CS100, Spring 2000
Project 6 Write-Up: What's the Frequency, Kenneth?
Due in lecture, Thursday, April 13

There are separate write-ups for Project 6. This is the first of 4 documents for Project 6:

→ 1. Project 6 Write-Up
2. Project 6 Milestones
3. Project 6 Tips
4. Project 6 What To Hand In

1. Overview

You should read all write-ups to do this project. The fate of the world rests upon you!

1.1 The Story Line

Assume that you work for Bureau 13, a top-secret government organization that protects the public from all kinds of nasty things. In the past week, Bureau 13 has intercepted an email message from a nefarious society called the Evil Group of People (EGOP) that's bent on destroying the world. Unfortunately, the email message seems garbled even though EGOP used ASCII. Unclear as to EGOP's intent, your supervisors, Yan and Dis, have assigned you the task of deciphering the message. You are the perfect choice because you took CS100 in college and learned how to decrypt messages. Having become a highly skilled programmer, you will hopefully save the world in completing the NEXT project (Project 7). But first, you need to build your skills with this project.

1.2 Goals

By the end of *Project 7*, you will have learned to break substitution ciphers using simple, yet flexible and powerful, techniques. You will also save the world from destruction....

1.3 Skills for Projects 6 and 7

- Encapsulation, static, non-static
- 1-D Arrays and 2-D Arrays
- Sorting and Searching
- Text processing with **`String`**s and **`char`**s
- Artificial intelligence, optimization, heuristic search, hill-climbing
- Cracking simple encryption: breaking substitution ciphers
- Unigram and bigram frequencies of natural languages

1.4 Recommendations

We strongly recommend that you:
- CAREFULLY and COMPLETELY read ALL write-ups.
- Monitor the Announcements webpage and the course newsgroup.
- Go to consultants and/or office hours for interactive help.

2. Background: Encryption and Decryption

2.1 Characters

Your bosses informed you that the information EGOP sent is ***text*** written in English. Each English letter is called a ***character***. In general, a language uses a set of characters to build words. Since sets have no implied order, we organize the character sets into ***alphabets***, which usually sort characters.

2.2 Hidden Information

Suppose part of EGOP's message contains the passage of text `TUBSU SIF  BUUBDL`. Assuming EGOP uses English, this portion doesn't appear to contain meaningful text. However, you might notice a pattern to the letters – try to substitute a letter "one-lower" in the alphabet for each given letter. So, the letter `B` becomes `A`, the letter `C` becomes `B`, and so on. Eventually, you will discover that EGOP "hid" the message `START THE ATTACK`.

2.3 Encryption and Decryption

Zounds! EGOP hid a secret message! How did they do that? What should you do? First, read everything in this report:

- The process of hiding information is called ***encryption*** or ***encoding***.
- The process of "unhiding" information is called ***decryption*** or ***decoding***.

Both processes are required! Encryption provides privacy and protection for information, like messages proclaiming intentions to take over the world. But, what good is the ability to hide something if you can't find ("unhide") it, and what good is the ability to unhide if nothing can be hidden?

To hide or unhide information, something must change the information. The set of rules for performing this ***transformation*** is called a ***key***:

- Encryption transforms text using an ***encryption key***.
- Decryption transforms text using an ***decryption key***.

An ***cryptosystem*** consists of both the rules for encryption and the rules for decryption. That is, a cryptosystem is an encryption key together with the corresponding decryption key:

- To encrypt text, transform unencrypted text with an encryption key.
- To decrypt text, transform encrypted text with a decryption key.

You might sometimes hear or see the term ***cracking***, which means figuring out the decryption key from encrypted text without knowing the decryption key[1].

2.4 Substitution Ciphers

A ***cipher*** is a cryptosystem where the encoding and decoding is done character-by-character, as opposed to being done on words, phrases, or blocks of 1024 characters. From undercover agent Artemov, you know that EGOP failed to study any computer science and, fortunately, chose one of the simplest cryptosystems: a ***substitution cipher***. A substitution cipher, or ***mapping/permutation***, changes each character to another character. For example, in Section 2.2, EGOP's substitution cipher exchanges each letter of the English alphabet with the "next" letter. Note that 'Z' "wraps around" to 'A'. The notation 'A'→'B' means, "'A' maps to 'B'," which also means, "replace each 'A' with 'B'." You must follow two rules with a substitution cipher:

1. Use every character from a given character set.
2. Map every character to only one other character.

Rules 1 and 2 means a character uniquely maps to another character. For example, if 'A'→'B' is part of the encryption key, then neither 'A'→'C' nor 'C'→'B' can be part of the encryption key.

2.5 Encryption Key

Sometimes we might say ***encipher*** or ***encode*** instead of *encrypt*. A cipher uses an ***encryption key*** to encode text. So, applying the encryption key to text encrypts, or "hides," that text. Represent an encryption key by writing two lines:

1. On the ***top line***, write the character set.
2. On the ***bottom line***, write the conversion for each character in another line.

Figure 1 shows the encryption key for the famous substitution cipher called the ***Caesar cipher***, which Julius Caesar supposedly used. The Caesar cipher simultaneously renames each letter with the letter 3 positions "down" in the alphabet and "wraps around" back to 'a' after the letter 'z'.

a b c d e f g h i j k l m n o p q r s t u v w x y z ← top line: alphabet
↓ ↓
d e f g h i j k l m n o p q r s t u v w x y z a b c ← bottom line: encoding of each letter of the alphabet

Figure 1: Encryption Key for Caesar Cipher

An encryption key renames each character on the first row to the character just below on the second row. Using mapping notation, you may express the encryption key in Figure 1 as 'a'→'d', 'b'→'e', ..., 'w'→'z', 'x'→'a', 'y'→'b', and 'z'→'c', where each character is used exactly once on the top and exactly once on the bottom.

How do you encipher a message using an encryption key? Call the unencrypted message ***plaintext*** and the encrypted message ***ciphertext***. Figure 2 places the plaintext on the top line above the ciphertext on the bottom line. Observe that each plaintext letter is replaced by the appropriate character from the encryption key for Caesar cipher.

1. In many cryptosystems, the decryption key is "easy" to compute from the encryption key. Therefore, the encryption key must be hidden for the cryptosystem to be secure. However, there are some cryptosystems where it is (believed to be) infeasible to compute the decryption key from the encryption key, in which case it is OK for the encryption key to be public. So-called *public key cryptosystems* have this property, e.g. PGP/RSA.

For example, `'w'`→`'z'`, `'e'`→`'h'`, the space maps to a space, `'a'`→`'d'`, `'p'`→`'s'`, and so forth. In both Figures 1 and 2, the bottom line is the encoded version of the top line. (But, see also Figures 6–8.)

```
w e  a p o l o g i z e  f o r  t h e  i n c o n v e n i e n c e ← top line: unenciphered plaintext
↓ ↓  ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓  ↓ ↓ ↓  ↓ ↓ ↓  ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
z h  d s r o r j l c h  i r u  w k h  l q f r q y h q l h q f h ← bottom line: enciphered ciphertext
```

Figure 2: Example of Encryption

Note that the encryption key does not require the top line to be sorted. See Figure 3 for an equivalent representation of the Caesar cipher. Once more, each line uses each character from the character set exactly once.

```
a b n o c d g p w x y e f q r s t u v h i j k l m z ← top line: alphabet in random order
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
d e q r f g j s z a b h i t u v w x y k l m n o p c ← top line: encoding of each character
```

Figure 3: Encryption Key with Unsorted Top Line

2.6 Decryption Key

To produce understandable decrypted text, you need to decrypt the encrypted text. Sometimes we might say ***decode*** or ***decipher*** instead of *decrypt*. To decrypt, we undo encryption, i.e., "run encryption in reverse" to transform ciphertext into plaintext, as shown Figure 4.

```
w e  a p o l o g i z e  f o r  t h e  i n c o n v e n i e n c e ← unenciphered plaintext
↑ ↑  ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑  ↑ ↑ ↑  ↑ ↑ ↑  ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
z h  d s r o r j l c h  i r u  w k h  l q f r q y h q l h q f h ← enciphered ciphertext
```

Figure 4: Intermediate Decryption

Observe that Figure 4 has flipped the arrows upside down to show that decryption transforms the bottom line into the top line. Now, compare the character mappings used in Figures 2 and 4:

- Figure 2: `'w'`→`'z'`, `'e'`→`'h'`, the space maps to a space, `'a'`→`'d'`, `'p'`→`'s'`, and so forth.
- Figure 4: `'w'`←`'z'`, `'e'`←`'h'`, the space maps to a space, `'a'`←`'d'`, `'p'`←`'s'`, and so forth.

The mappings in Figure 4 reverse the mappings in Figure 2, which corresponds to the idea of "running encryption in reverse." So, to form the ***decryption key*** for decoding ciphertext, reverse all the mappings in the encryption key. Figure 5 shows this process for the encryption key from Figure 1.

```
a b c d e f g h i j k l m n o p q r s t u v w x y z ←  alphabet
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
d e f g h i j k l m n o p q r s t u v w x y z a b c ←  encoding of each letter of the alphabet
```

Figure 5: Intermediate Decryption Key

For conciseness and consistency omit arrows but understand that they implicitly point down. So, you must flip Figures 4 and 5 upside down, yielding Figures 6 and 7. Figure 8 summarizes the different types and arrangements of keys.

```
d e f g h i j k l m n o p q r s t u v w x y z a b c ←  top line: encoded alphabet
a b c d e f g h i j k l m n o p q r s t u v w x y z ←  bottom line: alphabet
```

Figure 6: Final Decryption Key for Caesar Cipher

```
zh dsrorjlch iru wkh lqfrqyhqlhqfh ← top line: enciphered ciphertext
we apologize for the inconvenience ← bottom line: unenciphered plaintext
```

Figure 7: Final Decryption

	General Key	Encryption Key	Decryption Key
Top Line	alphabet	unencoded alphabet	encoded alphabet
Bottom Line	transformed alphabet	encoded alphabet	unencoded alphabet

Figure 8: Summary of Keys

2.7 Inversion

What is an ***inverse***? The inverse of a process is its “opposite.” For example, the opposite of *increment by 2* is *decrement by 2*. For a given operation, the inverse of a value is its opposite value. For example, for addition, the inverse of 2 is -2. For cryptosystems, encryption and decryption are inverse processes. For the operation of transforming text, encryption and decryption keys are inverse “values.”

To form a decryption key, you swap the top and bottom lines of an encryption key. More formally, swapping the top and bottom lines is called ***inverting***:

- inverting an encryption key produces a decryption key
- inverting a decryption key produces an encryption key

Why the term *inverse*? If you transform plaintext using an encryption key, i.e., encrypt, you produce ciphertext. If you then transform the ciphertext using the decryption key, i.e., decrypt with the inverse of the encryption key, you produce the original plaintext. That is, the encryption key and decryption key “undo” each other and are inverses of each other. Mathematically speaking, for plaintext p

$$decode(encode(p)) = encode(decode(p)) = p \tag{1}$$

For instance, `'a'`→`'d'` followed by `'d'`→`'a'` yields the mapping `'a'`→`'d'`→`'a'`.

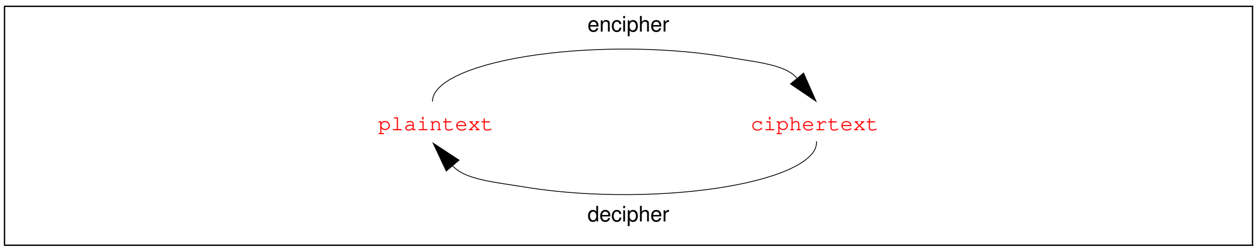

Figure 9: Encryption and Decryption are Inverse Processes

3. Foundation for Cracking: Frequency Analysis

To crack EGOP’s message, you need to know how to compute a decryption key without knowing the encryption key–EGOP tries to keep theirs a secret. Before finding the decryption key, you need to review some basics of natural languages.

3.1 Natural Languages

In computer science, a ***language*** is any set of words over some alphabet. A ***natural language***, as opposed to a programming language, has native human speakers. Natural languages have lots of structure, such as grammar, semantics, and syntax, which assist decryption. To perform your task, consider only frequencies, as discussed below.

3.2 Frequencies

How might you crack a cryptosystem that uses character mappings? Consider how frequently certain characters appear in text. If you could spot repeated patterns in encoded text and then match them to common or known patterns in “regular” text, you might be able to crack the cryptosystem! For example, the letter `'u'` almost always follows the letter `'q'`[2]. Natural languages have other patterns, too. A ***frequency*** is a measure of how often a pattern appears in a body of text. You may measure frequency of a pattern as either:

- a tally: the number of times the pattern appears
- a percentage: a ratio of (the number of times the pattern appears) / (the total number of patterns).

There are published tables of frequencies of single letters and pairs of letters for different languages[3]. We refer to these frequencies as ***unigram*** and ***bigram*** frequencies:

- unigram = 1 letter, where *uni* =1 and *gram* = letter
- bigram = 2 letter pair, where *bi* =2 and *gram* = letter

Note that the order of letters matters, e.g. `qu` and `uq` are different! Higher-order frequencies, like *trigrams*, are also studied and would help our task, but for simplicity, do not consider them.

2. Why not *always*? Because of typos, abbreviations, words borrowed from other languages, and people in advertising that like to mess around with spelling.
3. For an example of frequencies in just words, see http://www.cs.wright.edu/people/faculty/fdgarber/740/ascii/ascii2freq.html.

3.3 Example

For brevity, consider an invented language with the 4-letter alphabet {a,b,c,d}. Mark spaces between words with a hyphen (-) and assume no punctuation. Figure 10 shows a portion of text using this invented language. The following sections demonstrate how to collect and organize frequency data using ***frequency tables***.

```
-a-aaa-abba-abc-ac-ad-ada-add-baa-bad-cab-cb-cdc-dab-dad-dada-dc-
```

Figure 10: Example Text for Invented Language

3.4 Unigram Frequencies

You may collect unigram frequencies in tables using either tallies or percentages. From Figure 10, count the number of times '-', 'a', 'b', 'c', and 'd' each appear. Each character count produces a tally, tabulated in Figure 11. Compute each character's frequency as a ratio of the number of times that character appears and the total number of characters. You may tabulate the frequencies as percentages, as shown in Figure 12. These tables are called ***unigram tables***.

char	-	a	b	c	d
freq	17	20	8	7	12

or

char	-	d	b	c	a
freq	17	12	8	7	20

Figure 11: Unigram Frequencies (Tallies)

char	-	a	b	c	d
freq (%)	27	31	13	11	19

or

char	-	d	b	c	a
freq (%)	27	19	13	11	31

Figure 12: Unigram Frequencies (Percentages)

Although the tables might appear two dimensional, the numbers are in a single row. So, think of the set of unigram frequencies as a 1-D table. Why two tables for each table of unigram frequencies? You may choose to count characters in any order, so the tables in each pair are equivalent. However, note that the count and percent frequencies differ! To access a frequency for a particular character, use the notation $freq_j$, where j is any character from the character set, including '-'. For example, $freq_{'a'}$ = 31 % tells you that 'a' occurs 31% of the time.

3.5 Bigram Frequencies

A bigram frequency measures how often a pair of letters occurs. For instance, take the ratio of the number of times 'c' comes before 'd' (1 time) with the total number of pairs (64 times). You will find that the pair cd appears 2% (1/64) of the time in the text shown in Figure 10. To collect all bigram frequencies, use a 2-D table called a ***bigram table***, as shown in Figure 13.

j = 'd' j = 'c'

i = 'c'

	-	a	b	c	d
-	0	13	3	5	6
a	9	5	6	2	9
b	5	5	2	2	0
c	6	2	2	0	2
d	6	8	0	3	2

or

	-	d	b	c	a
-	0	6	3	5	13
d	6	2	0	3	8
b	5	0	2	2	5
c	6	2	2	0	2
a	9	9	6	2	5

i = 'c'

Figure 13: Bigram Frequencies (%)

Using the bigram table, how do you store and access particular frequencies? Let the notation $freq_{i,j}$ indicate a frequency stored in the bigram table located at row i and column j:

- Row i indicates the first letter in a pair.
- Column j indicates the second letter in a pair.

In Figure 13, the i labels are to the left of the j labels, just like how they appear in words. So, you may express the pair i, j as, "character i before character j." More formally, determine the frequencies of pairs of characters with the following formula:

$$freq_{i,\,j} = \frac{number\ \ of\ \ times\ \ the\ \ pair\ \ ij\ \ appears}{number\ \ of\ \ pairs}. \tag{2}$$

For instance, $freq_{\text{'c'},\,\text{'d'}}$ refers to the frequency 2% located at row 'c' and column 'd' in Figure 13. Other examples include $freq_{\text{'a'},\,\text{'-'}} = 9\,\%$, $freq_{\text{'-'},\,\text{'a'}} = 13\,\%$, and $freq_{\text{'a'},\,\text{'b'}} = 6\,\%$. Some observations you should note:

- $freq_{i,\,j}$ doesn't necessarily equal $freq_{j,\,i}$ because the pairs might occur a different number of times.
- $freq_{\text{'-'},\,\text{'-'}}$ is 0 because the example has no double-spaces.
- By convention, we require the labels on the top be in the same order as the labels to the left.
- The two bigram tables above are equivalent.
- Accounting for roundoff-error, adding up percentages in each row or column in the bigram table yields the percentages in the unigram table. For example, the sum of Column 'b' in the bigram tables produces 3 + 6 + 2 + 2 =13, which matches the unigram frequency of 'b'.

3.6 Enciphering and Deciphering

What happens to the frequencies when plaintext is scrambled after applying an encryption key? Figure 15 shows the ciphertext generated by encrypting the plaintext from Figure 10 using the encryption key shown in Figure 14.

a	b	c	d
c	b	d	a

Figure 14: Example Encryption Key

```
-a-aaa-abba-abc-ac-ad-ada-add-baa-bad-cab-cb-cdc-dab-dad-dada-dc-
-c-ccc-cbbc-cbd-cd-ca-cac-caa-bcc-bca-dcb-db-dad-acb-aca-acac-ad-
```

Figure 15: Encoded Text for Figure 10

What happened to the frequencies? Inspect the unigram and bigram tables of frequency tallies in Figures 16 and 17. As expected, the tables do change. The numbers appear to change, but do they really? For instance, yes, $freq_{\text{'a'}}$ changes from 20 to 12 in unigram table. However, notice that number 20 still appears, but now is $freq_{\text{'c'}}$. In retrospect, this is not surprising: the encryption key maps each 'a' to 'c'. So, the old frequency of 'a' is the new frequency of 'c'. Similarly, in the bigram table, $freq_{\text{'d'},\,\text{'a'}} = 5$ in the plaintext table moves to $freq_{\text{'a'},\,\text{'c'}} = 5$ in the ciphertext table because the encryption key simultaneously maps 'd' to 'a' and 'a' to 'c'. That is, enciphering "scrambles" frequencies by rearranging them.

plaintext

-	a	b	c	d
17	20	8	7	12

ciphertext

-	a	b	c	d
17	12	8	20	7

Figure 16: Unigram Tallies

plaintext

	-	a	b	c	d
-	0	8	2	3	4
a	6	3	4	1	6
b	3	3	1	1	0
c	4	1	1	0	1
d	4	5	0	2	1

ciphertext

	-	a	b	c	d
-	0	4	2	8	3
a	4	1	0	5	2
b	3	0	1	3	1
c	6	6	4	3	1
d	4	1	1	1	0

Figure 17: Bigram Tallies

Decryption maps the ciphertext back to plaintext, restoring the frequencies back to their positions in the original table. Thus, decryption not only "unscrambles text," but also "unscrambles frequencies."

We can therefore hope that if we could somehow unscramble frequencies, then we would unscramble ciphertext, i.e. crack the cryptosystem.