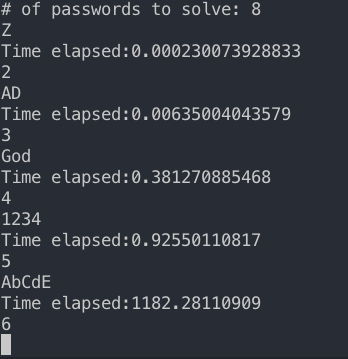
***ITP Final Project – Python MD5 password cracker analysis***

***NOTE:*** Run the program by cdíng the right directory and typing python bruteforce.py

Here were the passwords the program was able to crack:



Explanation and analysis of the algorithm:

Since it was an un-optimized brute force algorithm that checks the passwords by hashing a list of permutations of a set of string from length 1-8 against the 8 hashed passwords, the algorithm is really inefficient. (Most definitely more than polynomial time, my best guess is that it takes O(n!))

***Password length v. Time Analysis***

For length 1 passwords, it usually takes 0.2 milliseconds for the computer to go through every possible character in the given string, hash it and check it against the pre-hashed passwords.

For length 2 passwords, it usually takes around 6 milliseconds for the computer to go through every possible 2-permutation of a string, hash it and check it against the pre-hashed passwords.

For length 3 passwords, it usually takes around 0.4 second for the computer to go through every possible 3-permutation of a string, hash it and check it against the pre-hashed passwords.

For length 4 passwords, it usually takes around 1 second for the computer to go through every possible 4-permutation of a string, hash it and check it against the pre-hashed passwords.

For length 5 passwords, it usually takes around 1182 second for the computer to go through every possible 5-permutation of a string, hash it and check it against the pre-hashed passwords.

However, once I scramble the order of the string that itertool takes in to generate permutations, the time needed to generate a permutation of 1234 takes approximately 50 seconds. Hence, order of the given string also matters.

Here’s the analysis anyways:

Since it takes the computer 1 second to check a search space of 62^4 permutations but 1182 second to check a search space of 62ˆ5, the analysis is hard to perform as the search time grows either exponentially or by factorials.

To get a rough estimate, I’m using the same guess of computer taking approximately 1 second to go through a search space of 9.5^7. (62^4 / 9.5^7 = 2.12 seconds)

Assuming that it takes 1 second for the computer to search for 9.5^7 passwords and hashing them, a length 5 password with a 62^5 search space would require approximately 131 seconds.

This here is wildly inconsistent with the actual result, but the guestimate continues:

A length 6 password with a 62^6 search space would require approximately 8133 seconds or 2.26 hours.

A length 7 password with a 62^7 search space would require approximately 140 hours.

A length 8 password with a 62^8 search space would require approximately 8685 hours or 361 days.

If the grader’s computer can set up a high tier AWS EC2 instance and run the script on the virtual machine, the grader can test this hypothesis.

***Proposal for optimizing the algorithm***

1. Pre-generate a list containing every possible combination of characters from length 1-8 and check the 5 hashes against the list. The runtime would be reduced to only the time required to find the correct hash value in a list -> O(n). The caveat is that the list is going to be very large as a length 8 password has a 62ˆ8 search space.
2. I should also create a varying set of possible characters based on the user’s needs. If the password doesn’t require a special character such as @, I would only generate the possible combinations for a-z,0-9 & A-Z. If it requires special characters, I would run it against a different list containing special characters.
3. I could also split up the task of generating permutations and distribute it to multiple processes with python’s multiprocessing module’s Pool class.
4. Running the algorithm, after optimized, on a AWS EC2 high tier instance would also be an option. Preferably, the code will be running on a multicore instance after implementing the above suggestions.

<https://stackoverflow.com/questions/14092125/python-permutations-threads>