CS 327 Spring 2018
Programming Assignment 1

Introduction

For this assignment you will write a utility for encrypting, decrypting, and deciphering ASCII text documents using an affine cipher.

An *affine cipher* is a slight generalization of the shift ciphers we discussed in chapter 6. For an affine cipher we select two integers a and b as our key. Since we will deal with ASCII characters only, each character of our input text is represented by a number between 0 and 127. For example, the lower case letter 'a' is represented by the number 97. As with the Caesar and general shift ciphers, we encrypt our message character by character. Given a character of our plaintext message m, the encryption function $E(m, a, b) = (am + b) \bmod 128$. Here a is the *multiplier* and b is the *increment*.

Using what you have learned about modular arithmetic, you ought to be able to come up with a decryption function $D(c, a, b) = m$ where $c = E(m, a, b)$. You should consider the following questions.

- Are all choices of integers (a, b) valid keys? In other words, does $D(c, a, b)$ exist for all possible choices of (a, b)? If not, what restrictions must be placed on a and b to ensure that the decryption function exists? (**Hint**: You are going to need multiplicative inverses for the multiplier modulo 128—so what consequences does that have for the multiplier?)
- What is a reasonable upper bound on the number of different key pairs (a, b) that can be used to encrypt ASCII text? (**Hint**: remember that when you are doing modular arithmetic with modulus 128, you are really working with the residue classes $[0]_{128}, [1]_{128}, \ldots [127]_{128}$.) Given such an upper bound, what do we need to process in breaking an affine cypher?

The Extended Euclidean Algorithm

As noted in class, this is a tough little program to write. Here is an implementation in Ruby that you can use in your program if you like (suitably modified for your language).

```
# Find integers s and t such that gcd(a,b) = s*a + t*b
# pre: a,b >= 0
# post: return gcd(a,b), s, t
def egcd(a, b)
  # let A, B = a, b
  s, t, u, v = 1, 0, 0, 1
  while 0 < b
    # loop invariant: a = sA + tB and b = uA + vB and gcd(a,b) = gcd(A,B)
    q = a/b
    a, b, s, t, u, v = b, (a%b), u, v, (s-u*q), (t-v*q)
  end
  [a, s, t]
end
```

Program Specifications

For this assignment you will write a command line utility called `affine` with the following functionality (if you use a scripting language, then your utility may include an appropriate extension, such as `affine.py`).

1. The utility must be invoked from the command line to encrypt, decrypt, or decipher plaintext files containing ASCII text (which may include symbols, various sorts of whitespace, and so forth—in short, any ASCII characters).

2. The utility must accept a first command line parameter that is either `encrypt`, `decrypt`, or `decipher`.

3. The utility must accept a second command line parameter that is a path to an ASCII input file. For encrypting this file will be a plain text file, while for decrypting it will be a cipher text file.

4. The utility must accepts a third command line parameter that is a path to an ASCII output file (that the utility will write). For encrypting the file will be a cipher file, while for decrypting or deciphering it will be a plain text file.

5. The utility must accept a fourth and fifth parameter if its first parameter is either `encrypt` or `decrypt`. These parameters will consist of integers `a` and `b` designating the affine cipher multiplier and increment parameters, respectively.

6. The utility must accept a fourth parameter if its first parameter is `decipher`. This parameter is a pathname to a dictionary file used to break the affine cypher.

7. If the first parameter to the utility is `encrypt` or `decrypt`, and the pair of integers (`a`, `b`) is a valid encryption key, the utility must encrypt or decrypt the text in the input file and write the result to the output file.

8. If the first parameter to the utility is `encrypt` or `decrypt`, and the pair of integers (`a`, `b`) is not a valid encryption key, the utility must neither read an input file nor write an output file, but instead must print the following message on the standard output stream (with the user-supplied values for `a` and `b` in place of `<a>` and `<b>`).

   ```
   The key pair (<a>, <b>) is invalid, please select another key.
   ```

9. If the first parameter to the utility is `decipher`, then the utility must read the input file and dictionary file and attempt to decrypt the input file by trying each valid pair (`a`, `b`) as a decryption key for an affine cypher. The utility must break the resulting plain text into words and then count the number of valid words in the plain text. A word is valid if it is contained in the dictionary. The plain text with the most valid words must be considered the deciphered text. The utility must write the following data to its output file.
   - The the key pair used to decrypt the deciphered text (`a`, `b`) in the format "`<a> <b>`" on the first line of the output file. In other words, `a` and `b` appear as the first line of the output file, separated by a space.
   - A line with the text "`DECRYPTED MESSAGE:`". In other words, these words are the second

line of the output file.
- The decrypted message, occupying as many lines as it needs.

10. The dictionary file must contain words, one per line, in lowercase.

Example

Suppose the following text appears in a the file `plain.txt`.

This is a simple message to test the affine utility.
It has three lines that include a few number like 3.1416 and 2.1718.
It should easily be deciphered by the affine utility.

A command line like the following (which is for the utility written in Ruby) should encrypt this text using the key (29,34) and place it in `cypher.txt`.

```
ruby affine.rb encrypt plain.txt cipher.txt 29 34
```

Running the following command should decrypt the file and place the result in `tmp.txt`. The `tmp.txt` file should be identical at every character to `plain.txt`.

```
ruby affine.rb decrypt cipher.txt tmp.txt 29 34
```

Finally, running the following command should also write the file `tmp.txt`. The dictionary file is `words.txt`.

```
ruby affine.rb decipher cipher.txt tmp.txt words.txt
```

The contents of `tmp.txt` after this command should be the following.

29 34
DECRYPTED MESSAGE:
This is a simple message to test the affine utility.
It has three lines that include a few number like 3.14 and 2.17.
It should easily be deciphered by the affine utility.

Hints

Test your decipher code on text files with at least ten words with at least five characters in them. Deciphering is more error-prone on shorter texts with short words due to false positives (decrypted words that appear when the wrong key is used).

You may want to strip symbols out of your decrypted text to aid lookups (not necessary ut perhaps useful). You may also want to lowercase everything (probably necessary).

You may want to ignore words that are too short. For instance, your decipher utility may work better if you only match words of three or more, or perhaps four or more characters (not necessary but perhaps helpful).

You can find a decent dictionary of English words in the `words.txt` file at the following url: `https://github.com/dwyl/english-words/blob/master/words.txt`.

In general your code should work with any dictionary file with the specified format (so that it can work with different languages).

Deliverables

There are two deliverables:

- Your source code in file(s) with appropriate extensions.
- Instructions for running your program.

You can include the instructions as comments in Canvas or as a separate txt file. I will run your `affine` utility on my own test files.