Department of Information and Communications Technology

Higher Diploma in Software Engineering (IT114105)
AY 2018/2019

ITP4510 Data Structure and Algorithms

ASSIGNMENT
Deadline: 5 Apr 2019 (Fri) 11:55pm

Solitaire Encryption Algorithm

In Neal Stephenson's novel "Cryptonomicon", two of the main characters are able to communicate with one another secretly using an encryption algorithm code-named "Pontifex". Bruce Schneier actually created this encryption algorithm in real life for the novel and is actually called the **Solitaire Encryption Algorithm,** with the help of a deck of playing cards. In this assignment, we will implement the algorithm (in a slightly modified way). **LinkedList** (or **CircularList**) will be used to store and manipulate our deck of playing cards.

What is Encryption?

Encryption is the process of converting a plain message (referred to as *plaintext*) to secret coded message (*ciphertext*) so that only those who have got a key can convert it back to the original message. The process that converts back the ciphertext to plaintext is called **decryption**.

The Deck of Playing Cards

In this assignment, the deck of playing cards will be represented as a list of integers, as a single line of text, stored in a different file. We will use only two suits (say, spades ♠ and hearts ♥) from a deck of cards, plus the two jokers. Ace to King of spades is represented by 1 to 13, followed by Ace to King of hearts (14 – 26). Joker A will be 27 and Joker B will be 28. The following example shows how a deck will be stored in a file:

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

Note that the sample deck is to be used to encrypt and decrypt a single message.

Generating the Keystream Values

To encrypt or decrypt a message, we need to use a given deck as the tool to generate the keystream values. Here are the steps used in our version of the algorithm, assuming that we start with the deck represented by the numbers 1 – 28 we described above:

1. Find the joker A (27). Move the card down the deck by one position, i.e., swap the joker with the card immediately after it. Using the previous deck as an example, we will swap 27 and 18 as follows:

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

   If the joker is the last card in the deck, move it to the top – just imagine that the deck of cards form a continuous loop.

2. Find the joker B (28). Move it two cards down the deck. So, 28 will be placed after 2 in our given example:

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

   Again, treat the deck as a continuous loop if the joker is the last or the second last card in the deck.

3. Perform a triple cut by swapping the cards above the first joker (the one closest to the top of the deck) with the cards below the second joker. Thus, everything above the first joker (28 in this case) goes to the bottom of the deck and everything below the second one (27) goes to the top:

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

   ⇩

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

4. Perform a count cut. A number of cards at the top will be placed just before the last card in the deck. The number of cards to be cut is determined by the value of that last card. In our example, the last card is 2. Thus, we take two cards from the top and place it just ahead of 2:

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

   ⇩

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

   If the last card is a joker, take the value as 27 no matter which joker it is.

5. Take the value of the top card (say N, set N to 27 if it is a joker). The first keystream value is determined by the value of (N+1)-th card. In our example, as the top card is 11, the first keystream value is the 12-th card in the deck, that is 21.

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

   Joker (27 or 28) is not a valid keystream value. If the (N+1)-th card is a joker, leave the deck as it is and repeat steps 1 to 5 again until the keystream value is not a joker.

To generate the next value, we take the deck as it is after the last step and repeat the algorithm. The number of keystream values to be generated is determined by the length of the plaintext after trimming all whitespaces and non-letters.

Encryption

1. To encrypt a message with Solitaire, all non-letters character characters are removed from the plaintext message and all letters are converted into upper-case.
2. Convert the letters to numbers (A=1, B=2, etc.).
3. Add the keystream values to the letters correspondingly, i.e., first keystream value is added to the first letter and second keystream value to the second letter and so on.
4. If the resultant value is greater than 26, minus 26 from that value. Convert the values to letters and you are done for your ciphertext.

Decryption

1. To decrypt a message with Solitaire is just reversing the steps of encryption. Convert the letters to numbers (A=1, B=2, etc.). The ciphertext should contain upper-case letters only.
2. Subtract (instead of add) the keystream values from the letters correspondingly. The keystream values should be generated from the same deck sequence that is used for encryption.
3. If the resultant value is smaller than 1, add 26 back to that value. Convert the values to letters and you should be able to get the original plaintext.

Example of Encryption and Decryption

Take the first Solitaire message mentioned in novel as an example - “Do not use PC.”:

1. Removing the non-letters and capitalizing the word gives the plaintext: DONOTUSEPC.
2. Convert the message to numbers, we will have: 4 15 14 15 20 21 19 5 16 3
3. The message contains 10 letters. So, we need to generate ten keystream values using the given deck on Page 1. The first keystream value should be 21 as shown before. All ten values are: 21 26 13 1 18 20 25 24 9 7.
4. Keystream values are added to corresponding numbers and the numbers are then converted back to letters as follows:

Plaintext	D	O	N	O	T	U	S	E	P	C
(number representation)	4	15	14	15	20	21	19	5	16	3
Keystream values	21	26	13	1	18	20	25	24	9	7
(encoded numbers)	25	15	1	16	12	15	18	3	25	10
Ciphertext	Y	O	A	P	L	O	R	C	Y	J

   Therefore, the original message is encrypted as “YOAPLORCYJ”.

5. Decryption is just the reverse process of encryption.

Implementation

In this assignment, write a complete, well-documented and suitably object-oriented program that reads in a deck of 28 cards (numbers) from a file and encrypt (or decrypt) the input message using the modified Solitaire algorithm described above.

Your Java program should be able to handle the inputs for the action type, deck file and plaintext message through command line input as follows:

```
java SolitaireEncryption [option] [deck_file] [message_string]
```

where,

`[option]` can either be **`en`**, **`de`**, **`keygen`**, which represent the actions of encrypting the message, decrypting the message and generating (and showing) the keystream values respectively;

`[deck_file]` is the name of the file that holds the single line text of integers representing the deck; and

`[message_string]` is the string (quoted with "") to be encrypted or decrypted. If **`option`** is set to **`keygen`**, only the length of the string will be used.

In this exercise, you are required to store the deck in a **LinkedList** (or **CircularList**). You **MUST** use the **LinkedList** (and/or other data structures) developed in the Lab exercise, but you may add new methods to the class if necessary. The additional methods in your **LinkedList** (or **CircularList**) class should help you to do the following tasks:

- A search method should be added so that you can find the position of your jokers. The method should return an integer value indicating the position of your search item.
- A method that moves an element in the list down ***n*** positions (for steps 1 and 2 of your key generation process). You may also consider writing a swapping method to facilitate the move.
- Methods that will remove ***n*** items from the head or tail from your list (for the triple cut and count cut). These methods should return a **LinkedList** storing the removed items. An empty list should be returned if nothing is removed.
- Methods that will insert a **LinkedList** to the head or at the tail of your existing **LinkedList** (for the triple cut). Take extra care to check if the **LinkedList** is ***null*** or ***empty***.
- You may want to add another method that handles the insertion that occurs in the middle of the list (for count cut), but you can always use some other tricks to obtain the result.
- You can use the ***getItemAt()*** method in the **LinkedList** you have worked on during the Lab to get the face value of the top card, the bottom card and the (N+1)-th card, i.e., no method needs to be added for this purpose.

Expected Output

For the option of key generation, I suppose you should print out the content of your deck in each step of generating your keystream values. **S1** to **S4** shown in the sample output correspond to **Step 1** to **Step 4** for each keystream value in the **Solitaire Encryption Algorithm**.

```
C:\>java SolitaireEncryption keygen deck1.txt "Testing"
S1: [ 21 16 10 9 5 1 13 14 17 22 23 4 20 28 6 2 24 19 15 18 27 26 25 11 12 7 8 3 ]
S2: [ 21 16 10 9 5 1 13 14 17 22 23 4 20 6 2 28 24 19 15 18 27 26 25 11 12 7 8 3 ]
S3: [ 26 25 11 12 7 8 3 28 24 19 15 18 27 21 16 10 9 5 1 13 14 17 22 23 4 20 6 2 ]
S4: [ 11 12 7 8 3 28 24 19 15 18 27 21 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 ]
Key 1: 21
S1: [ 11 12 7 8 3 28 24 19 15 18 21 27 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 ]
S2: [ 11 12 7 8 3 24 19 28 15 18 21 27 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 ]
S3: [ 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 28 15 18 21 27 11 12 7 8 3 24 19 ]
S4: [ 21 27 11 12 7 8 3 24 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 28 15 18 19 ]
Key 2: 26
S1: [ 21 11 27 12 7 8 3 24 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 28 15 18 19 ]
S2: [ 21 11 27 12 7 8 3 24 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 15 18 28 19 ]
S3: [ 19 27 12 7 8 3 24 16 10 9 5 1 13 14 17 22 23 4 20 6 26 25 2 15 18 28 21 11 ]
S4: [ 1 13 14 17 22 23 4 20 6 26 25 2 15 18 28 21 19 27 12 7 8 3 24 16 10 9 5 11 ]
Key 3: 13
S1: [ 1 13 14 17 22 23 4 20 6 26 25 2 15 18 28 21 19 12 27 7 8 3 24 16 10 9 5 11 ]
S2: [ 1 13 14 17 22 23 4 20 6 26 25 2 15 18 21 19 28 12 27 7 8 3 24 16 10 9 5 11 ]
S3: [ 7 8 3 24 16 10 9 5 11 28 12 27 1 13 14 17 22 23 4 20 6 26 25 2 15 18 21 19 ]
S4: [ 20 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 28 12 27 1 13 14 17 22 23 4 19 ]
Key 4: 1
S1: [ 20 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 28 12 1 27 13 14 17 22 23 4 19 ]
S2: [ 20 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 1 28 27 13 14 17 22 23 4 19 ]
S3: [ 13 14 17 22 23 4 19 28 27 20 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 1 ]
S4: [ 14 17 22 23 4 19 28 27 20 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 ]
Key 5: 18
S1: [ 14 17 22 23 4 19 28 20 27 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 ]
S2: [ 14 17 22 23 4 19 20 27 28 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 ]
S3: [ 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 27 28 14 17 22 23 4 19 20 ]
S4: [ 28 14 17 22 23 4 19 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 27 20 ]
Key 6: 20
S1: [ 28 14 17 22 23 4 19 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 20 27 ]
S2: [ 14 17 28 22 23 4 19 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 20 27 ]
S3: [ 28 22 23 4 19 6 26 25 2 15 18 21 7 8 3 24 16 10 9 5 11 12 13 1 20 27 14 17 ]
S4: [ 10 9 5 11 12 13 1 20 27 14 28 22 23 4 19 6 26 25 2 15 18 21 7 8 3 24 16 17 ]
Joker: Key skipped
S1: [ 10 9 5 11 12 13 1 20 14 27 28 22 23 4 19 6 26 25 2 15 18 21 7 8 3 24 16 17 ]
S2: [ 10 9 5 11 12 13 1 20 14 27 22 23 28 4 19 6 26 25 2 15 18 21 7 8 3 24 16 17 ]
S3: [ 4 19 6 26 25 2 15 18 21 7 8 3 24 16 17 27 22 23 28 10 9 5 11 12 13 1 20 14 ]
S4: [ 17 27 22 23 28 10 9 5 11 12 13 1 20 4 19 6 26 25 2 15 18 21 7 8 3 24 16 14 ]
Key 7: 25
Keystream values: [ 21 26 13 1 18 20 25 ]
```

The expected output of the encryption should look somewhat like this:

```
C:\>java SolitaireEncryption en deck1.txt "Do not use PC"
D      4      21     25     Y
O      15     26     15     O
N      14     13     1      A
O      15     1      16     P
T      20     18     12     L
U      21     20     15     O
S      19     25     18     R
E      5      24     3      C
P      16     9      25     Y
C      3      7      10     J
Encrypted message: YOAPLORCYJ
```

And the expected output of the decryption should look like this:

```
C:\>java SolitaireEncryption de deck1.txt "YOAPLORCYJ"
Y      25     21     4      D
O      15     26     15     O
A      1      13     14     N
P      16     1      15     O
L      12     18     20     T
O      15     20     21     U
R      18     25     19     S
C      3      24     5      E
Y      25     9      16     P
J      10     7      3      C
Decrypted message: DONOTUSEPC
```

Instructions to Students

This assignment is an individual assignment. Each student has to submit his/her own work. Plagiarism will be treated seriously. All assignments that have been found involved wholly or partly in plagiarism (no matter these assignments are from the original authors or from the plagiarists) will score ZERO marks. Further, disciplinary action will be followed.

Adequate in-program comments should be placed as appropriate. All user-defined names should be descriptive as much as possible. Marks are given based on correctness, programming quality, and style.

You are required to hand in:

- Well-documented program listings (source programs) and **the executable results (screen dumps).**
- Upload your source programs, executable programs and report to moodle. It is required that your programs can be executed directly.

Marks allocation

Design (7.1)	10%
Implementation	60%
Testing (7.3)	10%
Coding Standard (5)	10%
Appropriate Comments in programs (7.2)	10%

1. Exceptional/abnormal cases. You should create your own Exception class. Submit a brief description of all such cases (e.g**. invalid input**) handled by the program.
2. Program structure and in-program comments.
3. Evidence of testing. Test the program and submit the logged **listing of run samples**.

**** This assignment is counted for 40% of your End of Module Assessment.**

Reference

The original Solitaire algorithm: http://www.schneier.com/solitaire.html