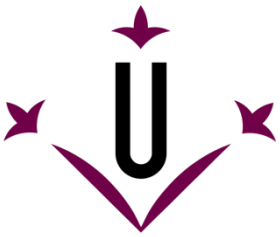


Universitat de Lleida

Escola Politècnica Superior

Grau en Enginyeria Informàtica

Programació II

Pràctica 1

BigNaturals

Aniol Serrano Ortega

GPraLab 2

Índex

1. Introducció

En aquesta pràctica es tracta de resoldre una sèrie de problemes en forma de funcions. La majoria d'aquestes funcions retornen una sèrie d'enters en forma d'*array* que contenen la solució al problema plantejat. La particularitat d'aquesta pràctica és que la representació dels números està feta de forma inversa a la natural, és a dir, el número 593 és representaria pel *array* [3, 9, 5].

L'objectiu principal d'aquesta pràctica és resoldre aquestes funcions d'operacions de forma que no hi hagi un límit determinat pel tipus de variable com pot ser *int* amb 32 bits o *long* amb 64 bits. Per a resoldre aquests problemes, en alguns dels casos s'ha hagut de programar els mètodes tradicionals d'operació. Mitjançant la combinació de mètodes prèviament implementats, és possible implementar operacions més complexes com la multiplicació o el factorial utilitzant la suma, per exemple.

2. Documentació

La documentació detallada de cada funció és important per comprendre millor la pràctica i les solucions als problemes plantejats. D'altra banda, en totes aquestes funcions se suposa que les representacions dels números són correctes. Això vol dir que totes les posicions del vector contenen dígits vàlids i que no conté dígits no significatius.

2.1. Les funcions *Zero* i *One*

Aquestes dues funcions són realment senzilles, però tenen certa utilitat per a ser usades en altres funcions. En la funció zero s'inicialitza un *array* d'enters amb una única posició i valor, en el cas de *Zero* és el 0 i en el cas de *One* és el 1.

2.2. La funció de comparació *equals*

La funció *equals* rep com a paràmetres dos *arrays* d'enters i retorna un booleà segons si aquests dos enters són iguals o no ho són. Aquests dos vectors d'enters són la representació dels seus nombres naturals.

Per a implementar aquesta funció correctament primerament s'ha de verificar primerament si la llargada d'aquests números és la mateixa, ja que al ser números correctament representats, si la llargada d'un és diferent respecte a la de l'altre, per tant, no correspon al mateix número.

Seguidament, es fa un bucle *for* per iterar sobre cada element del vector i en el moment en què es troba amb una posició on els dos números no són iguals, retorna falç. D'altra banda, si recorre els dos vectors i no troba cap diferència entre cap parell d'enters, llavors retorna verdader.

2.3. La suma de dos números

n

EXPLICACIO

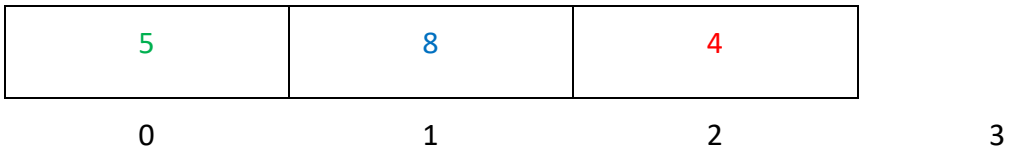
Este código es una función llamada **add** que toma dos arrays de enteros **num1** y **num2**, y devuelve un nuevo array que contiene la suma de los dos números. Aquí te explico en detalle lo que hace cada sección del código:
En las líneas 2 y 5, el código comprueba si alguno de los números es cero. Si es así, devuelve el otro número sin procesarlo, ya que sumar cero no tiene ningún efecto.
En las líneas 8-10, el código determina la longitud del array de resultado, que será la longitud máxima entre **num1** y **num2**. También crea un nuevo array de enteros llamado **result** que es lo suficientemente grande para contener el resultado de la suma, más un espacio adicional para el caso en que se tenga que agregar un dígito de acarreo.
En las líneas 13-25, el código realiza la suma de los dos números dígito por dígito, comenzando desde los dígitos menos significativos. Si los números tienen diferentes longitudes, el código llena los dígitos faltantes con ceros. Cada dígito es sumado junto con el dígito de acarreo de la suma anterior, si es que existe. Si el resultado de la suma es mayor que 9, se establece el dígito de acarreo en 1, de lo contrario se establece en 0. El resultado se almacena en el array **result** en la posición correspondiente.
En la línea 27, el código comprueba si hay un dígito de acarreo sobrante al final de la suma. Si es así, se agrega al array de resultado.
En las líneas 30-40, el código elimina los ceros sobrantes al final del array de resultado. Comienza comprobando el último dígito del array. Si es cero, mueve el índice hacia atrás y repite la comprobación hasta que encuentre el último dígito no cero. Luego crea un nuevo array de enteros llamado **shorterResult** que contiene solo los dígitos no cero del resultado y devuelve este nuevo array.
Finalmente, en la línea 43, la función devuelve el array **result** que contiene la suma de los dos números con ceros sobrantes eliminados

---

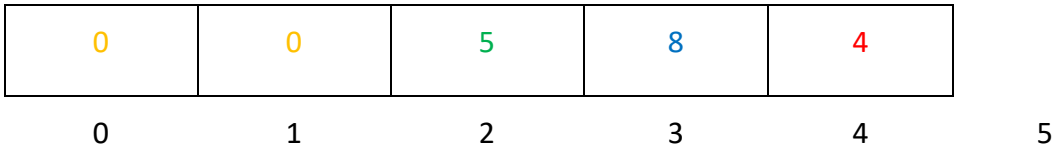
2.4. El desplaçament cap a l'esquerra

La funció *shiftLeft* rep com a paràmetres un *array* d'enters anomenat *number* i un enter anomenat *positions*. Aquest mètode retorna aquest *array* d'enters però els seus enters desplaçats a la dreta tantes posicions com el valor de *positions*, és a dir, s'afegeixen numeros al final de la representació natural del nombre.

Per a veure que fa aquesta funció de forma més esquemàtica, es pot veure en la següent figura la representació del número 485 abans i després d'usar la funció *shitLeft*:



- *Positions* = 2



Que fa referència al número: 48500.

Per resoldre aquesta funció, primer s'ha de verificar si es compleixen les condicions necessàries per aplicar el desplaçament. Això implica comprovar si l'array conté només un zero o si no hi ha cap desplaçament perquè la variable *positions* és igual a zero. En cas de no complir aquestes condicions, es retorna el número original.

Seguidament, es crea un *array* anomenat *result* per a emmagatzemar el resultat amb les posicions originals més les ocasionades pel desplaçament. Com que en Java al crear un *array* s'inicialitza tot a zeros, no és necessari afegir els zeros del desplaçament. Finalment, es copia el número original en les posicions desplaçades del vector i es retorna el resultat.

2.5. Multiplicació d'un número per un dígit

n

2.6. Multiplicació de dos números

n

2.7. El Factorial

n

2.8. El Fibonacci

n

3. Conclusions

Gràcies a la bona previsió de la pràctica, s'ha pogut resoldre els problemes de múltiples formes per trobar una solució millor.

Mencionar funcions descartades

Annex

a. Versió alternativa de la funció *multiplyByDigit*

En aquesta versió en comptes d'utilitzar les funcions prèviament definides com són la funció *zero()* i *add()*, es resol el problema de forma totalment aïllada de les altres funcions. Aquesta solució funciona correctament, però la seva llegibilitat és clarament pitjor a més que no cohesiona el codi i, per tant, no fa ús del treball prèviament fet.

```
public int[] multiplyByDigit(int[] number, int digit) {
        if ((number.length == 1 && number[0] == 0) || digit == 0) {
            return new int[]{0};
        }

        int carry = 0, product;
        int[] result = new int[number.length + 1];
        for (int i = 0; i < number.length; i++) {
            product = (digit * number[i]) + carry;
            result[i] = product % 10;
            carry = product / 10;
        }

        if (carry > 0) {
            result[number.length] = carry;
            return result;
        }

        int[] shortResult = new int[result.length - 1];
        for (int i = 0; i < shortResult.length; i++) {
            shortResult[i] = result[i];
        }
        return shortResult;
    }
```

En canvi, el codi final ha estat molt més esclaridor, ja que sí que usa les funcions prèviament realitzades i, per tant, el codi resulta més senzill i intuïtiu.

```
public int[] multiplyByDigit(int[] number, int digit) {
        if ((number.length == 1 && number[0] == 0) || digit == 0) {
            zero();
        }

        int[] result = zero();
        for (int i = 1; i <= digit; i++) {
            result = add(result, number);
        }
        return result;
    }
```

b. Funcions auxiliars descartades

Aquestes versions són funcions que s'havien implementat primerament però no funcionen per a números molt grans, i per tant, no implementen correctament la solució d'acord amb l'enunciat de la pràctica. Igualment aquestes funcions auxiliars són útils en cas que s'usés per a números que no provoquen desbordament, és a dir, petits.

```
// From int gets the length as if it was in one array
public int getLength(int num) {
        int length = 1;
```

```
        while (num >= 10) {
            num /= 10;
            length++;
        }
        return length;
    }
```

```
// From array returns it in an int
public int getIntFromArray(int[] num) {
        int result = 0;
        for (int i = 0; i < num.length; i++) {
            int currentNum = num[i];
            result += currentNum * Math.pow(10, (num.length - i - 1));
        }
        return result;
    }
```