readme_gflesher

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: gflesher</td></tr>
<tr><td>2</td><td>Team members names and netids: Grant Flesher - gflesher</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: I attempted the 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: I completed the program! My program was able to come up with combinations for varying outputs!</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 7 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/grantflesher/-coin_combination-</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>coincombination_gflesher.py</td><td>This file is my project code. It takes input of a list of coin denominations, and a goal amount. It outputs if there exists a possible combination of the coin denominations that exist that add up to the goal amount.</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>Test_files (foulder)</td><td>The text files include inputs that showcase varying outputs. They show the edge cases working, and varying</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>coin amounts succeeding.</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>Time_files (foulder)</td><td>50 output files were created to give points to put on a graph. There inputs and outputs for each output file is in this folder.</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>plots_gflesher.png</td><td>This shows a plot of the times. It showed how while run times could stay short no matter how many coins, as the number of coins increased, the number of longer run times increased.</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries: Python (sys, time)</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project): The Key Data structure I used was a list. I parsed user input and made it into a list in descending order to prioritize larger coins over smaller ones.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):

The program recursively searches for a combination of given coins that adds up to the goal. The largest coin will check the maximum possible number of coins that can be used, and search that first. If not possible, it will then check the second most and so on. It recursively explores all valid combinations by decrementing the coin count in each recursive call. If there is no combination found, it will output such that no output was found</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.

I added test cases for when the goal was zero, and I added a test case for when more than 5 coins were added. This was done to prevent the program from ever doing more than five coins. The zero test case was added, because the recursive algorithm would not catch it. I added these to fix edge cases in my code so my output would be better. By adding these test cases, i was able to</td></tr>
<tr><td>12</td><td>How you managed the code development: I started by handling the input for the goal</td></tr>
</table>

	and the coins, and then making the coins input into a sorted list. From there, I implemented a recursive search inspired by my time in Bui's class. I tested the recursive function multiple times before I submitted.
13	Detailed discussion of results: My project is a good example of how projects can grow exponentially complex! While it was always possible for any number of coins to have a short run time, the more coins that were provided, the greater the longer run times were possible. I believe this is because my program is of n^k complexity, with k being the number of coins provided in the input. If i was to go back and do it again, I would provide even more data for the graph, to see more of how the complexity gets exponentially complex.
14	How team was organized: solo.
15	What you might do differently if you did the project again: Not be sick!
16	Any additional material: This helps to this day: https://www3.nd.edu/~pbui/teaching/cse.20312.fa23/