Project 1 Readme Team Belugas

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: Belugas</td></tr>
<tr><td>2</td><td>Team members names and netids: Izzy Molnar - imolnar, Emily Norton - enorton3</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: SAT Solver using Brute Force and Backtracking</td></tr>
<tr><td>4</td><td>Overall success of the project: Successful - Our algorithms produced the results we expected for the test cases that we verified by hand.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 17 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/emilyynorton/Project1-TOC</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>src</td><td>sat.py – wrote backtracking & brute force functions</td></tr>
<tr><td>scripts</td><td>timeit_belugas.py – used to run SatSolver, determine the times, and plot them</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>input</td><td>cnffile.cnf – contains 3 SAT problems
data_kSAT2_belugas.cnf – contains 41 SAT problems</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>results</td><td>brute_force_cnffile_sat_solver_results_belugas.csv – results from brute force algorithm on cnffile.cnf file
btracking_cnffile_sat_solver_results_belugas.csv – results from backtracking algorithm on cnffile.cnf file
brute_force_kSAT2_sat_solver_results_belugas.csv – results from brute force algorithm on kSAT2.cnf file</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>btracking_kSAT2_sat_solver_results_belugas.csv – results from backtracking algorithm on kSAT2.cnf file</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>results</td><td>output_cnffile_bruteforce_belugas.png – plot from brute force algorithm on cnffile.cnf file
output_cnffile_backtrack_belugas.png – plot from backtracking algorithm on cnffile.cnf file
output_kSAT2_bruteforce_belugas.png – plot from brute force algorithm on kSAT2.cnf file
output_kSAT2_backtrack_belugas.png – plot from backtracking algorithm on kSAT2.cnf file</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries: Language: Python3, Associated Libraries: matplotlib.pyplot, os, itertools, sys</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
Brute Force: iterator, list (with set to remove duplicates), and dictionary
Backtracking: 2D list and dictionary</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):
Brute Force: Create iterator that produces every possible combination of True/False corresponding to the number of variables in the SAT problem. Loop through combinations of True/False, assign them to variables in the SAT problem, and check if each clause in the CNF expression holds True. If all clauses are True, then the expression holds true and the function returns the assignments for each variable that made the expression true. As soon as one clause is false, the entire expression cannot be true (because of the “and” connecting clauses), so the function moves to the next True/False assignment.

Backtracking: Recursively assigns True/False values to variables and checks whether each clause is already satisfied or potentially satisfiable. If a clause becomes impossible to satisfy, it backtracks and tries a different assignment. If all clauses become satisfied, it returns the assignment.</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the correctness of your code.
We used the kSAT file from the Canvas/Resources/Project Files/SAT folder on canvas to test our code on a variety of SAT expressions containing a different number of variables. Testing cases containing different numbers of variables allowed us to verify the correctness by looking for an exponential curve for the worst case time for unsatisfiable cases (red markers on the graph).</td></tr>
<tr><td>12</td><td>How you managed the code development:
We each created a branch to work on a specific function or portion of the assignment, then we pushed the branch and merged it with main once we finished implementing the given feature.</td></tr>
</table>

13	Detailed discussion of results: For brute force, the time it takes to solve the SAT problem increases exponentially for unsatisfiable problems. The exponential curve follows the form of the line $y = 2^n$, as there are 2 possible values for each of the n-variables (True or False). For the satisfiable expressions, the time to solve each problem also increases greatly as the number of variables increases, but the time slightly varies depending on whether a solution was found early in the process of testing True/False assignments, or later in that process. The unsatisfiable expressions always took as much or more time to run than the satisfiable expressions. For backtracking, the maximum time that it takes to solve a SAT problem generally follows an exponential curve, but there is more variety in terms of the amount of time that it takes to find a solution or deem an expression unsatisfiable. On average, backtracking takes less time than brute force. For example, the SAT expression with 18 variables took over 0.25 seconds to be labeled unsatisfiable by the brute force algorithm, but only took around 0.001 seconds to be labeled unsatisfiable by the backtracking algorithm. Backtracking is able to determine the solutions for CNF expressions more quickly because it reduces the search space by pruning invalid branches early, rather than testing every possible solution.
14	How team was organized: We collaborated throughout the entire duration of the project, but each primarily focused on implementing one of the functions. Izzy focused on backtracking, and writing the script to record the time and number of variables for each expression and plot them. Emily focused on the brute force function and managing the Github repository.
15	What you might do differently if you did the project again: We would structure our Github better. We should be more intentional in creating branches and ensure that each branch has a descriptive name so that we can keep track of our work. This would be especially helpful in situations where we need to reference previous versions of our work.
16	Any additional material: N/A