Readme_gflesher

Version 1 8/22/24

1. Team name:
    a. gflesher
2. Names of all team members:
    a. Grant Flesher
3. Link to github repository:
    a. https://github.com/grantflesher/-coin_combination-
4. Which project options were attempted:
    a. An equivalent of DumbSAT for a solver that returns a coin combination that works
5. Approximately total time spent on project:
    a. 6/7 hours
6. The language you used, and a list of libraries you invoked.
    a. Python (sys, time)
7. How would a TA run your program (did you provide a script to run a test case?)
    a. Utilize input redirection. For example:
        i. python3 coincombination_gflesher.py < test_files/check1_gflesher.txt
8. A brief description of the key data structures you used, and how the program functioned.
    a. 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.
    b. 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
9. A discussion as to what test cases you added and why you decided to add them (what did they tell you about the correctness of your code). Where did the data come from? (course website, handcrafted, a data generator, other)
    a. 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 for 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. I generated the data myself. I added multiple of each number of coins.
10. An analysis of the results, such as if timings were called for, which plots showed what? What was the approximate complexity of your program?
    a. The plot showed, while each number of coins could have quick solutions, the runs got longer as more coins were added. I think the program got exponentially more complex with each coin added to the program.
11. A description of how you managed the code development and testing.

a. I started by handling the input for the goal 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.

12. Did you do any extra programs, or attempted any extra test cases

- I checked for when the goal was zero. This made the program output there was no solution, when there was a solution, the solution was nothing!