Exam of Computer Science
SCAN 1st year - Juin 2021

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Total duration : *2h*
Allowed materials : *Any personal or lecture notes.*
Smartphones are not allowed

— The graduation can still change.

— The assignment is on 6 pages.

— Questions are independent. If it is necessary for A to be solved to solve B, then you can do as if you had solved A (and clearly indicate your assumption) to solve B.

Thought of the day:
« Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. » *Martin Golding*

Indicative marking scheme :

- Exercise 1 : 9 points
- Exercise 2 : 11 points

Before you start :

- Parts are independent. You can answer them in the order of your choice.
- The subject is long. Take some time to read all the questions before you start.
- The neatness of your exam script will be taken into account.

1 Digital image processing (9 pts)

In digital image processing, histogram equalization is a contrast adjustment method based on the image histogram. It performs a transformation on each pixel of the image, and thus one obtains a new image only by applying an independent operation on each pixel. This transformation is done using the cumulative histogram of the original image.
Histogram equalization (Fig.1) allows a better spreading of the intensity over the possible range of values by spreading the histogram (source https ://fr.wikipedia.org/).

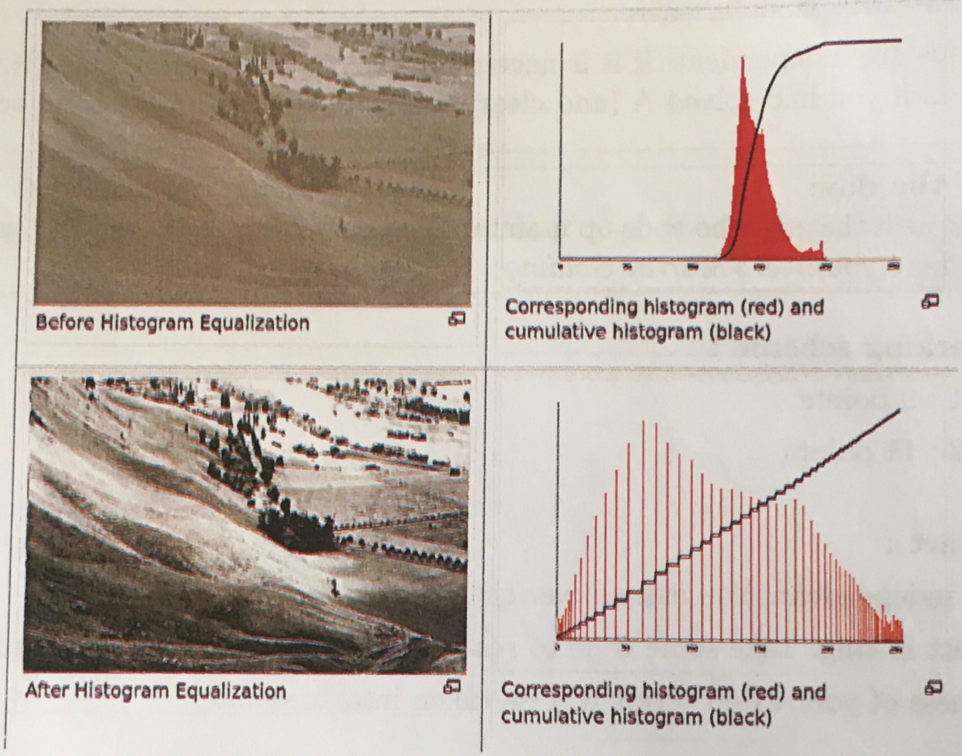

FIGURE 1 – Histogram equalization

In the following, a digital image in gray scale is a 2D array containing integer values (pixels) ranging from 0 to 255 (0 for black and 255 for white). There is a total of L=256 levels of gray.

The histogram is a 1D array of integers that stores the number of occurrences of a pixel value in the image : $histo[k] = n_k$ where n_k is the number of pixels with value k ($k \in [0..255]$).

For instance, if there are 250 pixels with value 0 in the image, the cell 0 of the histogram will contain the value 250; if there are 50 pixels with value 128, the cell 128 of the histogram will contain the value 50.

(Q1.1) Write the `histogram` method that takes as parameter an image and returns the histogram of this image (1,5 pt).

The cumulative histogram of an image is defined as : $histoCumul[k] = \sum_{j=0}^{k} histo[j]$

(Q1.2) Write the `histogramCumul` method that takes as parameter the histogram of an image and returns the cumulative histogram of this image (1 pt).

We want to display the histogram of an image in the terminal (line by line from the top to the bottom) using the `System.out.print` and `System.out.println` methods. For each pixel value in x coordinate, we display the corresponding value of the histogram using stars '*' (see Fig.2). We will compute and use the maximum value (ordinate) of the histogram.

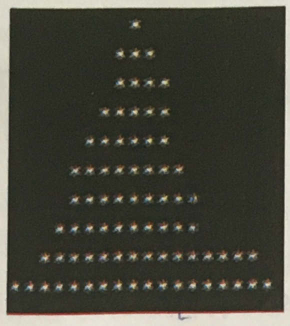

FIGURE 2 – Display of an histogram ; abscissa : pixel value k ; ordinate : n_k ; in this example, only a subset of the histogram is shown.

(Q1.3) Write the `displayHistogram` method that takes the histogram of an image and displays it using the format presented above (3 pts).

The histogram equalization produces a new image by replacing the value of each pixel x_k by $T(x_k)$:

$T(x_k) = \frac{L-1}{N} \times histoCumul[x_k]$
where L is the number of gray levels (256) and N is the total number of pixels in the image.

(Q1.4) Write the `equalizationHistogram` method that takes a digital image and returns the image equalized. You should use the previously written methods (2 pts).

We consider that the digital image has been loaded from the disk using the method (`loadImage`) and stored in the variable `int[][] image` declared in the main.

```
public static void main ( String[] args){
int[][] image = loadImage("image.png");
// to be completed
}
```

(Q1.5) Complete the `main` method in order to : (1,5 pt)

— display the histogram of the image

— create an equalized version of this image

— display the histogram of the equalized image.

2 Small problem on big numbers (11 points)

In Java, the primitive type *int* is a finite subset of integers, which depends on the size of memory words used to represent integers. The type *int* is coded on 4 bytes and can represent numbers between -2147483648 and +2147483647. When a number is too large to be represented by an *int*, we can use the type *long* (on 8 bytes) instead. But if we want to use big numbers that do not fit in 20 digits, we need a new representation. To achieve this goal, we have chosen to model those big numbers, here not signed, by using a class and adopting the Object Oriented Programming (OOP) paradigm.

The objective of this exercise is to use big integers that will be represented by the following **BigNumber** class :

```
BigNumber
---------------------------------------
+ val : int []
+ size : int
+ SIZEMAX : int
---------------------------------------
+ BigNumber(int size)
+ generateRandom() : void
+ sum(BigNumber bN) : BigNumber
+ countPatternABA(int n) : int
```

To facilitate the computation with those big numbers, we choose to represent a big number by an array of integers (*val*) of size (SIZEMAX) that includes (*size*) useful integers, each of them representing a digit and being in the interval 0..9 in such a way that :

- units are stored in cell of index 0
- tens in cell of index 1
- hundreds in cell of index 2, etc...
- the remainder of the array is filled until the end with 0 .

No number can exceed *SIZEMAX* integers, where *SIZEMAX* is fixed as an integer constant of value 100.

For instance, the integer 65789560765445900431020, will be represented as shown in Fig.3 :

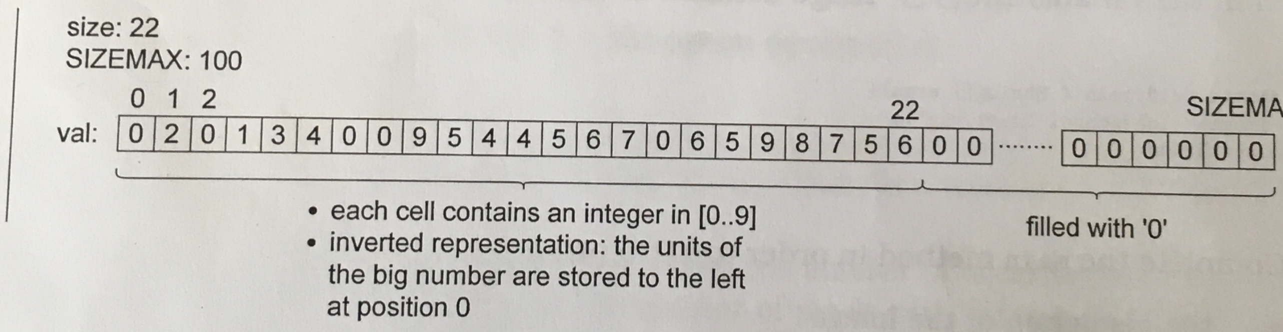
size: 22
SIZEMAX: 100
0 1 2
22
SIZEMAX-1
val: 0 2 0 1 3 4 0 0 9 5 4 4 5 6 7 0 6 5 9 8 7 5 6 0 0 .... 0 0 0 0 0 0
• each cell contains an integer in [0..9]
• inverted representation: the units of the big number are stored to the left at position 0
filled with '0'

FIGURE 3 – Example of representation of a number with the BigNumber class.

(Q2.1) To create a **BigNumber**, it is necessary to call a constructor. How many constructors can a class have and how to distinguish them?

To facilitate the manipulation of the attributes, we will declare them as *public*.

(Q2.2) Write the beginning of the **BigNumber** class that will contain the attributes and the constructor as shown on the UML diagram. Once created, the array `val` is filled with zero by default in java, i.e. no need to set all cells to 0 (1 pt).

(Q2.3) To initialize a **BigNumber**, we propose a random filling of the *size* first cells of the array. During the generation, we will make sure that the most significant digit (in our example '6') is not equal to 0. Write the `generateRandom()` method that implement this initialization (1,5 pt).

(Q2.4) Modify the constructor of the **BigNumber** class so that it replaces the initialization with zero by a random initialization of the array. In the following, you will use this version of the constructor. Then, describe , without coding them, the changes required to allow the creation of a **BigNumber** either with a random array or with an array provided by the user (1.5 pt).

We want to perform the sum of two big numbers. To do so, we suggest the following `sum` method :

```java
    public BigNumber sum(BigNumber bN){
    int sizeNewBG = ... ;
     BigNumber bNSum = ...;
    int sum = 0;
    int carry = 0;
    for (int i=0; i< ...; i++){
       bNSum.val[i] = ...;
       carry = ...;
    }
    if (carry > 0){
       if (sizeNewBN == this.SIZEMAX){
          bNSum.size = -1;
       } else {
          bNSum.size = ...;
          bNSum.val[...] = carry ;
       }
    }
    else {
       bNSum.size = ...;
    }
    return bNSum;
}
```

(Q2.5) On your copy, finish the `sum` method by filling the missing pieces of code. If you want to make some minor modification to some instructions, make sure that those modifications are clearly indicated (1,5 pts).

We want to analyze the sequence of digits in a number in search of simple patterns *'a-b-a'*.

(Q2.6) Write the `countPatternABA` method that counts the number of patterns *'a-b-a'* found in the number. The method takes as parameter an integer representing *b*, the central value of the pattern (1 pt).

We want to implement an encapsulation of the attributes of the class to prevent any direct access from the `main` class.

(Q2.7) Give the modifications that need to be applied to the **BigNumber** class to prevent this direct access and provide the role and the signature (without detailing the implementation) of the methods that will allow an access to the attributes from outside the class (1.5 pt).

(Q2.8) Assuming that the attributes are *public*, write the `main` performing the following operations (2 pts) :

— create a big number *bn* including 25 digits with a random generation of its content

- perform the sum of this number with itself until obtaining the largest possible number that can be represented with the type **BigNumber**. Display this last BigNumber
- create another big number with 25 digits and with at least 2 patterns '*-4-*'.