Problem Set 2 – Due Wednesday, April 13, 2016

Problem 4. For this problem you are to implement the ciphertext-only cryptanalytic method for substitution ciphers described in class and in the paper by Diaconis. You can use C, C++, Python, Java, or Go. Email the TAs for permission to use a different language.

Our version of the enciphering scheme works as follows. The key is a random permutation ε on the set $\Sigma = \{\mathrm{a}, \ldots, \mathrm{z}\}$ of lowercase English letters. A message is a string $M = M_1 \cdots M_m$ of arbitrary bytes. For each $i \in [1..|M|]$, if $M_i \in \Sigma$ then let $C_i = \varepsilon(M_i)$; otherwise let $C_i = M_i$. The ciphertext is $C = C_1 C_2 \cdots C_m$. We use the same algorithm to decipher except that the inverse permutation $f = \varepsilon^{-1}$ is then the key.

Compute bigram (two-letter) frequencies of English based on the English translation of *War and Peace*, by Leo Tolstoy. You can find a lower-case version of it at

`http://web.cs.ucdavis.edu/ rogaway/classes/127/spring16/war-and-peace.txt`

Then decipher the following ciphertext using Diaconis's method:

```
qkne l knixw tkn onixenw iytxrerjnx,
qkne tkn uxrray, tkn almbxny, qnxn xiemnw le crobjey hnarxn jn,
qkne l qiy ykrqe tkn ckixty iew wlimxijy, tr iww, wlvlwn, iew jniybxn tknj,
qkne l ylttlem knixw tkn iytxrerjnx qknxn kn onctbxnw qltk jbck iuuoibyn le tkn
onctbxn xrrj,
krq yrre beiccrbetihon l hncijn tlxnw iew ylcd,
tloo xlylem iew molwlem rbt l qiewnxnw raa hz jzynoa,
le tkn jzytlcio jrlyt elmkt ilx, iew axrj tljn tr tljn,
orrdnw bu le unxanct ylonecn it tkn ytixy.
```

About how many iterations (steps) did you need until the ciphertext was decrypted? Submit the plaintext as well as your program.

Hints

You will compute a table M that maps a bigram $(a, b) \in \Sigma^2$ to its frequency $M[a, b] \in [0, 1]$. As described in class, the *plausibility* of a deciphering key f relative to C is then defined as

$$\mathrm{Pl}(f) = \prod_{i=1}^{n-1} M[f(C_i), (C_{i+1})]$$

Due to the tiny sizes of numbers, you will probably want to work with $\ln(\mathrm{Pl}(f))$ instead of $\mathrm{Pl}(f)$. Taking the natural log of both sides yields:

$$\ln(\mathrm{Pl}(f)) = \sum_{i=1}^{n-1} \ln M[f(C_i), f(C_{i+1})]$$

To maximum of $\mathrm{Pl}(f)$ corresponds to the maximum of $\ln(\mathrm{Pl}(f))$.

Here is an example of a C procedure to generate a pseudorandom number in some range:

```c
int randint(int min, int max) {
  unsigned long bins = max - min + 1,
                rands = (unsigned long)RAND_MAX + 1,
                bin_size = rands / bins,
                overflow = rands % bins;
  int r;
  do {
    r = rand();
  } while (rands - overflow <= (unsigned long)r);

  return r / bin_size + min;
}
```

Most languages provide a library implementation of something like this. In Python, for example, you would use `random.randint()`.

Finally, here is an example of a C program to generate a biased coin flip (1 with probability about p and 0 otherwise).

```c
int bernouli(double p) {
  double r = (double)rand() / (double)RAND_MAX;
  return (r <= p);
}
```

Again, none of these example mean that you need to program in C.