**LSP 121**

**Activity 15**

**Compression and Encryption/Security**

**Compression**

Compression of computer data is a very common occurrence. Data compression is beneficial because it allows us to pack more data into the same storage space. This can also affect how data is transmitted, as compressed data takes less space and thus is transmitted in a more timely fashion. Examples of applications that transmit and/or store compressed data includes digital television broadcasts, DVDs, Blu-Ray discs, MP3 players (such as iPods and smart phones), voice over IP (VoIP) telephones, and cable TV.

Many forms of compression are quite complex, but there is one technique that is fairly common and can easily be shown: run-length compression. Run-length compression is part of a class of *lossless compression* techniques. We call them lossless because when we compress the data and then de-compress it, we retain the original data. A lossy compression technique loses some of its content between compression and de-compression. Music and video files are typically (but not always) lossy.

Run-length compression replaces any repetitions of the same bit or byte that occur in a sequence of data with a run count of the bit/byte. For example, this technique works at the binary level by counting either long strings (or runs) of binary 0s or long strings of binary 1s. Let us consider the following data string, which is composed predominantly of binary 0s:

0000010000000001100000000000000010000110000000000000000000010000001

A compression technique based on run-length encoding would compress the 0s by first counting the “runs” of 0s—that is, it would start by counting the 0s until a binary 1 is encountered. If there are no 0s between a pair of 1s, then that would be considered a run that contains zero 0s. Performing this on our data string, we find the following runs:

5 9 0 15 4 0 20 6

Thus, in the first run, we encountered five 0s, while the second run had nine 0s. The third run had zero 0s because a 1 immediately followed a 1. In the next run, we encountered fifteen 0s, followed by a run of four 0s, zero 0s, twenty 0s, and finally six 0s.

The next step in this compression technique would be to convert each of the decimal values (5, 9, 0, 15, and so on) into 4-bit binary values, or nibbles. Knowing our binary arithmetic, we will use the following nibbles:

Nibble Decimal Value

0000 0

0001 1

0010 2

0011 3

0100 4

0101 5

0110 6

0111 7

1000 8

1001 9

1010 10

1011 11

1100 12

1101 13

1110 14

1111 15

The only unique rule to follow during this conversion comes into play when you encounter a decimal value of 15 or greater. Because the largest decimal number that a 4-bit binary nibble can represent is 15 (which corresponds to four binary 1s—1111), you must convert a run that has a decimal value that is greater than 15 into multiple 4-bit nibbles. For example, a run of 20 would be converted into 1111 0101, in which the first nibble is the value 15, and the second nibble is the value 5. A caveat to this rule is that if you are converting the value of 15 itself, then you would also create two nibbles: 1111 followed by 0000. The reason for this is simply to be consistent—so that whenever a binary nibble of 1111 (or 15) is encountered, the following nibble (0000, which corresponds to a decimal value of 0) is added to that nibble.

Thus, converting the above runs of 5, 9, 0, 15, 4, 0, 20, and 6, would produce the following nibbles:

0101 1001 0000 1111 0000 0100 0000 1111 0101 0110

In this example, note that the original bit string, which consisted of 67 bits, is compressed to 40 bits—a reduction of 40.3%—and that no data has been lost (hence, the name lossless). (Recall from LSP 120: (Original – New Value) / Original = percentage change, or in our example: (67 – 40) / 67). One disadvantage of this technique is that it is worthwhile only if the original data consists predominantly of binary 0s. Run-length encoding is used in compressing video images (JPEG and MPEG) due to the presence of many zero values, as well as compressing other documents that have repeating characters.

1. What are the decimal runs of 0s for the following binary string:

0000000001000001100000000000000010000000100000000000000000000011000001

2. What are the corresponding 4-bit nibbles for the above runs?

3. What percentage reduction results? (Recall percentage change from LSP 120 and the example given above.)

4. Given the following compressed bit string, de-compress it:

00111000000010101111010000000111

**Encryption**

Substitution-based Ciphers

Despite its daunting name, the substitution-based cipher is actually a fairly simple encryption technique. A substitution-based cipher replaces a character or group of characters with a different character or group of characters. Consider the following simple example. Each letter in the plaintext row maps onto the letter below it in the ciphertext row.

Plaintext: abcdefghijklmnopqrstuvwxyz

Ciphertext: POIUYTREWQLKJHGFDSAMNZVCXB

This ciphertext simply corresponds to the letters on a keyboard, scanning right to left, top to bottom. To send a message using this encoding scheme, each plaintext letter of the message is replaced with the ciphertext character directly below it. Thus, the message

how about lunch at noon

would encode to

EGVPO GNMKN HIEPM HGGH

A space has been placed after every five ciphertext characters to help disguise obvious patterns.

4. Using the above plaintext and corresponding ciphertext, encode the message “this is cool”.

A famous substitution cipher is the Caesar Cipher, created and used by Julius Caesar.

Transposition-based Ciphers

A transposition-based cipher is different from a substitution-based cipher in that the order of the plaintext is not preserved. Rearranging the order of the plaintext characters makes common patterns unclear and the code much more difficult to break. Let us consider a simple example of a transposition cipher. Choose a keyword that contains no duplicate letters, such as COMPUTER. Over each letter in the keyword, write the number that corresponds to the order in which that letter appears in the alphabet when compared to the other letters in the keyword. For the keyword COMPUTER, C appears first in the alphabet, E is second, M is third, O is fourth, and so on.

14358726

COMPUTER

Take a plaintext message such as “this is the best class i have ever taken” and write it under the keyword in consecutive rows going from left to right.

14358726

COMPUTER

thisisth

ebestcla

ssihavee

vertaken

To encode the message, read down each column starting with the column numbered 1 and proceeding through to the column numbered 8. Reading column 1 gives us TESV, and column 2 gives us TLEE. Encoding all eight columns gives us the following message:

TESVTLEEIEIRHBSESSHTHAENSCVKITAA

Two interesting observations can be made about this example. First, the choice of the keyword is once again very important, and care must be taken to make sure the keyword does not fall into the wrong hands. Second, you could make the encryption even more difficult by performing an additional substitution-based cipher on the result of the transposition cipher. In fact, why stop there? You could create a very difficult code if you repeated various patterns of substitution- and transposition-based ciphers, one after another.

5. Using the keyword RACQUET, encode the message “I want an ice cream cone to eat before I eat my dinner”.

6. One of the most common forms of communication today is email. It is used for everyday conversations as well as important business decisions. If email is so important, do you or anyone you know send encrypted email? Using the email program of your choice, tell how you can encrypt your emails. Is there an option in your email program to create an encrypted message? If not, is there a third party vendor on the Internet that will encrypt your messages? Describe it.