Solitaire Encryption

Lester McCann
Editted Lightly by Andrew Rosen

February 3, 2019

Abstract

This assignment was originally made by Lester McCann. As a fan of Neal Stephenson, this assignment appealed to me. This assignment is perfect for teaching my students how to manipulate Linked Lists. Most of the wording is Dr. McCann's and the examples are unchanged. I've edited a word here and there, formatted it to more closely render like what my assignments are, and added objective relevant to my class.

1 The Somewhat Simplified Solitaire Encryption Algorithm

In Neal Stephenson's novel **Cryptonomicon**, two of the main characters are able to covertly communicate with one another with a deck of playing cards (including the jokers) and knowledge of the Solitaire encryption algorithm, which was created (in real life) by Bruce Schneier. The novel includes the description of the Solitaire algorithm in an appendix, but you can also find a revised version on the web (see below).

The idea here is that you might want to write secret messages without having access to a computer, and a deck of cards arranged in a specific way is inconspicuous enough not be looked at too closely if you're apprehended by the bad guys.

For this assignment, we'll simplify the algorithm in several ways. For example, we'll be assuming that we have just two suits (say, hearts and spades) from a deck of cards, plus the two jokers, just to keep things simple. Further, let's assume that the values of the 26 suit cards are 1 to 26 (Ace to King of hearts, followed by Ace to King of spades), that the "A" joker is 27, and that the "B" joker is 28. Thus, 15 represents the 2 of spades. Now that you've got the idea, note that because we are doing this in a computer, we can just use the numbers 1–28 and forget about the suits and ranks.

2 Generating Keystream Values

The hard part of Solitaire is the generation of the *keystream values*. (They will be used to encrypt or decrypt our messages.) Here are the steps used in our variant of the algorithm, assuming that we start with a list of the values from 1–28 as described above.

1. Find the A joker (27). Swap Joker A with the card beneath (after) it in the deck. This will effectively move the card down one position. (What if the joker is the last card in the deck? Imagine that the deck of cards is continuous; the card following the bottom card is the top card of the deck, and you'd just exchange them.)[1]
2. Find the B joker (28). Move Joker B two cards down by performing two exchanges.
3. Perform the *triple cut*. To do this, take all the cards above the first joker (the one closest to the top of the deck, A and B don't matter in this step) and swap it with all the cards below the second joker.
4. Remove the bottom card from the deck. Count down from the top card by a quantity of cards equal to the value of that bottom card. (If the bottom card is a joker, let its value be 27, regardless of which joker it is.) Take that group of cards and move them to the bottom of the deck. Return the bottom card to the bottom of the deck.
5. (Last step!) Look at the top card's value (which is again 1-27, as it was in the previous step). Keep it at the top of the deck. Count down the deck by that many cards. Record the value of the NEXT card in the deck, but don't remove it from the deck. If that next card happens to be a joker, don't record anything. Leave the deck the way it is, and start again from the first step, repeating until that next card is not a joker.

The value that you recorded in the last step is one value of the keystream, and will be in the range 1 – 26, inclusive (to match with the number of letters in the alphabet). To generate another value, we take the deck as it is after the last step and repeat the algorithm. We need to generate as many keystream values as there are letters in the message being encrypted or decrypted.

[1] Looks like a good enough reason to do CircularLinkedList

3 Example

As usual, an example will really help make sense of the algorithm. Let's say that this is the original ordering of our half–deck of cards:

```
1 4 7 10 13 16 19 22 25 28 3 6 9 12 15 18 21 24 27 2 5 8 11 14 17 20 23 26
```

Step 1

Swap 27 with the value following it. So, we swap 27 and 2:

```
1 4 7 10 13 16 19 22 25 28 3 6 9 12 15 18 21 24 2 27 5 8 11 14 17 20 23 26
           ^^^^
```

Step 2

Move 28 two places down the list. It ends up between 6 and 9:

```
1 4 7 10 13 16 19 22 25 3 6 28 9 12 15 18 21 24 2 27 5 8 11 14 17 20 23 26
           ^^^^^^
```

Step 3

Do the triple cut. Everything above the first joker (28 in this case) goes to the bottom of the deck, and everything below the second (27) goes to the top:

```
5 8 11 14 17 20 23 26 28 9 12 15 18 21 24 2 27 1 4 7 10 13 16 19 22 25 3 6
^^^^^^^^^^^^^^^^^^^^^^                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Step 4

The bottom card is 6. The first 6 cards of the deck are 5, 8, 11, 14, 17, and 20. They go just ahead of 6 at the bottom end of the deck:

```
23 26 28 9 12 15 18 21 24 2 27 1 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6
^^^^^^^^^^^^^^^
```

Step 5

The top card is 23. Thus, our generated keystream value is the 24^{th} card, which is 11.

Repeat

Self Test: What is the next keystream value? The answer is provided at the end of this document.

4 Encrypting and Decrypting

OK, so what do you do with all of those keystream values? The answer depends on whether you are encoding a message or decoding one. For this two work, the sender and the recipient have to have a deck in the same starting configuration (Both either memorized it or know an algorithm for generating a starting configuration.)

4.1 Encryption

To encode a message with Solitaire, remove all non–letters and convert any lower–case letters to upper–case. (If you wanted to be in line with traditional cryptographic practice, you'd also divide the letters into groups of five.) Convert the letters to numbers (A=1, B=2, etc.). Use Solitaire to generate the same number of values as are in the message. Add the corresponding pairs of numbers, modulo 26. Convert the numbers back to letters, and you're done.

Decryption is just the reverse of encryption. Start by converting the message to be decoded to numbers. Using the same card ordering as was used to encrypt the message originally, generate enough keystream values. (Because the same starting deck of cards was used, the same keystream will be generated.) Subtract the keystream values from the message numbers, again modulo 26. Finally, convert the numbers to letters and read the message.

Let's give it a try. The message to be sent is this:

```
Dr. McCann is insane!
```

Removing the non-letters and capitalizing gives us:

```
DRMCCANNISINSANE
```

The message has 16 letters, and 16 is not a multiple of 5.[2] So, we'll pad out the message with X's. Next, convert the letters to numbers:

```
D  R  M  C  C  A  N  N  I  S  I  N  S  A  N  E  X  X  X  X
4 18 13  3  3  1 14 14  9 19  9 14 19  1 14  5 24 24 24 24
```

Rather than actually generating a sequence of 20 keystream values for this example, let's just pretend that we did:

```
21 6 2 19 15 18 12 23 23 5 1 7 14 6 13 1 26 16 12 20
```

Just add the two groups together pairwise. To get the modulo 26: If the sum of a pair is greater than 26, just subtract 26 from it. For example, 14 + 12 = 26, but 14 + 23 = 37 - 26 = 11. (Note that this isn't quite the result that the operator `%` would give you in Java.)

```
   4 18 13  3  3  1 14 14  9 19  9 14 19  1 14  5 24 24 24 24
+ 21  6  2 19 15 18 12 23 23  5  1  7 14  6 13  1 26 16 12 20
-------------------------------------------------------------
  25 24 15 22 18 19 26 11  6 24 10 21  7  7  1  6 24 14 10 18
```

[2] We like our encrypted messages to be some factor of 5 long. This makes it so ostensibly an adversary can't use anything about the message length to figure out what the message is. Also, it's what the textbooks do.

And convert back to letters:

```
YXOVRSZKFXJUGGAFXNJR
```

4.2 Decryption

Here's how the recipient would decrypt this message. Convert the encrypted message's letters to numbers, generate the same keystream (by starting with the same deck ordering as was used for the encryption), and subtract the keystream values from the message numbers. To deal with the modulo 26 this time, just add 26 to the top number if it is equal to or smaller than the bottom number.

```
  25 24 15 22 18 19 26 11  6 24 10 21  7  7  1  6 24 14 10 18
- 21  6  2 19 15 18 12 23 23  5  1  7 14  6 13  1 26 16 12 20
------------------------------------------------------------
   4 18 13  3  3  1 14 14  9 19  9 14 19  1 14  5 24 24 24 24
```

Finally, convert the numbers to letters, and viola: Another accurate medical diagnosis!

```
4 18 13  3  3  1 14 14  9 19  9 14 19  1 14  5 24 24 24 24
D  R  M  C  C  A  N  N  I  S  I  N  S  A  N  E  X  X  X  X
```

5 Assignment

Use `CircularLinkedList.java` as a starting point to make a new program that reads in a 'deck' of 28 numbers (from a file, from a command line, or an array, your choice), asks the user for one or more messages to decrypt, and decrypts them using the modified Solitaire algorithm described above. Note that if your program is decrypting multiple messages, all but the first should be decrypted using the deck as it exists after the decryption of the previous message. (The first uses the deck provided, of course.)

5.1 Why a Circular Linked List?

We use Circular Linked List here for a few reasons

- Good practice solving and thinking for the upcoming exam.
- The wrap around nature of the Circular Linked List allows us to not worry about Steps 1 and 2.
- Linked Lists are **ideal** for these kinds of manipulations, as we can move entire sections of the list around much, much quicker than an ArrayList. For example, in step 3, we can split our deck into multiple smaller decks and recombine them very quickly.

5.2 Output

Your output will be just the decrypted messages — lists of characters without spaces or punctuation.

5.3 Want to Learn More?

- The original Solitaire algorithm is described on this web page: `http://www.schneier.com/solitaire.html`.

Other Requirements and Hints:

- Start early! There are lots of little things that need to be done to write this program.
- Make sure that you understand our modified Solitaire algorithm before you start writing the program; you can't write a program to solve a problem you don't understand.
- Don't try to write the whole program at once; start small. For example, complete the add method first, then test it. Then move on to remove after you confirm it's working.
- You can check your program's work by hand. Take the data file, manually generate the first few keystream letters, and check that your program generated the same ones.
- Create some encrypted messages for your program to decrypt. The easiest way to do this? Write an encoding method for your program! It's not too hard.
- Exchange encrypted messages with your classmates. Why? If you only test with your own encryption and decryption routines, any logical error with the algorithm implementation is likely to appear in both routines. Without independent verification, you may think that your logically–flawed code is correct.
- Each of the steps of the encryption algorithm are actually very simple. If you need to do something complicated, like it step 3, think about how you can break your linked list into new, temporary, smaller linked lists and recombine them!

Answer to the Self Test: After Step 1:

```
23 26 28 9 12 15 18 21 24 2 1 27 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6
```

After Step 2:

```
23 26 9 12 28 15 18 21 24 2 1 27 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6
```

After Step 3:

```
4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6 28 15 18 21 24 2 1 27 23 26 9 12
```

After Step 4:

```
14 17 20 6 28 15 18 21 24 2 1 27 23 26 9 4 7 10 13 16 19 22 25 3 5 8 11 12
```

After Step 5:

The deck is the same as it was after step 4. The 15^{th} card, the next keystream value, is 9.