Project 1 Readme Team msulli52

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: msulli52</td></tr>
<tr><td>2</td><td>Team members names and netids: Molly Sullivan, msulli52</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: The "Knapsack" Problem: An equivalent of DumbSAT for a solver that returns a coin combination that works</td></tr>
<tr><td>4</td><td>Overall success of the project: My project was successful.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 14 hours (I worked on this over several days, restarting a few times because of misunderstandings leading me in the wrong direction)</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/msulli52/theory_project1_msulli52</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>generate_coin_results_msulli52.py</td><td>This Python file stores the main algorithm to solve the coin knapsack satisfiability problem. When you run it, it takes output_test_cases_msulli52.csv as input data and then determines the execution time and satisfiability of each test case, storing the output in output_results_msulli52.csv.</td></tr>
<tr><td>plot_results_msulli52.py</td><td>This Python file uses matplotlib and numpy to graph the data in output_results_msulli52.csv. It compares the total number of coins in the knapsack vs time and also reveals the satisfiability of</td></tr>
</table>
</td></tr>
</table>

	each test case. After running the code, an example of a graph result is in output_plot_timevsnumcoins_msulli52.png.
Test Files	
test_case_generator_msulli52.py	**This Python file generates a csv file of test cases to run. It creates random target values and knapsack sizes. The numbers are adjustable depending on how much data you want. Right now I have it set up to create 600 tests, 40 different cases for each knapsack size 1-15. When you run this file, it puts all the data to output_test_cases_msulli52.csv. Each time you run it you will get new random test cases.**
Output Files	
output_test_cases_msulli52.csv **output_results_msulli52.csv**	**These are two csv files that store the output data from the two above Python files. Output_test_cases_msulli52.csv stores the output from test_case_generator_msulli52.py. The columns are target value (in cents), knapsack count (number of 3, 11, 23-cent coins), and total number of coins in the knapsack. Output_results_msulli52.csv stores the output from generate_coin_results_msulli52.py. This stores the execution time and satisfiability needed for plotting, among other data, in the following columns: target value (in cents), knapsack count (number of 3, 11, 23-cent coins), total number of coins in the knapsack, execution time (in microseconds), the coin combination that sums to the target (if found), and the satisfiability of every test case.**

<table>
<tr><td></td><td>
<table>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>output_plot_timevsnumcoins_msulli52.png</td><td>This is a screenshot that shows my graph output that plots the data from output_results_msulli52.csv after running plot_results_msulli52.py. It shows the total number of coins in the knapsack vs execution time in microseconds. It demonstrates a 2^n time complexity pattern as the data gets larger. The green points are satisfiable cases, and the red points are unsatisfiable cases.</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
Python, csv, random, time, matplotlib.pyplot, numpy, curve_fit</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
When generating the random test cases in test_case_generator_msulli52.py, I used lists for coins (to store the values of each type of coin) and knapsack (to store all the coins that are in the knapsack).

Then in generate_coin_results_msulli52.py, I used a list to store the different coin combinations that could be generated to reach the target. A parameter called coins was also a list that I passed into my algorithm to find the combinations of those coins. I also used a csv reader to read from and write into a csv file.

Finally, in plot_results_msulli52.py, I used many lists as well: total_coins, execution_times, colors, satisfiable_points, unsatisfiable_points, and indices. I used numpy arrays to store the number of unique coins as well as max execution times for finding the boundary curve to plot. These were all used to store the values of all the data points that I plotted on my graph. I also used a csv reader to read from a csv file.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):
First, in test_case_generator_msulli52.py, I create random test cases to run through my coin combination algorithm. It distributes a random number of coins of 3, 11, and 23 cent values in each knapsack, which also has a random total number of coins. It creates a csv file (output_test_cases_msulli52.csv) with random target values, coin distribution, and total number of coins in the knapsack.

Second, in generate_coin_results_msulli52.py, I take in the test cases and try to find if there is a coin combination that will add to the target value given the random number of 3, 11, and 23-cent coins in the knapsack. My algorithm finds all combinations/subsets of the coins in the knapsack and then checks to see if the</td></tr>
</table>

	sum of the combination equals the target, meaning a valid combination is found and it is satisfiable. After running the test cases, an output_results_msulli52.csv file is created with the target value, knapsack values, execution time, combination, and satisfiability, which are used for plotting. **Finally, in plot_results_msulli52.py, I read the results csv file and use matplotlib and numpy to graph a plot that shows the total number of coins in the knapsack vs execution time in microseconds. The points are color-coded by satisfiability and the graph features an exponential 2^n shape. There is also a dotted boundary line to show the shape of the curve.**
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code: **I created my own random test cases for my project. I decided to make my own to really understand how the test cases would be created and used as input for my algorithm. I essentially create knapsacks with a random total number of coins (with randomly assigned amounts of 3, 11, and 23-cent coins in them). I also create a random target value. This information is stored in a csv file and used as input for my algorithm to find the satisfiable coin combinations, if any. Using these test cases allowed me to have a wide range of random data to show how execution time grows at an exponential 2^n rate as knapsack size increases. The random numbers help give my data a diversity of inputs to create a valid output graph.**
12	How you managed the code development: **I initially picked this project because in Programming Challenges we are learning about memoization and dynamic programming and one of the main examples we use is choosing coins to satisfy a target amount or find the minimum number of coins to reach a target. I liked exploring this CS problem and felt I had enough background information on different solving algorithms to start. I started out by creating a few different algorithms (brute force with for loops, depth-first search, and dynamic programming) and got feedback that I was kind of going in the wrong direction. I originally spent a lot of time working on these algorithms that had different time complexities (not consistently 2^n worst case) so I spent a lot of time troubleshooting my incorrect graph results until I finally found the algorithm that worked (brute force using exhaustive binary subset generation). I decided to create my own test cases to have more control and understanding of my inputs and outputs. Once I was sure my data output made sense, I finalized my plot with satisfiability coloring and a boundary line to show the shape. I was able to manage everything myself.**
13	Detailed discussion of results: **My results are shown in a graph of total number of coins (x-axis) vs executive time in microseconds (y-axis). It shows how time to find a coin combination to reach a target value exponentially increases based on the knapsack size. My test cases yield a variety of data points to fully show the 2^n behavior that this relation follows as the knapsack size grows as well as showcase satisfiability performance. We can see that as the data set gets bigger, the problem becomes more challenging and takes more time. More detailed information about values of the test cases and outputs are stored in csv files to complement the visual**

	representation of the graph.
14	How team was organized: **I worked by myself on this project, which I think worked well for me and my understanding. I had a lot of "aha" moments throughout the process which was really helpful for me to encounter on my own.**
15	What you might do differently if you did the project again: **I would have asked more questions and done more thinking and research about the problem and the overall goal of the project earlier before jumping into writing the algorithms. At first, I immediately started writing a few different algorithms to compare because I didn't really understand the direction of the project. I probably spent too many hours going down the wrong path before checking with Professor Kogge to get feedback on my work. Once, I had my aha moment on the correct approach, it was much smoother. I also started out by using one of the provided test generators, which I didn't fully understand how to integrate since I didn't make it myself. It was helpful for me to create my own test case database to make manipulating it simpler for me.**
16	Any additional material: **None**