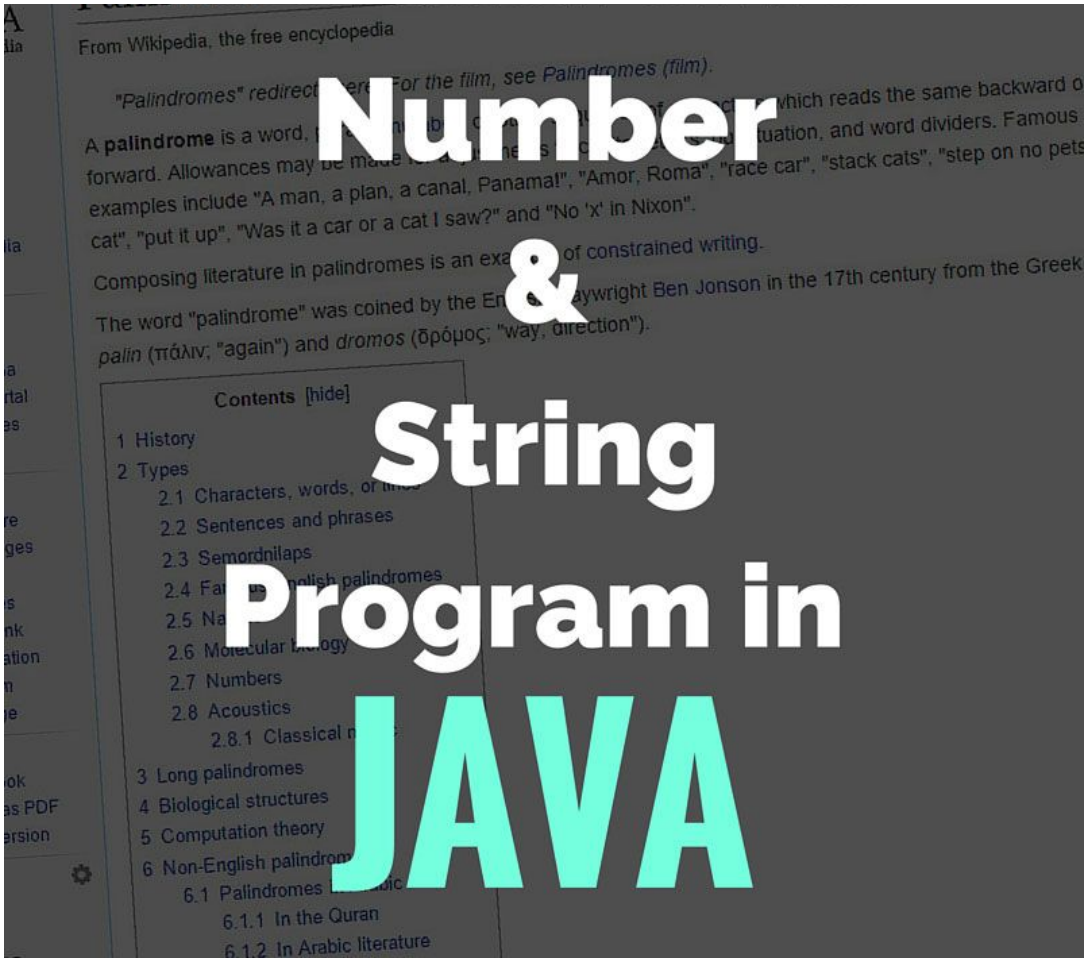

From Wikipedia, the free encyclopedia
Contents [hide]
1 History
2 Types
2.1 Characters, words, or
2.2 Sentences and phrases
2.3 Semordnilaps
2.4 Fa
2.5 Na
2.6 Molecular
2.7 Numbers
2.8 Acoustics
2.8.1 Classical
3 Long palindromes
4 Biological structures
5 Computation theory
6 Non-English palindrom
6.1 Palindromes
6.1.1 In the Quran
6.1.2 In Arabic literature
Number
&
String
Program in
JAVA

Índice.

1. Funciones para la compatibilidad de las funciones. 3
1.1 Función validBignumber para comprobar que los BigNumber son validos. 3
Diagrama de flujo de la función validBigNumber. 3
1.2 Función removeZero para eliminar ceros a la izquierda. 4
Diagrama de flujo de la función removeZero. 4
1.3 Función addZero para añadir ceros al String. 5
Diagrama de flujo de la función addZero. 5
2. Función equals para comprobar si dos BigNumber son iguales. 6
Diagrama de flujo de la función equals. 6
3. Función compareTo para comprobar que BigNumber es el mayor. 7
Diagrama de flujo de la función compareTo. 7
4. Función add para sumar dos BigNumber. 8
Diagrama de flujo de la función add. 8
5. Función sub para restar dos BigNumber. 9
Diagrama de flujo de la función sub. 10
6. Función mult para multiplicar dos BigNumber. 11
Diagrama de flujo de la función mult. 12
7. Función div para dividir dos BigNumber. 13
Diagrama de flujo de la función div. 14
8. Función power para elevar un numero a una potencia. 15
Diagrama de flujo de la función power. 15
9. Función para calcular el factorial de un BigNumber. 16
Diagrama de flujo de la función factorial. 16
10. Función para calcular el MCD de dos BIgNumber. 17
Diagrama de flujo de la función mcd. 18
11. Función sqrt para hallar la raíz cuadrada de un BigNumber. 19
Diagrama de flujo de la función sqrt. 21

1. Funciones para la compatibilidad de las funciones.

En este apartado estarán todas las funciones que se usarán en todo el código para prevenir posibles errores en los BigNumber, estas funciones no pertenecen al esqueleto del proyecto, las he añadido ya que se usan en varias partes, así nos ahorramos repetir código, por ejemplo, código para comprobar que son valores numéricos. Todas las funciones de prevención de errores devolverán un null.

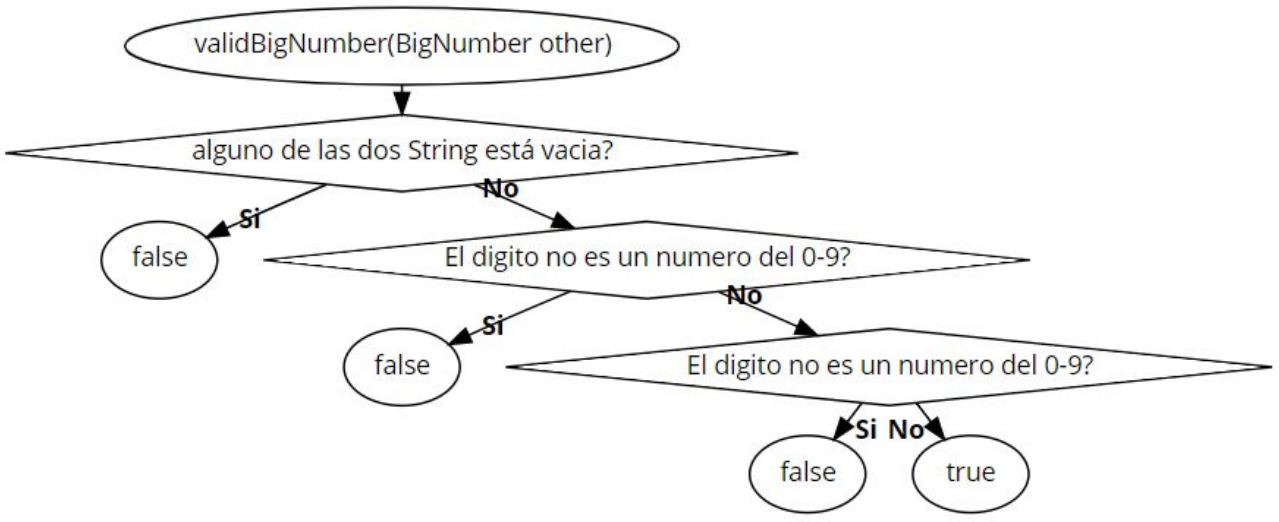
1.1 Función validBignumber para comprobar que los BigNumber son validos.

Esta función la usaremos para comprobar que los BigNumber son adecuados para hacer las operaciones en ellos. Para esto comprobaremos que la longitud del número no sea 0, ya que si el número es un String vacío nos romperá el código.

A continuación haremos dos bucles, cada uno recorre un BigNumber caracter a caracter comprobando que no sean letras ni caracteres especiales, solo pueden ser números del 0-9 o del 48-57 en código ASCII.

```
public boolean validBigNumber(BigNumber other){
    if ((this.numeroString.length() == 0) || (other.numeroString.length() == 0)) return false;
    for (int i = 0; i < this.numeroString.length(); i++) {
        if ((this.numeroString.charAt(i) < 48) || (this.numeroString.charAt(i) > 48+9)) return false;
    }
    for (int i = 0; i <other.numeroString.length() ; i++) {
        if ((other.numeroString.charAt(i) < 48) || (other.numeroString.charAt(i) > 48+9)) return false;
    }
    return true;
}
```

Diagrama de flujo de la función validBigNumber.



1.2 Función removeZero para eliminar ceros a la izquierda.

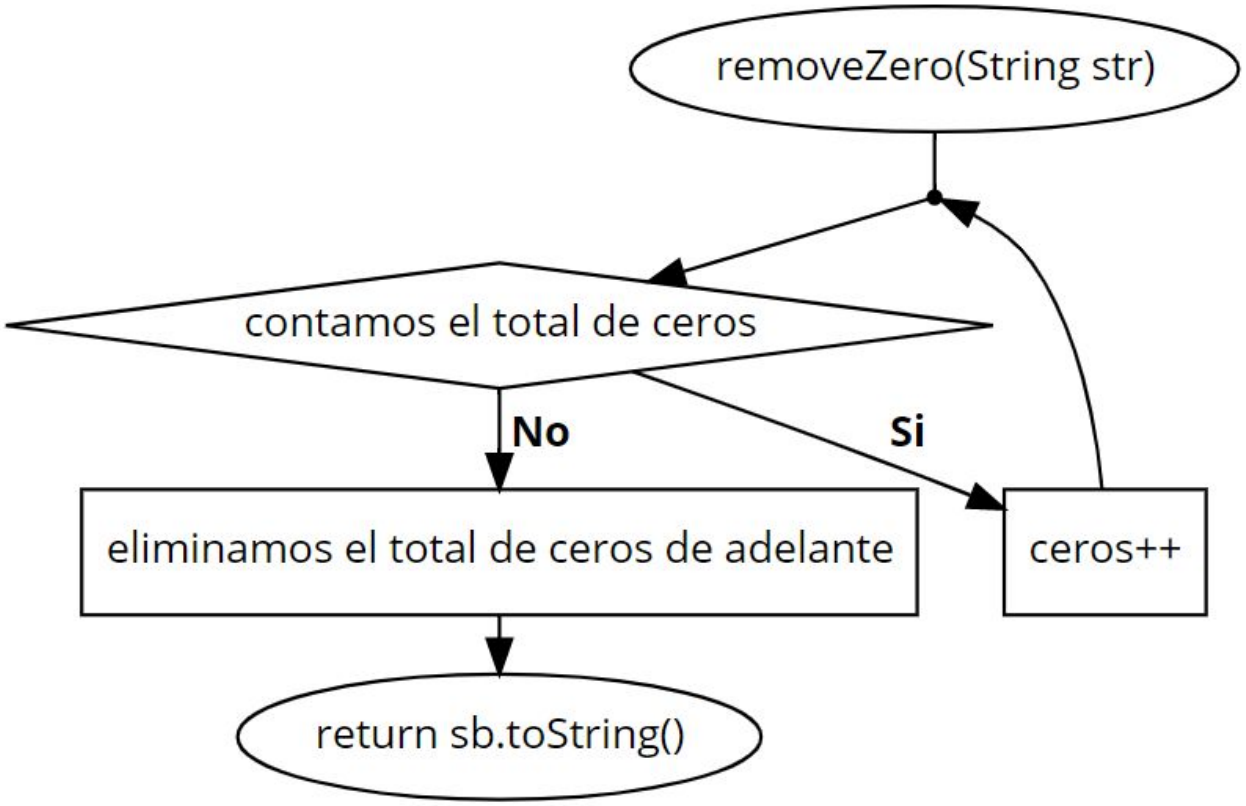
La función removeZero la usaremos para eliminar los ceros de la izquierda de la String de un objeto, esto nos facilita el trabajo a la hora de hacer las comparaciones, ya que se puede hacer un código más óptimo.

0000045 → 45

En el codigo simplemente usamos un bucle para contar el total de ceros que hay que quitar y luego con el objeto StringBuilder usamos el replace desde la posición 0 hasta el contador de ceros.

```
public String removeZero(String str) {
    int i = 0;
    while (i < str.length() && str.charAt(i) == '0') i++;
    StringBuilder sb = new StringBuilder(str);
    sb.replace( start: 0, i,  str: "");
    return sb.toString();
}
```

Diagrama de flujo de la función removeZero.



1.3 Función addZero para añadir ceros al String.

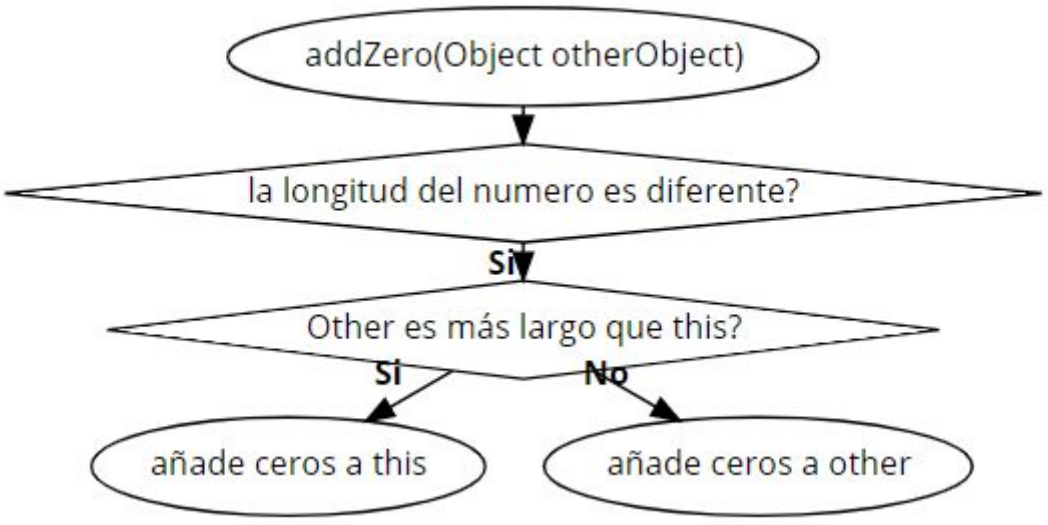
Esta se encargará de poner ceros a la izquierda para igualar los dos objetos. Esto nos es muy útil para la mayoría de operaciones ya que si no, podemos tener una excepción de index out of bounds al ir caracter por caracter.

65432 & 45 → 00045

Lo único que hace la función es comparar las dos Strings de entrada y si hay una de ellas de diferente tamaño le añadirá tantos ceros como diferencia de longitud halla entre las dos Strings, en este caso añadiria tres ceros para igualarlo.

```
public void addZero(Object otherObject) {
    BigNumber other = (BigNumber) otherObject;
    int numDiferencia = 0;
    if (other.numeroString.length() != this.numeroString.length()) {

        if (other.numeroString.length() > this.numeroString.length()) {
            numDiferencia = other.numeroString.length() - this.numeroString.length();
            for (int i = numDiferencia; i > 0; i--) {
                this.numeroString = "0" + this.numeroString;
            }
        } else {
            numDiferencia = this.numeroString.length() - other.numeroString.length();
            for (int i = numDiferencia; i > 0; i--) {
                other.numeroString = "0" + other.numeroString;
            }
        }
    }
}
```

Diagrama de flujo de la función addZero.



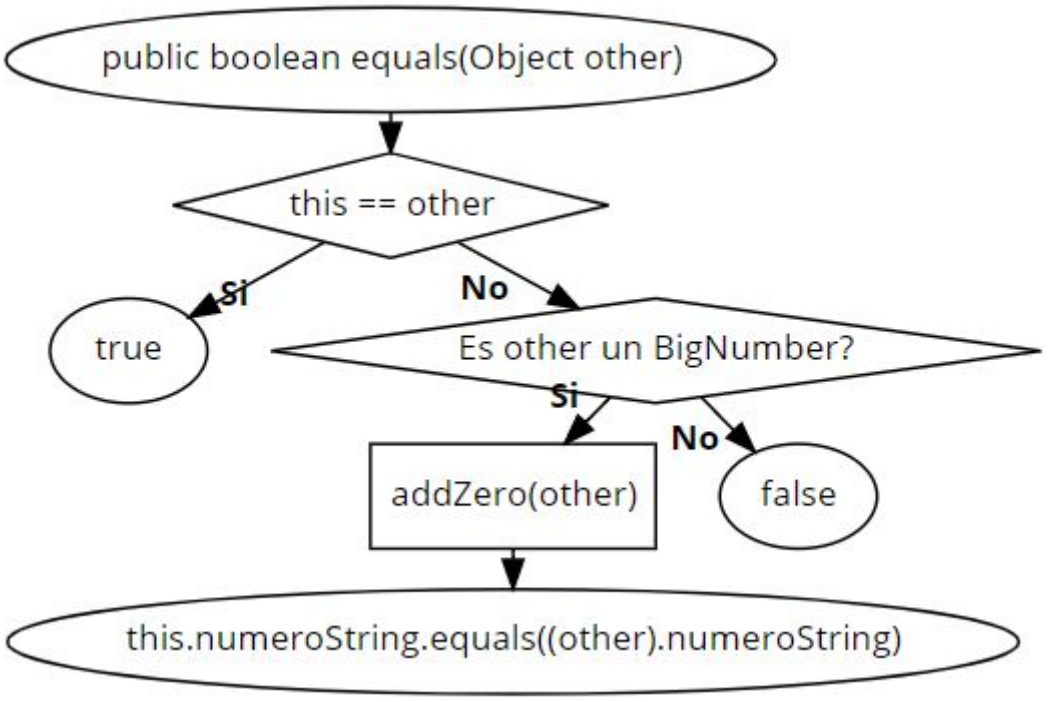
2. Función equals para comprobar si dos BigNumber son iguales.

Esta función consiste en comparar dos Objetos y devolver un boolean con el resultado de su comparación, si son iguales será un true, si no, un false. Para implementarla tenemos hacer un Override al método ya existente en la clase Object de Java.

Lo primero que hacemos es comprobar si literalmente los objetos son idénticos como tal, luego comprobamos que el Objeto que nos entra por parámetro sea una instancia de BigNumber, si es así pasamos a la última fase que es comprobar si el valor de su atributo es el mismo, para esta fase necesitamos la función addZero o removeZero para que sean del mismo tamaño.

```java
@Override
public boolean equals(Object other) {
    if (this == other) return true;
    if (other instanceof BigNumber){
        addZero(other);
        return this.numeroString.equals(((BigNumber) other).numeroString);
    }
    return false;
}
```

Diagrama de flujo de la función equals.



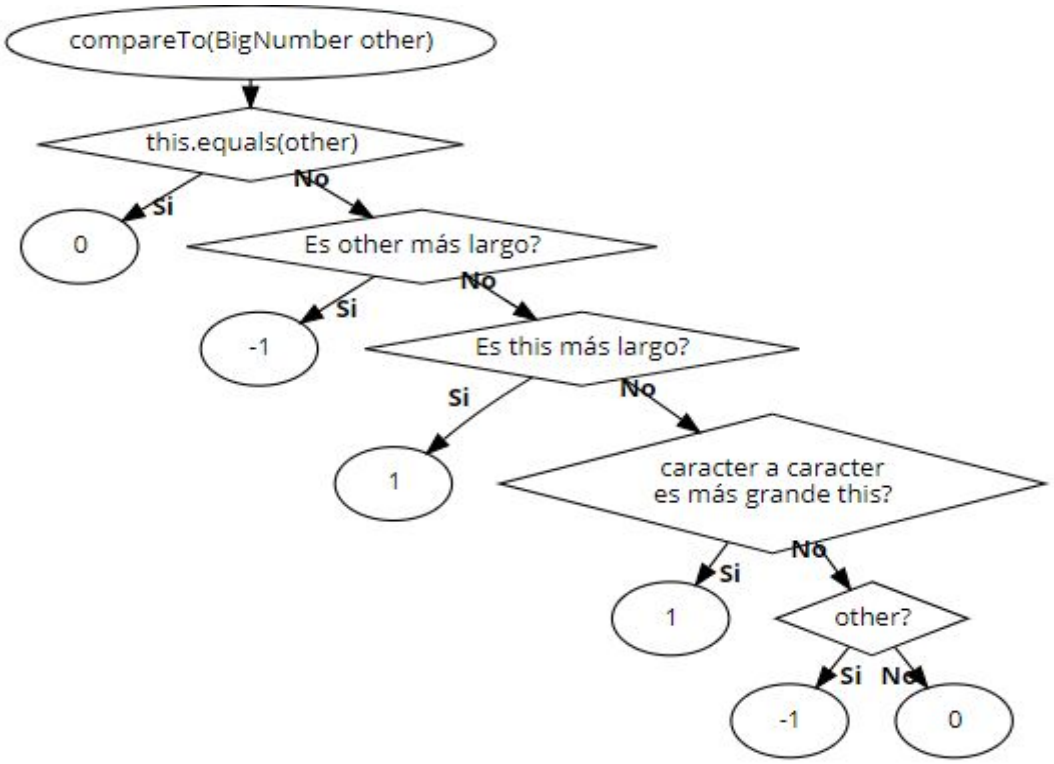
3. Función compareTo para comprobar que BigNumber es el mayor.

En este apartado tendremos que comparar el valor de dos número. De los dos números que nos pasan tendremos que devolver “1” si el primero es mayor, “-1” si el primero es menor y “0” si son iguales.

Lo primero es comparar con la función equals para saber si directamente son iguales, lo siguiente es comprobar según la longitud la longitud para esto eliminaremos los 0 a la izquierda con la función removeZero y, para los que tienen la misma longitud se soluciona con un “for” que recorre las mismas posiciones hasta que encuentre cuál es el mayor, o en su defecto devolverá “0” porque significa que son iguales.

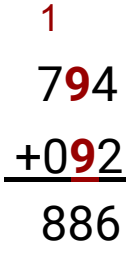
```
public int compareTo(BigNumber other) {

    this.numeroString = this.removeZero(this.numeroString);
    other.numeroString = other.removeZero(other.numeroString);

    if (this.equals(other)) return 0;
    if (this.numeroString.length() < other.numeroString.length()) return -1;
    if (this.numeroString.length() > other.numeroString.length()) return 1;
    for (int i = 0; i < this.numeroString.length(); i++) {
        if (this.numeroString.charAt(i) > other.numeroString.charAt(i)) return 1;
        if (this.numeroString.charAt(i) < other.numeroString.charAt(i)) return -1;
    }
    return 0;
}
```

Diagrama de flujo de la función compareTo.



4. Función add para sumar dos BigNumber.

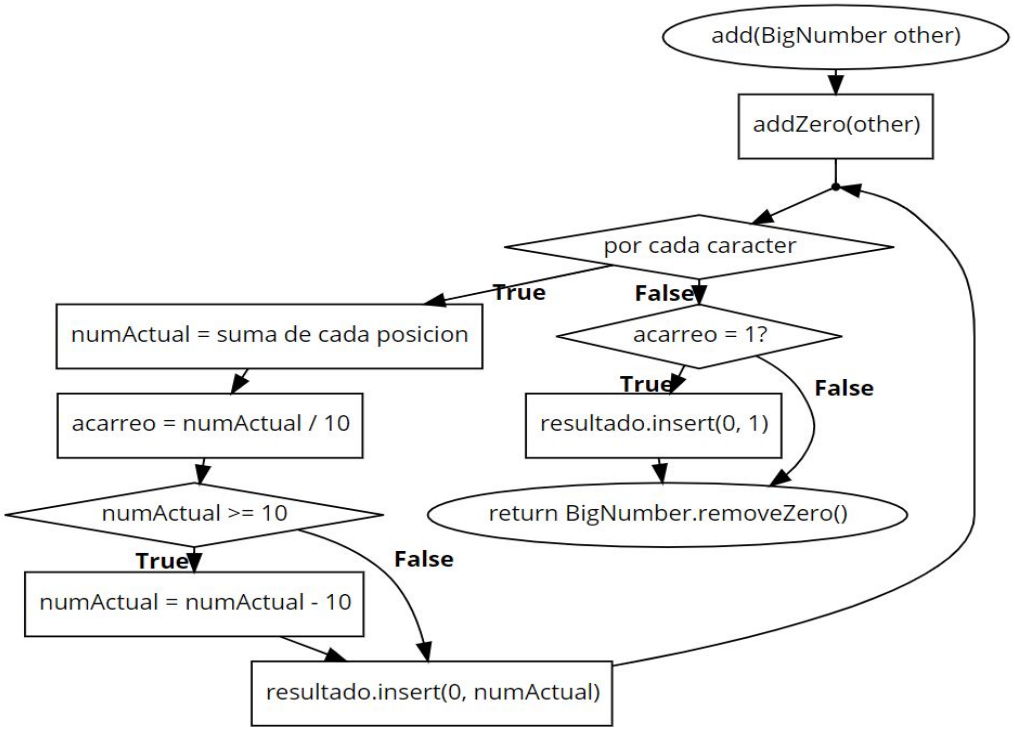
La función de la suma consiste en replicar el método que usamos a papel ya que resulta ser de los más efectivos, este consiste en ir sumando verticalmente los dígitos por columnas, cuando la suma supere el total de 10 tenemos que restarle 10 y acarrear uno a la siguiente unidad.



Para poder realizar esta operación necesitamos la función addZero para que estén igualados y poder acceder a la misma posición en ambos números. a continuación usaremos un for para ir recorriendo los dígitos de la misma posición e ir sumándolos, si la suma de estos dígitos es mayor que 10 le restamos 10 y añadiremos 1 al acarreo. Lo último que queda es ir concatenando el resultado de la suma actual y al final si el acarreo es 1 sumarlo.

```
for (int i = this.numeroString.length()-1; i >= 0 ; i--) {
    numActual = ((this.numeroString.charAt(i) -48) + (acarreo) + (other.numeroString.charAt(i)-48));
    acarreo = numActual/10;
    if (numActual >= 10){
        numActual = numActual-10;
    }
    resultado.insert( offset: 0, numActual);
}
if (acarreo == 1) resultado.insert( offset: 0, i: 1);
return new BigNumber(removeZero(resultado.toString()));
```

Diagrama de flujo de la función add.



5. Función sub para restar dos BigNumber.

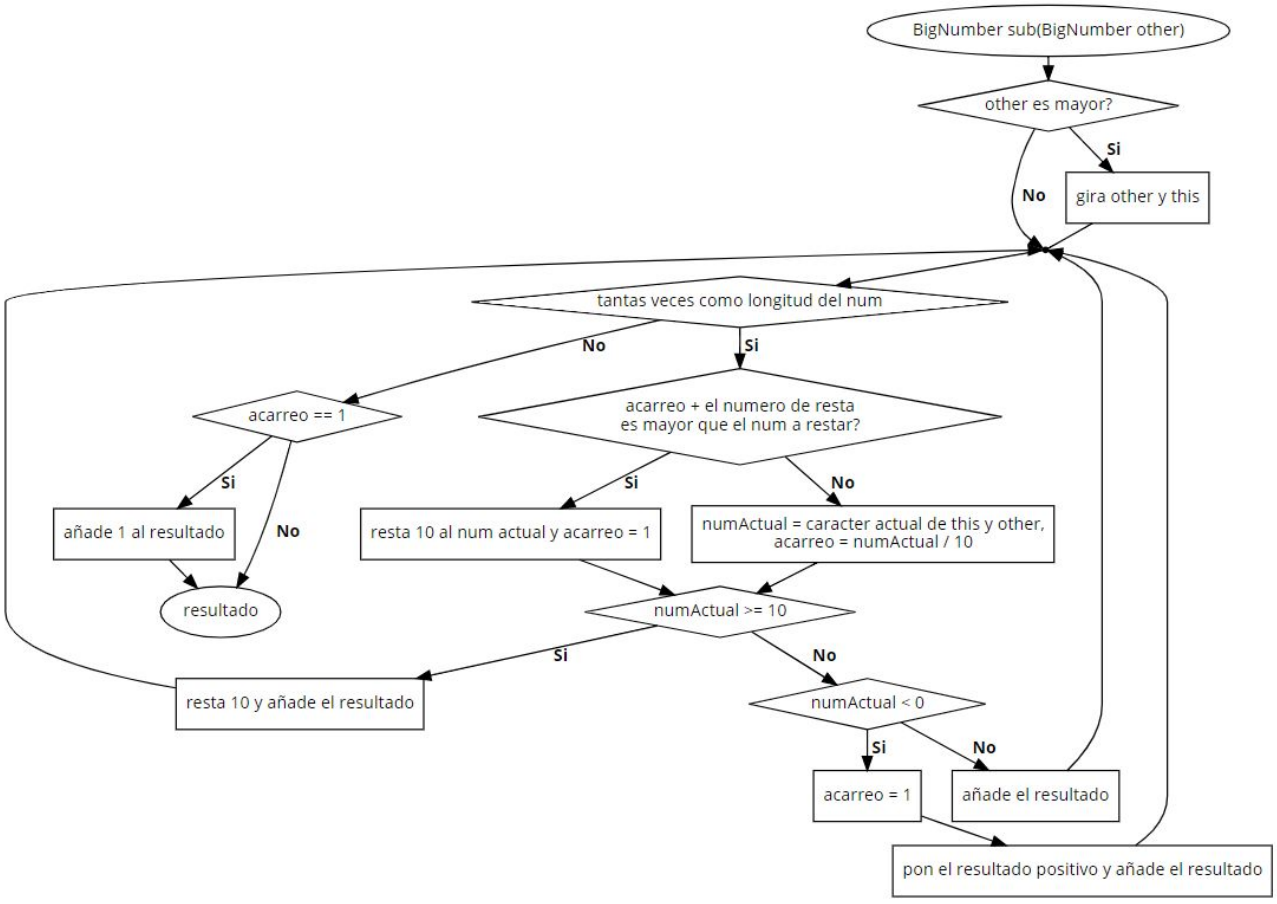
La función de resta funciona de la misma manera que la de add solo que en lugar de sumar las posiciones, se restan.

```
  1 1
  794
 -095
 ----
  699
```

Para la resta tenemos que asegurarnos de el mayor esté arriba y el menor abajo ya que si nó nos dará un número negativo para hacer las operaciones necesitamos recorrer los dos numero de derecha a izquierda y cuando el número actual de “abajo” sea mayor que el de arriba tendremos que llevarnos una.

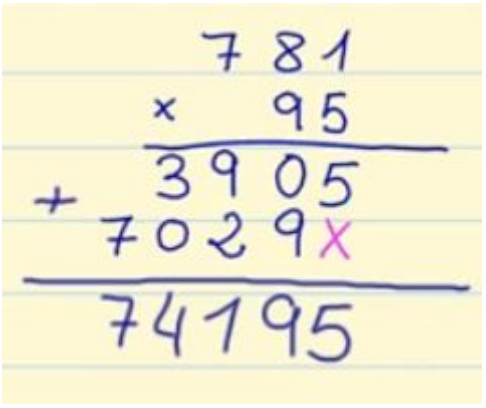
```
BigNumber sub(BigNumber other) {

    if (!this.validBigNumber(other)) return null;
    addZero(other);
    int acarreo = 0;
    int numActual;

    // Si this es inferior a other se cambian this por other.
    if (this.compareTo(other) == -1) {
        String temp = other.numeroString;
        other.numeroString = this.numeroString;
        this.numeroString = temp;
    }

    StringBuilder resultado = new StringBuilder();
    for (int i = this.numeroString.length() - 1; i >= 0; i--) {

        if ((this.numeroString.charAt(i) - 48) < ((acarreo) + (other.numeroString.charAt(i) - 48))) {
            numActual = ((this.numeroString.charAt(i) - 38) - ((acarreo) + (other.numeroString.charAt(i) - 48)));
            acarreo = 1;
        } else {
            numActual = ((this.numeroString.charAt(i) - 48) - ((acarreo) + (other.numeroString.charAt(i) - 48)));
            acarreo = numActual / 10;
        }
        if (numActual >= 10) {
            numActual = numActual - 10;
            resultado.insert( offset: 0, numActual);
        } else if (numActual < 0) {
            acarreo = 1;
            numActual = numActual * -1;
            resultado.insert( offset: 0, numActual);
        } else {
            resultado.insert( offset: 0, numActual);
        }
    }
    if (acarreo == 1) resultado.insert( offset: 0,  i: 1);
    return new BigNumber(resultado.toString());
}
```

Diagrama de flujo de la función sub.



6. Función mult para multiplicar dos BigNumber.

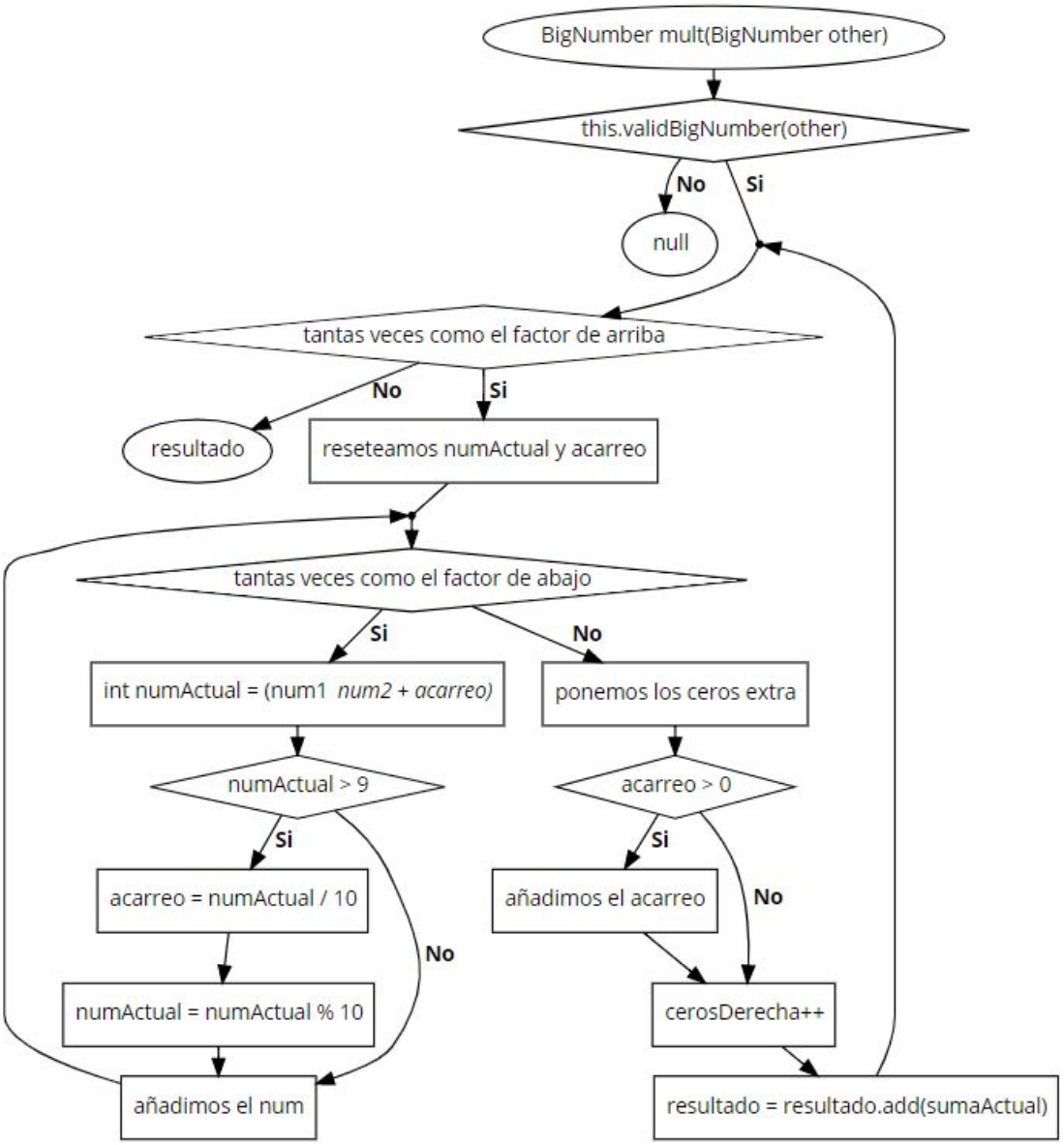
La función multiplicación consiste en hacer la multiplicación como se hace a papel, para esto tenemos que tener en cuenta los pasos necesarios para llevarla a cabo.



Es simple por cada dígito del número de abajo hay que realizar una multiplicación por todos los de arriba. por ejemplo en este caso seria 5x1, 5x8, 7x8 = 3905, cada vez que se hace un dígito se añade un 0 a el siguiente cálculo, 9x1, 9x8, 7x8 = 7029 pero como es el segundo cálculo le añadimos un 0. Una vez tenemos los cálculos hechos se suma el resultado.

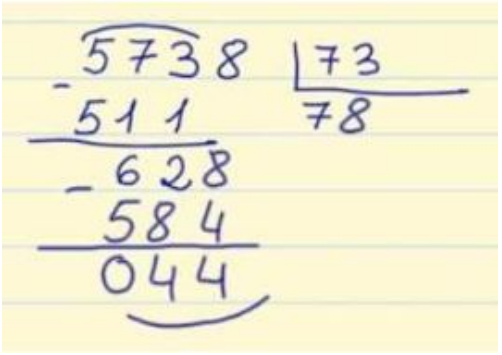
```
for (int i = this.numeroString.length() - 1; i >= 0; i--) {
    BigNumber sumaActual = new BigNumber( s: "");
    int acarreo = 0;
    for (int j = other.numeroString.length() - 1; j >= 0; j--) {
        int numActual = ((this.numeroString.charAt(i)-48) * (other.numeroString.charAt(j) -48) + acarreo);
        acarreo = 0;

        if (numActual > 9) {
            acarreo = numActual / 10;
            numActual = numActual % 10;
        }
        sumaActual.numeroString = numActual + sumaActual.numeroString;
    }
    for (int j = cerosDerecha; j > 0; j--) sumaActual.numeroString += "0";
    if (acarreo > 0) sumaActual.numeroString = acarreo + sumaActual.numeroString;
    cerosDerecha++;
    resultado = resultado.add(sumaActual);
}
return resultado;
```

Diagrama de flujo de la función mult.



7. Función div para dividir dos BigNumber.

La función de división también funciona como si estuviera hecha a papel.

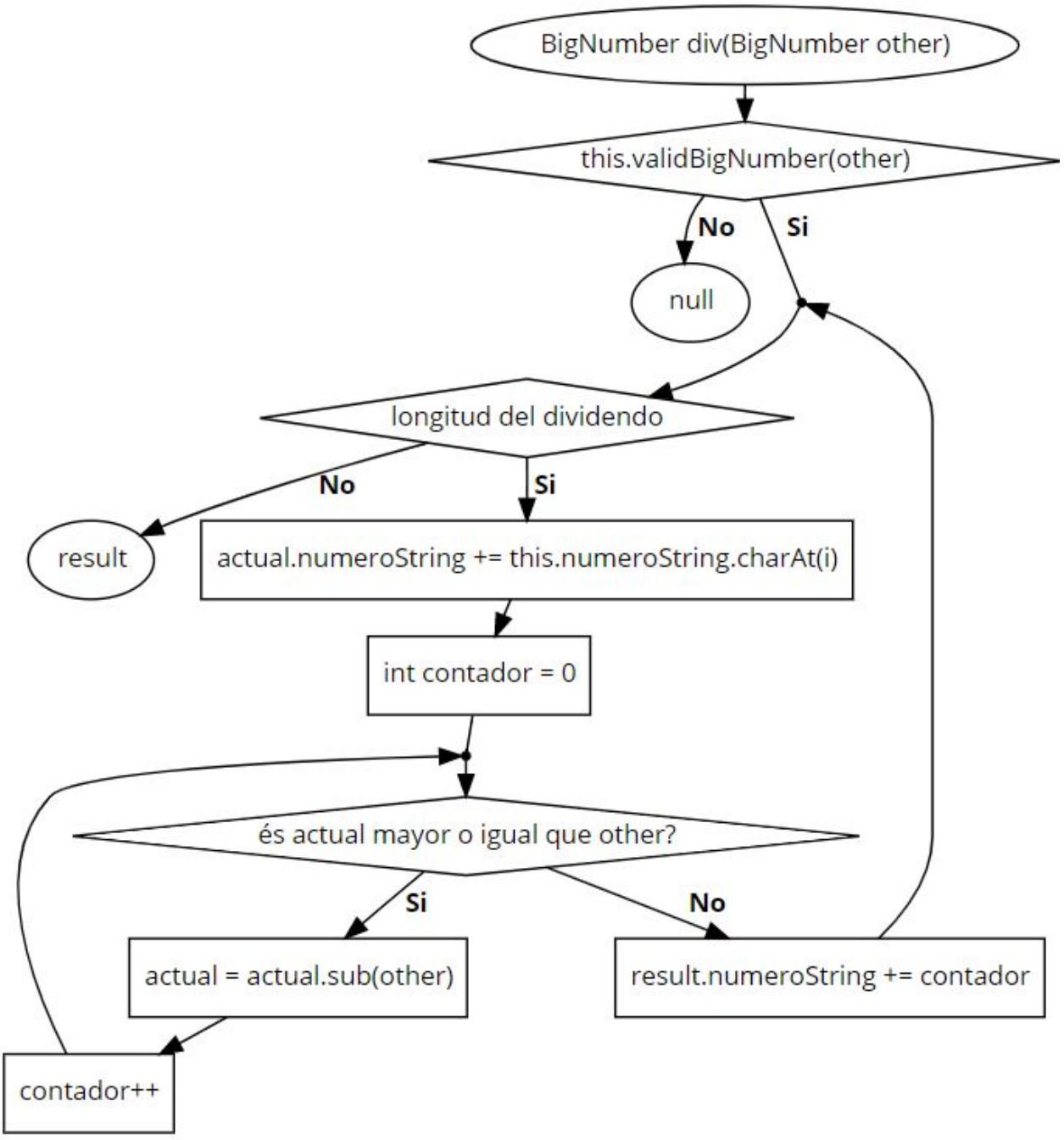


Para hacer la división lo que haremos es ir cogiendo cifras del dividendo una a una y comprobar cual es mayor, si el dividendo o el divisor, si el divisor es mayor cogeremos otra cifra más, si el dividendo es mayor le restamos el divisor tantas veces como sea necesario para que el resto de la resta sea menor que el divisor.

Para hacer esto simplemente tenemos un for que se repite tantas veces como dígitos tenga el divisor y un while que va a ir restando al dividendo el divisor y contando cuantas veces se hace para hallar el resultado, si no se puede dividir se añade un 0 al resultado y se añade un número más.

```
BigNumber div(BigNumber other) {

    if (!this.validBigNumber(other)) return null;

    BigNumber result = new BigNumber( s: "");
    BigNumber actual = new BigNumber( s: "");
    for (int i = 0; i < this.numeroString.length(); i++) {
        actual.numeroString += this.numeroString.charAt(i);
        int contador = 0;
        while(actual.compareTo(other) != -1){
                actual = actual.sub(other);
                contador++;
        }
        result.numeroString += contador;
    }
    return result;
}
```

Diagrama de flujo de la función div.



8. Función power para elevar un numero a una potencia.

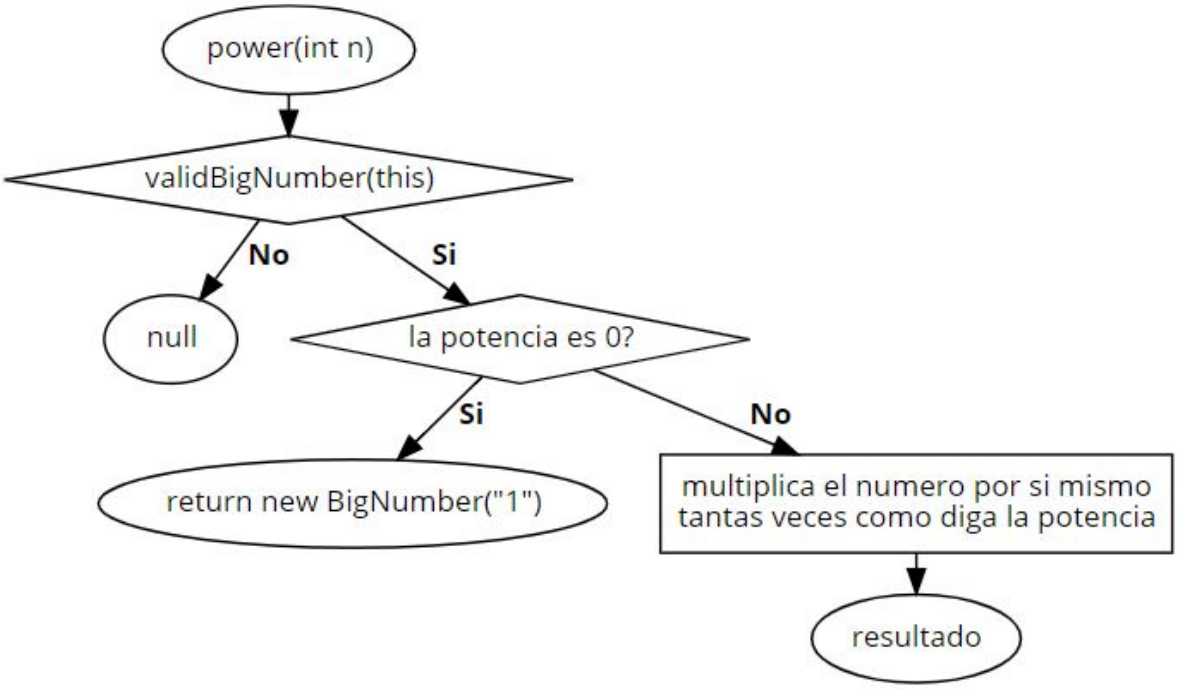
Para calcular la potencia de un número es tan simple como multiplicar el numero por si mismo tantas veces como nos diga la potencia

$$2^3 = 2x2x2 = 8$$

Simplemente necesitamos un bucle que se repita n veces donde en es la potencia, este bucle multiplicará this por el resultado. Tenemos que tener en cuenta que si la potencia es 0 tenemos que devolver 1, ya que cualquier número elevado a 0 es 1.

```
BigNumber power(int n) {

    if (!validBigNumber( other: this)) return null;
    /* Si la potencia a la que se eleva es 0 */
    if (n == 0) return new BigNumber( s: "1");

    BigNumber resultado = new BigNumber(this.numeroString);

    for (int i = n; i > 1; i--) resultado = this.mult(resultado);
    return resultado;
}
```

Diagrama de flujo de la función power.



9. Función para calcular el factorial de un BigNumber.

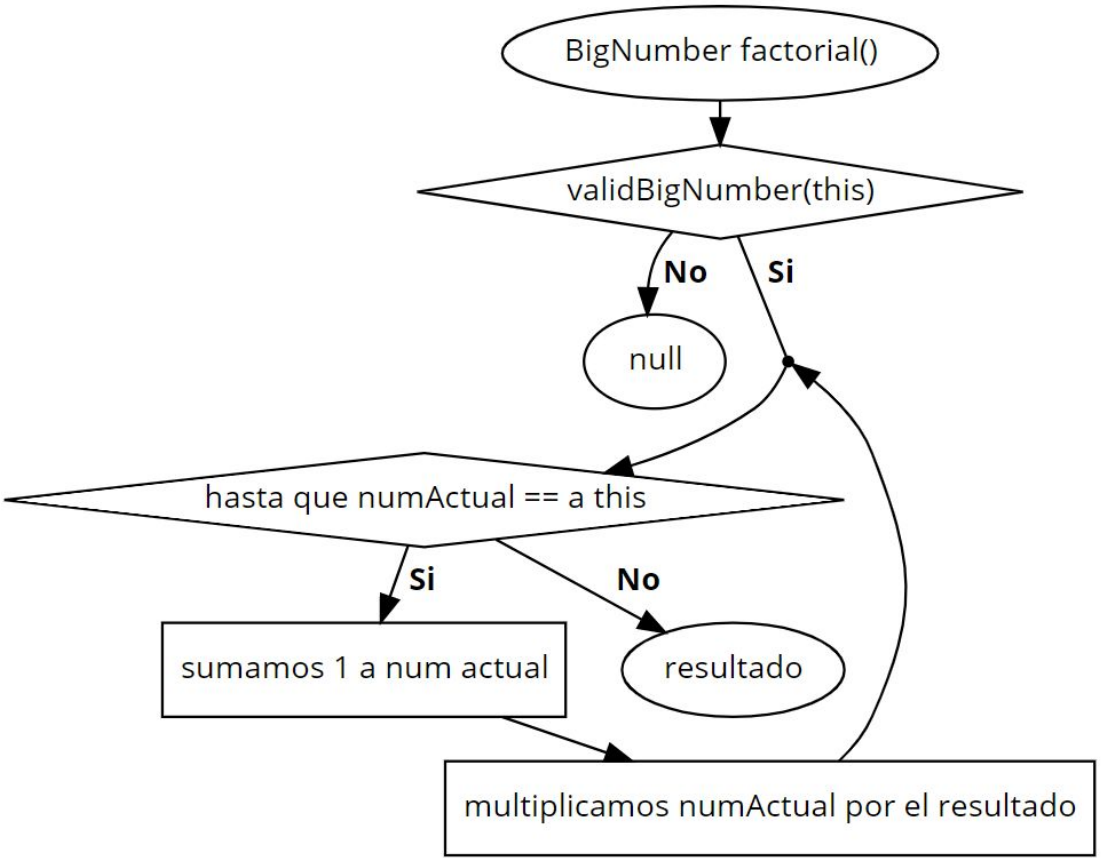
Esta función trata de calcular el factorial de un BigNumber, para calcular el factorial de un número tenemos que multiplicar ese número por todos sus inferiores hasta llegar a 1.

Factorial de 10 = 10x9x8x7x6x5x4x3x2x1 = 3628800

Para hacer el cálculo simplemente tendremos que ir multiplicando desde el número 1 hasta el número que deseemos con la condición de multiplicar hasta que el numActual sea igual que el de this, para eso usaremos la función compareTo, también se puede hacer al revés.

```
BigNumber factorial() {
    if (!validBigNumber( other: this)) return null;
    BigNumber resultado = new BigNumber( s: "1");
    BigNumber numActual = new BigNumber( s: "1");

    while (numActual.compareTo(new BigNumber(this.numeroString)) != 0) {
        numActual = (numActual.add(new BigNumber( s: "1")));
        resultado = numActual.mult(resultado);
    }
    return resultado;
}
```

Diagrama de flujo de la función factorial.



10. Función para calcular el MCD de dos BIgNumber.

Para hacer el cálculo del MCD he usado los dos algoritmos de euclides he tenido problemas de rendimiento con los dos ya que creo que mi resta no está optimizada adecuadamente y las dos lo utilizan.

De los métodos los cuales he probado el primero consiste en, simplemente restar al número mayor del que se quería sacar el mcd el más pequeño hasta que llegaran a se iguales cuando fuesen iguales ese numero seria el máximo común divisor.

```
BigNumber mcd(BigNumber other) {
    if (!this.validBigNumber(other)) return null;
    BigNumber num = new BigNumber(this.numeroString);
    while(num.compareTo(other) != 0) {
        if (this.compareTo(other) == 1) {
            num = num.sub(other);
        } else {
            other = other.sub(num);
        }
    }
    return num;
}
```

La otra forma de hacer esta operación de forma más efectiva es usando otro método de euclides el cual consiste en ir cambiando el residuo por el divisor y el divisor por el dividendo hasta llegar a 0, entonces el dividendo anterior será el MCD.

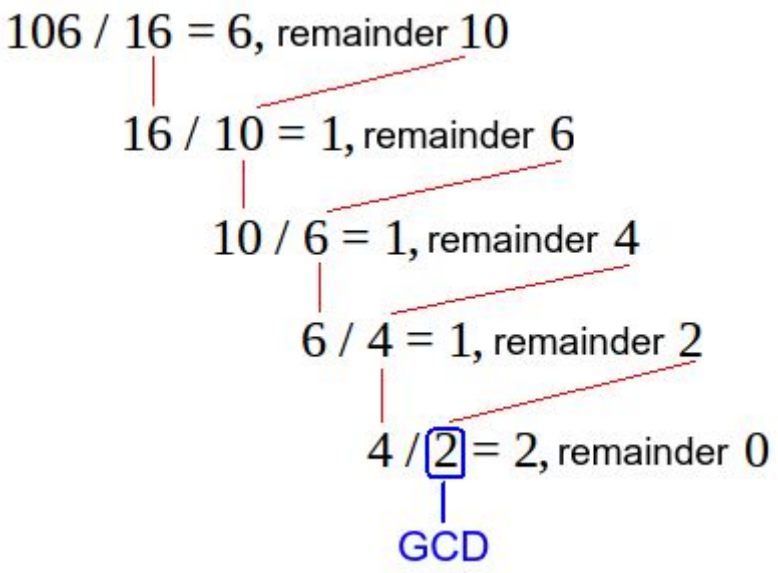
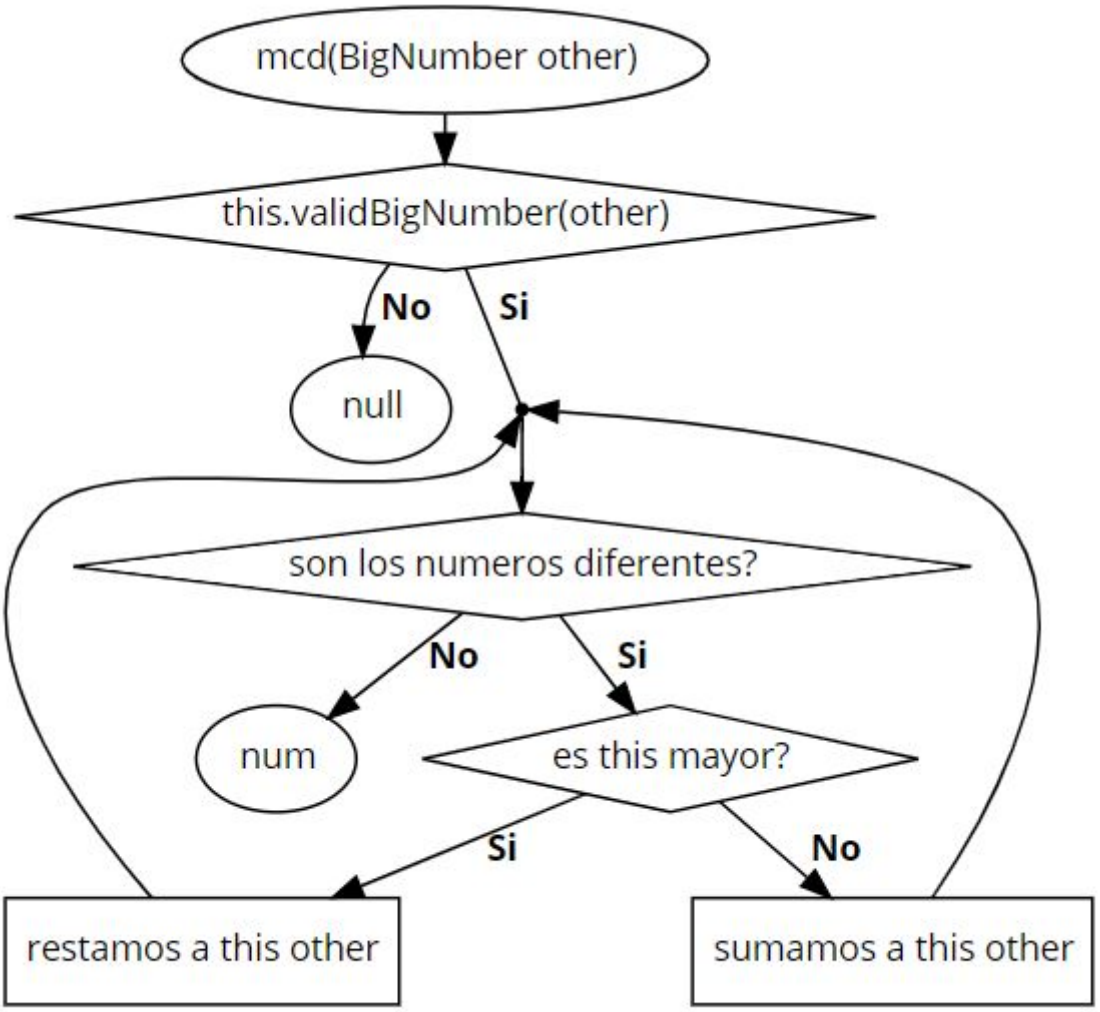
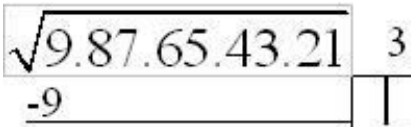


Diagrama de flujo de la función mcd.

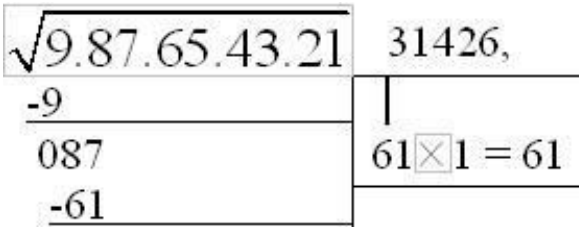


11. Función sqrt para hallar la raíz cuadrada de un BigNumber.

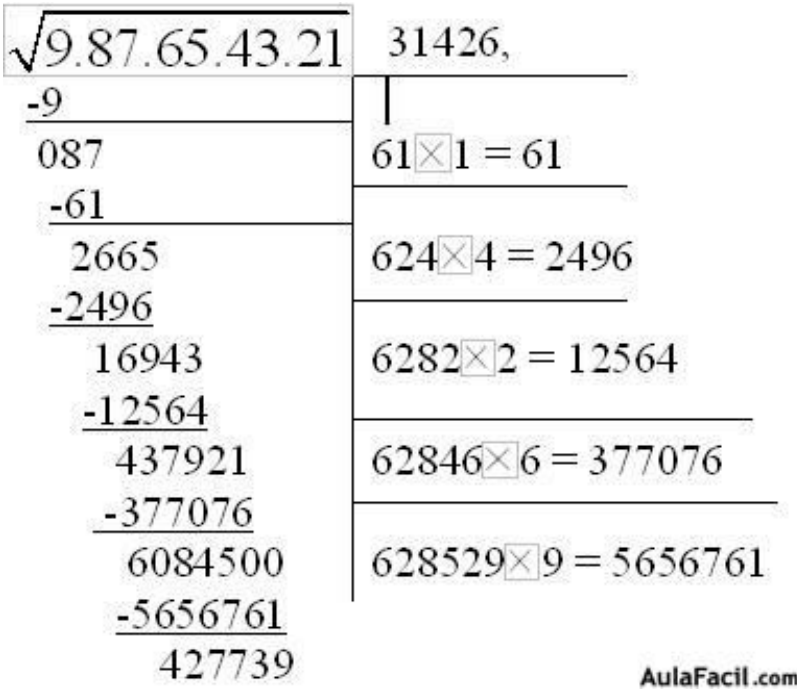
Para calcular la raíz cuadrada de un número tenemos que dividir el total del número en grupos de dos dígitos, si el número es impar le añadiremos un cero delante. A continuación tenemos que buscar un número que multiplicado se acerque lo máximo posible al primer par sin poder pasarnos, en este caso 3x3 = 9 a si que le restamos 9 al primer par de números.



A continuación bajamos el siguiente par para dividir el cual tiene el valor de 87 y también multiplicamos el número anterior x2 y buscamos un numero que multiplicando 6_ x _ se acerque lo máximo posible, en este caso sería 61x1 = 61 y lo volveríamos a restar en el residuo.



Estos pasos se repetirán tantas veces como pares de dígitos tenga el número



Para realizar estos pasos en el código crearemos una array de Strings en la que almacenaremos los pares de dígitos, usaremos un for para recorrer el array cogiendo los dígitos que necesitamos y otro for para ir haciendo las operaciones correspondientes.

Dentro de este for hay un if que es el encargado de que cuando el resultado se pase restarle 1 y hacer los cálculos con el resultado restado para calcular el resultado.

```
BigNumber sqrt() {

    BigNumber resto = new BigNumber( s: "");
    BigNumber resultado = new BigNumber( s: "");

    if ((this.numeroString.length()%2) != 0){
        this.numeroString = "0" + this.numeroString;
    }
    String[] digitArray = new String [this.numeroString.length()/2];

    for (int i = 0, j = 0; i < this.numeroString.length()/2 ; i++, j+=2) {
        String num = "" + (this.numeroString.charAt(j)) + (this.numeroString.charAt(j+1));
        digitArray[i] = num;
    }

    for (int i = 0; i <digitArray.length; i++) {

        BigNumber actual = new BigNumber( s: "");

        resto.numeroString += digitArray[i];

        for (int j = 0; j <= 9; j++) {
            StringBuilder sb = new StringBuilder();
            sb = new StringBuilder(new BigNumber(resultado).mult(new BigNumber( s: "2")).toString());
            sb = sb.append(j + 1);
            actual = new BigNumber(sb.toString());
            if (actual.mult(new BigNumber(Integer.toString( i: j+1))).compareTo(resto) == 1  ){

                sb = new StringBuilder(new BigNumber(resultado).mult(new BigNumber( s: "2")).toString());
                sb = sb.append(j);
                actual = new BigNumber(sb.toString());
                actual = actual.mult(new BigNumber(Integer.toString(j)));
                resto = resto.sub(actual);
                resultado.numeroString += Integer.toString(j);
                break;
            }
        }
    }
    return resultado;
}
```

Diagrama de flujo de la función sqrt.