Roxanne Calderon

2613650

2/26/2015

Miniproject 2

Results:
1. "MSOKKJCOSXOEEKDTOSLGFWCMCHSUSGX";
key length = 2; firstWordLength = 6
Result:
KS: CAESARSWIFEMUSTBEABOVESUSPICION
Time: 208.281 ms

2."OOPCULNWFRCFQAQJGPNARMEYUODYOUNRGWORQEPVARCEPBBSCEQYEARAJUYGWWYACYWBP
RNEJBMDTEAEYCCFJNENSGWAQRTSJTGXNRQRMDGFEEPHSJRGFCFMACCB"
keyLength=3; firstWordLength = 7
Result:
JAY:
FORTUNEWHICHHASAGREATDEALOFPOWERINOTHERMATTERSBUTESPECIALLYINWARC
ANBRINGABOUTGREATCHANGESINASITUATIONTHROUGHVERYSLIGHTFORCES
Time: 5605.86 ms

3. "MTZHZEOQKASVBDOWMWMKMNYIIHVWPEXJA"
keyLength=4; firstWordLength = 10
Result:
IWKD: EXPERIENCEISTHETEACHEROFALLTHINGS
Time: 132329 ms

4. "HUETNMIXVTMQWZTQMMZUNZXNSSBLNSJVSJQDLKR"
keyLength=5; firstWordLength = 11
Result:
ZIENF: IMAGINATIONISMOREIMPORTANTTHANKNOWLEDGE
Time: 2.61088e+06 ms

As you are incrementing by 27^n each time, it is no wonder that there is a rapid rise the larger the key becomes. While the first trials were quite fast and appeared efficient. However, once trial 3 ran, in around 2:30, it was apparent the time it was taking was increasing rapidly. The 4^{th} trial took a bit under 45 minutes and if trial 6 was run it would take an estimated 18 hours to complete. While the rest of the program is quite efficient, iterating through all the possible values that could be generated by a key take up a very large amount of time. As such, to make it as efficient as possible, the amount that for loops occurred were limited, keeping 1 for loop inside an if statement to be only run when absolutely necessary. In addition, the code was changed from Java to C++ to make it more efficient

overall. If the efficiency was to be improved, the best way to do so would likely be to clean up the way that that key strings are generated.