Esquemas Iterativos y Recursivos con Secuencias y Acumuladores

Análisis y Diseño de Datos y Algoritmos
Estructuras de Datos y Algoritmos

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA INFORMÁTICA
Departamento de Lenguajes y Sistemas Informáticos
Curso 2022-2023

Índice

- Secuencias
  - Factorías de secuencias
- Acumuladores
  - Factorías de acumuladores
- Transformación de algoritmos iterativos y recursivos
- Diseño de algoritmos iterativos

Notación

- Rangos – [a, b), [a, b]
- Tuplas – t = (t0, t1, … , tn-1)
- Listas – ls = [e0, e1, … , en-1], |ls|, ls[a,b], ls[i], []
- Conjuntos – ss = {e0, e1, … , en-1}, |ss|, {}
- Multiconjuntos – ms = {e0:m0, e1:m1, … , en-1:mn-1}
- Diccionarios – m = {k0:v0, k1:v1, … , kn-1:vn-1}, m[ki]
- Agregados de datos – d
- Secuencias – s

Algoritmos iterativos

Ejemplo iterativo

```
Double sumLista(List<Double> list) {
      Integer i = 0;
      Double b = 0.;
      while(i < list.size()){
         b = b + list.get(i);
         i = i+1;
      }
      return b;
}
```

Versión funcional

```
Double sumListaFunc(List<Double> list) {
      IntStream s = IntStream.range(0,list.size());
      DoubleStream s2 = s.mapToDouble(i->list.get(i));
      return s2.sum();
}
```

Algoritmos recursivos:tipos

Algoritmo recursivo final:

$$mcd(a,b) = \begin{cases} a, & b = 0 \\ mcd(b, a\%b), & a, b > 0 \end{cases}$$

Algoritmo recursivo simple no final:

$$n! = \begin{cases} 1, & n = 0 \\ n * (n-1)!, & n > 0 \end{cases}$$

Algoritmo recursivo múltiple:

$$fib(n) = \begin{cases} n, & 0 \leq n \leq 1 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

Algoritmos recursivos

```
Double sumLista(List<Double> list) {
        return sumLista(0,0.,list);
}

Double sumLista(Integer i, Double b, List<Double> list) {
        if(i<list.size()){
                Double e = list.get(i);
                b = sumLista(i+1,b+e,list);
        }
        return b;
}

Double r = sumLista(List.of(3.,4.,11.,10.));
```

i	0	1	2	3	4	3	2	1.	0
b	0.	3.	7.	18.	28.	28.	28.	28.	28.
r									28.

Algoritmos iterativos

Estructura clásica general

```
R f(x) {
        E e = e0;
        B b = b0;
        while(g(e)){
                b = c(b,e);
                e = nx(e);
        }
        return r(b);
}
```

Esquema funcional (abstracto)

```
R f(x) (
     S s = Stream.iterate(e0, e->g(e), e->nx(e));
     Collector cl = (b0, (b,e)->c(b,e), b->r(b), ...);
     return s.collect(cl);
}
```

Algoritmos iterativos

Estructura clásica general

```
R f(x) {
        E e = e0;
        B b = b0;
        while(g(e)){
                b = c(b,e);
                e = nx(e);
        }
        return r(b);
}
```

secuencia

Esquema funcional (abstracto)

```
R f(x) (
     S s = Stream.iterate(e0, e->g(e), e->nx(e));
     Collector cl = (b0, (b,e)->c(b,e), b->r(b), ...);
     return s.collect(cl);
}
```

Algoritmos iterativos

Estructura clásica general

acumulador

```
R f(x) {
        E e = e0;
        B b = b0;
        while(g(e)){
                b = c(b,e);
                e = nx(e);
        }
        return r(b);
}
```

Esquema funcional (abstracto)

```
R f(x) (
     S s = Stream.iterate(e0, e->g(e), e->nx(e));
     Collector cl = (b0, (b,e)->c(b,e), b->r(b), ...);
     return s.collect(cl);
}
```

Recursividad final y algoritmos iterativos

Esquema iterativo

```
R f(x) {
        E e = e0;
        B b = b0;
        while(g(e)){
                b = c(b,e);
                e = nx(e);
        }
        return r(b);
}
```

Esquema recursivo

```
R f(X x){
        E e = e0;
        B b = b0;
        b = f(e,b,x);
        return r(b);
}
B f(E e, B b, X x){
        if(g(e)){
                b = f(nx(e),c(b,e),x);
        }
        return b;
}
```

Recursividad final y algoritmos iterativos

Esquema iterativo

```
R f(x) {
        E e = e0;
        B b = b0;
        while(g(e)){
                b = c(b,e);
                e = nx(e);
        }
        return r(b);
}
```

Esquema recursivo

```
R f(X x){
        E e = e0;
        B b = b0;
        b = f(e,b,x);
        return r(b);
}
B f(E e, B b, X x){
        if(g(e)){
                b = f(nx(e),c(b,e),x);
        }
        return b;
}
```

secuencia

Recursividad final y algoritmos iterativos

Esquema iterativo

```
R f(x) {
        E e = e0;
        B b = b0;
        while(g(e)){
                b = c(b,e);
                e = nx(e);
        }
        return r(b);
}
```

Acumulador

Esquema recursivo

```
R f(X x){
        E e = e0;
        B b = b0;
        b = f(e,b,x);
        return r(b);
}
B f(E e, B b, X x){
        if(g(e)){
                b = f(nx(e),c(b,e),x);
        }
        return b;
}
```

Transformación de alg. iterativos

Ejemplo, funcional a imperativo: Sumar los enteros contenidos en una lista

```
public static Integer ejemplo1Java8(List<Integer> ls) {
      return ls.stream().mapToInt(x -> x).sum();
}
```

```
public static Integer ejemplo1Java7For(List<Integer> ls) {
      Integer a = 0;
      for (Integer e: ls) {
            a = a + e;
      }
      return a;
}
```

```
public static Integer ejemplo1Java7While(List<Integer> ls) {
      Integer a = 0;
      Integer i = 0;
      while (i < ls.size()) {
            Integer e = ls.get(i);
            a = a + e;
            i = i + 1;
      }
      return a;
}
```

Transformación de alg. iterativos

Ejemplo, funcional a imperativo: Sumar los enteros contenidos en una lista

```
public static Integer ejemplo1Java7While(List<Integer> ls) {
      Integer a = 0;
      Integer i = 0;
      while (i < ls.size()) {
            Integer e = ls.get(i);
            a = a + e;
            i = i + 1;
      }
      return a;
}
```

```
Integer ejemplo1Java7Fnl(List<Integer> ls){
        Integer e = 0;
        Integer b = 0;
        b = ejemplo1Java7Fnl(e,b,ls);
        return b;
}
Integer ejemplo1Java7Fnl(Integer i, Integer b,
                                  List<Integer> ls){
        if(i<ls.size()){
                b = ejemplo1Java7Fnl(i+1,b+ls.get(i));
        }
        return b;
}
```

Implementación de secuencias

Son flujos secuenciales de elementos que se pueden recorrer uno detrás de otro

Algoritmo iterativo → tiene asociada una secuencia finita

Tipos más utilizados para su implementación:

- Stream<E> en Java
- Iterator<E> en Java

```
interface Iterator<E> {
        boolean hasNext(); //Indica si hay más elementos
        E next(); //Devuelve el siguiente y pone otro disponible
        …
}
```

- Cada iterator se concreta en una clase que implementa Iterator<E>
- Podemos obtener un iterator de un stream con el método iterator()

Implementación de secuencias

```
public interfaz Iterable<E> {    // Interfaz iterable
    public Iterator<E> iterator();
}
public class ArrayList implements List<E>, Iterable<E> {
    public Iterator<E> iterator() {
        return new ListIterator(this); // Clase interna que itera sobre la lista
    }
}

List<E> ls = new ArrayList<>();
Iterator<E> it = ls.iterator();
while (it.hasNext()) {   //  Uso de un iterador
   E o = it.next();
   System.out.println(o);
}

for (E o: ls) {   // for extendido
    System.out.println(o);
}
```

Implementación de secuencias

```
public interfaz Iterable<E> {    // Interfaz iterable
    public Iterator<E> iterator();
}
publi
    pu

    }
}

List<I
Iterat
while
   E e
   Sy
}

for (E
   Sy
}
```

```
public class ListIterator<E> implements Iterator<E> {
        private List<E> ls;
        private Integer i;
        public ListIterator(List<E> ls) {
                super();
                this.ls = ls;
                this.i = 0;
        }
        @Override
        public boolean hasNext() {
                return i < ls.size();
        }
        @Override
        public E next() {
                Integer old = i;
                i = i+1;
                return ls.get(old);
        }
}
```

Factoría de secuencias

Rango: secuencia (enteros o reales) de *a* hasta *b* → *[a,b)*

$$range(a,b) \equiv (a, e \rightarrow e < b, e \rightarrow e + 1)$$

Iterate: secuencia general, definida por valor inicial, predicado y operador unario.

$$iterate(e0,g,nx) \equiv (e0, e \rightarrow g(e), e \rightarrow nx(e))$$

Secuencias asociadas a agregados: el agregado exporta un iterador para poder recorrer los elementos que contiene.
En Java:

- *c.stream()* para colecciones
- *Files.lines(file)* para ficheros
- *text.chars()* para cadenas de caracteres
- *Arrays.stream(a)* para arrays
- …

Factoría de secuencias

Ejemplos

```
Integer sumRange(Integer a, Integer b){
      return IntStream.range(a,b).sum();
}
Integer sumRange(Integer a, Integer b){
      Integer ac = 0;
      Integer e = a;
      while(e<b){
            ac = ac +e;
            e = e+1;
      }
      return ac;
}
```

Factoría de secuencias

Ejemplos

```
Double sumIterate(Double a, Double b, Double c){
      Double e0 = a;
      Predicate<Double> g = e->e<b;
      UnaryOperator<Double> nx = e->e*c;
      return Stream.iterate(e0,g,nx).mapToDouble(x->x).sum();
}

Double sumIterate(Double a, Double b, Double c){
      Double ac = 0;
      Double e = a;
      while(e<b){
            ac = ac+e;
            e = e*c;
      }
      return ac;
}
```

Operaciones sobre secuencias

- **Transformación:** *s2 = s1.map(t),* siendo *t* una función, sustituye cada elemento de la secuencia por el valor obtenido al aplicar la función

- **Filtro:** *s2 = s1.filter(h),* siendo *h* un predicado, elimina de la secuencia todos los elementos que no cumplan el predicado

- **Aplanamiento:** *s2 = s1.flatMap(seq),* siendo *seq* una función que por cada elemento de la secuencia original obtiene una nueva secuencia, y sustituye los elementos de la secuencia por los valores de la nueva secuencia obtenidos al aplicar la función

- **Límite:** *s2 = s1.limit(n)*, obtiene otra secuencia con los primeros *n* elementos

Operaciones sobre secuencias

Ejemplos – Transformación

```
Double sumaCuadrados(Integer a, Integer b){
      return IntStream.range(a,b).mapToDouble(e->e*e).sum()
}

Double sumaCuadrados(Integer a, Integer b){
     Double ac = 0;
     Integer e = a;
     while (e<b){
            t = e * e;
            ac = ac + t;
            e = e + 1;
      }
      return ac;
}
```

Operaciones sobre secuencias

Ejemplos – Filtro

```
Integer sumaMultiplos(Integer a, Integer b, Integer m){
      return IntStream.range(a,b).filter(e->e%m==0).sum();
}

Integer sumaMultiplos(Integer a, Integer b, Integer m){
      Double ac = 0;
      Double e = a;
      while(e<b){
            if(e%m==0){
                  ac = ac+e;
            }
            e = e+1;
      }
      return ac;
}
```

Operaciones sobre secuencias

Ejemplos – Aplanamiento

```
Integer sumaElems(List<Set<Integer>> ls){
      return ls.stream()
                  .flatMap(e->e.stream())
                  .mapToInt(Integer::intValue)
                  .sum();
}

Integer sumaElems(List<Set<Integer>> ls){
      Integer ac = 0;
      Integer i = 0;
      while(i<ls.size()){
            Iterator<Integer> it = ls.get(i).iterator();
            while(it.hasNext()){
                  Integer e = it.next();
                  ac = ac +e;
            }
            i = i+1;
      }
      return ac;
}
```

Operaciones sobre secuencias

Ejemplos – Límite

```
Long sumaPrimerosPrimos(Integer n){
      return Stream.iterate(2,e->true,e->Math2.siguientePrimo(e))
                  .limit(n)
                  .mapToLong(Long::longValue)
                  .sum();
}

Long sumaPrimerosPrimos(Integer n){
      Long ac = 0;
      Integer i = 0;
      Long p = 2;
      while(i<n){
            ac = ac + p;
            p = Math2.siguientePrimo(p);
            i = i+1;
      }
      return ac;
}
```

Operaciones sobre secuencias

- **Prefijo:** *s2 = s1. takeWhile(p),* siendo *p* un predicado*,* prefijo más largo de elementos de *s1* que cumplen *p*

- **Sufijo:** *s2 = s1. dropWhile(p),* siendo *p* un predicado*, s*ufijo de s1 que resulta cuando se elimina el prefijo anterior

- **Concatenar:** *concat(s1,s2)*
- **Enumerate:** *enumerate(s), s*ecuencia formada por los pares de elementos construidos por los elementos de *s* y su posición en la misma.
- **Pares consecutivos**
- **Producto cartesiano**
- **Zip**

Acumuladores

Forma general de algoritmo iterativo y funcional

```
R f(x) {
      E e = e0;
      B b = b0;
      while(g(e)&&!d(b)){
         b = c(b,e);
         e = s(e);
      }
      return r(b);
}
```

```
R f(x) {
      A a = (b0,(b,e) → c(b,e),b → r(b),b → d(b));
      S s = = (e0,e → g(e),e → s(e));
      R r = s.collect(a);
      return r;
}
```

Acumulador secuencial

$$a = (b0,(b,e) \rightarrow c(b,e),b \rightarrow r(b),b \rightarrow d(b))$$

Base del acumulador (valor inicial)

Función de acumulación

Función de retorno

Función de cortocuito

Acumulador simple

$$a = (b0,(b,e) \rightarrow c(b,e))$$

Recursividad final y algoritmos iterativos

Un algoritmo iterativo tiene un algoritmo recursivo final equivalente

```
R seqCollectLeft(X x) {
      E e = e0;
      B b = b0;
      while(g(e)&&!d(b)){
         b = c(b,e);
         e = nx(e);
      }
      return r(b);
}
```

```
R seqCollectLeft(X x) {
      B b = seqCollectLeftP(x,b0,e0);
      return r(b);
}
B seqCollectLeftP(X x, B b, E e) {
      B r = b;
      if(g(e)&&!d(b)){
            r = seqCollectLeftP(x,c(b,e),nx(e));
      }
      return r;
}
```

Factorías de acumuladores

- Método general para acumular: *collect*
  - s.allMatch(p)
  - s.noneMatch(p)
  - s.anyMatch(p)
  - s.sum()
  - s.count()

All(p): $B = R = Boolean, a = (true, (b, e) \rightarrow p(e), b \rightarrow b, b \rightarrow !\, b)$

None(p): $B = R = Boolean, a = (false, (b, e) \rightarrow p(e), b \rightarrow !\, b, b \rightarrow b)$

Any(p): $B = R = Boolean, a = (false, (b, e) \rightarrow p(e), b \rightarrow b, b \rightarrow b)$

Sum(): $B = R = Number, a = (0, (b, e) \rightarrow b + e)$

Count(): $B = R = Integer, a = (0, (b, e) \rightarrow b + 1)$

Factorías de acumuladores

- Método general para acumular: *collect*
  - s.joining(sp,pf,sf)

Joining(sp,pf,sf):

$$B = (String, Boolean), E = R = String$$
$$a = ((pf, true), ((b1, b2), e) \rightarrow c((b1, b2), e), (b1, b2) \rightarrow b1 + sf, b \rightarrow false)$$

Con

$$c((b1, b2), e) = \begin{cases} (b1 + e, false), & b2 = true \\ (b1 + sp + e, false), & b2 = false \end{cases}$$

Factorías de acumuladores

- Método general para acumular: *collect*
  - s.reduce(id,bo)

Reduce(e0,bo): $B = E = R,\ a = (e0, (b, e) \rightarrow bo(b, e))$

Reduce(bo): $B = E, R = Optional<E>$

$$a = \big((?, true), \big((b1, b2), e\big) \rightarrow c(b1, b2, e), (b1, b2) \rightarrow Opt.of(b1), b \rightarrow false\big)$$

$$c(b1, b2, e) = \begin{cases} (e, false), & b2 \\ (bo(b1, e), false), & !\,b2 \end{cases}$$

Combinando secuencias y acumuladores

```
s  <- (e0, e->g(e), e->nx(e))
a  <- (b0, (b,e)->c(b,e), b->r(b), b->d(b))
```

```
R seqCollectLef(S s, A a) {
        E e = s.e0;
        B b = a.b0;
        while (s.g(e) && !a.d(b)) {
                b = a.c(b, e)
                e = s.nx(e) ;
        }
        return a.r(b) ;
}
```

Transformación de alg. iterativos

- Transformación de la notación funcional a la imperativa y viceversa
- Se trata de identificar la secuencia y el acumulador, y a partir de ahí escribir el algoritmo
- Ejemplo: algoritmo para decidir si un número es primo

```
boolean esPrimo1(Long n) {
      Long sqrt = (long)Math.sqrt((double)n);
      return LongStream.rangeClosed(2, sqrt)
                  .noneMatch(x->Math2.esDivisible(n, x));
}
```

Secuencia, acumulador y predicado:

$s = (2, e \rightarrow e \leq \sqrt{n}, e \rightarrow e + 1)$
$B = Boolean$
$a = (false, (b, e) \rightarrow p(e), b \rightarrow !b, b \rightarrow b)$
$p(e) = Math2.esDivisible(n, e)$

```
boolean esPrimo2(Long n){
      Long sqrt = (long)Math.sqrt((double)n);
      Long e = 2L;
      Boolean b = false;
      while(e <= sqrt && !b){
            b = Math2.esDivisible(n, e);
            e = e + 1;
      }
      return !b;
}
```

Transformación de alg. iterativos

```
boolean esPrimo2(Long n){
      Long sqrt = (long)Math.sqrt((double)n);
      Long e = 2L;
      Boolean b = false;
      while(e <= sqrt && !b){
            b = Math2.esDivisible(n, e);
            e = e + 1;
      }
      return !b;
}
```

```
Boolean esPrimoFnl(Long n){
        Long sqrt = (long)Math.sqrt((double)n);
        Long e = 2L;
       Boolean b = false;
       b = esPrimoFnl(e,b,sqrt,n);
       return !b;
}
Boolean esPrimoFnl(Integer e,Boolean b,Long sqrt,Long n){
       if(e<= sqrt&& !b){
              b = esPrimoFnl(e+1,Math.esDivisible(n,e));
       }
       return b;
}
```

Transformación de alg. iterativos

Ejemplo, imperativo a funcional: Dada una lista de enteros, comprobar si todos son impares

```
public static boolean ejemplo3Java7For(List<Integer> ls) {
      boolean a = true;
      for (Integer e: ls) {
            a = e%2 == 1;
            if (!a) break;
      }
      return a;
}
```

```
public static boolean ejemplo3Java7While(List<Integer> ls) {
      boolean a = true;
      Integer i = 0;
      while (i < ls.size() && a) {
            Integer e = ls.get(i);
            a = e%2 == 1;
            i = i + 1;
      }
      return a;
}
```

```
public static boolean ejemplo3Java8(List<Integer> ls) {
      return ls.stream().allMatch(x -> x%2 == 1);
}
```

Transformación de alg. iterativos

Ejemplo, imperativo a funcional: Dada una lista de enteros, comprobar si todos son impares

```java
public static boolean ejemplo3Java7While(List<Integer> ls) {
	boolean a = true;
	Integer i = 0;
	while (i < ls.size() && a) {
		Integer e = ls.get(i);
		a = e%2 == 1;
		i = i + 1;
	}
	return a;
}
```

```java
Boolean ejemplo3Java7Fnl(List<Integer> ls){
		Integer e = 0;
		boolean b = true;
		b = ejemplo3Java7Fnl(e,b,ls);
		return b;
}
Boolean ejemplo3Java7Fnl(Integer i, Boolean b,
                                    List<Integer> ls){
		if(i<ls.size()&&b){
			b = ejemplo3Java7Fnl(i+1,ls.get(i)%2==1,ls);
		}
		return b;
}
```

Diseño de algoritmos iterativos

- **Diseñar un algoritmo iterativo** supone encontrar un algoritmo iterativo, imperativo o funcional, dada una restricción entrada-salida para el mismo

- Estrategia más sencilla:
  - Buscar un estado, una secuencia y el acumulador adecuado
  - Hay que demostrar que la secuencia es finita
  - El resultado debe ser la consecuencia de acumular la secuencia mediante el acumulador
  - Hay casos, los menos, dónde debemos escoger un estado y un invariante. Encontrar el estado se denomina generalizar el problema. A partir del estado y el invariante se deberá escoger la función siguiente para que se mantenga el invariante y la función de cota cumpla las propiedades exigidas.

Diseño de algoritmos iterativos

Ejemplo: Encontrar el siguiente primo de un número n

```
Long siguientePrimo(Long n){
      Long e0 = n%2==0?n+1:n+2;
      return Stream.iterate(e0, e->e+2)
            .filter(e->Math2.esPrimo(e))
            .findFirst()
            .get();
 }
```

```
Long siguientePrimo(Long n){
      Long e = n%2==0?n+1:n+2;
      Long r = null;
      while(r == null){
            if(esPrimo2(e)) {
                  r = e;
            }
            e = e+2;
      }
      return r;
}
```

Diseño de algoritmos iterativos

```
Long siguientePrimo(Long n){
      Long e = n%2==0?n+1:n+2;
      Long r = null;
      while(r == null){
            if(esPrimo2(e)) {
                  r = e;
            }
            e = e+2;
      }
      return r;
}
```

Ejemplo: Encontrar el siguiente primo de un número n

```
Integer siguientePrimoFnl(Long n){
        Long e = n%2==0?n+1:n+2;
        Long r = null;
        r = siguientePrimoFnl(e,r,ls);
        return r;
}
Integer siguientePrimoFnl(Integer e, Integer b){
        if(b==null){
            if (esPrimo(e)) {
              b = e;
           }
           b = siguientePrimoFnl(e+2,b);
         }
        return b;
}
```

Diseño de algoritmos iterativos

Ejemplo: Obtener una lista con los primos menores que n

Ejemplo: Obtener la suma de los primos mayores o iguales que *m* y menores que *n*

Diseño de algoritmos iterativos

Ejemplo: Comprobar si una lista está ordenada

```
Boolean ord(List<E> ls, Comparator<E> cmp){
      return IntStream.range(0,ls.size()-1)
      .allMatch(i->cmp.compare(ls.get(i),ls.get(i+1))<=0);
}
```

```
Boolean ord(List<E> ls, Comparator<E> cmp) {
      Integer i = 0;
      Boolean a = true;
      while(i<=ls.size()-2 && a) {
            a = cmp.compare(ls.get(i),ls.get(i+1))<=0;
            i++;
      }
      return a;
}
```

Diseño de algoritmos iterativos

```
Boolean ord(List<E> ls, Comparator<E> cmp) {
      Integer i = 0;
      Boolean a = true;
      while(i<=ls.size()-2 && a) {
            a = cmp.compare(ls.get(i),ls.get(i+1))<=0;
            i++;
      }
      return a;
}
```

Ejemplo: Comprobar si una lista está ordenada

```
Boolean ordFnl(Long n){
        Integer i = 0;
        Boolean a = true;
        a = ordFnl(i,a,ls);
        return a;
}
Boolean ordFnl(Integer i, Boolean a){
        If (i<=ls.size()-2 && a){
            a = cmp.compare(ls.get(i),ls.get(i+1))<=0;
            a = ordFnl(i+1,a);
        }
        return a;
}
```

Diseño de algoritmos iterativos

Ejemplo: Obtener una lista de pares de un primo y su siguiente, tal que el primero sea mayor o igual que m y menor que n, y que la diferencia entre ellos sea una cantidad k>=2 dada

```
List<Tuple2<Long,Long>> primosPar(Long m, Long n, Integer k){
      var r = Stream.iterate(Math2.siguientePrimo(m-1),
            x->x < n, x->Math2.siguientePrimo(x));
      var r2 = Streams2.consecutivePairs(r);
      var rr2 = r2.filter(t->t.v2-t.v1==k)
            .collect(Collectors.toList());
      return rr2;
}
```

Ejercicio propuesto: hacer la versión imperativa y recursiva

Diseño de algoritmos iterativos

Ejemplo: Obtener el número de veces que se repite cada divisor, mayor que 1, entre los divisores de los números que van de *m* a *n*

```
Stream<Long> divisores(Long n){
      return Stream.iterate(
            2L, x-> x <= (long) Math.sqrt(n), x -> x+1)
            .filter(x->n%x==0);
}

Multiset<Long> rr4 = Stream.iterate(m1,x->x<n1,x->x+1)
      .flatMap(x->divisores(x))
      .collect(Collectors2.toMultiset());
```

Ejercicio propuesto: hacer la versión imperativa y recursiva

Diseño de algoritmos iterativos

Ejemplo: Búsqueda binaria: dada una lista ls, ordenada con respecto a un orden, encontrar, si existe, la posición de un elemento dado e o -1 si no lo encuentra

Generalizar problema

$$(i, j, k),\ i \in [0, n],\ j \in [0, n], k \in [0, n]$$

Invariante

$$e \in ls = e \in ls[i, j],\ i \leq j,\ k = \frac{i+j}{2}$$

Tamaño, estado inicial, guarda, siguiente, función de cota

$$s(i, j, k) = \begin{cases} (i, k, \frac{i+k}{2}), & e < ls[k] \\ (k+1, j, \frac{k+1+j}{2}), & e > ls[k] \end{cases}$$

Diseño de algoritmos iterativos

Ejemplo: Búsqueda binaria: dada una lista ls, ordenada con respecto a un orden, encontrar, si existe, la posición de un elemento dado e o -1 si no lo encuentra

```
bb(ls,e) {
      n = |ls|;
      (i, j, k) = (0, n, n/2);
      while(j-i >0 && !(ls[k] == e)){
            if(e < ls[k]){
                  (i,j,k) = (i,k,(i+k)/2);
            } else {
                  (i,j,k) = (k+1,j,(k+1+j)/2);
            }
      }
      return j-i>0?k:-1;
}
```

$$E = (i, j, k), \qquad sq = (\left(0, n, \frac{n}{2}\right), (i,j,k) \to j - i > 0, (i,j,k) \to s(i,j,k))$$

$$B = E, \qquad a = (?, ?, (i,j,k) \to j - i > 0 ? k : -1, (i,j,k) \to e = ls[k])$$

Diseño de algoritmos iterativos

```
bb(ls,e) {
      n = |ls|;
      (i, j, k) = (0, n, n/2);
      while(j-i >0 && !(ls[k] == e)){
            if(e < ls[k]){
                  (i,j,k) = (i,k,(i+k)/2);
            } else {
                  (i,j,k) = (k+1,j,(k+1+j)/2);
            }
      }
      return j-i>0?k:-1;
}
```

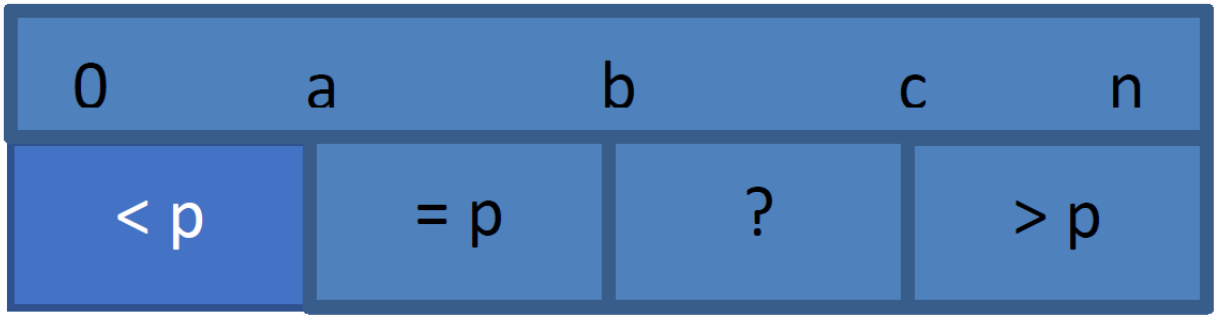
Ejemplo: Búsqueda binaria: dada una lista ls, ordenada con respecto a un orden, encontrar, si existe, la posición de un elemento dado e o -1 si no lo encuentra

```
Integer bbFnl(List<integer> ls, Integer e){
        n = |ls|
       (i,j,k) = ordFnl(0, n, n/2);
       return j-i>0?k:-1;
}
Tuple bbFnl(Integer i, Integer j, Integer k){
       if (j-i>0 && !(ls[k]==e)) {
            if (e<ls[k]){
               (i,j,k) = bbFnl(i,k,(i+k)/2);
            } else {
               (i,j,k) = bbFnl(k+1,j,(k+1+j)/2);
            }
        }
       return (i,j,k);
}
```

Diseño de algoritmos iterativos

Ejemplo: Bandera holandesa:

Dada una lista y un elemento del mismo tipo que las casillas, que llamaremos pivote, reordenarla, de menor a mayor, para que resulten tres bloques: los menores que el pivote, los iguales al pivote y los mayores que el pivote. El algoritmo debe devolver dos enteros que son las posiciones de las casillas que separan los tres bloques formados.



Diseño de algoritmos iterativos

Ejemplo: Bandera holandesa:

```
bh(ls, p){
      n = |ls|;
      (a,b,c) = (0,0,n);
      while(c-b>0){
            if(ls[b] < p ){
                  it(a,b,ls);
                  a++;
                  b++;
            }else if(ls[b] == p){
                  b++;
            }else{
                  it(b,c-1,ls);
                  c--;
            }
      }
      return (a,b);
}
```

```
it(a,b,ls){
      (ls[a],ls[b]) = (ls[b],ls[a]);
}
```

Esquemas Iterativos y Recursivos con Secuencias y Acumuladores

Análisis y Diseño de Datos y Algoritmos
Estructuras de Datos y Algoritmos

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA INFORMÁTICA
Departamento de Lenguajes y Sistemas Informáticos
Curso 2022-2023

Tuplas con record (Java)

Simulamos tuplas mediante *record*:

```
record T(Integer i,Double b){}

T t = new T(0,0.);
```

Acceso a las propiedades mediante *t.i()*, *t.b()*.
Normalmente, la declaración se complementa de un método de factoría.

```
record T(Integer i,Double b){
       public static T of(Integer i,Double b) {
              return new T(i,b);
}

T t = T.of(0,0.);
```

```
public static record T(Integer a, Double b, String c) {
       public static T of(Integer a, Double b, String c){
              return new T(a,b,c);
}
T t = T.of(2,5.4,"Hola");
T t2 = t;
String s = t.c();
```