PRACTICO N° 3

Diseño de algoritmos iterativos y planteos recursivos
Patrones de algoritmos
Implementación en Java

EJERCICIO 1. Dado un número natural **N**=d_m,d_{m-1},...,d_2,d_1 y las siguientes definiciones:

- La **suma de los dígitos** se define como sumadig=$d_m + d_{m-1} + ... + d_2 + d_1$
- Un dígito **dig** está en **N** si dig =d_i para algún i, 1<=i<=m
- Un número natural **N** verifica la propiedad **es creciente** si $d_i < d_{i-1}$ para 1<i<=m

a. Complete la clase *ProcesadorNumero* implementando cada definición:

```
class ProcesadorNumero {
/*Cada uno de los servicios provistos por la clase recibe un número natural n y lo
procesa para computar un valor*/
public static int sumaDig(int n){
/* Retorna la suma de los dígitos del número natural n */
}
public static boolean estaDig(int n,int d){
/* Retorna true si y solo si d es un dígito del número natural n */
}
public static boolean esCreciente(int n){
    /* Retorna true si y solo si los dígitos de n están ordenados en forma
estrictamente creciente (desde la posición más significativa hacia la menos
significativa) */
}
}
```

OBSERVACIÓN: PARTE DEL OBJETIVO DE ESTE EJERCICIO ES QUE SE FAMILIARICE CON LA NOTACIÓN DEL ENUNCIADO.

b. Agregue a la clase ProcesadorNumero la implementación de un método:

```
public static boolean esPerfecto (int n){
/*Un número es perfecto si es igual a la suma de sus divisores, excluyéndolo a sí
mismo */
}
```

c. Agregue a la clase ProcesadorNumero los siguientes métodos recursivos considerando que cada uno de ellos corresponda a un planteo recursivo:

```
public static int contadorDig(int n, int d){
/* Retorna la cantidad de apariciones del dígito d en el número natural n*/
}
public static int mayorDigito(int n){
/* Retorna el mayor dígito de n */
}
```

d. Agregue a la clase ProcesadorNumero el método todosImpares,

```
public static boolean todosImpares(int n){
/* Retorna true si y solo si todos los dígitos de n son impares */
}
```

implementando el siguiente planteo recursivo:

Caso Trivial: Un número con un dígito cumple la propiedad todosImpares sí y solo sí el dígito es impar.
Caso Trivial: Un número con dos o más dígitos NO cumple la propiedad todosImpares si el dígito menos significativo es par.
Caso Recursivo: Un número con dos o más dígitos cumple la propiedad todosImpares si el dígito menos significativo es impar y N' cumple la propiedad, siendo N' igual a N sin su dígito menos significativo.

e. Implemente una clase tester para verificar los servicios de ProcesadorNumero con casos significativos.

EJERCICIO 2. **Dada una secuencia S de n elementos leídos por consola**

a) Analice qué algoritmo corresponde a cada problema:

i. Contar la cantidad de apariciones de elementos de S que cumplen una propiedad p.
ii. Decidir si algún elemento de S cumple una propiedad p.
iii. Decidir si todos los elementos de S cumplen una propiedad p.
iv. Computar la posición de la primera aparición de un elemento de S que cumple una propiedad p. La posición es 0 si ninguno cumple la propiedad p.
v. Decidir si hay al menos m elementos seguidos de S que cumplen una propiedad p.

```
propiedad ← true
para i tomando valores entre 1 y n, mientras se cumple la propiedad
  leer s
  si s NO cumple la propiedad p
    propiedad ← falso
```

```
valor ← 0
para i tomando valores entre 1 y n, mientras valor = 0
  leer s
  si s cumple la propiedad p
    valor ← i
```

```
valor ← 0
para i tomando valores entre 1 y n, mientras valor < m y i+(m-valor)-1<=n
  leer s
  si S cumple la propiedad p
    valor ←valor + 1
  sino
    valor ← 0
```

```
valor ← 0
para i tomando valores entre 1 y n
  leer s
  si s cumple la propiedad p
    valor ← valor + 1
```

```
valor← falso
para i tomando valores entre 1 y n, mientras valor es falso
  leer s
  si s cumple la propiedad p
    valor ← verdadero
```

b) El siguiente algoritmo decide si S contiene al menos m elementos que cumplen una propiedad p. Analice su eficiencia.

```
cont ← 0
para i tomando valores entre 1 y n
  leer s
  si s cumple la propiedad p
    cont ← cont + 1
si cont >= m cumple = verdadero
sino cumple = falso
```

c) Muestre una solución eficiente para cada uno de los dos enunciados que siguen y describa qué diferencia al problema y qué diferencia a la solución.

- Decidir si una secuencia S tiene al menos m elementos que cumplen la propiedad p.
- Decidir si una secuencia S tiene exactamente m elementos que cumplen la propiedad p.

d) Diseñe algoritmos para los siguientes problemas:

i. Computar la posición de la última aparición de un elemento de S que cumple una propiedad p. La posición es 0 si ninguno cumple la propiedad p.
ii. Decidir si hay al menos 2 elementos de S que cumplen una propiedad p, separados por otro elemento que no cumple la propiedad.

EJERCICIO 3. **Procesar una secuencia *S*** de **n** números $s_1\ s_2\ \ldots\ s_n$, con **n** > 0, **leídos por consola**

a) Complete la clase SecuenciaConsola escribiendo implementaciones iterativas:

```
class SecuenciaConsola {
/*Cada uno de los servicios provistos por la clase lee una secuencia de n números
enteros por consola y computa un valor*/
public static int primer(int n){
/* Retorna el primer elemento mayor que 100 que aparece en la secuencia*/}
public static int ultimaPosicion(int n, int m){
/* Retorna la posición de la última aparición de m, si no existe retorna 0*/}
public static int sumaSec(int n){
/* Retorna la suma de los números de la secuencia*/}
public static boolean estaNum(int n,int num){
/* Retorna true si y solo si num es está en la secuencia leída */}
public static boolean esCreciente(int n){
/* Retorna true si y solo si los elementos de la secuencia están ordenados en forma
estrictamente creciente (desde el primero en ser leído hasta el último) */}
}
```

b) Para cada uno de los siguientes servicios escriba un planteo recursivo, implemente un método consistente con el planteo e inclúyalo en la clase SecuenciaConsola.

```
public static int contadorNum(int n,int num){
/* Retorna la cantidad de apariciones de num en la secuencia de n elementos*/
public static int mayorNum(int n){
/* Retorna el mayor número de la secuencia de n elementos */
}
```

c) Implemente una clase tester para SecuenciaConsola.

d) Usando los servicios provistos por ProcesadorNumero, aumente los servicios provistos por la clase SecuenciaConsola completando el código de los métodos iterativos:

```
public static int mayorPar(int n){
/* Retorna el mayor número par en la secuencia*/
}
public static int contarPerfectos(int n){
/* Computa la cantidad de números perfectos de la secuencia*/
}
public static boolean dosTodosImpares(int n){
/* Retorna true si solo si la secuencia contiene dos números seguidos, con todos los
dígitos impares */
}
}
```

e) Dada la siguiente definición, proponga un planteo recursivo para computar la suma de productos de una secuencia S de n números enteros. Definimos la *suma de productos* de S como:

$d_1*d_n + d_2*d_{n-1} + ... + d_{n/2}*d_{(n/2)+1}$ cuando **n** es par y

$d_1*d_n + d_2*d_{n-1} + ... + d_{(n/2)+1}$ cuando **n** es impar.

Por ejemplo

para **n** = 6 y **S** = 10 4 3 5 7 8 entonces ***M*** = 10* 8 + 4 * 7 + 3 * 5 = 123

para **n** = 7 y **S** = 10 4 3 5 7 8 9 entonces ***M*** = 10 * 9 + 4 * 8 + 3 * 7 + 5 = 148

Implemente un método recursivo consistente con el planteo e inclúyalo en la clase SecuenciaConsola.

f) Implemente una clase tester que verifique los nuevos servicios.

EJERCICIO 4. Procesar una secuencia de n caracteres leída por consola

a) Implemente soluciones iterativas para cada uno de los servicios de la clase *SecuenciaCaracteres*:

```
class SecuenciaCaracteres {
/*Cada uno de los servicios provistos por la clase lee una secuencia de n caracteres
por consola y computa un valor*/
public static boolean dosVocales (int n){
/* retorna true si la secuencia de n caracteres contiene dos vocales seguidas*/
}
public static boolean estanTodas(int n){
/* retorna true si la secuencia de n caracteres contiene al menos una aparición de cada
letra mayúscula*/
}
}
```

b) Para cada uno de los siguientes servicios escriba un planteo recursivo, implemente un método consistente con el planteo e inclúyalo en la clase SecuenciaCaracteres.

```
public static int contadorChar(int n,char ch){
/* Retorna la cantidad de apariciones de ch en la secuencia */
}
public static boolean todasLetras(int n){
/* Retorna true si y solo si todas los caracteres de la secuencia son letras*/
}
```

c) Implemente una clase tester que verifique los servicios provistos por la clase SecuenciaCaracteres

d) Agregue un método maxDistancia(n) que compute la máxima distancia entre dos vocales implementando el siguiente algoritmo:

```
Algoritmo maxDistancia
maxdis ← -1
Buscar la posición de la primera vocal
si existe
  maxdis ← 0   dis ← 0
 mientras no se termina la secuencia S
   leer s
   dis ← dis + 1
   si s es una vocal
     si dis > maxdis
        maxdis = dis
     dis = 0
```

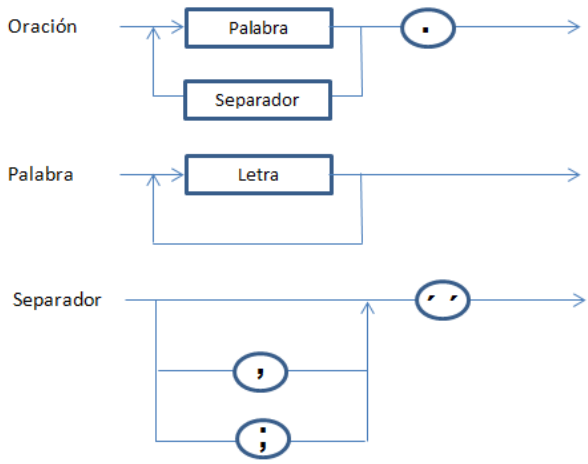
OBSERVE QUE EL ALGORITMO COMPUTA -1 SI NO HAY NINGUNA VOCAL Y 0 SI SOLO HAY UNA VOCAL.

e) Una secuencia **S** = $a_1\ a_2\ ..\ a_k\ b_k\ ..\ b_2\ b_1$ de **longitud par** (2k) se dice que "tiene algún reflejo" si existe i en [1..k] tal que $a_i=b_i$.
Por ejemplo
* **S** = abcdbb (longitud=6) "tiene algún reflejo" (ya que $a_2=b_2=b$).
* **S** = abab (longitud=4) **NO** "tiene algún reflejo"
* Si S es vacía, **NO** "tiene algún reflejo".
Dada la definición de tiene algún reflejo, proponga un planteo recursivo para decidir si una secuencia de caracteres tiene algún reflejo.

f) Dada una secuencia de caracteres S = $a_1\ a_2\ ..\ a_k\ b_k\ ..\ b_2\ b_1$ de **longitud par** (2k) su secuencia "explosiva desde el centro" se genera como $a_k\ b_k\ a_{k-1}\ b_{k-1}\ ...\ a_1\ b_1$
Proponga un planteo recursivo para generar la secuencia explosiva desde el centro de una secuencia S como la descripta anteriormente.

g) Implemente los métodos tieneAlgunReflejo(n) y explosiva(n), inclúyalos en la clase SecuenciaCaracteres y modifique la clase tester para verificar estos servicios con casos de prueba significativos.

EJERCICIO 5. **Procesar Oracion** Dado el siguiente diagrama que describe la forma de una oración:



a. Analice si las siguientes cadenas de caracteres son válidas de acuerdo al diagrama sintáctico:
"Martes, a la mañana."
"Martes,a la mañana."
"Martes, a la mañana."
"Martes, a la mañana ."
"Martes, a la 12 hs."
"Martes, no. Lunes."

b. Para cada uno de los servicios provistos por la clase Oracion diseñe un algoritmo y luego implemente la clase completa:

```
class Oracion{
public static int cantLetras (){
/* Lee una oración y retorna la cantidad letras */
}
public static int cantPalabras (){
/* Lee una oración y retorna la cantidad Palabras */
}
public static int cantPalabrasLetra (char ch){
/* Lee una oración y retorna la cantidad  de palabras que empiezan con el carácter
ch*/
}
```

c. Proponga un algoritmo informal para computar la longitud de la palabra más larga de una oración, implemente el algoritmo propuesto e incluya este servicio en la clase Oracion.

d. Defina una clase tester para verificar los servicios de la clase Oracion.

EJERCICIO 6. **Procesar una secuencia S de números enteros**, **leídos por archivo**

a. Implemente una clase SecuenciaArchivo con la misma funcionalidad que SecuenciaConsola pero con la secuencia S leída de un archivo, cuyo nombre es recibido como parámetro por cada servicio. En este caso no se conoce la cantidad de elementos en el momento que comienza la lectura.
b. Implemente una clase tester para SecuenciaArchivo.

Observación
Los planteos recursivos con dos casos triviales se modelan en Pascal con dos condicionales. En Java el cortocircuito permite modelar el problema con un solo condicional, pero es importante notar que sigue habiendo dos condiciones de corte.