CS2020: Data Structures and Algorithms (Accelerated)

Problem Set 1

Due: January 21th, 11:59pm

Overview. Your first job this week is to set up your environment for writing programs in CS2020. By the end of this problem set, you should be able to write simple Java programs, and measure their performance.

Your second job is to implement a *linear feedback shift register*. This is a simple data structure that can be used as part of a simple encryption schemes, or as a part of a pseudorandom number generator.

Your third (optional) job is to classify a set of plays, determining whether they are by Shakespeare or not.

For each problem, upload the Java files to Coursemology as individual files, and write any textual answers in the text box.

Collaboration Policy. You are encouraged to work with other students on solving these problems. However, you **must** write up your solution **by yourself**. We may, randomly, ask you questions about your solution, and if you cannot answer them, we will assume you have cheated. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Problem 1. (**Getting up and running**)

In CS2020, we will be writing programs in Java. We will be using *Eclipse* as our basic development environment. Eclipse is an open source IDE (integrated development environment) consisting of editing, debugging, and performance measurement tools. You can download Eclipse from: `http://eclipse.org/`.

Problem 1.a. Install Eclipse on your machine.

Problem 1.b. Start Eclipse, and specify a workspace where Eclipse can store your projects. Create a new Java project (File→New→Java Project) called CS2020-PS1-1. Within the new project, create a new package (File→New→Package) called sg.edu.nus.cs2020.

Problem 1.c. Within the cs2020 package, create a new class (File→New→Class) called PSOne. Within this class, create the following method:

```
static int MysteryFunction(int argA, int argB)
{
     int c = 1;
     int d = argA;
     int e = argB;
     while (e > 0)
     {
          if (2*(e/2) !=e)
          {
               c = c*d;
          }
          d = d*d;
          e = e/2;
     }
     return c;
}
```

Also, create the following main function:

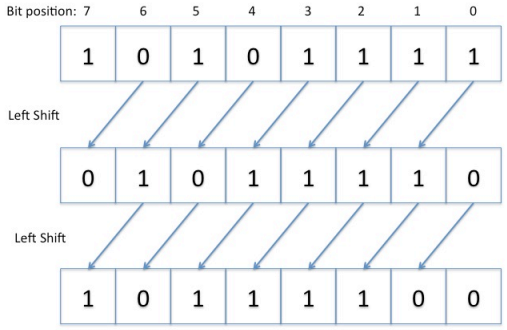
```
public static void main(String args[])
{
     int output = MysteryFunction(5, 5);
     System.out.printf("The answer is: " + output + ".");
}
```

Run your program. What is the answer output in your solution? Submit your program, with a comment indicating the answer output by your program.

Problem 1.d. (Optional.) What is MysteryFunction calculating? If you try a few examples, you might be able to guess.

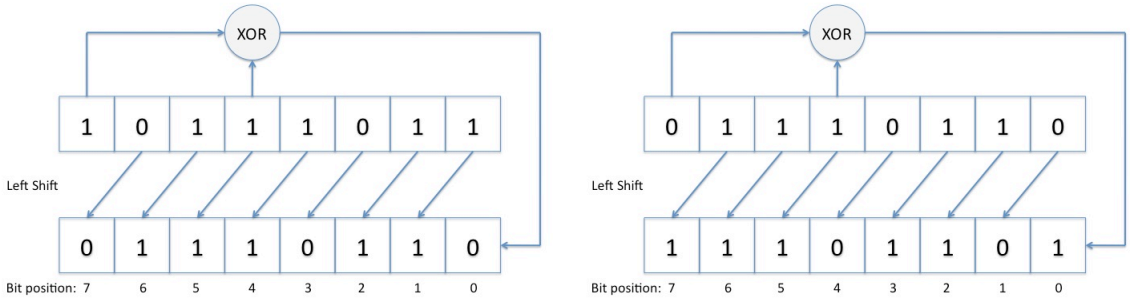
Problem 2. Linear Feedback Shift Register

A *shift register* is an array of bits that supports a *shift* operation which moves every bit one slot to the left. For example, below is an example of a shift register initially containing the seed "10101111". It is shifted twice. Each time it is shifted, the leftmost bit is dropped, every other bit is moved one place to the left, and the rightmost bit is replaced with a '0'. Notice that after it is shifted 8 times, the shift register will be all zeros.



A *linear feedback shift register* is a special type of shift register that, instead of setting the rightmost bit to zero, updates the rightmost bit with some linear function of the other bits. That is, when a shift operation occurs, it feeds back some information into the low-order bit. We will build a linear feedback shift register based on the *exclusive-or* (XOR) function. (In Java, `(a^b)` calculates the XOR of `a` and `b`.)

Our linear feedback shift register takes two parameters: a *size* and a *tap*. The size refers to the number of bits. The tap refers to one of the bits and must be between 0 and *size*−1 (inclusive). Every time a shift operation occurs, the following four things happen: (1) The feedback bit is calculated as the XOR of the leftmost bit and the tap bit. (2) The leftmost bit is dropped. (3) Every bit is moved one slot to the left. (4) The rightmost bit is set to the feedback bit. Here are two examples of a linear feedback shift register in action. In both cases, the size is 8 and the tap is 4.



Problem 2.a. Implement a linear feedback shift register that implements the ILFShiftRegister interface:

```
public interface ILFShiftRegister {

        public void setSeed(int[] seed);

        public int shift();

        public int generate(int k);
}
```

The interface requires that the following methods be supported:

- `void setSeed(int[] seed)`: sets the shift register to the specified initial seed. The initial seed is specified as an array of 0's and 1's (represented as integers). If the seed contains any other value, that is an error. (Recall that the *seed* is the initial set of bits stored in the shift register.) Notice that `seed[0]` is the low-order bit, i.e., the least significant bit in the seed.
- `int shift()`: executes one shift step and returns the rightmost bit of the resulting register.
- `int generate(int k)`: extracts k bits from the shift register. It executes the `shift` operation k times, saving the k bits returned. It then converts these k bits from binary into an integer. For example: if `generate` is called with a value of 3, then it calls `shift` 3 times. If the `shift` operations return 1 and then 1 and then 0, then `generate` returns the value 6 (i.e., "110" in binary).

Your implementation should also support the following constructor:

```
ShiftRegister(int size, int tap)
```

This constructor initializes the shift register with its size and with the appropriate tap. Submit your implementation.

Problem 2.b. Test your code with a variety of test cases. Make sure to test the corner cases (e.g., when the shift-register has size 1, etc.). Here are two test cases to use. The first tests the `shift` functionality, while the second tests the `generate` functionality.
Test 1: seed = '101111010', tap = 7

```
int[] array = new int[9];
array[0] = 0;
array[1] = 1;
array[2] = 0;
array[3] = 1;
array[4] = 1;
array[5] = 1;
array[6] = 1;
array[7] = 0;
array[8] = 1;

ShiftRegister shifter = new ShiftRegister(9,7);
shifter.setSeed(array);

for (int i=0; i<10; i++){
        int j = shifter.shift();
        System.out.print(j);
}
```

This code should produce the following output:

```
1100011110
```

Test 2: seed = '101111010', tap = 7

```
int[] array = new int[9];
array[0] = 0;
array[1] = 1;
array[2] = 0;
array[3] = 1;
array[4] = 1;
array[5] = 1;
array[6] = 1;
array[7] = 0;
array[8] = 1;

ShiftRegister shifter = new ShiftRegister(9,7);
shifter.setSeed(array);

for (int i=0; i<10; i++){
```

```
        int j = shifter.generate(3);
        System.out.println(j);
}
```

This code should produce the following output:

```
6
1
7
2
2
1
6
6
2
3
```

Submit your test code, along with several test cases.

Problem 2.c. Now test your code using the provided "JUnit test." (These are a subset of the tests that the tutors will use when evaluating your code.) Load the file `ShiftRegisterTest.java` in Eclipse, and update the method `getRegister` to use your shift register implementation. Then execute the tests.

Look at the last test: it is testing what happens when the seed is larger than the specified register. Explain what you think a proper response is to this erroneous situation, and what is the right way to test this case? (We will cover error handling later in the semester.)

Problem 2.d. Think about how you might use a shift register to perform a simple encryption scheme. (Hint: initially, the seed of the shift register should be set to your "password.") How would you encrypt a text file? How would you encrypt an image?

The two released classes `ImageEncode.java` and `SimpleImage.java` implement a simple encryption/decryption scheme using a shift register. You will need to add a couple lines of code to `ImageEncode.java` so that it properly uses your shift register. To decode the picture, using the following "code":

- Tap: `7`
- Seed: `11100010110`

If your shift register works correctly, you should then be able to decode the included picture. Who is it a picture of? (The only thing you are required to submit for this part is an identification of the decoded picture.)

Problem 2.e. (Extra, for those who are interested) Think of the output of a linear feedback shift register as the sequence of 0's and 1's that you get if you call `getBit()` after each shift. A linear feedback shift register is only useful for encryption if the resulting output sequence does not repeat very often. Experiment with how many times a shift register has to be shifted in order to cause the output pattern to repeat. Are all taps equally good?

Also, notice that a short binary password will not provide very good security. Optionally write a modified version that takes a text string (such as, "TheCowJumpedOverTheMoon") and converts it to a (long) binary string to use as a seed for a shift register.

Another use of linear feedback shift registers is to generate pseudorandom numbers. One basic property of a pseudorandom number generator is that it outputs roughly the same number of 0's as 1's. Is that true of your linear feedback shift register?

Hints for question 2:

Do I need to comment my code? Yes. Be sure to write a comment explaining: the purpose of each class instance variable, and the purpose/workings of each method. You should also explain any critical steps in your code.

What data structure should I use to store the current value of the shift register? One of the advantages of encapsulation is that you can store the value of the register in any way that you want. For simplicity, we recommend that you use an array of integers (as is passed to `setSeed`). However, if you want to optimize the running time of the `shift` operation, there are better more efficient solutions. Implement the best solution you can!

For `generate`, how do I convert the bits produced by `shift` into an integer? Begin with a variable v set to zero. Every time you get a new bit from `shift`, multiply v times two, and add the new bit (either 0 or 1). The final result will be the integer representation of the bits.

How do I create a variable to store the array of integers in the register when I don't know its size? Declare it simple as an array of integers: `int[] intArray`. Then, initialize it in the constructor once you have learned the correct size.

For Test 2, my first call to `generate` produces the value 3 instead of the value 6. You are interpreting the bits in the wrong order. Notice that the binary string '110' is 6, while the binary string '011' is 3.

I get an ArrayOutOfBounds error (or a NullPointerException). Did you remember to initialize all your class instance variables?

How should I debug my program, when I cannot see the internal state of my object? Wouldn't it be nice if you could just type `System.out.println(shifter)` and have it print out the value of the shift register? In fact, you can if you implement:
`public String toString()`
This method should convert the value of the register into a string and return it. Then, you can modify your test code to include:

```
System.out.println(shifter + " " + j);
```

and you will get the following output for the first test:

```
011110101 1
111101011 1
```

```
111010110 0
110101100 0
101011000 0
010110001 1
101100011 1
011000111 1
110001111 1
100011110 0
```

Problem 3. (Document Distance Experiments (Extra Credit))

In this question, you are going to try to differentiate texts by Shakespeare from texts by some clever forgers. Attached to the problem set, you will find six files.

- *hamlet.txt*, a play by William Shakespeare
- *henryv.txt*, a play by William Shakespeare
- *macbeth.txt*, a play by William Shakespeare
- *mystery.txt*, a mystery text
- *cromwell.txt*, a play sometimes attibuted to William Shakespeare
- *vortigern.txt*, a play forged to sound like William Shakespeare

Using the provided VectorTextFile.java class, determine which plays are by Shakespeare and which are not. You should be able to use the VectorTextFile class, even without understanding the details of how it works! Just read the comments on its public functionality.

All five are written to sound like Shakespeare, and hence the "angle" between any two vectors will not differ much. Even so, the plays by Shakespeare form a cluster, where every pair is close together. Find a threshold angle α such that:

- if text A and B are both by Shakespeare, then the angle between A and B is $< \alpha$, and if
- if text A is by Shakespeare and text B is not by Shakespeare, then the angle between A and B is $> \alpha$.

The existence of such a threshold α indicates a cluster of texts by the same author.

Submit three things: (i) the Java code you used to find the threshold, (ii) the threshold value α, and (iii) a classification of which texts are by Shakespeare and which are not.