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_bruscinators

Also change the title of this template to “Project x Readme Team xxx”

1	Team Name: **Automata Avengers**
2	Team members names and netids **Mary Brusco mbrusco**
3	Overall project attempted, with sub-projects: **Rewrite DumbSAT to use an incremental search through possible solutions**
4	Overall success of the project: **Succeeded! Shows the exponential relationship between the size of a problem and the time it takes to output it.**
5	Approximately total time (in hours) to complete: **5-7 hours**
6	Link to github repository: **https://github.com/marybrus/project1_bruscinators**
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	
main_bruscinator.py	**Consists of all of the code used to generate output files including reading input, running tests, and parsing files.**
Test Files	
input_bruscinator.py	**The input contents include the test cases and samples of inputs. They are in the formate “(number of variables) (number of clauses) (number of literals) (number of trials)” where each line is a separate test case.**
Output Files	
results_bruscinator.csv	**This file holds a summary of each test case in csv form.**

trace_bruscinator.csv	**This file is similar to the results file but it has more detailed information for each test case in CSV form.**
cnffile_bruscinator.cnf	**The cnf file contains the formulas in Conjunctive Normal Form that were tested for each trial.**

8	Programming languages used, and associated libraries: **Programming language: Python** **Libraries: time and random and matplotlib.pyplot as plt and numpy as np**
9	Key data structures (for each sub-project): **1. def parse_file(file_path):** The key data structure used for this function is a 2D array because when we parse the text file of input, we are aiming to convert it into a 2D array which is what is used in the run_cases() function. **2. def plot_results(sizes, times, flags):** The main data structure for this is the array. I use an array to loop through every result and plot based on its properties (i.e. if it is unsatisfiable it results in a red triangle, otherwise a green circle). Using arrays made it really easy to extract necessary data because the index represented the sample number. **3. def increment(assignment):** The key data structure for this function is a list of integers in binary representation that represent the current truth assignment of the variables in the formula (0 being false, 1 being true) **4. def check(Wff, Nvars, Nclauses, Assignment):** For this the key data structure is a list of clauses (or integers) which represents variables in the CNF formula. Positives show it is in normal form, and negative show it is in negative form. We also use various variables to hold the number of variables, clauses, the assignment (a list of true or false values based on the CNF formula).
10	General operation of code (for each subproject) **1. def parse_file(file_path):** This function is used to parse the input file in order to test and run cases on the different samples. This increases the functionality of the code because we are able to test different test cases very easily. **2. def plot_results(sizes, times, flags):** The purpose of this function is to plot the results of the satisfiability of the samples in a visually appealing way. The function also adds an exponential line of best fit in order to compare the data points and show how the time grows exponentially as the size, or number of variables, increases. **3. def increment(assignment):** The purpose of this is a binary counter that generates the next possible truth assignment. It starts with the first element of the assignment list and checks if it is 0, if it is then it changes to 1 and returns true meaning it was incremented. Otherwise, if it is 1 it resets to zero and moves to the next element in the list. This happens until a 0 to 1 is found and it returns true, or it goes through all the elements. **4. def check(Wff, Nvars, Nclauses, Assignment):**

11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. **1. Input_bruscinator.txt** I created my own test case. The reason I decided to create my own is because the incremental search takes a very long time, so I have to create my own in order to ensure I would be able to run it in a reasonable amount of time. The set up of the file is each line is a different sample, the first number is the amount of variables, the second is the number of clauses, the third is the literal per clause, and the last is the number of trials each test case with test. My thought process mainly surrounded increasing the number of variables for each sample in order to show through my plot that the more variables we use, the more difficult and longer the problem is, concluding in the fact that based on size and time, the time grows exponentially. Thus, I increment the number of variables per sample. Next, the number of clauses which represent logical clauses like OR. More clauses make the problem more complex and harder to solve, so I varied the amount of clauses to show variance in my collection of samples. Next, the list per clause, for each sample I used 2 which shows a variant of the 2-SAT problem. Lastly, the trials. I kept the trials consistent so each sample had the same chance at satisfiability, and so I could run the program in a timely manner.
12	How you managed the code development **I managed the code development by modularizing. I began by determining how to integrate the incrementer. From there I added on the other necessities, i.e. a function for plotting, and finally a function for parsing input files. This worked for me as I was able to focus on the code little by little ensuring it worked correctly before I moved on.**
13	Detailed discussion of results: **1. Results_bruscinator.csv** **The results file holds a brief summary about each of the test cases. Each line corresponds to a different test case and holds the following information: the problem number, number of variables, number of clauses, the literals per clause, the results (satisfiable or unsatisfiable), the execution time, and the assignment. This is useful because it gives us a concise view of the test case's outcome so we can analyze trends.** **2. Trace_bruscinator.csv** **The trace file is really similar to the results file but it contains more information such as: how many trials were unsatisfiable and satisfiable, the maximum time it took for execution, and the average times it took to execute. This is helpful to provide more detailed insights on how the system works over a number of trial and helps us learn about its performance.** **3. Cnffile_bruscinator.cnf** **The cnf file contains the formulas in Conjunctive Normal Form that are tested per trial. Each line starts with c to describe the problem, literals per clause, and whether it was satisfiable or not. This shows a representation of the formulas used and can be used in other solvers. We are also able to analyze this file to see how different configurations affect the results.**
14	How team was organized **I organized my individual project by first reading through all the necessary files, then familiarizing myself with the provided files. Once I had a basis of**

	understanding for what needed to be done and what was already provided, I was able to integrate the code necessary to fulfill the requirements of the project.
15	What you might do differently if you did the project again **If I were to do this project again, I would probably meet with the professor/TA's earlier in order to gain a good understanding as to if where I was headed was correct. I found myself struggling at some points because I was not sure what exactly was required.**
16	Any additional material: