Project x Readme Team ishin

Version 1 9/11/24

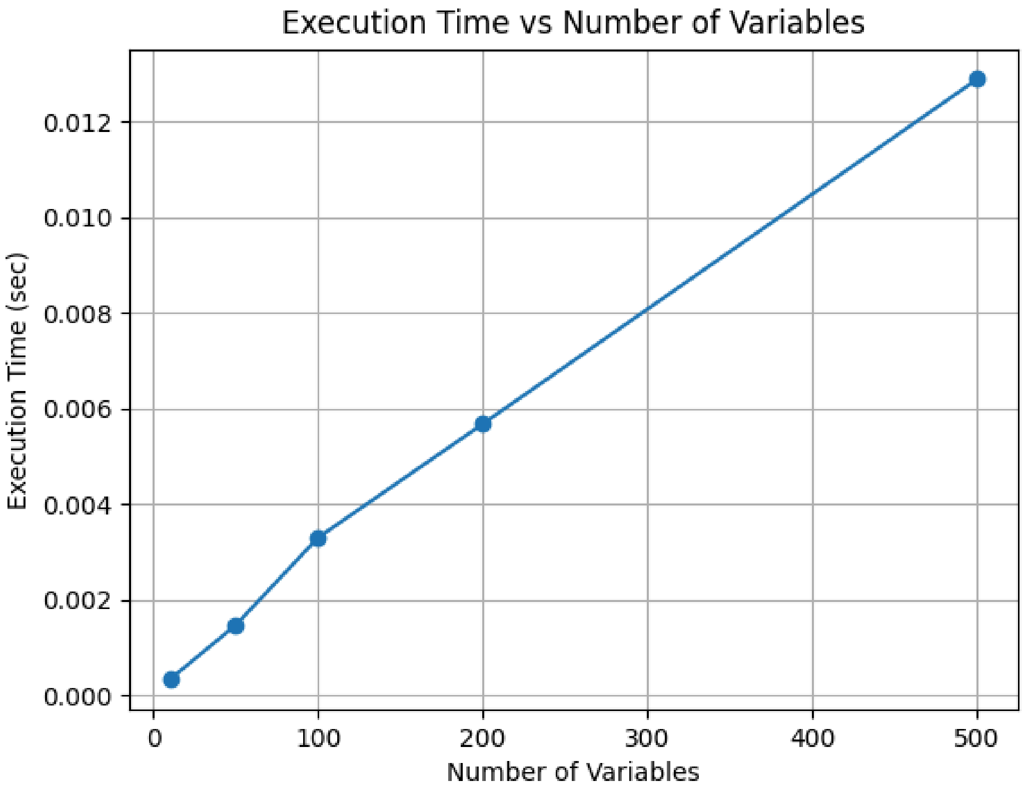
A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_”teamname”

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

<table>
<tr><td>1</td><td>Team Name: ishin</td></tr>
<tr><td>2</td><td>Team members names and netids: Ian Shin, ishin</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: The overall project was to implement a 2-SAT solver using Kosaraju’s algorithm to determine satisfiability and execution time. Subprojects include creating a CNF parser to process input files, developing an implication graph builder, finding SCCs, generating various test cases, creating performance plots, and outputting detailed logs.</td></tr>
<tr><td>4</td><td>Overall success of the project: The process was generally successful. The 2-SAT solver was able to determine the satisfiability over 5 test cases and produced accurate results within expected time limits. The algorithm also worked well for detecting SCCs, and implication graphs were accurately built. The solver’s performance was consistent across various input sizes, as seen in the performance plot.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 9</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/ianshin/ishin-project1-2satsolver</td></tr>
<tr><td>7</td><td>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.
<table>
<tr><td>File/folder Name</td><td>File Contents and Use</td></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>cnf_parser.py</td><td>CNF parser for input files</td></tr>
<tr><td>generate_2sat.py</td><td>Random 2-SAT test case generator in CNF with certain number of variables and clauses</td></tr>
<tr><td>implication_graph.py</td><td>Implication graph builder from CNF clauses</td></tr>
<tr><td>scc.py</td><td>Kosaraju’s algorithm for SCC detection</td></tr>
<tr><td>solver.py</td><td>Main solver implementation</td></tr>
<tr><td colspan="2">Test Files</td></tr>
</table>
</td></tr>
</table>

test_10vars.cnf.csv	2-SAT test case with 10 variables and 30 clauses
test_50vars.cnf.csv	2-SAT test case with 50 variables and 75 clauses
test_100vars.cnf.csv	2-SAT test case with 100 variables and 150 clauses
test_200vars.cnf.csv	2-SAT test case with 200 variables and 300 clauses
test_500vars.cnf.csv	2-SAT test case with 500 variables and 1000 clauses
Output Files	
results.txt	General summary of results for each test file processed by the solver. Provides basic information about parsing success, satisfiability, and execution time.
detailed_output.txt	Detailed information about each test file, including parsed CNF file contents, the implication graph, and the SCCs identified during the solving process
Plots (as needed)	
performance_plot.png	Performance plot graphing execution time vs. number of variables

Attached here if necessary:

	
8	Programming languages used, and associated libraries: Language - Python Libraries - csv, random, collections.defaultdict, matplotlib, matplotlib.pyplot
9	Key data structures (for each sub-project): Implication graph (defaultdict(list)); Stack (list), Visited Set (set), SCCs List (list of lists), Clauses List (list of lists)
10	General operation of code (for each subproject): - CNF Parsing: CNF file read using csv library. Parser processes the problem line and each clause and then converts into integers, storing into list - 2-SAT Generator: Random 2-SAT problems generated by selecting random variables and assigning random signs; clauses are saved in CNF format - Implication graph: For each clause in CNF, program generates implications and stores in directed graph represented using defaultdict(list) - SCC: Uses Kosaraju's algorithm to run a DFS on implication graph, processing nodes in order of completion time, reversing the graph, and then running DFS again to find strongly connected components - Solver: execution time of each test file measured using time.time(). Results are stored and plotted using matplotlib.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code.

	Test cases as stated above in the file names were used to verify if the solver works correctly for small to large number of variables and clauses, scaling the problem size slightly to anticipate differing situations of satisfiability. The tests helped confirm that the code correctly parses the CNF file, builds the implication graph, detects SCC/satisfiability, and spits out the execution time. It also verified that the code remained efficient and accurate, handling larger instances as well.
12	How you managed the code development: Took an incremental approach by implementing the CNF parser first to ensure that the input files could be correctly parsed before moving onto more complex parts such as graph building and satisfiability/SCC checking. After completing each of these, I wrote the generator that would write test cases. Then I wrote the solver (main) file so that it would output text files that stored a summary of the results.
13	Detailed discussion of results: The CNF parser successfully parsed all test files; parsing results go into detail in the detailed_output.txt file. Implication graphs and SCCs were generated correctly, and satisfiability/execution time results were consistent with the expected behavior for each test file. The results confirmed that the algorithm is running in polynomial time – as the number of variables increased, the execution time scaled linearly, which is consistent with the expected behavior for a 2-SAT solver based on Kosaraju's algorithm. This can be verified in both .txt files.
14	How team was organized: N/A
15	What you might do differently if you did the project again: I would like to analyze more diverse test cases, optimize the parser for larger datasets, and develop a graphical representation of the implication graph and SCCs.
16	Any additional material: N/A