Divide y Vencerás.

La idea básica de este esquema consiste en lo siguiente:

1. Dado un problema P, con datos D, si los datos permiten una solución directa, se ofrece ésta.

2. En caso contrario, se siguen las siguientes fases:

a) Se dividen los datos D en varios conjuntos de datos más pequeños, D_i.
b) Se resuelven los problemas $P(D_i)$ parciales, sobre los conjuntos de datos D_i, recursivamente.
c) Se combinan las soluciones parciales, resultando así la solución final.

Esquema genérico:

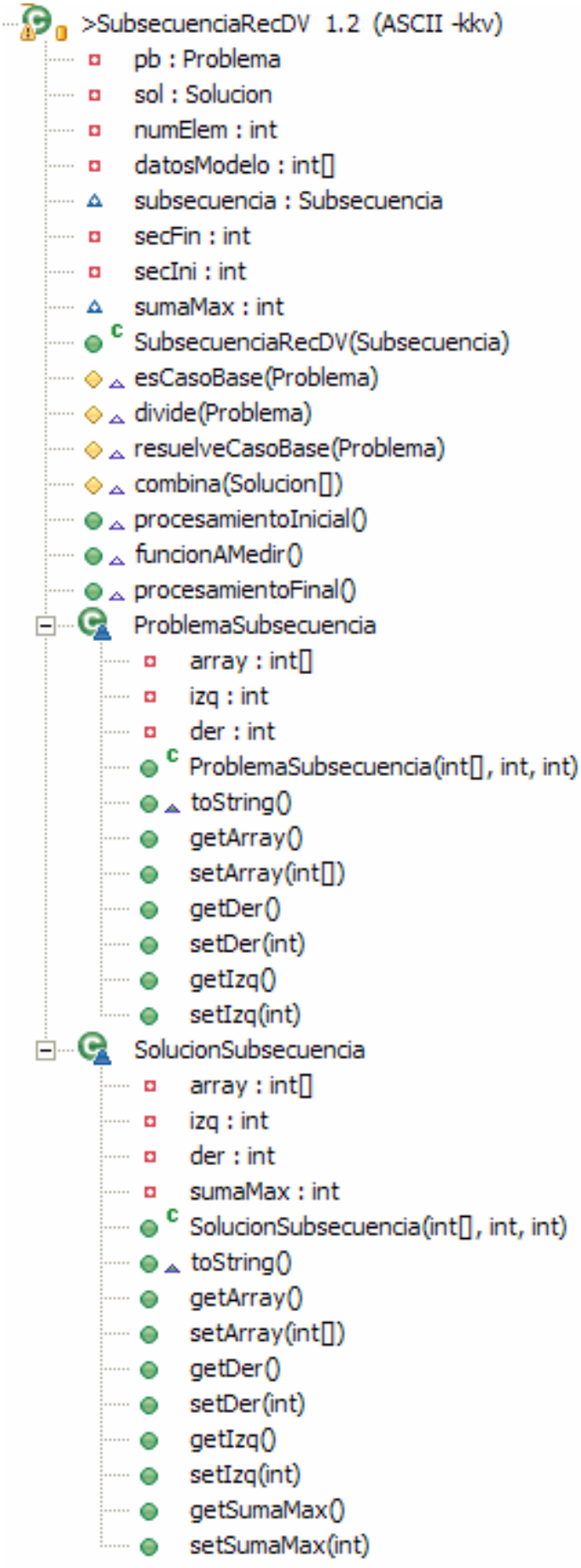
El esquema (traducido a *J*ava) es el siguiente:

```
protected public Solucion dYV(Problema p) {
  Problema[] subproblemas;
  Solucion[] subsoluciones;
  Solucion s;
  int a;
  if (esCasoBase(p)) {
    s = resuelveCasoBase(p);
  } else {
      subproblemas = divide(p);
      a = subproblemas.length;
      subsoluciones = new Solucion[a];
      for (int i = 0; i < a; i++) {
        subsoluciones[i] = dYV(subproblemas[i]);
      }
      s = combina(subsoluciones);
    }
  return s;
}
```

```java
abstract public class EsquemaDYV {
    /**
    protected Solucion dYV(Problema p) {
        Problema[] subproblemas;
        Solucion[] subsoluciones;
        Solucion s;
        int a;

        if (esCasoBase(p)) {
            s = resuelveCasoBase(p);
        } else {
            subproblemas = divide(p);
            a = subproblemas.length;
            subsoluciones = new Solucion[a];
            for (int i = 0; i < a; i++) {
                subsoluciones[i] = dYV(subproblemas[i]);
            }
            s = combina(subsoluciones);
        }
        return s;
    }

    abstract protected boolean esCasoBase(Problema p);

    abstract protected Solucion resuelveCasoBase(Problema p);

    abstract protected Problema[] divide(Problema p);

    abstract protected Solucion combina(Solucion[] s);

    /**
    public interface Problema {
    }
    /**
    public interface Solucion {
    }
}
```

Problema de la Subsecuencia

```java
public class Subsecuencia {

    private int numElem;
    private int rango;
    public int[] datosModelo;

    public Subsecuencia(int numElem, int rango) {
        this.numElem = numElem;
        this.rango = rango;
        datosModelo = new int[numElem];
        inicializaModelo();
    }

    private void inicializaModelo() {
        Random rn = new Random();
        for (int i = 0; i < numElem; i++) {
            datosModelo[i] = rn.nextInt() % rango + 1;
        }
    }

    public String getModelo() {
        if (datosModelo == null) {
            return null;
        } else {
            String s = "{";
            for (int i = 0; i < datosModelo.length; i++) {
                s += datosModelo[i];
                s += (i < datosModelo.length - 1 ? ", " : "}");
            }
            return s;
        }
    }

    public String toString() {
        String s = "Problema subsecuencia con " + numElem;
        return s;
    }
    public int getNumElem() {
    public int[] getDatosModelo() {
    public void setDatosModelo(int[] datosModelo) {
    public void setNumElem(int numElem) {
    public int getRango() {
    public void setRango(int rango) {
}
```


>SubsecuenciaRecDV 1.2 (ASCII -kkv)
pb : Problema
sol : Solucion
numElem : int
datosModelo : int[]
subsecuencia : Subsecuencia
secFin : int
secIni : int
sumaMax : int
SubsecuenciaRecDV(Subsecuencia)
esCasoBase(Problema)
divide(Problema)
resuelveCasoBase(Problema)
combina(Solucion[])
procesamientoInicial()
funcionAMedir()
procesamientoFinal()
ProblemaSubsecuencia
array : int[]
izq : int
der : int
ProblemaSubsecuencia(int[], int, int)
toString()
getArray()
setArray(int[])
getDer()
setDer(int)
getIzq()
setIzq(int)
SolucionSubsecuencia
array : int[]
izq : int
der : int
sumaMax : int
SolucionSubsecuencia(int[], int, int)
toString()
getArray()
setArray(int[])
getDer()
setDer(int)
getIzq()
setIzq(int)
getSumaMax()
setSumaMax(int)

```java
class ProblemaSubsecuencia implements Problema {
    private int[] array;
    private int izq;
    private int der;

    public ProblemaSubsecuencia(int[] array, int izq, int der) {
        super();
        this.array = array;
        this.izq = izq;
        this.der = der;
    }
    public String toString() {
        String s = "{";
        for (int i = izq; i <= der; i++) {
            if (i < izq) {
                s += array[i] + ", ";
            } else {
                s += array[i] + "}";
            }
        }
        return s;
    }

    public int[] getArray() {

    public void setArray(int[] array) {

    public int getDer() {

    public void setDer(int der) {

    public int getIzq() {

    public void setIzq(int izq) {
}
```

```java
class SolucionSubsecuencia implements Solucion {

    private int[] array;
    private int izq;
    private int der;
    private int sumaMax;

    public SolucionSubsecuencia(int[] array, int izq, int der) {
        super();
        this.array = array;
        this.izq = izq;
        this.der = der;
    }
    public String toString() {
        String s = "{";
        for (int i = izq; i <= der; i++) {
            if (i < izq) {
                s += array[i] + ", ";
            } else {
                s += array[i] + "}";
            }
        }
        return s;
    }
    public int[] getArray() {
    public void setArray(int[] array) {
    public int getDer() {
    public void setDer(int der) {
    public int getIzq() {
    public void setIzq(int izq) {
    public int getSumaMax() {
    public void setSumaMax(int sumaMax) {
}
```

```java
public class SubsecuenciaRecDV extends EsquemaDYV implements Temporizable {

    private Problema pb;
    private Solucion sol;
    private int numElem;
    private int[] datosModelo;
    private Subsecuencia subsecuencia;
    private int secIni = 0;
    private int secFin = 0;
    int sumaMax = 0;

    public SubsecuenciaRecDV(Subsecuencia subsecuencia) {
        super();
        this.subsecuencia = subsecuencia;
        this.numElem = subsecuencia.getNumElem();
        this.datosModelo = subsecuencia.getDatosModelo();
    }

    public void procesamientoInicial() {
        pb = new ProblemaSubsecuencia(datosModelo, 0, datosModelo.length - 1);
    }

    public void funcionAMedir() {
        sol = dYV(pb);
    }

    public void procesamientoFinal() {

    }
```

```
protected boolean esCasoBase(Problema p) {
    if (((ProblemaSubsecuencia) p).getDer() == ((ProblemaSubsecuencia) p).getIzq()) {
        // el array tiene un solo elemento
        return true;
    } else {
        return false;
    }
}
protected Solucion resuelveCasoBase(Problema p) {
    ProblemaSubsecuencia p1 = (ProblemaSubsecuencia) p;
    if (p1.getIzq() > 0) {
        SolucionSubsecuencia s1 = new SolucionSubsecuencia(p1.getArray(),
                p1.getIzq(), p1.getDer());
        s1.setSumaMax(p1.getIzq());
        return s1;

    } else {
        SolucionSubsecuencia s2 = new SolucionSubsecuencia(p1.getArray(),
                p1.getIzq(), p1.getDer());
        s2.setSumaMax(0);
        return s2;
    }
}

protected Problema[] divide(Problema p) {
    Problema[] pbs = new Problema[2];
    ProblemaSubsecuencia p1 = (ProblemaSubsecuencia) p;
    int centro = (p1.getIzq() + p1.getDer()) / 2;
    pbs[0] = new ProblemaSubsecuencia(p1.getArray(), p1.getIzq(), centro);
    pbs[1] = new ProblemaSubsecuencia(p1.getArray(), centro + 1, p1
            .getDer());

    return pbs;
}
```

```
protected Solucion combina(Solucion[] s) {
    int sumaMaxIzq = ((SolucionSubsecuencia) s[0]).getSumaMax();
    int sumaMaxDer = ((SolucionSubsecuencia) s[1]).getSumaMax();
    int izq = ((SolucionSubsecuencia) s[0]).getIzq();
    int der = ((SolucionSubsecuencia) s[1]).getDer();
    int centro = (izq + der) / 2;
    int sumaMaxIzqBorde = 0;
    int sumaBordeIzq = 0;
    for (int i = centro; i >= izq; i--) {
        sumaBordeIzq = sumaBordeIzq + subsecuencia.datosModelo[i];
        if (sumaBordeIzq > sumaMaxIzqBorde) {
            sumaMaxIzqBorde = sumaBordeIzq;
            secIni = i;
        }
    }
    int sumaMaxDerBorde = 0;
    int sumaBordeDer = 0;
    for (int i = centro + 1; i <= der; i++) {
        sumaBordeDer = sumaBordeDer + subsecuencia.datosModelo[i];
        if (sumaBordeDer > sumaMaxDerBorde) {
            sumaMaxDerBorde = sumaBordeDer;
            secFin = i;
        }
    }
    int suma_bordes = sumaMaxIzqBorde + sumaMaxDerBorde;
    int suma_sub_max = 0;

    if (sumaMaxIzq > sumaMaxDer) {
        if (sumaMaxIzq > suma_bordes) {
            suma_sub_max = sumaMaxIzq;
            secIni = izq;
            secFin = centro;
        } else {
            suma_sub_max = suma_bordes;
        }
    } else {
        if (sumaMaxDer > suma_bordes) {
            suma_sub_max = sumaMaxDer;
            secIni = centro;
            secFin = der;

        } else {
            suma_sub_max = suma_bordes;
        }
    }
   SolucionSubsecuencia soluc = new SolucionSubsecuencia(datosModelo,
            ((SolucionSubsecuencia) s[0]).getIzq(),
            ((SolucionSubsecuencia) s[1]).getDer());
   soluc.setSumaMax(suma_sub_max);
   System.out.println(suma_sub_max + " " + secIni + " " + secFin);
   sumaMax = suma_sub_max;
   return soluc;
}
```

```java
public class PruebaSubsecuencia {

    public static void main(String[] args) {

            int numPruebas = 1;
            int factorRetardo = 1;

            System.out.println("---------------------------------");
            Subsecuencia subsecuenciaProblema = new Subsecuencia(6,10);
            System.out.println(subsecuenciaProblema.getModelo());
            System.out.println("---------------------------------");

            SubsecuenciaRecDV subsecuenciaRec =
                                new SubsecuenciaRecDV(subsecuenciaProblema);

            Temporizador t = new Temporizador(numPruebas);

            t.setFactorRetardo(factorRetardo);
            t.cronometra(subsecuenciaRec);

            System.out.println(t.getInformeTiempoMedio());

    }
}
```

```
---------------------------------
{-3, -1, 1, 10, 1, 10}
---------------------------------
1 0 1
2 1 2
11 3 4
21 3 5
22 2 5
Tiempo medio: 16.0 con un factor retardo de: 1
```

```
---------------------------------
{6, -5, 10, 4, 10, 6}
---------------------------------
6 0 0
11 0 2
14 3 4
20 3 5
31 0 5
Tiempo medio: 0.0 con un factor retardo de: 1
```

```
---------------------------------
{-2, 0, 10, -6, 6, -5}
---------------------------------
1 0 1
10 0 2
6 0 4
6 4 4
10 2 4
Tiempo medio: 0.0 con un factor retardo de: 1
```

Algoritmos voraces (devoradores)

Estos algoritmos toman decisiones basándose en la información que tienen disponible de modo inmediato, sin tener en cuenta los efectos que estas decisiones puedan tener en el futuro. Por tanto, **resultan fáciles de inventar, fáciles de implementar y, cuando funcionan, son eficientes**. Sin embargo, como el mundo no suele ser tan sencillo, hay muchos problemas que no se pueden resolver correctamente con dicho enfoque.

La estrategia de estos algoritmos es básicamente iterativa, y consiste en una serie de etapas, en cada una de las cuales se consumen una parte de los datos y se construye una parte de la solución, parando cuando se hayan consumido totalmente los datos. El nombre de este esquema es muy descriptivo: **en cada fase (bocado) se consume una parte de los datos**. Se intentará que la parte consumida sea lo mayor posible, bajo ciertas condiciones.

Esquema genérico:

```
public abstract class EsquemaVZ {
* el método <code>resuelve</code> contiene el esquema voraz
    public void resuelve()
    {
        inicializa();

        while(!fin()) {
            seleccionaYElimina();
            if(prometedor())
                anotaEnSolucion();
        }
    }
    abstract protected void inicializa();
    abstract protected boolean fin();
    abstract protected void seleccionaYElimina();
    abstract protected boolean prometedor();
    abstract protected void anotaEnSolucion();
}
```

Problema del cambio de monedas

Un algoritmo sencillo

```
public int cambioMonedaVoraz(int[] monedas, int cambio, int monedasDistintas) {
    int resultado = 0;
    for (int i = monedas.length - 1; i >= 0; i--) {
        while (monedas[i] <= cambio) {
            resultado++;
            cambio = cambio - monedas[i];
        }
    }
    return resultado;
}
```

Solución basándose en el esquema voraz

```
public class Cambio {

    protected int[] monedas;

    protected int aCambiar;

    public Cambio(int[] monedas, int aCambiar) {
        super();
        this.monedas = monedas;
        this.aCambiar = aCambiar;
    }

    public int getACambiar() {
        return aCambiar;
    }

    public void setACambiar(int cambiar) {
        aCambiar = cambiar;
    }

    public int[] getMonedas() {
        return monedas;
    }

    public void setMonedas(int[] monedas) {
        this.monedas = monedas;
    }

    public String toString() {
        String s = "{";
        for (int i = 0; i < monedas.length; i++) {
            if (i < monedas.length - 1) {
                s += monedas[i] + ", ";
            } else {
                s += monedas[i] + "}";
            }
        }
        return s;
    }
}
```

```java
public class CambioVoraz extends EsquemaVZ implements Temporizable {

    private Cambio cambio;
    private int k = -1;
    private int numMinimoMonedas = 0;

    public CambioVoraz(Cambio cambio) {
        super();
        this.cambio = cambio;
    }
    protected void inicializa() {

    }
    protected boolean fin() {
        if (cambio.aCambiar > 0) {
            return false;

        } else {
            return true;
        }
    }
    protected void seleccionaYElimina() {
        k = k + 1;
    }
    protected boolean prometedor() {
        if (cambio.monedas[k] <= cambio.aCambiar) {
            return true;
        } else {
            return false;
        }
    }
    protected void anotaEnSolucion() {
        numMinimoMonedas++;
        cambio.aCambiar = cambio.aCambiar - cambio.monedas[k];
    }

    public void procesamientoInicial() {..
    public void funcionAMedir() {

        resuelve();
    }
    public void procesamientoFinal() {..
    public int getNumMinimoMonedas() {..
    public Cambio getCambio() {..
    public void setCambio(Cambio cambio) {..
    public void setNumMinimoMonedas(int numMinimoMonedas) {..
}
```

```java
public class PruebaCambio {

    public static void main(String[] args) {

        int[] monedas = { 100, 50, 20, 10, 5, 2, 1 };
        int aCambiar = 127;
        int numPruebas = 1;
        int factorRetardo = 1;
        Cambio cambio = new Cambio(monedas, aCambiar);
        System.out.println(cambio.getACambiar());
        System.out.println("-----------------------------------");
        System.out.println(cambio);
        Temporizador t = new Temporizador(numPruebas);
        CambioVoraz cambioVoraz = new CambioVoraz(cambio);
        t.setFactorRetardo(factorRetardo);
        t.cronometra(cambioVoraz);
        // Imprime el resultado
        System.out.println(cambioVoraz.getNumMinimoMonedas());
        System.out.println(t);
    }
}
```

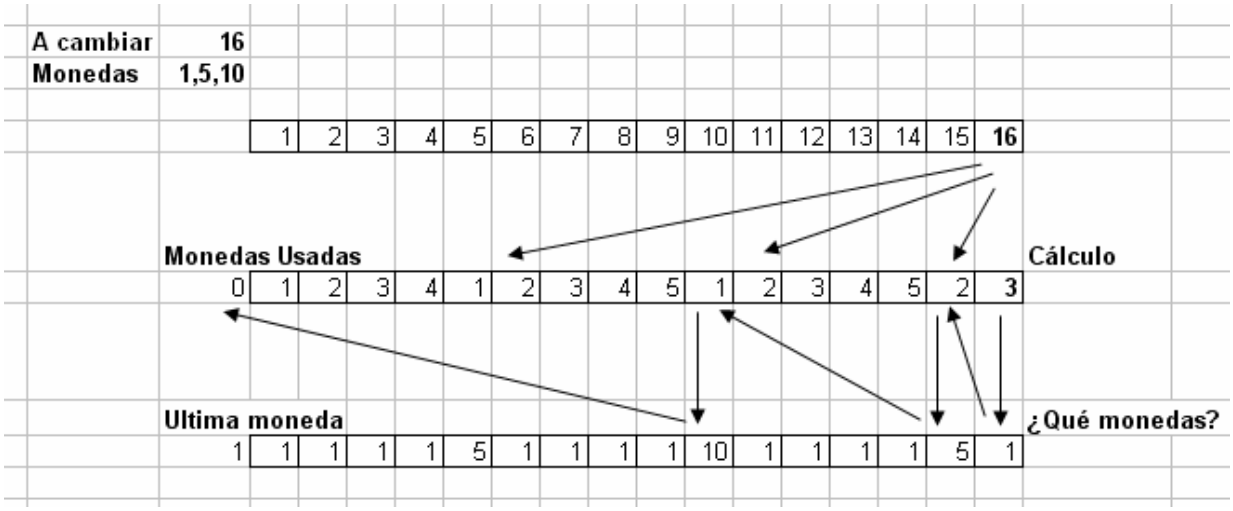
```
127
-----------------------------------
{100, 50, 20, 10, 5, 2, 1}
4
Tiempo mínimo: (0 con un factor retardo de: 1)  Tiempo mínimo real: 0.0 ms
```

Programación dinámica.

La idea de la técnica "divide y vencerás" es llevada al extremo en la programación dinámica. Si en el proceso de resolución del problema resolvemos un subproblema, almacenamos la solución, por si esta puede ser necesaria de nuevo para la resolución del problema**. Estas soluciones parciales de todos los subproblemas se almacenan en una tabla (array)** sin tener en cuenta si van a ser realmente necesarias posteriormente en la solución total.

Con el uso de esta tabla se evitan hacer cálculos idénticos reiteradamente, mejorando así la eficiencia en la obtención de la solución.

Problema del cambio de monedas



minMonedas	16	3	3
nuevaMoneda	1	5	10
j	0	1	2
valor moneda	1	5	10
[16-moneda[j]]	15	11	6
[16-moneda[j]]+1	16	12	7
monedasUsadas[16-moneda[j]]	2	2	2
monedasUsadas[16-moneda[j]]+1	3	3	3
minMoneda	3	nada	nada
nuevaMoneda	1	nada	nada
monedasUsadas[16]	3	nada	nada
ultimaMoneda[16]	1	nada	nada

```
public int cambioMonedasDin(int [] monedas, int monedasDistintas, int maxCambio,
                            int [] monedasUsadas, int [] ultimaMoneda)
{
    monedasUsadas[0]=0;
    ultimaMoneda[0]=1;

    for(int unidades=1; unidades<=maxCambio; unidades++)
    {
        int minMonedas=unidades;
        int nuevaMoneda=1;

        for(int j=0; j<monedasDistintas; j++)
        {
            if(monedas[j]>unidades)
                continue;
            if(monedasUsadas[unidades-monedas[j]]+1<minMonedas)
            {
                minMonedas=monedasUsadas[unidades-monedas[j]]+1;
                nuevaMoneda=monedas[j];
            }
        }
        monedasUsadas[unidades]=minMonedas;
        ultimaMoneda[unidades]=nuevaMoneda;
    }
    return monedasUsadas[maxCambio];
}
```

El tiempo de ejecución es el de los dos bucles for anidados, y por tanto es **O(NK),** donde N es número de monedas con distintos valores y K la cantidad máxima de cambio que queremos devolver.

```java
public class CambioDinamico implements Temporizable {

    private Cambio cambio;
    private int numMinimoMonedas = 0;
    private int[] monedasUsadas;
    private int[] ultimaMoneda;

    public CambioDinamico(Cambio cambio) {
        super();
        this.cambio = cambio;
        this.monedasUsadas = new int[cambio.getACambiar() + 1];
        this.ultimaMoneda = new int[cambio.getACambiar() + 1];
    }

    private void cambioMonedasDin(int[] monedas, int monedasDistintas,
            int maxCambio, int[] monedasUsadas, int[] ultimaMoneda) {

        monedasUsadas[0] = 0;
        ultimaMoneda[0] = 1;
        for (int unidades = 1; unidades <= maxCambio; unidades++) {
            int minMonedas = unidades;
            int nuevaMoneda = 1;
            for (int j = 0; j < monedasDistintas; j++) {
                if (monedas[j] > unidades)
                    continue;
                if (monedasUsadas[unidades - monedas[j]] + 1 < minMonedas) {
                    minMonedas = monedasUsadas[unidades - monedas[j]] + 1;
                    nuevaMoneda = monedas[j];
                }
            }
            monedasUsadas[unidades] = minMonedas;
            ultimaMoneda[unidades] = nuevaMoneda;
        }
        numMinimoMonedas = monedasUsadas[maxCambio];
    }
    public void procesamientoInicial() {

    }
    public void funcionAMedir() {
        cambioMonedasDin(cambio.getMonedas(), cambio.getMonedas().length,
                cambio.getACambiar(), monedasUsadas, ultimaMoneda);
    }
    public void procesamientoFinal() {
        int m = cambio.getACambiar();
        for (int i = 0; i < numMinimoMonedas; i++) {
            System.out.print(ultimaMoneda[m] + " ");
            m = m - ultimaMoneda[m];
        }
    }
    public Cambio getCambio() {..
    public void setCambio(Cambio cambio) {..
    public int[] getMonedasUsadas() {..
    public void setMonedasUsadas(int[] monedasUsadas) {..
    public int getNumMinimoMonedas() {..
    public void setNumMinimoMonedas(int numMinimoMonedas) {..
    public int[] getUltimaMoneda() {..
    public void setUltimaMoneda(int[] ultimaMoneda) {..
}
```

```java
public class PruebaCambio {

    public static void main(String[] args) {

        int[] monedas = { 1, 5, 10};
        int aCambiar = 18;
        int numPruebas = 1;
        int factorRetardo = 1;
        Cambio cambio = new Cambio(monedas, aCambiar);
        System.out.println(cambio.getACambiar());
        System.out.println("-----------------------------------");
        System.out.println(cambio);
        Temporizador t = new Temporizador(numPruebas);
        CambioDinamico cambioDinamico = new CambioDinamico(cambio);
        t.setFactorRetardo(factorRetardo);
        t.cronometra(cambioDinamico);
        // Imprime el resultado
        System.out.println();
        System.out.println("Numero de monedas "+cambioDinamico.getNumMinimoMonedas());
        System.out.println(t);
    }
}
```

```
18
-----------------------------------
{1, 5, 10}
1 1 1 5 10
Numero de monedas 5
Tiempo mínimo: (0 con un factor retardo de: 1)
```

```
25
-----------------------------------
{1, 5, 10}
5 10 10
Numero de monedas 3
Tiempo mínimo: (0 con un factor retardo de: 1)
```

Algoritmos de vuelta atrás.

Los algoritmos de vuelta atrás (también conocidos como de **backtracking**) hacen **una búsqueda sistemática de todas las posibilidades**, sin dejar ninguna por considerar. Cuando intenta una solución que no lleva a ningún sitio, retrocede deshaciendo el último paso, e intentando una nueva variante desde esa posición (es normalmente de naturaleza recursiva).

El proceso general de los algoritmos de vuelta atrás se contempla como un **método de prueba y búsqueda**, que gradualmente construye tareas básicas y las inspecciona para determinar si conducen a la solución del problema. Si una tarea no conduce a la solución, prueba con otra tarea básica. Es una prueba sistemática hasta llegar a la solución, o bien determinar que no hay solución por haberse agotado todas las opciones que probar.

La característica principal de los algoritmos de vuelta atrás es intentar realizar pasos que se acercan cada vez mas a la solución completa. Cada paso es anotado, borrándose tal anotación si se determina que no conduce a la solución, esta acción constituye **la vuelta atrás**. Cuando se produce una vuelta atrás se ensaya otro paso (otro movimiento). En definitiva, se prueba sistemáticamente con todas las opciones posibles hasta encontrar una solución, o bien agotar todas las posibilidades sin llegar a la solución.

```java
public class LaberintoSencillo {

    public static void imprimir(char[][] labe) {
        int N = labe.length;
        int M = labe[0].length;
        System.out.println();
        for (int i = 0; i < labe.length; i++) {
            for (int j = 0; j < labe[0].length; j++) {
                System.out.print(labe[i][j]);
            }
            System.out.println();
        }
        System.out.println();
    }

    public static boolean valido(char[][] labe, int x, int y) {
        int N = labe.length;
        int M = labe[0].length;
        boolean ok = false;
        if (x < N && x >= 0 && y < M && y >= 0
                && (labe[x][y] == ' ' || labe[x][y] == 's')) {
            ok = true;
        }
        return ok;
    }

    public static boolean resolver(char[][] labe, int x, int y) {
        boolean fin = false;

        if (valido(labe, x, y)) {
            if (labe[x][y] == 's')
                fin = true;

            else {
                labe[x][y] = 'x';
                fin = resolver(labe, x - 1, y); // arriba
                if (!fin) {
                    fin = resolver(labe, x, y - 1); // izquierda
                }
                if (!fin) {
                    fin = resolver(labe, x + 1, y); // abajo
                }
                if (!fin) {
                    fin = resolver(labe, x, y + 1); // derecha
                }

            }

            if (fin) {
                labe[x][y] = 'o';
            }
        }
        return fin;
    }
```

```
    public static void main(String[] args) {
        char[][] laberinto = { { '*', '*', '*', '*', ' ', '*' },
                { '*', ' ', ' ', ' ', ' ', '*' },
                { ' ', ' ', ' ', '*', '*', '*' },
                { '*', '*', ' ', ' ', '*', ' ' },
                { ' ', '*', ' ', ' ', '*', ' ' },
                { '*', '*', '*', 's', '*', '*' } };

        int i_x = 0;
        int i_y = 4;

        imprimir(laberinto);

        if (resolver(laberinto, i_x, i_y)) {
            System.out.println("laberinto resuelto");
            imprimir(laberinto);
        } else {
            System.out.println("laberinto no tiene solucion");
            imprimir(laberinto);
        }

    }
}
```

```
**** *
*    *
   ***
**  *
 *  *
***s**

laberinto resuelto

****o*
*oooo*
xoo***
**ox*
 *oo*
***o**
```

Esquema genérico:

```
protected void btOptima(Etapa x) {
    Etapa xsig;
    Candidatos cand;

    if (esSolucion(x)) {
        if (esMejor())
            actualizaSolucion();
    }
    xsig = nuevaEtapa(x);
    cand = calculaCandidatos(x);
    while (quedanCandidatos(cand)) {
        seleccionaCandidato(cand, xsig);
        if (prometedor(cand, xsig)) {
            anotaSolucion(cand, xsig);
            btOptima(xsig);
            cancelaAnotacion(cand, xsig);
        }
    }
}
```

```java
abstract public class EsquemaBtOptima {
    /**
* el método <code>resuelve</code> contiene el esquema Backtracking
* para una solución óptima
*/
protected void btOptima(Etapa x) {
        Etapa xsig;
        Candidatos cand;
        if (esSolucion(x)) {
            if (esMejor())
                actualizaSolucion();
        }
        else {
            xsig = nuevaEtapa(x);
            cand = calculaCandidatos(x);
            while (quedanCandidatos(cand)) {
                seleccionaCandidato(cand, xsig);
                if (esPrometedor(cand, xsig)) {
                    anotaSolucion(cand, xsig);
                    btOptima(xsig);
                    cancelaAnotacion(cand, xsig);
                }
            }
        }
    }
    /**
    * Devuelve un booleano indicando si estamos en un nodo hoja del árbol
    * de expansión.
    */
    abstract protected boolean esSolucion(Etapa x);
    /**
    * Se comprueba si el valor acumulado supera al de la solución óptima
    * encontrada hasta el momento.
    */
    abstract protected boolean esMejor();
    /**
    * Sustituye la solución óptima por una copia de la solución alcanzada
    * por esa rama.
    */
    abstract protected void actualizaSolucion();
    /**
    * Devuelve la nueva etapa a partir de la que aparece en el primer
    * parámetro del esquema.
    */
    abstract protected Etapa nuevaEtapa(Etapa x);
    /**
    * Devuelve los candidatos para la nueva etapa.
    */
```

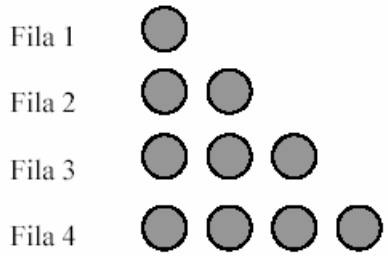
```
abstract protected Etapa nuevaEtapa(Etapa x);
/**
* Devuelve los candidatos para la nueva etapa.
*/
abstract protected Candidatos calculaCandidatos(Etapa x);
/**
* Indica si aún quedan candidatos en la nueva etapa.
*/
abstract protected boolean quedanCandidatos(Candidatos cand);
/**
* Selecciona un candidato de los candidatos de la nueva etapa.
*/
abstract protected void seleccionaCandidato(Candidatos cand, Etapa xsig);
/**
* Indica si el candidato seleccionado cumple una serie de condiciones
* que lo hacen admisible para ser añadido a la solcuión encontrada
* hasta ese momento en la rama actual del árbol de expansión.
*/
abstract protected boolean esPrometedor(Candidatos cand, Etapa xsig);
/**
* Actualiza la solución alcanzada por la rama actual con el candidato
* prometedor.
*/
abstract protected void anotaSolucion(Candidatos cand, Etapa xsig);
* Con el diseño propuesto, este método no hace nada.
abstract protected void cancelaAnotacion(Candidatos cand, Etapa xsig);
/**la clase Solucion para cada problema
implementa esta interface*/
public interface Solucion {
}
/**la clase Etapa para cada problema
implementa esta interface*/
public interface Etapa {
}
/**la clase Candidatos para cada problema
implementa esta interface*/
public interface Candidatos {
}
```

Relación de problemas: Vuelta atrás

1. ***Salto del caballo*** : Disponemos de un tablero de ajedrez de N*N casillas, situamos inicialmente un caballo en la casilla de ordenadas (X_0,Y_0), el problema consiste en encontrar, si existe, un camino que permita al caballo pasar una sola vez por todas y cada una de las casillas del tablero.

2. ***Problema de las ocho reinas*** : El problema consiste en colocar ocho reinas en el tablero de ajedrez, de manera que ninguna reina pueda amenazar a otra.

3. ***Juego del Nim*** : En este juego hay una serie de fichas colocadas en filas. Hay tantas filas como se deseen, con una ficha en la primera, dos en la segunda, tres en la tercera, etc.



Las reglas del juego son muy simples. Hay dos contrincantes que juegan por turnos, en cada movimiento se pueden quitar tantas fichas de una sola fila como se quiera, y gana el jugador que deja el tablero vacío. Por ejemplo, si quedan dos filas con tres fichas, una en la primera fila y dos en la segunda, si nos toca jugar tomaríamos una sola ficha de la fila con dos. Así, nuestro contrincante solo puede coger una ficha de alguna de las dos filas que quedan (sólo hay una ficha en cada una y no se pueden coger fichas de más de una fila); el resultado es que queda una ficha que cogeríamos nosotros, dejando el tablero vacío y ganando la partida

Relación de problemas: Voraces

1. ***Problema de la mochila*** : Se desea llenar una mochila hasta un volumen máximo V, y para ello se dispone de n objetos, en cantidades limitadas v1, ..., vn y cuyos valores por unidad de volumen son p1, ..., pn, respectivamente. Puede seleccionarse de cada objeto una cantidad cualquiera ci con tal de que ci <= vi.

   El problema consiste en determinar las cantidades c1, ..., cn que llenan la mochila maximizando el valor $\sum_{i=1}^{n} v_i \ p_i$ total.

2. ***Guardar ficheros en un disquete:*** Se trata de solucionar un problema de eficiencia de espacio en un diskete. Se pretende, en un caso, introducir el mayor número posible de ficheros dados, cada uno con un tamaño, en el diskete sin que se exceda, en ningún caso, el tamaño del diskete.

Relación de problemas: divide vencerás

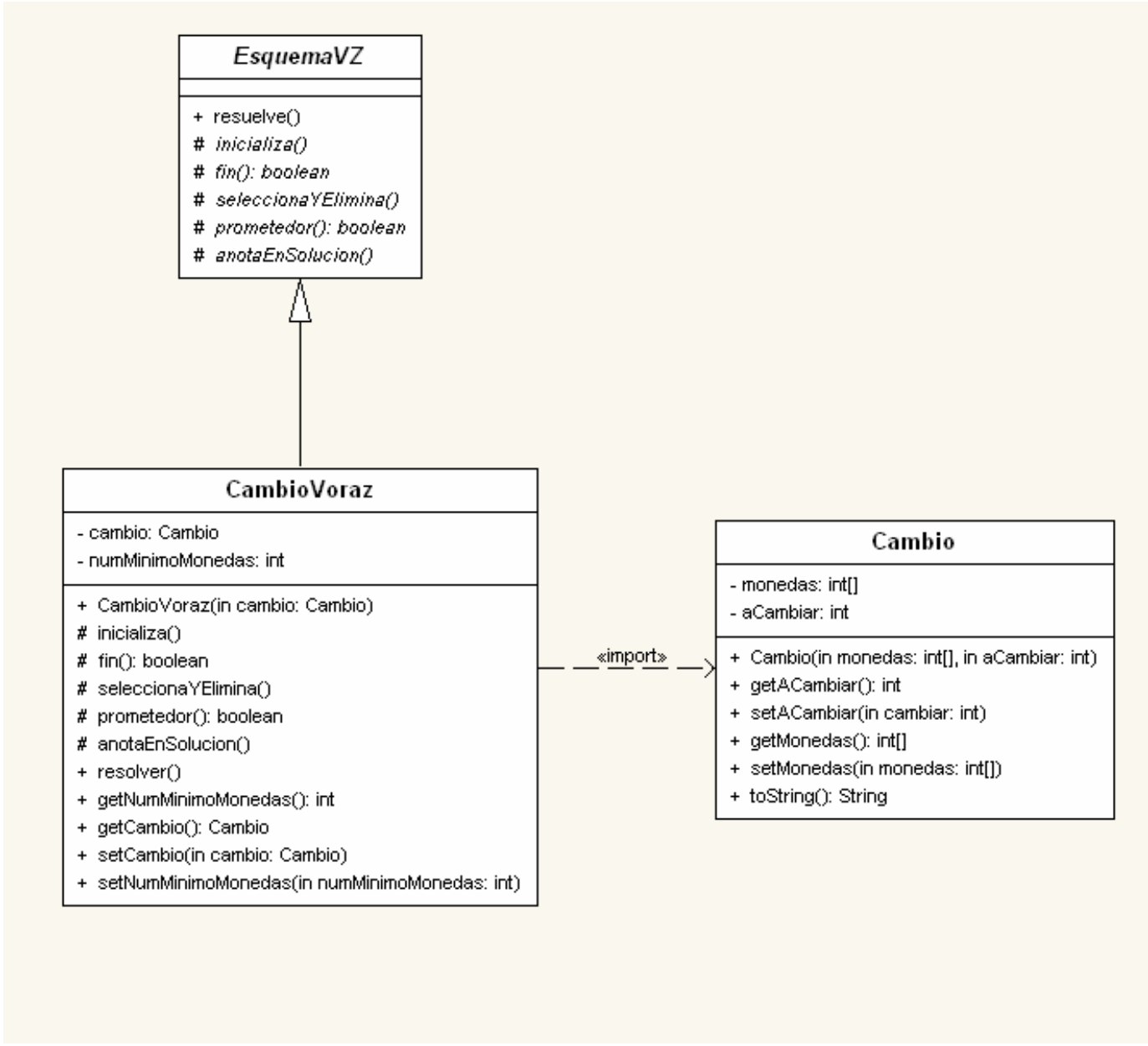
1. ***Multiplicación de Matrices Cuadradas:*** Hallar un algoritmo capaz de multiplicar dos matrices cuadradas de dimensión NxN, donde N es un número potencia de 2, mediante el uso de la técnica divide y vencerás.

2. ***Máximo y el mínimo de un vector***: Dado un vector de n elementos (enteros) no ordenado buscar los elementos máximo y mínimo.

Relación de problemas: programación dinámica

1. ***Beneficio empresas:*** Se dispone de una cantidad de dinero M, que se quiere invertir en D empresas. Para cada empresa se sabe que invertir j euros en la empresa i permite obtener una rentabilidad definida de la siguiente manera: B(i,j). Determinar cuál es el máximo beneficio que se puede obtener con los M euros y decir como se debe invertir el dinero.

2. ***Problema de la mochila Dinámico.***


EsquemaVZ
+ resuelve()
# inicializa()
# fin(): boolean
# seleccionaYElimina()
# prometedor(): boolean
# anotaEnSolucion()
CambioVoraz
- cambio: Cambio
- numMinimoMonedas: int
+ CambioVoraz(in cambio: Cambio)
# inicializa()
# fin(): boolean
# seleccionaYElimina()
# prometedor(): boolean
# anotaEnSolucion()
+ resolver()
+ getNumMinimoMonedas(): int
+ getCambio(): Cambio
+ setCambio(in cambio: Cambio)
+ setNumMinimoMonedas(in numMinimoMonedas: int)
«import»
Cambio
- monedas: int[]
- aCambiar: int
+ Cambio(in monedas: int[], in aCambiar: int)
+ getACambiar(): int
+ setACambiar(in cambiar: int)
+ getMonedas(): int[]
+ setMonedas(in monedas: int[])
+ toString(): String

```java
public class CambioVoraz extends EsquemaVZ {
    private Cambio cambio;

    private int k;

    private int numMinimoMonedas;

    public CambioVoraz(Cambio cambio) {
        super();
        inicializa();
        this.cambio = cambio;
    }

    protected void inicializa() {
        this.numMinimoMonedas = 0;
        this.k = 0;
    }

    protected boolean fin() {
        if (cambio.getACambiar() > 0) {
            return false;

        } else {
            return true;
        }
    }

    protected void seleccionaYElimina() {
        if ((cambio.getACambiar() >= cambio.getMonedas()[k])) {

        } else {
            k = k + 1;
        }
    }

    protected boolean prometedor() {
        if (cambio.getMonedas()[k] <= cambio.getACambiar()) {
            return true;
        } else {
            return false;
        }
    }

    protected void anotaEnSolucion() {
        numMinimoMonedas++;
        cambio.setACambiar(cambio.getACambiar() - cambio.getMonedas()[k]);
    }

    public void resolver() {

        resuelve();
    }

    public int getNumMinimoMonedas() {

    public Cambio getCambio() {

    public void setCambio(Cambio cambio) {

    public void setNumMinimoMonedas(int numMinimoMonedas) {
}
```