Readme_teamkyle

Version 1 10/16/24

1. Team name: Team Kyle (teamkyle)
2. Names of all team members: Kyle Crosby

3. Link to github repository: https://github.com/kylecrosby2/Project1TeamKyle

4. Which project options were attempted: Knapsack problem – “An equivalent of DumbSAT for a solver that returns a coin combination that works”

5. Approximately total time spent on project: 9 hours

6. The language you used, and a list of libraries you invoked: Python. Libraries: time, csv, os, pandas, matplotlib, numpy

7. How would a TA run your program (did you provide a script to run a test case?):

   If the input data does not already exist, run test_caseGen_teamkyle.py to generate all of the input files (make sure to first create a directory in your cwd called “input”). Next, generate the CSV output by running dumbSatKnapsack_teamkyle.py (make sure to first create a directory in your cwd called “output”). Now, a CSV file named output_teamkyle.csv is in your output directory. Finally, run makePlot_teamkyle.csv to generate a plot from the data in the output CSV. The plot will be created using matplotlib.

8. A brief description of the key data structures you used, and how the program functioned.

   The Python program works by taking an input of a target value for the coins to sum to, the jar of coins (list of coins, likely of different denominations, generated by the test case file), and the start time of execution. The function then iteratively finds every combination of these coins using nested for loops, and stores found combinations in a nested list. Every time a new combination is found, the sum is checked and if it equals the target value, the for loop breaks and returns the found combination. Otherwise, it continues through every combination and returns an empty list if no satisfiable combination is found. Execution time is calculated at completion, and it is written to an output CSV file along with the number of variables, working combination (or empty list), and target value, for that test case.

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)

   My test data came from an edited version of the provided knapsack test generator. I

adjusted the test generator to loop 20 times, each time creating a new data file and changing the number of variables. This created 20 separate data text files. Each had 100 tests of a certain number of variables. I verified that my combination algorithm worked correctly for several simple test cases (e.g [1, 2, 3]). Due to the simplicity of finding combinations, I did not see it necessary to create an entire test file to verify the output. The algorithm simply checks if the sum of any combination is ever equivalent to the target.

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?

    The results of my program indicated an increase in execution time as the number of variables (aka number of coins in the jar) increased. As seen in my plot, a bounded curve is fit to the scatter plot points of execution time vs. number of variables. The bounded curve's equation is y = 46 + 0.3*2^x, indicating an exponential increase in execution time. The approximate complexity of my program is O(n*2^n), where n is the number of variables (or "coins in the jar"). In the worst case (unsatisfiable), the program must loop through every combination of coins in the jar, of which there are 2^n. Additionally, the program must add every combination, which has O(n) complexity, thus the final complexity is O(n*2^n). However, it is worth noting that there are more efficient ways to keep track of the sum of each combination such that summing each iteration is unnecessary, so the most important attribute of this program's complexity is the O(2^n) time to loop through all combinations of coins in the worst case.

11. A description of how you managed the code development and testing.

    All code was developed locally in VSCode. I named output files iteratively to track the history of outputs. I tested by adjusting the test data generation parameters to run within a reasonable time on my algorithm. I then fed the output CSV into my plotting function to verify the timing worked correctly.

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

    No.