

Memoria Practica 1

Jesús carrascosa Carro
(jescarcar5)
jescarcar5@alum.us.es

Octubre 2022

Índice

1. Introducción	1
2. Ejercicio 1	2
2.1. Código	2
2.2. EnteroCadena:	4
3. Ejercicio 2:	5
3.1. Código:	5
3.2. Trio:	10
4. Ejercicio 3:	12
4.1. Código:	12
4.2. ParDePuntos2	21
5. Ejercicio 4:	21
6. Test:	24
6.1. Código:	24
6.2. Imágenes:	26
6.2.1. Ejercicio 1	26
6.2.2. Ejercicio 2	27
6.2.3. Ejercicio 3	27
6.2.4. Ejercicio 4	28

1. Introducción

Resumen

Este documento es la memoria de la 1ª práctica individual de la asignatura ADDA. El código que se muestra en este documento, también se encuentra disponible en el repositorio privado de github:

https://github.com/yisuscc/PII_escargar5

Puesto que dicho repositorio tiene la visibilidad en privado, si precisan acceder a él, ruego que me contacten por email que figura al inicio del documento

2. Ejercicio 1

2.1. Código

```
// versión del eninciado
public static Map<Integer,List<String>>ejercicioA ( Integer varA, String varB,
Integer varC,String varD, Integer varE){
UnaryOperator<EnteroCadena> nx = elem ->
{ String aux ;
//se que se puede hacer con el operador ternario ?
//pero prefiero utilizar if else por legibilidad
if(elem.a()%3==0) {
aux = elem.s()+elem.a().toString();
}else {
aux = elem.s().substring(elem.a()%elem.s().length());
}
return EnteroCadena.of(elem.a()+2,aux);
};
return Stream.iterate(EnteroCadena.of(varA, varB), elem->elem.a()<varC,nx)
.map(elem->elem.s()+varD)
.filter(nom->nom.length()<varE)
.collect(Collectors.groupingBy(String::length));

}

// funciones que no sirven de ayuda (se que esto es mas propio e python pero
//por legibilidad mato)

public static Map<Integer,List<String>>ej1Iterativo ( Integer varA, String varB,
Integer varC,String varD, Integer varE){
Map<Integer, List<String>> d = new HashMap<>();
EnteroCadena s = new EnteroCadena(varA, varB); // la semilla
//Predicate<EnteroCadena> prdct = x->{
//return x.a()< varC;
//};
UnaryOperator<EnteroCadena> nx = elem ->
{ String aux ;
// se puede sustituir el UnaryOperator<T>
// por una function<T,T>
//se que se puede hacer con el operador ternario ?
//pero prefiero utilizar if else por legibilidad
```

```

        if(elem.a()%3==0) {
            aux = elem.s()+elem.a().toString();
        }else {
            aux = elem.s().substring(elem.a()%elem.s().length());
        }
        return EnteroCadena.of(elem.a()+2,aux);
    };
    while (s.a()< varC) { // probar con un do while
        String valor = s.s()+varD;
        Integer clave = valor.length();
        if(clave<varE){
            if(d.containsKey(clave)) { // existe ya la clave
                List<String>ls = d.get(clave);
                ls.add(valor);
                d.put(clave, ls);
            }else {
                // no existe la clave
                List<String> ls = new ArrayList<>();
                ls.add(valor);
                d.put(clave, ls);
            }
        }
        s= nx.apply(s);
    }
    return d;
}

// versión recursiva final
public static Map<Integer,List<String>>>ej1RekursivoFinal ( Integer varA, String varB,
Integer varC,String varD, Integer varE){
    Map<Integer, List<String>>> d = new HashMap<>();
    EnteroCadena s = new EnteroCadena(varA, varB); // la semilla

    UnaryOperator<EnteroCadena> nx = elem ->
    { String aux ;
        if(elem.a()%3==0) {
            aux = elem.s()+elem.a().toString();
        }else {
            aux = elem.s().substring(elem.a()%elem.s().length());
        }
        return EnteroCadena.of(elem.a()+2,aux);
    };
    return ej1RekursivoAux(varA, varB, varC, varD, varE, s, nx, d);
}

```

```

private static Map<Integer,List<String>>ej1RecursivoAux ( Integer varA, String varB,
Integer varC,String varD,
Integer varE,EnteroCadena s,
UnaryOperator<EnteroCadena> nx,
Map<Integer, List<String>> d){

String valor = s.s()+varD;
Integer clave = valor.length();

if (s.a()< varC){
if(clave<varE) {
if(d.containsKey(clave)) { // existe ya la clave
List<String>ls = d.get(clave);
ls.add(valor);
d.put(clave, ls);

}else {
// no existe la clave
List<String> ls = new LinkedList<>();
ls.add(valor);
d.put(clave, ls);

}
EnteroCadena ns= nx.apply(s);
ej1RecursivoAux(varA, varB, varC, varD, varE, ns, nx, d);

}
}

return d;

}

```

2.2. EnteroCadena:

```

public record EnteroCadena(Integer a, String s) {
public static EnteroCadena of (Integer a, String s) {
return new EnteroCadena(a, s);
}
}

```

3. Ejercicio 2:

3.1. Código:

```
    public static Integer Ej2RecNF(Integer a, Integer b, String s) {
return Ej2RecNFAux(a, b, s);

}
private static Integer Ej2RecNFAux(Integer a, Integer b, String s){
// if(a<0|| b<0) {
// throw new IllegalArgumentException("A y b tienen que ser positivos");
// }
Integer r = null;
if(0<=a|| 0<=b){
if(s.length()==0) {
r = a*a+b*b;
}
else if (a<2 || b<2){
r = s.length()+a+b;
}
else if (a% s.length()<b% s.length()) {
r= a+b+Ej2RecNFAux(a-1, b/2, s.substring(a% s.length(), b% s.length()));
}

else {
r = a*b+Ej2RecNFAux(a/2, b-1, s.substring(b% s.length(), a% s.length()));
}

}

return r;

}
// version recursiva final
public static Integer Ej2RecFinal(Integer a, Integer b, String s) {
return Ej2RecFinalAux(a, b, s, 0);

}
private static Integer Ej2RecFinalAux(Integer a, Integer b, String s,Integer acum) {
Integer r = null;
if(0<=a|| 0<=b){
if(s.length()==0) {
r = a*a+b*b +acum;
}
else if (a<2 || b<2){
```

```

    r = s.length()+a+b+ acum;
  }
  else if (a%s.length()<b%s.length()) {
    acum = a+b +acum;
    r= Ej2RecFinalAux(a-1, b/2, s.substring(a%s.length(), b%s.length()),acum);
  }

  else {
    acum = a*b +acum;
    r = Ej2RecFinalAux(a/2, b-1, s.substring(b%s.length(), a%s.length()),acum);
  }

}
return r;

}

//versión iterativa while
public static Integer Ej2Iterativo(Integer a, Integer b, String s) {

  Integer r;

  Trio res  = Trio.of(a,b,s.length());
  Map<Trio, Integer> d = new HashMap<>();
  // con for queda mucho mas bonito y mejor
  Integer i = 0; // se encarga de a
  // se encarga de b
  // se encarga de s
  Integer sl = s.length();

  while(i<=a){
    Integer j =0;
    while(j<=b){
      Integer k = 0;
      while(k<=sl) {

        if(k==0) {
          r = i*i+j*j;
          d.put(Trio.of(i, j, k), r);
        }
        else if (i<2 || j<2){
          r = k+i+j;
          d.put(Trio.of(i, j, k), r);
        }
        else if (i%k<j%k) {

```

```

Integer n = j%k- i%k;
r= i+j+d.get(Trio.of(i-1, j/2, n));
d.put(Trio.of(i, j, k), r);
}

else {
Integer n = i%k- j%k;
r = i*j+d.get(Trio.of(i/2, j-1, n));
d.put(Trio.of(i, j, k), r);

}
k++;

}
j++;
}
i++;
}
return d.get(res);

}

//iterativo for
public static Integer Ej2IterativoFor(Integer a, Integer b, String s) {

Trio res = Trio.of(a,b,s.length());
Map<Trio, Integer> d = new HashMap<>();
for(Integer i = 0;i<=a;i++) {
for(Integer j=0;j<=b;j++) {
for(Integer k= 0;k<=s.length();k++) {
if(k==0) {
Integer r = i*i+j*j;
d.put(Trio.of(i, j, k), r);
}
else if (i<2 || j<2){
Integer r = k+i+j;
d.put(Trio.of(i, j, k), r);
}
else if (i%k<j%k) {
Integer n = j%k- i%k;
Integer r= i+j+d.get(Trio.of(i-1, j/2, n));
d.put(Trio.of(i, j, k), r);
}

else {
Integer n = i%k- j%k;

```

```

Integer r = i*j+d.get(Trio.of(i/2, j-1, n));
d.put(Trio.of(i, j, k), r);
}
}
}
}
return d.get(res);
}
// versiónn funcional
public static Integer Ej2FuncionalV1(Integer a, Integer b, String s) {
// version traducida directa del iterativo,
Trio res = Trio.of(a,b,s.length()); // uso mi propio record de trio
Map<Trio, Integer> d = new HashMap<>();
Integer sl = s.length();
Map<Trio, Integer> mapAux = new HashMap<>();
Consumer<Trio> con = x-> {
Integer r = null;
Integer i = x.a();
Integer j = x.b();
Integer k = x.c();

if(k==0) {
r = i*i+j*j;
d.put(Trio.of(i, j, k), r); // se puede sutstituir el trio.of() por x
}
else if (i<2 || j<2){
r = k+i+j;
d.put(Trio.of(i, j, k), r);
}
else if (i%k<j%k) {
Integer n = j%k- i%k;
r= i+j+d.get(Trio.of(i-1, j/2, n));
d.put(Trio.of(i, j, k), r);
}

else {
Integer n = i%k- j%k;
r = i*j+d.get(Trio.of(i/2, j-1, n));
d.put(Trio.of(i, j, k), r);
}

};
Trio.fullSequence(a, b, s.length()).stream().forEach(con);
return d.get(res);
}

```



```

}

public static Integer Ej2FuncionalV2(Integer a, Integer b, String s) {
// Similar al anterior
Map<Trio, Integer> d = new HashMap<>();

Function<Trio, Pair<Trio, Integer>> con = x-> {
Integer r = null;
Integer i = x.a();
Integer j = x.b();
Integer k = x.c();

if(k==0) {
r = i*i+j*j;
d.put(x, r);
}
else if (i<2 || j<2){
r = k+i+j;
d.put(Trio.of(i, j, k), r);
}
else if (i%k<j%k) {
Integer n = j%k- i%k;
r= i+j+d.get(Trio.of(i-1, j/2, n));
d.put(x, r);
}

else {
Integer n = i%k- j%k;
r = i*j+d.get(Trio.of(i/2, j-1, n));
d.put(x, r);
}
return Pair.of(x, r);
};
Predicate<Pair<Trio, Integer>> prd = p -> p.first().equals(Trio.of(a, b, s.length()));
return Stream.iterate(Trio.of(0, 0, 0), T-> T.hasNext(a, b, s.length()),
T->T.nextTrio(a, b, s.length())).map(t -> con.apply(t)).
filter(prd).findFirst().get().second();
}

public static Integer Ej2FuncionalV3(Integer a, Integer b, String s) {

Integer sl = s.length();
Map<Trio, Integer> d = new HashMap<>();
//d.put(Trio.of(0,0,0), 0); N va a hacer falta creo yo pero lo dejo comentado por si acaso
UnaryOperator<Pair<Trio, Integer>> con = y-> {

```

```

Trio x = y.first().nextTrio(a, b, sl);
Integer r = null;
Integer i = x.a();
Integer j = x.b();
Integer k = x.c();

if(k==0) {
    r = i*i+j*j;
    d.put(x, r);
}
else if (i<2 || j<2){
    r = k+i+j;
    d.put(Trio.of(i, j, k), r);
}
else if (i%k<j%k) {
    Integer n = j%k- i%k;
    r = i+j+d.get(Trio.of(i-1, j/2, n));
    d.put(x, r);
}

else {
    Integer n = i%k- j%k;
    r = i*j+d.get(Trio.of(i/2, j-1, n));
    d.put(x, r);
}

return Pair.of(x, r);
};

Predicate<Pair<Trio, Integer>> prd = p -> p.first().equals(Trio.of(a, b, s.length()));

return Stream.iterate(Pair.of(Trio.of(0, 0, 0),0 ),
    p1-> p1.first().hasNext(a, b, sl),p -> con.apply(p)).
    filter(prd).findFirst().get().second();
}

```

3.2. Trio:

```

    public record Trio(Integer a , Integer b, Integer c) {
    public static Trio of (Integer a , Integer b, Integer c) {
    return new Trio(a, b, c);
    }
    public Trio nextTrio( Integer maxA, Integer maxB ,Integer maxC) {
    // Inicializamos
    Integer nC = this.c();

```

```

Integer nB = this.b();
Integer nA = this.a();

if(this.hasNext(maxA, maxB, maxC)) {
    nC = nC+1;
    if (nC%(maxC+1)==0){
        nC =0;
        nB = nB+1;
        if (nB%(maxB+1)==0) {
            nB = 0;
            nA++;
            if((nA%(maxA+1)==0)) {
                // así el hasNext da false
                nC= maxC+1;
                nB = maxB+1;
                nA = maxA+1;
            }
        }
    }

}

return Trio.of(nA, nB, nC);
}

public static Trio nextTrioStatic(Trio T, Integer maxA, Integer maxB ,Integer maxC) {
    // Inicializamos
    Integer nC = T.c();
    Integer nB = T.b();
    Integer nA = T.a();
    if(T.hasNext(maxA, maxB, maxC)) {
        nC = nC+1;
        if (nC%(maxC+1)==0){
            nC =0;
            nB++;
            if (nB%(maxB+1)==0) {
                nB = 0;
                nA++;
                if((nA%(maxA+1)==0)) {
                    nC= 0;
                    nB = 0;
                    nA = 0;
                }
            }
        }
    }
}

```

```

}

}
}
return Trio.of(nA, nB, nC);
}
public Boolean hasNext(Integer maxA, Integer maxB ,Integer maxC) {
    Trio T = this;
    Boolean r = false ;
    if(T.c()<=(maxC)&&T.a() <=maxA  && T.b()<=maxB ) {
        r = true;
    }
    return r;
}

public static List<Trio> fullSequence(Integer maxA, Integer maxB ,Integer maxC){
    List<Trio> res = new LinkedList<>();
    for(Integer i =0;i<= maxA;i++) {
        for(Integer j =0;j<= maxB;j++) {
            for(Integer k =0;k<= maxC;k++) {
                res.add(Trio.of(i, j, k));
            }
        }
    }
    return res;
}

}

```

4. Ejercicio 3:

4.1. Código:

```

public class Ej3 {
    public static List<Punto2D> ej3IterativoV1(String file1, String file2) {
        List<Punto2D> res = new ArrayList<>();
        // convertmos a iterator file
        // IteratorFile

        Function<String, Punto2D> funk = e -> {
            String[] p = e.split(",");
            Double x = Double.parseDouble(p[0].trim());

```

```

Double y = Double.parseDouble(p[1].trim());
return Punto2D.of(x, y);

};
// Hacemos el predicado
Predicate<Punto2D> condicion = p -> {
Boolean r = false;
if (p.getCuadrante() == Cuadrante.TERCER_CUADRANTE) {
r = true;
} else if (p.getCuadrante().equals(Cuadrante.PRIMER_CUADRANTE)) {
r = true;
}

return r;
};
Iterator<String> f1 = new IteratorFile(file1);
Iterator<String> f2 = new IteratorFile(file2);
return acumIterativo(f1, f2, condicion, funk);
}

// private static Iterator<Punto2D> transformadorAPunto2d(Iterator<String>ItS){
//
// }
private static List<Punto2D> acumIterativo(Iterator<String> i1, Iterator<String> i2,
Predicate<Punto2D> prd,
Function<String, Punto2D> conver) {
List<Punto2D> res = new ArrayList<>();

Punto2D p1 = (i1.hasNext()) ? conver.apply(i1.next()) : null;
Punto2D p2 = (i2.hasNext()) ? conver.apply(i2.next()) : null;

while ((p1 != null || p2 != null)) {
// Punto2D p = null;

if (p1 != null && (p2 == null || Comparator2.isLENull(p1, p2))) {
// esta 1 condición define el orden
// Le toca el turno a p1
if (p1 != null && prd.test(p1)) {
res.add(p1);
}
// else if (p2!= null&&prd.test(p2)) {
// res.add(p2);
// p2 = (i2.hasNext())?conver.apply(i2.next()):null;
// p1 = (i1.hasNext())?conver.apply(i1.next()):null;
// }else {

```

```

// p1 = (i1.hasNext())?conver.apply(i1.next()):null;
// p2 = (i2.hasNext())?conver.apply(i2.next()):null;
// }
p1 = (i1.hasNext()) ? conver.apply(i1.next()) : null;
} else { // er toca el turno a p2
if (p2 != null && prd.test(p2)) {
res.add(p2);
p2 = (i2.hasNext()) ? conver.apply(i2.next()) : null;
}
// else if (p1!= null&&prd.test(p1)) {
// res.add(p1);
// p1 = (i1.hasNext())?conver.apply(i1.next()):null;
// p2 = (i2.hasNext())?conver.apply(i2.next()):null;
// }else {
// p1 = (i1.hasNext())?conver.apply(i1.next()):null;
// p2 = (i2.hasNext())?conver.apply(i2.next()):null;
// }
p2 = (i2.hasNext()) ? conver.apply(i2.next()) : null;
}

}
return res;
}

// versión recursiva final
public static List<Punto2D> ej3RecursivoFinalV1(String file1, String file2) {
List<Punto2D> ac = new ArrayList<>(); // ACUMULADOR

Function<String, Punto2D> conver = e -> { // convierte a punto2d
String[] p = e.split(",");
Double x = Double.parseDouble(p[0].trim());
Double y = Double.parseDouble(p[1].trim());
return Punto2D.of(x, y);
};

// Hacemos el predicado
// e predicate, creo que se pueden meter sin problemas
// dentro del bucle recursivo
Predicate<Punto2D> condicion = p -> {
Boolean r = false;
if (p.getCuadrante() == Cuadrante.TERCER_CUADRANTE) {
r = true;
} else if (p.getCuadrante().equals(Cuadrante.PRIMER_CUADRANTE)) {
r = true;
}
}

```

```

return r;
};
// creamos los iteradores
Iterator<String> i1 = new IteratorFile(file1);
Iterator<String> i2 = new IteratorFile(file2);
Punto2D p1 = (i1.hasNext()) ? conver.apply(i1.next()) : null;
Punto2D p2 = (i2.hasNext()) ? conver.apply(i2.next()) : null;
return bucleRecFinalAux(p1, p2, ac, i1, i2, conver, condicion);

}

private static List<Punto2D> bucleRecFinalAux(Punto2D p1, Punto2D p2, List<Punto2D> res, Iterator<String> i2, Function<String, Punto2D> conver, Predicate<Punto2D> prd) {
    if ((p1 != null || p2 != null)) {
        // Punto2D p = null;

        if (p1 != null && (p2 == null || Comparator2.isLENull(p1, p2))) { // esta 1 condición define
            // Le toca el turno a p1
            if (p1 != null && prd.test(p1)) {
                res.add(p1);
            }

            // else if (p2 != null && prd.test(p2)) {
            // res.add(p2);
            // p2 = (i2.hasNext())?conver.apply(i2.next()):null;
            // p1 = (i1.hasNext())?conver.apply(i1.next()):null;
            // }else {
            // p1 = (i1.hasNext())?conver.apply(i1.next()):null;
            // p2 = (i2.hasNext())?conver.apply(i2.next()):null;
            // }
            p1 = (i1.hasNext()) ? conver.apply(i1.next()) : null;
        } else { // caso contrario
            if (p2 != null && prd.test(p2)) {
                res.add(p2);
            }
            p2 = (i2.hasNext()) ? conver.apply(i2.next()) : null;
        }

        // else if (p1 != null && prd.test(p1)) {
        // res.add(p1);
        // p1 = (i1.hasNext())?conver.apply(i1.next()):null;
        // p2 = (i2.hasNext())?conver.apply(i2.next()):null;
        // }else {
        // p1 = (i1.hasNext())?conver.apply(i1.next()):null;
        // p2 = (i2.hasNext())?conver.apply(i2.next()):null;
        // }
        p2 = (i2.hasNext()) ? conver.apply(i2.next()) : null;
    }
}

```

```

    bucleRecFinalAux(p1, p2, res, i1, i2, conver, prd);

}

return res;
}

// vo y a intentar la funcional para aclararme las ideas
public static List<Punto2D> ej3Funcional(String file1, String file2) {
    // Funtion que convierte string a punto2d
    Function<String, Punto2D> funk = e -> {
        String[] p = e.split(",");
        Double x = Double.parseDouble(p[0].trim());
        Double y = Double.parseDouble(p[1].trim());
        return Punto2D.of(x, y);
    };
    // Hacemos el predicado
    Predicate<Punto2D> condicion = p -> {
        Boolean r = false;
        if (p.getCuadrante() == Cuadrante.PRIMER_CUADRANTE ||
            p.getCuadrante() == Cuadrante.TERCER_CUADRANTE) {
            r = true;
        }

        return r;
    };
    Stream<String> lineas = null;
    // Stream de cadena con ambas lineas
    try {
        lineas = Stream.concat(Files.lines(Paths.get(file1)), Files.lines(Paths.get(file2)));
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    } //
    // convertimos a punto2d
    return lineas.map(funk).filter(condicion).sorted().toList();
}

public static List<Punto2D> ej3FuncionalV2 (String file1, String file2){

    Function<String, Punto2D> conver = e -> {
        String[] p = e.split(",");
        Double x = Double.parseDouble(p[0].trim());

```



```

Double y = Double.parseDouble(p[1].trim());
return Punto2D.of(x, y);

};
Iterator<Punto2D> i1 = Stream2.file(file1).map(conver).iterator();
Iterator<Punto2D> i2 = Stream2.file(file2).map(conver).iterator();
Punto2D p1 = (i1.hasNext()) ? i1.next() : null;
Punto2D p2 = (i2.hasNext()) ? i2.next() : null;

Predicate<Punto2D> condicion = p -> {
Boolean r = false;
if (p.getCuadrante() == Cuadrante.PRIMER_CUADRANTE ||
p.getCuadrante() == Cuadrante.TERCER_CUADRANTE) {
r = true;
}

return r;
};
Predicate<ParDePuntos2> frenos = pp -> pp.getP1() != null || pp.getP2() != null;
// Function<ParDePuntos, Punto2D> daElMenor = pp -> {
// Punto2D pp1 = pp.getP1();
// Punto2D pp2 = pp.getP2();
// Punto2D Pmenor;
// if( pp1 != null) {
// Pmenor = Comparator2.isLENull(pp1, pp2)? pp1 : pp2;
// }else if(pp2 != null) {
// Pmenor = pp2;
// }
// else {
// Pmenor = null;
// }
//
// return Pmenor;
// };
UnaryOperator<ParDePuntos2> next = pp -> {
Punto2D pp1 = pp.getP1();
Punto2D pp2 = pp.getP2();
if(pp1!=null || pp2 !=null) {
if(pp1 != null && (pp2 == null || Comparator2.isLENull(pp1, pp2))) {
pp1 = (i1.hasNext()) ? i1.next() : null;
}else if (pp2!=null) {
pp2 = (i2.hasNext()) ? i2.next() : null;
}}
return ParDePuntos2.create(pp1, pp2);
};

```

```

return Stream.iterate(ParDePuntos2.create(p1, p2), frenos, next)
    .map(p -> p.daElMenor()).filter(condicion).toList();
}

//
// public static List<Punto2D> ej3FuncionalV43434(String file1, String file2) {
// Function<String, Punto2D> conver = e -> {
// String[] p = e.split(",");
// Double x = Double.parseDouble(p[0].trim());
// Double y = Double.parseDouble(p[1].trim());
// return Punto2D.of(x, y);
//
// };
// // Hacemos el predicado
// Predicate<Punto2D> condicion = p -> {
// Boolean r = false;
// if (p.getCuadrante() == Cuadrante.PRIMER_CUADRANTE || p.getCuadrante() == Cuadrante.TERCER_CUADRANTE)
// r = true;
// }
//
// return r;
// };
// // Iterator<String> i1 = new IteratorFile(file1);
// // Iterator<String> i2 = new IteratorFile(file2);
// Iterator<Punto2D> i1 = Stream2.file(file1).map(conver).iterator();
// Iterator<Punto2D> i2 = Stream2.file(file2).map(conver).iterator();
// // se vuelve redundante
// Punto2D p1 = (i1.hasNext()) ? i1.next() : null;
// Punto2D p2 = (i2.hasNext()) ? i2.next() : null;
// Punto2D Pmenor = Comparator2.isLENull(p1, p2) ? p1 : p2;
// Pair<ParDePuntos, Punto2D> inicio = Pair.of(ParDePuntos.create(p1, p2), Pmenor);
// UnaryOperator<Pair<ParDePuntos, Punto2D>> siguiente = x -> {
// ParDePuntos s = x.first(); //Sig de siguiente
// Punto2D p1Sig = s.getP1();
// Punto2D p2Sig = s.getP2();
// Punto2D PmenorSig ;
// if ((p1Sig != null || p2Sig != null)){
// if (p1Sig != null && (p2Sig == null || Comparator2.isLENull(p1Sig, p2Sig))) {
// // esta 1 condición define el orden
// // Le toca el turno a p1
//
// if(p1 != null)
// p1Sig = (i1.hasNext()) ? i1.next() : null;
//

```

```

// // }
//
// } else { // er toca el turno a p2
// if(p2 != null) {
// p2Sig = (i2.hasNext()) ? i2.next() : null;
// }
//
// }
//
// if( p1 != null) {
// PmenorSig = Comparator2.isLENull(p1, p2)? p1 : p2;
// }else if(p2 != null) {
// PmenorSig = p2;
// }
// else {
// PmenorSig = null;
// }
// }
//
// return Pair.of(ParDePuntos.create(p1Sig, p2Sig), Pmenor);
// };
//
// Predicate<Pair<ParDePuntos,Punto2D>> pr = u -> u.first().getP1() != null || u.first().getP2() != null;
//
//
//
// List<Punto2D> res = new ArrayList<>();
// // primero concatenamos en zip
//
// return Stream.iterate(inicio, pr, siguiente).map(p -> p.second()).filter(condicion).distinct().collect(toList());
// }
// public static List<Punto2D> ej3FuncionalV3(String file1, String file2) {
// Function<String, Punto2D> conver = e -> {
// String[] p = e.split(",");
// Double x = Double.parseDouble(p[0].trim());
// Double y = Double.parseDouble(p[1].trim());
// return Punto2D.of(x, y);
// };
// // Hacemos el predicado
// Predicate<Punto2D> condicion = p -> {
// Boolean r = false;
// if (p.getCuadrante() == Cuadrante.PRIMER_CUADRANTE || p.getCuadrante() == Cuadrante.TERCER_CUADRANTE)
// r = true;
// }

```

```

//
// return r;
// };
//
// Iterator<Punto2D> i1 = Stream2.file(file1).map(conver).iterator();
// Iterator<Punto2D> i2 = Stream2.file(file2).map(conver).iterator();
// Punto2D p1 = (i1.hasNext()) ? i1.next() : null;
// Punto2D p2 = (i2.hasNext()) ? i2.next() : null;
// Punto2D Pmayor = p1.compareTo(p2) < 0 ? p1 : p2;
//
// UnaryOperator<Punto2D> siguiente = x -> {
// Punto2D P = x;
//
//
// if (p1 != null && (p2 == null || Comparator2.isLENull(p1, p2))) {
// // esta 1 condición define el orden
// // Le toca el turno a p1
// P = p1;
// p1 = (i1.hasNext()) ? i1.next() : null;
//
// // }
//
// } else { // er toca el turno a p2
// P = p2;
// p2 = (i2.hasNext()) ? i2.next() : null;
// }
//
// return P;
//
// };
//
// Predicate<Punto2D> pr = u -> p1 != null || p2 != null;
//
//
//
// List<Punto2D> res = new ArrayList<>();
// // primero concatenamos en zip
//
// return Stream.iterate(Pmayor, pr, siguiente).filter(condicion).distinct().collect(Collectors.toList());
//
// }

}

```

4.2. ParDePuntos2

```
package Ejercicios;

import us.lsi.common.Comparator2;
import us.lsi.geometria.Punto2D;

public record ParDePuntos2(Punto2D p1, Punto2D p2) {
    //a este le da igual si los puntos son nulos

    public Punto2D getP1() {
        return p1;
    }

    public Punto2D getP2() {
        return p2;
    }

    public Punto2D daElMenor() {
        Punto2D pp1 = this.getP1();
        Punto2D pp2 = this.getP2();
        Punto2D Pmenor;
        if( pp1 != null) {
            Pmenor = Comparator2.isLENull(pp1, pp2)? pp1 : pp2;
        }else if(pp2 != null) {
            Pmenor = pp2;
        }
        else {
            Pmenor = null;
        }

        return Pmenor;
    }

    public static ParDePuntos2 create(Punto2D pp1, Punto2D pp2) {
        // TODO Auto-generated method stub
        return new ParDePuntos2(pp1, pp2);
    }
}
```

5. Ejercicio 4:

```
public static String ej4Recsm(Integer a, Integer b, Integer c) {
    String s;
    // se puede usrra el string formatter pero ya lo he implementado de esta forma
```

```

    if (a<2&& b<= 2||c<2) {
    s = "(" +a.toString() + "+" +b.toString()+"+"+c.toString()+ ")";
    }else if (a<3||b<3&&c<=3) {
    s= "(" +c.toString() + "-" +b.toString()+"-"+a.toString()+ ")";
    }
    else if ((b%a== 0)&&((a%2==0)||(b%3==0))) {
    s= "(" +ej4Recsm(a-1, b/a, c-1)+ "*" +ej4Recsm(a-2, b/2, c/2)+ ")";
    }
    else {
    s= "(" +ej4Recsm(a/2, b-2, c/2)+ "/" +ej4Recsm(a/3, b-1, c/3)+ ")";
    }

    return s;
    }

    public static String ej4RecCm(Integer a, Integer b, Integer c) {
    //he creado un record trio, porque no consigo importar el de repositorio
    Map<Trio, String> d = new HashMap();
    return ej4RecCmAux(a, b, c, d);

    }

    private static String ej4RecCmAux(Integer a, Integer b, Integer c,Map<Trio,String> d ) {
    Trio clave = new Trio(a, b, c);
    String valor ;

    if (d.containsKey(clave)) {
    valor = d.get(clave);
    }else {
    if (a<2&& b<= 2||c<2) {
    valor = "(" +a.toString() + "+" +b.toString()+"+"+c.toString()+ ")";
    d.put(clave, valor);
    }else if (a<3||b<3&&c<=3) {
    valor= "(" +c.toString() + "-" +b.toString()+"-"+a.toString()+ ")";
    d.put(clave, valor);
    }
    else if ((b%a== 0)&&((a%2==0)||(b%3==0))) {
    valor= "(" +ej4Recsm(a-1, b/a, c-1)+ "*" +ej4Recsm(a-2, b/2, c/2)+ ")";
    d.put(clave, valor);
    }
    else {
    valor= "(" +ej4Recsm(a/2, b-2, c/2)+ "/" +ej4Recsm(a/3, b-1, c/3)+ ")";
    d.put(clave, valor);
    }
    }
    return valor;
    }

```

```

    }
    public static String ej4iterCM(Integer a, Integer b, Integer c) {
        // por limpieza del código lo voy a hacer con for
        Map<Trio, String> d = new HashMap();
        Trio claveFinal = new Trio(a, b, c);
        for (Integer i = 0; i <= a; i++) {
            for (Integer j = 0; j <= b; j++) {
                for (Integer k = 0; k <= c; k++) {
                    Trio clave = new Trio(i, j, k);
                    String valor;
                    if (i < 2 && j <= 2 || k < 2) {
                        valor = "(" + i.toString() + "+" + j.toString() + "+" + k.toString() + ")";
                        d.put(clave, valor);
                    } else if (i < 3 || j < 3 && k <= 3) {
                        valor = "(" + k.toString() + "-" + j.toString() + "-" + i.toString() + ")";
                        d.put(clave, valor);
                    }
                    else if ((j % i == 0) && ((i % 2 == 0) || (j % 3 == 0))) {
                        Trio auxA = new Trio(i - 1, j / i, k - 1);
                        Trio auxB = new Trio(i - 2, j / 2, k / 2);
                        valor = "(" + d.get(auxA) + "*" + d.get(auxB) + ")";
                        d.put(clave, valor);
                    }
                    else {
                        Trio auxC = new Trio(i / 2, j - 2, k / 2);
                        Trio auxD = new Trio(i / 3, j - 1, k / 3);
                        valor = "(" + d.get(auxC) + "/" + d.get(auxD) + ")";
                        d.put(clave, valor);
                    }
                }
            }
        }
        return d.get(claveFinal);
    }
}

```

6. Test:

6.1. Código:

```
public class Test {
    public static void testEj1() {
        Path filePath=Path.of("ficheros/alumnos/PI1Ej1DatosEntrada.txt");
        Consumer<String> con = x-> {
            String [] datos = x.split(",");
            //5,pera,10,pia,20
            Integer a = Integer.parseInt(datos[0]);
            String b = datos[1];
            Integer c= Integer.parseInt(datos[2]);
            String d = datos[3];
            Integer e = Integer.parseInt(datos[4]);

            System.out.println("Versión del enunciado: " + Ejercicios.E1.ejercicioA(a, b, c, d, e));
            System.out.println("Versión Iterativa 1 : " + Ejercicios.E1.ej1Iterativo(a, b, c, d, e));
            System.out.println("Versión Recursiva final : " + Ejercicios.E1.ej1RecursivoFinal(a, b, c,
            d, e));
        };
        try {
            @SuppressWarnings("resource")

            Stream<String> lineas = Files.lines(filePath);

            lineas.forEach(con);

        } catch (Exception err) {
            err.printStackTrace();
        }
    }

    public static void testEj2 (){
        String filePath = "ficheros/alumnos/PI1Ej2DatosEntrada.txt";

        Consumer<String> cnsmr = x -> {
            String[] datos = x.split(",");
            Integer a = Integer.parseInt(datos[0]);
            Integer b = Integer.parseInt(datos[1]);
            String s = datos[2];
            System.out.println("Cadena: " + s);
            System.out.println("Versión recursiva no final: " + Ejercicios.E2.Ej2RecNF(a, b, s));
            System.out.println("Versión recursiva final: " + Ejercicios.E2.Ej2RecFinal(a, b, s));
            System.out.println("Versión Iterativa: " + Ejercicios.E2.Ej2Iterativo(a, b, s));
            System.out.println("Versión Iterativa For: " + Ejercicios.E2.Ej2IterativoFor(a, b, s));
        };
    }
}
```



```

        System.out.println("Versión Funcional v1: " + Ejercicios.E2.Ej2FuncionalV1(a, b, s));
        System.out.println("Versión Funcional v2: " + Ejercicios.E2.Ej2FuncionalV2(a, b, s));
        System.out.println("Versión Funcional v3: " + Ejercicios.E2.Ej2FuncionalV3(a, b, s));
    };

    try {

        Stream<String> lineas = Files.lines(Paths.get(filePath));
        lineas.forEach(cnsmr);
    } catch (Exception err) {
        err.printStackTrace();
    }
}

public static void testEj3() {
    String file1A = "ficheros/alumnos/PI1Ej3DatosEntrada1A.txt";
    String file1B = "ficheros/alumnos/PI1Ej3DatosEntrada1B.txt";
    String file2A = "ficheros/alumnos/PI1Ej3DatosEntrada2A.txt";
    String file2B = "ficheros/alumnos/PI1Ej3DatosEntrada2B.txt";
    String file3A = "ficheros/alumnos/PI1Ej3DatosEntrada3A.txt";
    String file3B = "ficheros/alumnos/PI1Ej3DatosEntrada3B.txt";

    System.out.println("Test1 (Iterativo): " + Ej3.ej3IterativoV1(file1A, file1B));
    System.out.println("Test2 (Recursivo): " + Ej3.ej3RecursivoFinalV1(file2A, file2B));
    System.out.println("Test3 (Funcional): " + Ej3.ej3Funcional(file3A, file3B));
}

public static void testEj4 () {
    String filePath = "ficheros/alumnos/PI1Ej4DatosEntrada.txt";
    Consumer<String> consu = x->{
        String[] datos = x.split(",");
        Integer a = Integer.parseInt(datos[0]);
        Integer b = Integer.parseInt(datos[1]);
        Integer c = Integer.parseInt(datos[2]);
        System.out.println(a.toString() +","+b.toString()+","+c.toString());
        System.out.println("Versión recursivo sin memoria: " + Ej4.ej4Recsm(a, b, c));
        System.out.println("Versión recursivo con memoria: " + Ej4.ej4RecCm(a, b, c));
        System.out.println("Versión Iterativo: " + Ej4.ej4iterCM(a, b, c));
    };
    try {

        Stream<String> lineas = Files.lines(Paths.get(filePath));
        lineas.forEach(consu);
    } catch (Exception err) {
        err.printStackTrace();
    }
}
}

```

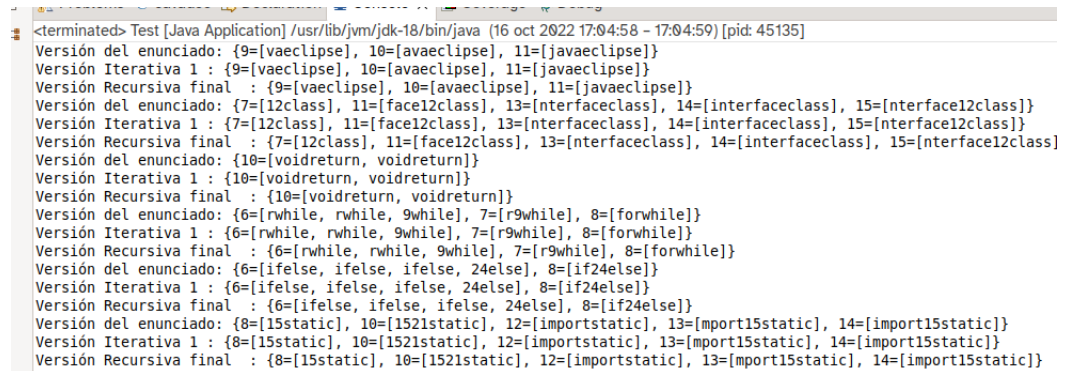
```

public static void main(String[] args) {
    testEj1();
    testEj2();
    testEj3();
    testEj4();
}
}

```

6.2. Imágenes:

6.2.1. Ejercicio 1



```

<terminated> Test [Java Application] /usr/lib/jvm/jdk-18/bin/java (16 oct 2022 17:04:58 - 17:04:59) [pid: 45135]
Versión del enunciado: {9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
Versión Iterativa 1 : {9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
Versión Recursiva final : {9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
Versión del enunciado: {7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
Versión Iterativa 1 : {7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
Versión Recursiva final : {7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
Versión del enunciado: {10=[voidreturn, voidreturn]}
Versión Iterativa 1 : {10=[voidreturn, voidreturn]}
Versión Recursiva final : {10=[voidreturn, voidreturn]}
Versión del enunciado: {6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
Versión Iterativa 1 : {6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
Versión Recursiva final : {6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
Versión del enunciado: {6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
Versión Iterativa 1 : {6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
Versión Recursiva final : {6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
Versión del enunciado: {8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
Versión Iterativa 1 : {8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
Versión Recursiva final : {8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}

```

6.2.2. Ejercicio 2

```

<terminated> Test [Java Application] /usr/lib
Cadena: adda
Versión recursiva no final: 623
Versión recursiva final: 623
Versión Iterativa: 623
Versión Iterativa For: 623
Versión Funcional v1: 623
Versión Funcional v2: 623
Versión Funcional v3: 623
Cadena: second course
Versión recursiva no final: 950
Versión recursiva final: 950
Versión Iterativa: 950
Versión Iterativa For: 950
Versión Funcional v1: 950
Versión Funcional v2: 950
Versión Funcional v3: 950
Cadena: analysis
Versión recursiva no final: 3278
Versión recursiva final: 3278
Versión Iterativa: 3278
Versión Iterativa For: 3278
Versión Funcional v1: 3278
Versión Funcional v2: 3278
Versión Funcional v3: 3278
Cadena: design
Versión recursiva no final: 3135
Versión recursiva final: 3135
Versión Iterativa: 3135
Versión Iterativa For: 3135
Versión Funcional v1: 3135
Versión Funcional v2: 3135
Versión Funcional v3: 3135
Cadena: data
Versión recursiva no final: 3810
Versión recursiva final: 3810
Versión Iterativa: 3810
Versión Iterativa For: 3810
Versión Funcional v1: 3810
Versión Funcional v2: 3810
Versión Funcional v3: 3810
Cadena: algorithms
Versión recursiva no final: 5553
Versión recursiva final: 5553
Versión Iterativa: 5553
Versión Iterativa For: 5553
Versión Funcional v1: 5553
Versión Funcional v2: 5553
Versión Funcional v3: 5553

```

6.2.3. Ejercicio 3

```
<terminated>Test [Java Application] /jdk-18/bin/java [24 oct 2023:19:26:58 - 19:26:58] [pid:18904]
Test1 (Iterativo): [-93.56, 33.78], (82.54, 58.64), (-76.79, 39.38), (-28.03, 59.54), (-19.29, -25.9), (-17.93, -20.26), (24.82, 68.2), (39.87, 48.37), (45.29, 97.59)]
Test2 (Recursivo): [-82.36, 49.76], (-74.69, 40.12), (-72.94, 56.81), (-65.53, 51.45), (-48.56, 81.69), (-47.56, 82.04), (-37.99, 90.32), (-36.56, 38.16), (-8.3, 69.67), (-6.82, -85.27), (23.9, 50.98)]
Test3 (Funcional): [-93.9, -6.76], (-81.49, 23.61), (-71.93, 51.44), (-71.64, -24.87), (-68.08, -8.76), (-62.34, 38.53), (-61.68, -1.78), (-56.16, 41.49), (-54.81, 62.67), (-53.48, 50.98), (-50.98, 97.59)]
Versión funcional con iteradores:
Test3 (Funcional Version 2): [-93.9, -6.76], (-81.49, 23.61), (-71.93, 51.44), (-71.64, -24.87), (-68.08, -8.76), (-62.34, 38.53), (-61.68, -1.78), (-56.16, 41.49), (-54.81, 62.67), (-53.48, -50.98), (-50.98, 97.59)]
```

6.2.4. Ejercicio 4

```
<terminated> Test [Java Application] /usr/lib/jvm/jdk-18/bin/java (16 oct 2022 17:07:43 - 17:07:45) [pid: 45409]
30,20,10
Versión recursivo sin memoria: (((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
Versión recursivo con memoria: (((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
Versión Iterativo: (((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
20,30,10
Versión recursivo sin memoria: (((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
Versión recursivo con memoria: (((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
Versión Iterativo: (((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
20,10,30
Versión recursivo sin memoria: (((3-4-2)/(2-5-1))/((2-5-1)/(1+6+1)))/(((2-5-1)/(1+6+1))/(3-8-2)))
Versión recursivo con memoria: (((3-4-2)/(2-5-1))/((2-5-1)/(1+6+1)))/(((2-5-1)/(1+6+1))/(3-8-2)))
Versión Iterativo: (((3-4-2)/(2-5-1))/((2-5-1)/(1+6+1)))/(((2-5-1)/(1+6+1))/(3-8-2)))
20,15,10
Versión recursivo sin memoria: (((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
Versión recursivo con memoria: (((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
Versión Iterativo: (((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
40,30,20
Versión recursivo sin memoria: (((((2+22+1)/(1+23+0))/(3+25+1))/((3+25+1)/(2+26+1)))/(((3+25+1)/(2+26+1))/((3+7+1)*(2+14+1))))
Versión recursivo con memoria: (((((2+22+1)/(1+23+0))/(3+25+1))/((3+25+1)/(2+26+1)))/(((3+25+1)/(2+26+1))/((3+7+1)*(2+14+1))))
Versión Iterativo: (((((2+22+1)/(1+23+0))/(3+25+1))/((3+25+1)/(2+26+1)))/(((3+25+1)/(2+26+1))/((3+7+1)*(2+14+1))))
60,50,40
Versión recursivo sin memoria: ((((((2+14+1)*(1+21+1))/(2+43+1))/((2+7+1)/(1+8+0))*(3+22+1)))/(((2+7+1)/(1+8+0))*(3+22+1)))/(((1+44+1)/(1+45+0)))
Versión recursivo con memoria: ((((((2+14+1)*(1+21+1))/(2+43+1))/((2+7+1)/(1+8+0))*(3+22+1)))/(((2+7+1)/(1+8+0))*(3+22+1)))/(((1+44+1)/(1+45+0)))
Versión Iterativo: ((((((2+14+1)*(1+21+1))/(2+43+1))/((2+7+1)/(1+8+0))*(3+22+1)))/(((2+7+1)/(1+8+0))*(3+22+1)))/(((1+44+1)/(1+45+0))))
```