Jack Stoetzel & Lauren Golian

Lab 4: Password Cracking

How to Run This Program

To run this program, use the commands

```
mpicc working.c -lcrypt
mpiexec -n <N> a.out
```

Where is the number of nodes to run the program on.

What Does This Program Do?

This program brute forces through a list of passwords by encrypting words from a dictoinary with numbers attached to them.
The program uses MPI to divide the computations amongst multiple nodes so that the work can be done quicker.
Currently the encrypted passwords are hard coded to the program and the words.txt file is read in sequentially.
Once the words are read in, half the nodes check encrypted words with numbers at the begining, and the other half with numbers at the end.
If a password is discovered, the username and their encrypted password is printed to the screen.

Questions

What is the theoretical time complexity of your algorithms (best and worst case), in terms of the input size?

The current version only operates with 3 numbers on either side of the passwords.
So every word is calculated 2002 times over.

- Once for reading in the word

- Once for calulating without numbers
- And the 2000 for numbers being attached to both sides (1000 numbers on the front, and 1000 numbers on the back).

Therefore, the time complexity of the algorithm in best case and worst case respectively is: O(n*2002), O((n * size of dicitonary * 2002)/# of processors).

According to the data, does adding more nodes perfectly divide the time taken by the program?

No, adding more nodes does not perfectly divide the time of the program.
Currently the program divides works in the following way:

- The number of nodes is subtracted by two if there is an odd number, or subtracted by two if it is an even number.
  - These one or two nodes will be used to compute words without any number attached to them. Ex) password
- The remaining nodes are then divided in half.
  - Even nodes will be used to compute words with numbers attached to the end. Ex) password123
  - Odd nodes will be used to compute words with numbers attached to the begining. Ex) 123password

So with large sets of nodes, the program runs very inefficiently since the encryption of words with no numbers will take significantly longer.

What are some real-world software examples that would need the above routines? Why? Would they benefit greatly from using your distributed code?

Off the top of my head, the only two practical uses of this software are:

- Hackers who want to use this code to brute force peoples passwords to access other poeple's accounts.
- A company might want to use this to test if any of their employees have weak passwords on their accounts.

This code is very computation heavy.
For anyone to use this and get results in a reasonable amount of time, the work would need to be distibuted amongst MANY processes.

How could the code be improved in terms of usability, efficiency, and robustness?

As stated in the second question, there is a lot of inefficency with how the work is divided.
To improve this, more nodes could be allocated to computing words with no numbers attached.
Though that would take away from the computations on words with numbers attached, there could be a formula to determine an optimal amount to allocate for each operation so that all three parts finish in relativley the same amount of time.

This could also be improved in with more effcient file reading.
Currently, it takes 13s to read from the words.txt file in serial.
After many attempts to read the words file in parallel with MPI, we decided to read in serial.
If this was done in parallel, that would significantly increase the overall speed of the program.

Another change to increase usablitity would be to read from a shadow file.
For the sake of time, we hard coded the salt and the passwords in the shadow file.
If this was read from an actual shadow file, the usability could universal for any shadow file.

Output

```
Dictionary has been read in.

bob      $1$ab$0I4CGceZU1wOu9PO3qn2p/ == fluffy
maria    $1$ab$DC1ifnjzzu8Za5qEt96Kq0 == password123
alice    $1$ab$DqPMoPPboZ2HTH7RfiqQs. == 99puppy
rmshifler        $1$ab$/rOLL5LFn/3ZIa2TFnN4G. == 1genius
jtanderson       $1$ab$Po1AuQSRCorWXHi8cIOhK/ == 100nodes
george   $1$ab$tYl84YhDM6bmCOuCusTKS. == red32
```