function, however, because of the 2-SAT solver's efficiency, the exponential curve was more of linear line. The line gave the formula y = 4E-05e^(0.029x). The approximate time complexity of my program was O(n) since the plot could produce a linear graph.

11. A description of how you managed the code development and testing.
    I managed the code development by starting off with writing the loop needed to parse through a csv file, specifically the one that was given. After completing this I tested the output by making my code print out the problem number and its clauses. Once I saw my program was able to do this without any errors, I began constructing the dpll algorithm. I created the dpll function first and then created the other two functions that the dpll function calls to reduce the number of clauses. Throughout this entire process I used print statements before and after function calls to verify that the clauses were modified correctly. I also used print statements before and after while loops and for loops to ensure the program was doing what it was supposed to. Before, giving my program the entire test case file, I first gave it a few test cases to make sure it could read those correctly. Once it achieved this, I moved on to having it read through the entire test case file.

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

    I attempted an extra program where I did not implement recursion (old_draft_asolis4.py). This program ultimately ended up not working since I had to use the stack to backtrack whenever a clause was not satisfied. This program classified 70 cases as satisfiable, which is not correct. After working on this program for almost 2 days I decided to start from scratch on my new program 2-Sat_Solver_asolis4.py. In this program I implement recursion and use parts of my functions from the previous program to create a working dpll algorithm that calls on the functions: unit_clause_prop and pure_literal. This program returns 50 satisfiable and 50 unsatisfiable cases from the test_cases_asolis4.csv file, which I was told was the correct output. Furthermore, after solving 10 cases by hand, I was able to verify that the program gave the correct output for the 10 cases. This, along with the 50/50 split between satisfiable and unsatisfiable, makes me believe that my program is correct.