Jay Offerdahl
EECS 565
Project 2 Report

Cracking the Vigenere Cipher

Introduction:

Writing this program was at first very simple. I began by creating my own hash table class, modifying the vigenere cipher I implemented in project one, and setting up several instances of nested for loops in order to calculate every possible key, decrypt the text, and check it against the dictionary that was stored in a hash table. This solution worked, but took a very long time (which I thought was acceptable at the time). I then discussed strategies with a few of my colleagues, and made some significant modifications to the program which greatly reduced the cracking time.

In a general discussion, this **lab demonstrated just how insecure the vigenere cipher is**. If I wanted to brute force the key and first word, all I would need to do it check keys for the first word lengths, which might be up to, say, 15. The keys could be anywhere from 0 – n, so I could pick an N and simple check all of those values. It would take much longer, but it's far from infeasible.

Results:

The initial results of the program I wrote, which used several nested for loops to calculate every key and decrypt the text with every possible key in order to check against the dictionary.

Initial Results (Poor efficiency)
Test #1: Key length of 2, first word length of 6
Possible key: KS --> output: CAESARSWIFEMUSTBEABOVESUSPICION
Time elapsed for #1: 0.001003 seconds

Test #2: Key length of 3, first word length of 7
Possible key: JAY --> output:
FORTUNEWHICHHASAGREATDEALOFPOWERINOTHERMATTERSBUTESPECIALLYINWARCANBRINGABOUT
GREATCHANGESINASITUATIONTHROUGHVERYSLIGHTFORCES
Time elapsed for #2: 0.007019 seconds

Test #3: Key length of 4, first word length of 10
Possible key: IWKD --> output: EXPERIENCEISTHETEACHEROFALLTHINGS
Time elapsed for #3: 0.158014 seconds

Test #4: Key length of 5, first word length of 11
Possible key: ZIENF --> output: IMAGINATIONISMOREIMPORTANTTHANKNOWLEDGE
Time elapsed for #4: 3.951686 seconds

Test #5: Key length of 6, first word length of 9
Possible key: HACKER --> output:
EDUCATIONISWHATREMAINSAFTERONEHASFORGOTTENWHATONEHASLEARNEDINSCHOOL
Time elapsed for #5: 147.139390 seconds

Test #6: Key length of 7, first word length of 13
Possible key: NICHOLS --> output: INTELLECTUALSSOLVEPROBLEMSGENIUSESPREVENTTHEM

Time elapsed for #6: 4877.133042 seconds **// 1 hour, 21 minutes, 28 seconds**

Next, after modifying the program to first build a list of valid keys and then only decrypt the full text for valid keys. Again, this change significantly reduced cracking time, which is shown below. For a discussion on efficiency, see the efficiency section below.

Initial Results (Maximum efficiency)

```
**********************************************************************************

                                    IamAHacker.exe

**********************************************************************************
Starting whole program timer...
**********************************************************************************
Testing key length: 2, and first word length: 6
Possible key: KS --> output: CAESARSWIFEMUSTBEABOVESUSPICION
Time elapsed: 0.000000 seconds

Testing key length: 3, and first word length: 7
Possible key: JAY --> output:
FORTUNEWHICHHASAGREATDEALOFPOWERINOTHERMATTERSBUTESPECIALLYINWARCANBRINGABOUT
GREATCHANGESINASITUATIONTHROUGHVERYSLIGHTFORCES
Time elapsed: 0.015649 seconds

Testing key length: 4, and first word length: 10
Possible key: IWKD --> output: EXPERIENCEISTHETEACHEROFALLTHINGS
Time elapsed: 0.015654 seconds

Testing key length: 5, and first word length: 11
Possible key: ZIENF --> output: IMAGINATIONISMOREIMPORTANTTHANKNOWLEDGE
Time elapsed: 0.046911 seconds

Testing key length: 6, and first word length: 9
Possible key: HACKER --> output:
EDUCATIONISWHATREMAINSAFTERONEHASFORGOTTENWHATONEHASLEARNEDINSCHOOL
Time elapsed: 0.215776 seconds

Testing key length: 7, and first word length: 13
Possible key: NICHOLS --> output: INTELLECTUALSSOLVEPROBLEMSGENIUSESPREVENTTHEM
Time elapsed: 0.099000 seconds

**********************************************************************************
Total cracking time: 0.392990 seconds
**********************************************************************************
Entire program length: 0.768873 seconds
**********************************************************************************
```

Jay Offerdahl
EECS 565
Project 2 Report

Efficiency:

Looking at the original program that I created, it was the true brute force method. I generated literally every key possible given the key size (26^2 for key lengths of 2, 26^7 for key lengths of 7). This clearly shows an ***exponential relationship***, every time the key length increased by one, the time to crack it didn't increase linearly, it increased exponentially. Next, I decrypted the first X letters of the string, where X = firstWordLength. If I found a match, I printed out a statement that showed the possible key and the output of the entire statement decrypted. This was highly inefficient all key sizes, even though the time to crack was low for the first few.

Before I discuss the new algorithm, it's worth mentioning the steps I took to make the HashTable and VigenereCipher programs as efficient as possible.

- Stored consistently accessed variables in memory
    - Instead of calling string.length() in my hash function, I store the string's length in an integer value in memory, and call that. This improved performance significantly (which showed more improvement in the earlier version).
    - In my hash function, I calculate the hash as the sum of the squares of the characters of the string, so instead of calling string.at(x) twice, I store it in an integer and then calculated the stored value * the stored value, which also showed an improvement.
- Tightened up the decrypt function
    - I didn't normalize the input because I only feed it strings with all capital letters.
    - I removed any redundant code in order to spend as little time parsing the string.

After implementing the new algorithm, I reduced times to crack by many orders of magnitude, especially at the key length of 7, which was ***49,264 times faster*** using the numbers I've given here. The difference in algorithms essentially comes down to building keys on the fly and keeping valid keys (as the loop progresses) in a list, which was constantly checked until the list was exhausted of invalid keys. This improvement allowed me to get my program under one second when calculating all six values.

It's worth mentioning that the times here were calculated using the bash command prompt on my computer. This method was about twice as fast (on the improved code) than using the standard command prompt. I have no idea why this is the case, but it allowed me to say I got this in under a second, so I'll take it.

Jay Offerdahl
EECS 565
Project 2 Report

Final Results & Conclusion:

Password cracking can take a long time, but only if the algorithm is solid. Again, this project demonstrated that the vigenere cipher is not secure at all, and can be easily broken using a performance based algorithm. The final results that I was able to generate (with formatting) are below:

1. Caesar's wife must be above suspicion.
2. Fortune which has a great deal of power in other matters but especially in war can bring about great change in a situation through very slight forces.
3. Experience is the teacher of all things.
4. Imagination is more important than knowledge.
5. Education is what remains after one has forgotten what one has learned in school.
6. Intellectuals solve problems, geniuses prevent them.

I'm confident in these outputs are correct after extensively testing this program. I had a great time programming this lab, and I enjoyed the technologies that I was exposed to while working through it. Thanks for reading this far, and as always, any questions can be directed to jayofferdahl@ku.edu.