Quiz 27: Encryption SOLUTION

CSCI 110 Section 1

Wednesday, November 16, 2016

1) Decrypt the following ROT9-encoded message: “yunjbn dbn j yjatrwp byjln cx yjat cqn enqrlun”. Tip: write out a translation chart on a separate piece of paper. [20 points]

Plaintext: “please use a parking space to park the vehicle”

A useful site for this: http://theblob.org/rot.cgi

2) Write a function that takes a string and returns a ROT-13 *encoded* version. The function only needs to work for lowercase letters. Hint: the ASCII char value for lowercase ‘a’ is 97. [30 points]

Looking at an ASCII table, simply adding 13 to a letter’s char value works for the first half of the alphabet. However for the second half of the alphabet, it turns those letters into invalid char values.

How do we get those second half letters to loop back around? Modulo. Because lowercase ‘a’ has an ASCII value of 97, we have to subtract 97 from each letter, then add 13 to do the rotation, then compute that mod 26 to have letters loop back around. Finally, we add 97 back to adjust the letters back to their ASCII values.

```
String rot13(String plaintext) {
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < plaintext.length(); i++) {
      char origLetter = plaintext.charAt(i);
      if (origLetter >= 97 && origLetter <= 122) {
            // If it's a lowercase letter, ROT13 it
            char encodedLetter =
                  (char) (((origLetter - 97 + 13) % 26) + 97);
            result.append(encodedLetter);
      } else {
            // Otherwise put it directly in the result
            result.append(origLetter);
      }
    }
    return result.toString();
}
```

3) Find five numbers that are congruent to 7 (mod 13). [20 points]

Any number n = 7 + 13k where k ∈ **Z** works here. Five examples are -6, 7, 20, 33, 46.

4) Write a function that generates a HashMap<Character, Character> that enables a user to quickly *decode* a message that's written in ROT20. [30 points]

Since the table should *decode*, that means that we should go backwards twenty characters in the alphabet. To avoid dealing with negative results, we'll add 6 instead of subtracting 20. We're assuming that this only needs to work for lowercase letters.

With a for loop:

```
HashMap<Character, Character> getRot20Table() {
    HashMap<Character, Character> table =
        new HashMap<Character, Character>();
    for (int i = 0; i < 26; i++) {
        table.put(
           (char) (i + 97),
           (char) (((i + 6) % 26) + 97));
    }
    return table;
}
```

Without a for loop:

```
HashMap<Character, Character> getRot20Table() {
    HashMap<Character, Character> table =
        new HashMap<Character, Character>();
    table.put('a', 'g');
    table.put('b', 'h');
    table.put('c', 'i');
    table.put('d', 'j');
    table.put('e', 'k');
    table.put('f', 'l');
    table.put('g', 'm');
    table.put('h', 'n');
    table.put('i', 'o');
    table.put('j', 'p');
    table.put('k', 'q');
    table.put('l', 'r');
    table.put('m', 's');
```

```
    table.put('n', 't');
    table.put('o', 'u');
    table.put('p', 'v');
    table.put('q', 'w');
    table.put('r', 'x');
    table.put('s', 'y');
    table.put('t', 'z');
    table.put('u', 'a');
    table.put('v', 'b');
    table.put('w', 'c');
    table.put('x', 'd');
    table.put('y', 'e');
    table.put('z', 'f');
    return table;
}
```

Sanity check: does the table make sense? Yeah: if we look up the value for ‘u’, it takes us back to ‘a’ which is a rotation back of 20.