PRACTICA INDIVIDUAL 1

MEMORIA 1: Jose Luis Navarro (josnavlop4)

ÍNDICE

Ejercicio 1……3

Ejercicio 2……5

Ejercicio 3……8

Ejercicio 4……11

Código Test.java……13

Ejercicio 1:

SOLUCIÓN FUNCIONAL

```
public static Map<Integer,List<String>> ejercicio1 (Integer varA, String varB, Integer varC, String varD, Integer varE) {
     UnaryOperator<EnteroCadena> nx = elem ->
            {
                return EnteroCadena.of(elem.a()+2,
                        elem.a()%3==0?
                        elem.s()+elem.a().toString():
                        elem.s().substring(elem.a()%elem.s().length()));
            };
    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));
}

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

SOLUCIÓN ITERATIVA

```
public static Map<Integer, List<String>> solucionIterativa(Integer varA, String varB, Integer varC, String varD, Integer varE) {
    Map<Integer,List<String>> ac = new HashMap<>();
    List<String> r;
    while(varA<varC) {
        String en=varB+varD;

        varB = varA%3==0?varB+varA.toString():varB.substring(varA%varB.length());
        varA+=2;

        if(en.length()<varE) {
            Integer clave = en.length();
            if(ac.containsKey(clave)) {
                r = ac.get(clave);
            }else {
                r = new ArrayList<>();
                ac.put(clave, r);
            }
            r.add(en);
        }

    }

    return ac;

}
```

SOLUCIÓN RECURSIVA FINAL

```
public static Map<Integer, List<String>> solucionRecursivaFinal(Integer a, String b, Integer c, String d, Integer e) {
    Map<Integer,List<String>> ac = new HashMap<>();
    ac = solucionRecursivaFinal(ac,a,b,c,d,e);
    return ac;
}

private static Map<Integer, List<String>> solucionRecursivaFinal(Map<Integer, List<String>> ac, Integer varA, String varB, Integer varC,
        String varD, Integer varE) {

    List<String> r;
    if (varA<varC) {
        String en=varB+varD;

        varB = varA%3==0?varB+varA.toString():varB.substring(varA%varB.length());
        varA+=2;

        if(en.length()<varE) {
            Integer clave = en.length();
            if(ac.containsKey(clave)) {
                r = ac.get(clave);
            }else {
                r = new ArrayList<>();
                ac.put(clave, r);
            }
            r.add(en);
            ac = solucionRecursivaFinal(ac,varA,varB,varC,varD,varE);

        }

    }
    return ac;
}
```

TEST

```
Datos: 5,java,10,eclipse,20
1) Solucion Funcional:
{9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
2) Solucion Iterativa:
{9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
3) Solucion Recursiva Final:
{9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
-----------------------------------------------------------------------------------------
Datos: 10,interface,20,class,30
1) Solucion Funcional:
{7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
2) Solucion Iterativa:
{7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
3) Solucion Recursiva Final:
{7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
-----------------------------------------------------------------------------------------
Datos: 4,void,8,return,16
1) Solucion Funcional:
{10=[voidreturn, voidreturn]}
2) Solucion Iterativa:
{10=[voidreturn, voidreturn]}
3) Solucion Recursiva Final:
{10=[voidreturn, voidreturn]}
-----------------------------------------------------------------------------------------
Datos: 5,for,15,while,25
1) Solucion Funcional:
{6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
2) Solucion Iterativa:
{6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
3) Solucion Recursiva Final:
{6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
-----------------------------------------------------------------------------------------
Datos: 20,if,30,else,40
1) Solucion Funcional:
{6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
2) Solucion Iterativa:
{6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
3) Solucion Recursiva Final:
{6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
-----------------------------------------------------------------------------------------
Datos: 15,import,25,static,50
1) Solucion Funcional:
{8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
2) Solucion Iterativa:
{8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
3) Solucion Recursiva Final:
{8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
-----------------------------------------------------------------------------------------
```

Ejercicio 2:

SOLUCIÓN ITERATIVA

```
public static Integer solucionIterativaWhile(Integer a, Integer b, String s) {
    Integer ac = 0;
    Integer res = null;

    while(!((s.length()==0)||(a<2||b<2))) {
        if(a%s.length()<b%s.length()) {
            ac = a+b+ac;
            s = s.substring(a%s.length(),b%s.length());
            a--;
            b/=2;
        }else {
            ac = a*b+ac;
            s = s.substring(b%s.length(),a%s.length());
            a/=2;
            b--;
        }
    }
    if(s.length()==0) {
        res = a * a + b * b + ac;
    } else if(a<2||b<2) {
        res = s.length()+ a + b + ac;
    }
    return res;
}
```

SOLUCIÓN RECURSIVA NO FINAL

```
public static Integer solucionRecursivaNoFinal(Integer a, Integer b, String s) {
    Integer ac = null;
    if(s.length()==0) {
        ac = a*a+b*b;
    }else if(a<2||b<2) {
        ac = s.length()+a+b;
    }else {
        if(a%s.length()<b%s.length()) {
            ac = a + b + solucionRecursivaNoFinal(a-1,b/2,s.substring(a%s.length(),b%s.length()));
        } else {
            ac = a * b + solucionRecursivaNoFinal(a/2,b-1,s.substring(b%s.length(),a%s.length()));
        }
    }
    return ac;
}
```

SOLUCIÓN RECURSIVA FINAL

```
public static Integer solucionRecursivaFinal(Integer a, Integer b, String s) {
    Integer ac = 0;
    ac = solucionRecursivaFinal(a,b,s,ac);
    return ac;
}


private static Integer solucionRecursivaFinal(Integer a, Integer b, String s, Integer ac) {
    if(s.length()==0) {
        ac = a * a + b * b + ac;
    }else if(a<2||b<2) {
        ac = s.length()+ a + b + ac;
    }else {
        if(a%s.length()<b%s.length()) {
            ac = solucionRecursivaFinal(a-1,b/2,s.substring(a%s.length(),b%s.length()),a+b+ac);
        } else {
            ac = solucionRecursivaFinal(a/2,b-1,s.substring(b%s.length(),a%s.length()),a*b+ac);
        }
    }
    return ac;

}
```

SOLUCIÓN FUNCIONAL

```
public static record Tupla(Integer ac,Integer a,Integer b, String s) {
    public static Tupla of (Integer ac, Integer a, Integer b, String s) {
        return new Tupla(ac,a,b,s);
    }
    public static Tupla first(Integer a, Integer b, String s) {
        //valor inicial de la secuencia
        return of(0,a,b,s);
    }
    public Tupla next() {
        //Siguiente elemento
        Tupla nx = null;
        if(a%s.length()<b%s.length()) {
            nx = of (a+b+ac,a-1,b/2,s.substring(a%s.length(),b%s.length()));
        } else {
            nx = of (a*b+ac,a/2,b-1,s.substring(b%s.length(),a%s.length()));
        }
        return nx;
    }
    public Boolean isBaseCase() {
        return (s.length()==0)||(a<2||b<2);
    }
}
public static Integer solucionFuncional(Integer a, Integer b, String s) {
    Tupla elementoFinal = Stream.iterate(Tupla.first(a,b,s), elem->elem.next())
            .filter(elem->elem.isBaseCase())
            .findFirst()
            .get();

    Integer res = 0;
    if (elementoFinal.s.length() == 0) {
        res = elementoFinal.a * elementoFinal.a + elementoFinal.b * elementoFinal.b + elementoFinal.ac;
    }else if (elementoFinal.a<2||elementoFinal.b<2) {
        res = elementoFinal.s.length() + elementoFinal.a + elementoFinal.b + elementoFinal.ac;
    }
    return res;
}
```

TEST

```
Datos: 10,20,adda
1) Solucion Iterativa:
623
2) Solucion Recursiva NO Final:
623
3) Solucion Recursiva Final:
623
4) Solucion Funcional:
623
-----------------------------------------------------------------------------------------
Datos: 20,30,second course
1) Solucion Iterativa:
950
2) Solucion Recursiva NO Final:
950
3) Solucion Recursiva Final:
950
4) Solucion Funcional:
950
-----------------------------------------------------------------------------------------
Datos: 30,40,analysis
1) Solucion Iterativa:
3278
2) Solucion Recursiva NO Final:
3278
3) Solucion Recursiva Final:
3278
4) Solucion Funcional:
3278
-----------------------------------------------------------------------------------------
Datos: 40,50,design
1) Solucion Iterativa:
3135
2) Solucion Recursiva NO Final:
3135
3) Solucion Recursiva Final:
3135
4) Solucion Funcional:
3135
-----------------------------------------------------------------------------------------
Datos: 50,75,data
1) Solucion Iterativa:
3810
2) Solucion Recursiva NO Final:
3810
3) Solucion Recursiva Final:
3810
4) Solucion Funcional:
3810
-----------------------------------------------------------------------------------------
Datos: 75,50,algorithms
1) Solucion Iterativa:
5553
2) Solucion Recursiva NO Final:
5553
3) Solucion Recursiva Final:
5553
4) Solucion Funcional:
5553
-----------------------------------------------------------------------------------------
```

Ejercicio 3:

SOLUCIÓN ITERATIVA

```
public static List<Punto2D> Ejercicio3Iterativo (String f1,String f2){
    List<Punto2D> ls = new ArrayList<>();

    Iterator<String> it1 = Stream2.file(f1).iterator();
    Iterator<String> it2 = Stream2.file(f2).iterator();

    Punto2D p1 = null;
    Punto2D p2 = null;

    p1 = next(it1,p1);
    p2 = next(it2,p2);

    while (p1 != null || p2 != null) {
        if (p2 == null || (p1 != null && p1.compareTo(p2) <= 0)) {
            ls.add(p1);
            p1 = next(it1,p1);
        } else if (p2 != null){
            ls.add(p2);
            p2 = next(it2,p2);
        }
    }
    return ls;
}

public static Punto2D parse(Iterator<String> it) {
    String [] v=null;
    if (it.hasNext()) {
        v=it.next().split(",");
        return Punto2D.of(Double.valueOf(v[0]),Double.valueOf(v[1]));
    }
    return null;
}

public static Punto2D next(Iterator<String> it,Punto2D p) {
    Retorna el siguiente punto si cumple las condiciones
    while(it.hasNext()) {
        p=parse(it);
        if(esPrimOTerCuadrant(p)) {
            return p;
        }
    }
    return null;
}

public static Boolean esPrimOTerCuadrant(Punto2D p) {
    return p.getCuadrante().equals(Cuadrante.PRIMER_CUADRANTE)||p.getCuadrante().equals(Cuadrante.TERCER_CUADRANTE);
}
```

SOLUCIÓN RECURSIVA FINAL

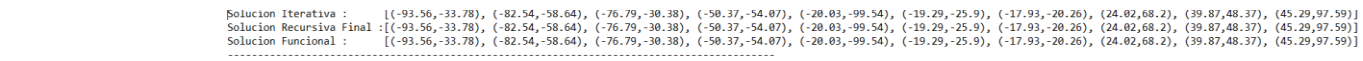
```
public static List<Punto2D> Ejercicio3RecursivoFinal (String f1,String f2){
    List<Punto2D> ac = new ArrayList<>();
    Iterator<String> it1 = Stream2.file(f1).iterator();
    Iterator<String> it2 = Stream2.file(f2).iterator();

    Punto2D p1 = null;
    Punto2D p2 = null;
    p1 = next(it1,p1);
    p2 = next(it2,p2);

    ac = Ejercicio3RecursivoFinal(it1,it2,p1,p2,ac);
    return ac;
}

private static List<Punto2D> Ejercicio3RecursivoFinal(Iterator<String> it1,Iterator<String> it2,Punto2D p1,Punto2D p2,
        List<Punto2D> ac) {
    List<Punto2D> res = new ArrayList<>();

    if (!(p1 != null || p2 != null)) {
        res = ac;
    } else {
        if (p2 == null || (p1 != null && p1.compareTo(p2) <= 0)) {
            ac.add(p1);
            p1 = next(it1,p1);
        } else if (p2 != null){
            ac.add(p2);
            p2 = next(it2,p2);
        }
        res = Ejercicio3RecursivoFinal(it1,it2,p1,p2,ac);
    }
    return res;
}
```

SOLUCIÓN FUNCIONAL

```
public record RecordEjercicio3(Iterator<String> it1, Iterator<String> it2, Punto2D p1,Punto2D p2,List<Punto2D> ac) {

    public static RecordEjercicio3 of(Iterator<String> it1, Iterator<String> it2, Punto2D p1,Punto2D p2,List<Punto2D> ac) {
        return new RecordEjercicio3 (it1, it2, p1, p2, ac);
    }

    public RecordEjercicio3 next2() {
        RecordEjercicio3 next=null;
        if (p2==null || (p1 != null && p1.compareTo(p2) <= 0)) {
            ac.add(p1);
            next=RecordEjercicio3.of(it1, it2, next(it1,p1), p2, ac);
        } else {
            ac.add(p2);
            next=RecordEjercicio3.of(it1, it2, p1, next(it2,p2), ac);
        }

        return next;
    }

    public Boolean isBaseCase() {
        return (!(p1 != null || p2 != null));
    }

}

public static List<Punto2D> Ejercicio3Funcional (String f1,String f2){
    Iterator<String> it1 = Stream2.file(f1).iterator();
    Iterator<String> it2 = Stream2.file(f2).iterator();

    Punto2D p1 = null;
    Punto2D p2 = null;
    p1 = next(it1,p1);
    p2 = next(it2,p2);

    RecordEjercicio3 elem=Stream.iterate( RecordEjercicio3.of(it1,it2,p1,p2,new ArrayList<>()), t->t.next2())
            .filter(t->t.isBaseCase())
            .findFirst()
            .get();
    return elem.ac();
}
```

TEST

Fichero 1A y 1B:



(El resto son muy largos como para una captura así que los copio y pego):

Fichero 2A y 2B

Solucion Iterativa : [(-82.35,-49.74), (-74.69,-40.12), (-72.94,-56.8), (-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), (3.45,70.0), (23.93,76.13), (30.7,8.47), (37.97,49.79), (40.55,83.01), (41.78,39.55), (49.46,51.93), (64.29,86.49), (74.78,41.09), (87.62,43.21)]
Solucion Recursiva Final :[(-82.35,-49.74), (-74.69,-40.12), (-72.94,-56.8), (-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), (3.45,70.0), (23.93,76.13), (30.7,8.47), (37.97,49.79), (40.55,83.01), (41.78,39.55), (49.46,51.93), (64.29,86.49), (74.78,41.09), (87.62,43.21)]
Solucion Funcional : [(-82.35,-49.74), (-74.69,-40.12), (-72.94,-56.8), (-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), (3.45,70.0), (23.93,76.13), (30.7,8.47), (37.97,49.79), (40.55,83.01), (41.78,39.55), (49.46,51.93), (64.29,86.49), (74.78,41.09), (87.62,43.21)]

Fichero 3A y 3B

Solucion Iterativa : [(-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,-26.67), (-53.48,-50.98), (-50.04,-96.54), (-46.99,-83.11), (-33.11,-92.17), (-32.08,-66.57), (-29.99,-72.32), (-20.6,-8.85), (-19.83,-5.01), (-19.58,-94.75), (-17.35,-76.96), (-16.97,-96.8), (-11.75,-13.63), (0.42,13.94), (9.07,33.36), (10.69,95.3), (14.7,82.66), (15.68,26.66), (16.33,54.0), (16.78,55.2), (28.38,81.47), (28.91,91.34), (35.75,38.79), (45.23,56.37), (45.41,82.21), (47.42,41.06), (53.42,66.34), (55.06,57.38), (58.08,11.18), (60.16,59.96), (60.68,8.38), (65.54,70.44), (68.32,23.46), (78.6,69.48), (79.09,80.75), (79.3,62.79), (79.76,69.36), (84.74,31.62), (86.21,86.12), (87.89,49.68), (90.47,25.64), (96.34,83.99)]

Solucion Recursiva Final :[(-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,-26.67), (-53.48,-50.98), (-50.04,-96.54), (-46.99,-83.11), (-33.11,-92.17), (-32.08,-66.57), (-29.99,-72.32), (-20.6,-8.85), (-19.83,-5.01), (-19.58,-94.75), (-17.35,-76.96), (-16.97,-96.8), (-11.75,-13.63), (0.42,13.94), (9.07,33.36), (10.69,95.3), (14.7,82.66), (15.68,26.66), (16.33,54.0), (16.78,55.2), (28.38,81.47), (28.91,91.34), (35.75,38.79), (45.23,56.37), (45.41,82.21), (47.42,41.06), (53.42,66.34), (55.06,57.38), (58.08,11.18), (60.16,59.96), (60.68,8.38), (65.54,70.44), (68.32,23.46), (78.6,69.48), (79.09,80.75), (79.3,62.79), (79.76,69.36), (84.74,31.62), (86.21,86.12), (87.89,49.68), (90.47,25.64), (96.34,83.99)]
Solucion 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,-26.67), (-53.48,-50.98), (-50.04,-96.54), (-46.99,-83.11), (-33.11,-92.17), (-32.08,-66.57), (-29.99,-72.32), (-20.6,-8.85), (-19.83,-5.01), (-19.58,-94.75), (-17.35,-76.96), (-16.97,-96.8), (-11.75,-13.63), (0.42,13.94), (9.07,33.36), (10.69,95.3), (14.7,82.66), (15.68,26.66), (16.33,54.0), (16.78,55.2), (28.38,81.47), (28.91,91.34), (35.75,38.79), (45.23,56.37), (45.41,82.21), (47.42,41.06), (53.42,66.34), (55.06,57.38), (58.08,11.18), (60.16,59.96), (60.68,8.38), (65.54,70.44), (68.32,23.46), (78.6,69.48), (79.09,80.75), (79.3,62.79), (79.76,69.36), (84.74,31.62), (86.21,86.12), (87.89,49.68), (90.47,25.64), (96.34,83.99)]

Ejercicio 4:

SOLUCION RECURSIVA SIN MEMORIA

```
public static String solucionRecursivaSinMem(Integer a, Integer b, Integer c) {
    String ac = null;
    if ((a<2&&b<=2)||c<2) {
        ac="("+a.toString()+"+"+b.toString()+"+"+c.toString()+")";
    }else if (a<3||(b<3&&c<=3)) {
        ac="("+a.toString()+"-"+b.toString()+"-"+c.toString()+")";
    }else {
        if (b%a==0&&(a%2==0||b%3==0)) {
            ac="("+solucionRecursivaSinMem(a-1, b/a,c-1)+"*"+solucionRecursivaSinMem(a-2, b/2,c/2)+")";
        }else {
            ac="("+solucionRecursivaSinMem(a/2, b-2,c/2)+"/"+solucionRecursivaSinMem(a/3, b-1,c/3)+")";
        }
    }
    return ac;
}
```

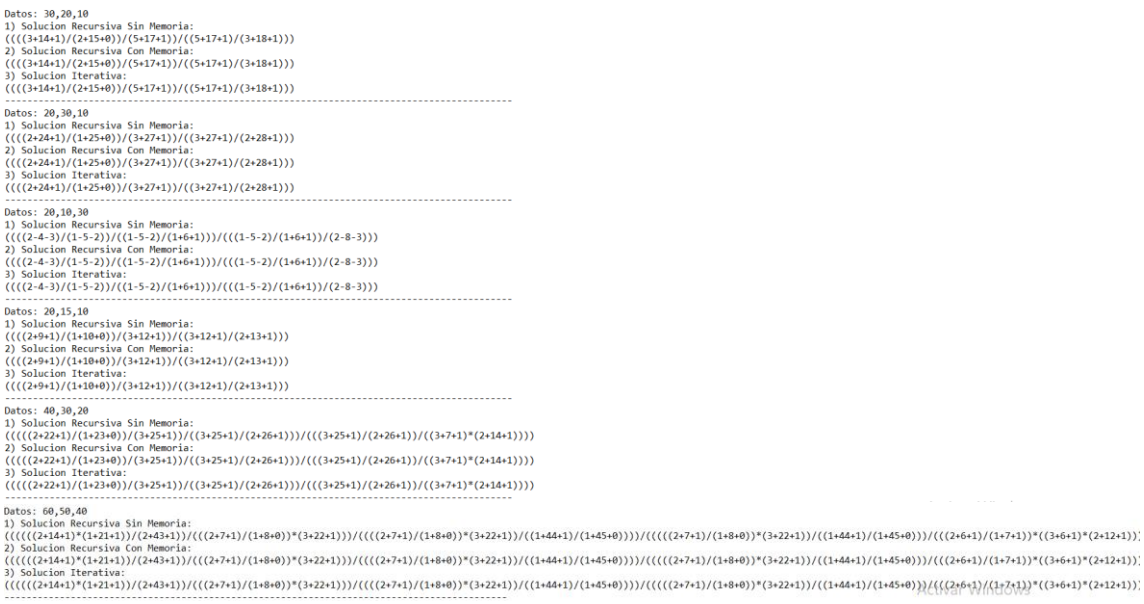
SOLUCION RECURSIVA CON MEMORIA

```
public static String solucionRecursivaConMem(Integer a, Integer b, Integer c) {
    Map<IntTrio,String> m = new HashMap<>();
    return solucionRecursivaConMem(a,b,c,m);
}

private static String solucionRecursivaConMem(Integer a, Integer b, Integer c, Map<IntTrio, String> m) {
    String ac="";
    IntTrio key = IntTrio.of(a,b,c);
    if (m.containsKey(key)) {
        ac = m.get(key);
    }else {
        if ((a<2&&b<=2)||c<2) {
            ac="("+a.toString()+"+"+b.toString()+"+"+c.toString()+")";
        }else if (a<3||(b<3&&c<=3)) {
            ac="("+a.toString()+"-"+b.toString()+"-"+c.toString()+")";
        }else {
            if (b%a==0&&(a%2==0||b%3==0)) {
                ac="("+solucionRecursivaConMem(a-1, b/a,c-1,m)+"*"+solucionRecursivaConMem(a-2, b/2,c/2,m)+")";
            }else {
                ac="("+solucionRecursivaConMem(a/2, b-2,c/2,m)+"/"+solucionRecursivaConMem(a/3, b-1,c/3,m)+")";
            }
        }
        m.put(IntTrio.of(a, b, c), ac);
    }
    return ac;
}
```

SOLUCION ITERATIVA

```
public static String solucionIterativa(Integer a, Integer b, Integer c) {
    Map<IntTrio,String> m = new HashMap<>();
    String ac = "";

    for (Integer i = 0;i<=a;i++) {
        for (Integer j= 0;j<=b;j++) {
            for (Integer k=0;k<=c;k++) {
                if ((i<2&&j<=2)||k<2) {
                    ac="("+i.toString()+"+"+j.toString()+"+"+k.toString()+")";
                }else if (i<3||(j<3&&k<=3)) {
                    ac="("+i.toString()+"-"+j.toString()+"-"+k.toString()+")";
                }else {
                    if (j%i==0&&(i%2==0||j%3==0)) {
                        ac="("+m.get(IntTrio.of(i-1, j/i,k-1))+"*"+m.get(IntTrio.of(i-2, j/2,k/2))+")";
                    }else {
                        ac="("+m.get(IntTrio.of(i/2, j-2,k/2))+"/"+m.get(IntTrio.of(i/3, j-1,k/3))+")";
                    }
                }
                m.put(IntTrio.of(i, j, k), ac);
            }
        }
    }
    return m.get(IntTrio.of(a,b,c));
}
```

TEST



```
Datos: 30,20,10
1) Solucion Recursiva Sin Memoria:
((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
2) Solucion Recursiva Con Memoria:
((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
3) Solucion Iterativa:
((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
--------------------------------------------------------------------------------
Datos: 20,30,10
1) Solucion Recursiva Sin Memoria:
((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
2) Solucion Recursiva Con Memoria:
((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
3) Solucion Iterativa:
((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
--------------------------------------------------------------------------------
Datos: 20,10,30
1) Solucion Recursiva Sin Memoria:
((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))
2) Solucion Recursiva Con Memoria:
((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))
3) Solucion Iterativa:
((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))
--------------------------------------------------------------------------------
Datos: 20,15,10
1) Solucion Recursiva Sin Memoria:
((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
2) Solucion Recursiva Con Memoria:
((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
3) Solucion Iterativa:
((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
--------------------------------------------------------------------------------
Datos: 40,30,20
1) Solucion Recursiva 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))))
2) Solucion Recursiva 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))))
3) Solucion Iterativa:
(((((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))))
--------------------------------------------------------------------------------
Datos: 60,50,40
1) Solucion Recursiva 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))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3+6+1)*(2+12+1)))
2) Solucion Recursiva 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))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3+6+1)*(2+12+1)))
3) Solucion Iterativa:
((((((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))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3+6+1)*(2+12+1)))
--------------------------------------------------------------------------------
```

Datos: 30,20,10
1) Solucion Recursiva Sin Memoria:
((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
2) Solucion Recursiva Con Memoria:
((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
3) Solucion Iterativa:
((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))

Datos: 20,30,10
1) Solucion Recursiva Sin Memoria:
((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
2) Solucion Recursiva Con Memoria:
((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
3) Solucion Iterativa:
((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))

Datos: 20,10,30
1) Solucion Recursiva Sin Memoria:
((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))
2) Solucion Recursiva Con Memoria:
((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))
3) Solucion Iterativa:
((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))

Datos: 20,15,10
1) Solucion Recursiva Sin Memoria:
((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
2) Solucion Recursiva Con Memoria:
((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
3) Solucion Iterativa:
((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))

Datos: 40,30,20
1) Solucion Recursiva 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))))
2) Solucion Recursiva 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))))
3) Solucion Iterativa:
(((((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))))

Datos: 60,50,40
1) Solucion Recursiva 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))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3+6+1)*(2+12+1)))))
2) Solucion Recursiva 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))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3+6+1)*(2+12+1)))))
3) Solucion Iterativa:
((((((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))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3+6+1)*(2+12+1)))))

Código Test.java

```
public static void main(String[] args) {
    Ejercicio1("ficheros/PI1Ej1DatosEntrada.txt");

    Ejercicio2("ficheros/PI1Ej2DatosEntrada.txt");

    Ejercicio3("ficheros/PI1Ej3DatosEntrada1A.txt",
            "ficheros/PI1Ej3DatosEntrada1B.txt");
    Ejercicio3("ficheros/PI1Ej3DatosEntrada2A.txt",
            "ficheros/PI1Ej3DatosEntrada2B.txt");
    Ejercicio3("ficheros/PI1Ej3DatosEntrada3A.txt",
            "ficheros/PI1Ej3DatosEntrada3B.txt");

    Ejercicio4("ficheros/PI1Ej4DatosEntrada.txt");

}

private static void Ejercicio1(String fichero) {
    List<String> f = Files2.linesFromFile(fichero);

    for (String linea:f) {
        String[] trozo = linea.split(",");

        Integer a = Integer.parseInt(trozo[0].trim());
        String b = trozo[1].trim();
        Integer c = Integer.parseInt(trozo[2].trim());
        String d = trozo[3].trim();
        Integer e = Integer.parseInt(trozo[4].trim());

        System.out.println("Datos: " + linea);
        System.out.println("1) Solucion Funcional: \n" + ejercicios.Ejercicio1.ejercicio1(a,b,c,d,e));
        System.out.println("2) Solucion Iterativa: \n" + ejercicios.Ejercicio1.solucionIterativa(a,b,c,d,e));
        System.out.println("3) Solucion Recursiva Final: \n" + ejercicios.Ejercicio1.solucionRecursivaFinal(a,b,c,d,e));
        System.out.println("-----------------------------------------------------------------------------------------------");
    }
}
private static void Ejercicio2(String fichero) {
    List<String> f = Files2.linesFromFile(fichero);

    for (String linea:f) {
        String[] trozo = linea.split(",");

        Integer a = Integer.parseInt(trozo[0].trim());
        Integer b = Integer.parseInt(trozo[1].trim());
        String s = trozo[2].trim();

        System.out.println("Datos: " + linea);
        System.out.println("1) Solucion Iterativa: \n" + ejercicios.Ejercicio2.solucionIterativaWhile(a, b, s));
        System.out.println("2) Solucion Recursiva NO Final: \n" + ejercicios.Ejercicio2.solucionRecursivaNoFinal(a,b,s));
        System.out.println("3) Solucion Recursiva Final: \n" + ejercicios.Ejercicio2.solucionRecursivaFinal(a,b,s));
        System.out.println("4) Solucion Funcional: \n" + ejercicios.Ejercicio2.solucionFuncional(a,b,s));

        System.out.println("-----------------------------------------------------------------------------------------------");


    }
}
private static void Ejercicio3(String f1, String f2) {
    System.out.println("Solucion Iterativa :      "+ejercicios.Ejercicio3.Ejercicio3Iterativo(f1,f2));
    System.out.println("Solucion Recursiva Final :"+ejercicios.Ejercicio3.Ejercicio3RecursivoFinal(f1,f2));
    System.out.println("Solucion Funcional :      "+ejercicios.Ejercicio3.Ejercicio3Funcional(f1,f2));
    System.out.println("-----------------------------------------------------------------------------------------------");
}

private static void Ejercicio4(String fichero) {
    List<String> f = Files2.linesFromFile(fichero);

    for (String linea:f) {
        String[] trozo = linea.split(",");

        Integer a = Integer.parseInt(trozo[0].trim());
        Integer b = Integer.parseInt(trozo[1].trim());
        Integer c = Integer.parseInt(trozo[2].trim());

        System.out.println("Datos: " + linea);
        System.out.println("1) Solucion Recursiva Sin Memoria: \n" + ejercicios.Ejercicio4.solucionRecursivaSinMem(a,b,c));
        System.out.println("2) Solucion Recursiva Con Memoria: \n" + ejercicios.Ejercicio4.solucionRecursivaConMem(a,b,c));
        System.out.println("3) Solucion Iterativa: \n" + ejercicios.Ejercicio4.solucionIterativa(a, b, c));

        System.out.println("-----------------------------------------------------------------------------------------------");
    }

}
```