AP Computer Science
Dr. Paul Bailey

Project 34a - Crypt
Monday, March 2, 2018

In this project, we implement several encryption and decryption algorithms. This will teach us about how to break down and analyze strings.

We will use only text files, capitalize all the letters, and strip out all the non-letters. The cryptographic schemes we will implement convert strings of uppercase alphabetic characters from their original form into their encrypted form. The source string is called *plaintext*, and the encrypted string is called *ciphertext*. The process of converting plaintext to ciphertext is called *encryption*; the process of converting ciphertext to plaintext is called *decryption*.

Each capital letter corresponds to a nonnegative integer from 0 to 25; A is 0, B is 1, C is 2, and so forth.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Once the letters are converted to numbers from 0 to 25, we used *arithmetic modulo 26* to manipulate them. This means that when we add or multiply two numbers, then take the remainder upon division by 26; in this way, we remain inside the set $\mathbb{Z}_{26} = \{0, 1, 2, \ldots, 25\}$.

Program 0. Create a new NetBeans project named *MyName*Crypt. For example, Bob's project is `BobCrypt`. Make a package inside this project named *myname*crypt; package names should use all lower case letters. For example, Bob's package name is `bobcrypt`.

Create three classes in the package.

- `Cipher` - this class will contain the encryption algorithms
- `Text` - this class will allow access to files
- `Program` - contains a `main` to test the algorithms

Create a directory (such that you know where it is, and can state the entire path!) to save all of your text files for the project. Look up your favorite speeches, poems, song lyrics, and so forth, and put them into the directory in text files for future testing.

Program 1. The `Text` class should contain the methods `public static String loadText(String path)` and `public static boolean saveText(String path, String content)`. The source code is in the appendix.

Create a `main` in the`Program` class and create `test` methods which saves a string to a text file, and load a string from a text file.

Program 2. The `Cipher` class should contain the method `public static String strip(String s)`. The source code is in the appendix. This converts all letters to upper case and strips out all non-letters.

Create `test` methods in the `Program` class which load a file, strip it, and save it into a different file name (in the same directory).

Program 3. The *Caesar cipher* takes each letter c in the plaintext, converts it to a number x between 0 and 25, adds a key k modulo 26 to x, and converts x back to a letter. The encrypted letters are concatenated into a new string, which is the ciphertext. The transformation is summarized as

$$x \mapsto x + k.$$

Decryption uses the same algorithm. The decryption key is $-k = 26 - k$.

In the `Cipher` class, create a method `public static String caesar(String s, int k)` which accepts the plaintext and returns the ciphertext.

Program 4. Since the Caesar cipher only uses 26 different keys, it is very vulnerable to a brute force attack.

In the `Cipher` class, recreate a method `public static void decaesar(String s)` which accepts the previously encrypted ciphertext, now viewed as plaintext, and, in a loop, applies every possible key to it. Display the key and the first 30 or so characters of the output. Eyeball the output to look for English; you will find the original text, and you will be able to say what the original key was.

Program 5. The *affine cipher* works just like the Caesar cipher, except that the key is a pair (a, b), where a and b are integers modulo 26, and a is invertible modulo 26.

$$x \mapsto ax + b.$$

If $y = ax+b$, decryption requires that $x = a^{-1}y+a^{-1}(-b)$. Thus the decryption key is the pair $(a^{-1}, -a^{-1}b)$.

In the `Cipher` class, create a method `public static String affine(String s, int a, int b)` which accepts the plaintext and returns the ciphertext.

Program 6. Since the affine cipher uses $26 \cdot 12 = 312$ different keys, the brute force attack it a little more difficult to implement with only your eyes, but it is possible.

In the `Cipher` class, recreate a method `public static void deaffine(String s)` which accepts the previously encrypted ciphertext, now viewed as plaintext, and, in a loop, applies every possible key to it. Display the key and the first 30 or so characters of the output. Eyeball the output to look for English; you will find the original text, and you will be able to say what the original key was.

```
package bobcrypt;

import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Text
{
    public static String loadText(String fileName)
    {
        String content = null;
        try
        {
            content = new String(Files.readAllBytes(Paths.get(fileName)));
        }
        catch (Exception ex)
        {}
        return content;
    }

    public static boolean saveText(String fileName, String content)
    {
        boolean success = false;
        try
        {
            PrintWriter out = new PrintWriter(fileName);
            out.print(content);
            out.close();
            success = true;
        }
        catch (Exception ex)
        {}
        return success;
    }
}

public class Cipher
{
    public static String strip(String s)
    {
        String t = "";
        char[] a = s.toCharArray();
        for (int i = 0; i < a.length; i++)
        {
            char c = a[i];
            if (c >= 'a' && c <= 'z')
            {
                c = (char)(c - 'a' + 'A');
            }
            if (c >= 'A' && c <= 'Z')
            {
                t += c;
            }
        }
        return t;
    }
}
```