Team name: nick

Names of all team members: Nicholas Cichoski

Link to github repository: https://github.com/ncichosk/theory-1 (https://github.com/ncichosk/theory-1)

Which project options were attempted: Implementing a polynomial time 2-SAT solver

Approximately total time spent on project: 8 hours

The language you used, and a list of libraries you invoked: python; time, sys, csv, matplotlib

How would a TA run your program (did you provide a script to run a test case?):
They would run "python 2sat_solver.py input.csv {0,1,2}" in the terminal where input.csv is a csv file containing the problems and {0,1,2} refers to the algorithm to be ran. 0 will run dumbSAT, 1 will run the polynomial time 2SAT, and 2 will run both.

A brief description of the key data structures you used, and how the program functioned:
I used a 2D array to store each individual wff problem problem. Each array inside the array corresponds to a clause in the wff and was read in from each line in the csv. I used a dictionary to store all 100 test cases with each key called wff_# and each value being a 2D array representing that wff. I iterated through the dictionary to solve each wff individually. In order to create a polynomial time solver, I first initialized a solution array to -1. I then went through iteratively and assigned values to the variable checking the wff after each assignment. If an assignement made a clause in the wff false, I would backtrack out of that "branch" of solutions and change the previous assignments. I also used several lists to track the satisfiable/unsatisfiable output, time per wff, and size of wff for each case in order to use in the matplotlib functions that I created.

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 used the 2SAT.cnf.csv file provided on Canvas as testing data. This is a substantial data set as it contains 100 wffs of varying length. I checked the correctness of my code by comparing the DPLL output (satisfiable or unstaisfiable) to the output generated by the dumbSAT solver on Canvas (which I assumed to produce correct output). I tracked any conflicting output in an array which is printed at the end of output_both_nick.txt. I also used this data to plot the solve times for each algorithm.

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 plot of time to solve v size of problem in plot_DPLL_time_complexity.pdf shows some upward curve in the worst case line for the DPLL algorithm, it is definately bounded inside of a polynmial time complexity. An approximate time complexity for this algorithm may be $O(n^{2)}$ or $O(n_3)$. This is advantagouse compared to the dumbSAT algorith in two ways: time complexity and scale. Looking at plot_dumbSAT_time_complexity.pdf we see a much steeper rise in time complexity for cases larger than 25 variables. This rapid increase is much faster than a polynomial time complexity. This time compleity is probably closer to O(k^n) Additionally, the scales used for these graphs are massively different. The slowest DPLL solver took less than half a second, while some dumbSAT solves took over 10 minutes to solve. This difference is best seen in plot_both_time_complexity.pdf where the DPLL times do not even register compared to the dumbSAT times which spike at 28 variables.

A description of how you managed the code development and testing. I broke down the project into four parts. First I developed the main function and made sure I could read in and process csv data files in the way I wanted to. Second, I implemented a copy of the dumbSAT algorithm to work with the data structures I was storing the wffs in and produce the output I wanted. Then I developed the DPLL algorithm. I ran both the DPLL and the dumbSAT at the same time for various tests and compared outputs to test my functions as I developed the DPLL algorithm. I also implemented a comparison that checked the satisfiable/unsatisfiable assignments for all 100 wffs in the data set and printed any contradictions. Finally, I created data structres that stored the output in a way that I could graph with matplotlib and changed my print statements to write to output files.

Did you do any extra programs, or attempted any extra test cases: No