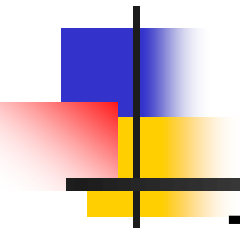


# Programació Avançada

Tècniques de disseny d'algorismes

Examen Curs 2017-18 Quadres

Examen Curs 2016-17 – Harry Potter

**Exercicis**: Backtracking


# Tècnica: Backtracking. Enunciat

La Maria, a qui li ha tocat la loteria, aprofitant que la crisi econòmica ha fet baixar el preu en el sector immobiliari ha comprat una casa, la casa dels seus somnis! Ara que ja li han lliurat les claus, ha de dedicar una partida econòmica per la decoració, entre d'altres adquisicions vol comprar una col·lecció de quadres de pintors emergents pel que pugui passar en el futur. Vol emplenar una de les parets del menjador amb una filera de quadres, l'un al costat de l'altre.

Cal escriure un programa que indiqui a la Maria **quins** són els quadres que ha de comprar a partir d'una primera selecció de quadres que ella ja ha fet. Tenint present que:


# Tècnica: Backtracking. Enunciat

- La Maria **no té cap restricció econòmica**, per tant el preu no ha d'influenciar en la selecció dels quadres.
- Els quadres es penjaran l'un al costat de l'altre **sense deixar cap espai** entre ells.
- Tots els quadres tenen un **prestigi associat, es volen seleccionar aquells que maximitzin el prestigi total, que vindrà determinat per la suma de prestigis dels quadres escollits. El prestigi sempre és un valor superior a 0.**
- Tots els quadres entre els que estriar tenen **la particularitat de què es poden penjar tant en vertical com en horitzontal** sense que això afecti al seu prestigi.
- La longitud de la filera de quadres a penjar **no pot sobrepassar** la dimensió de la paret a decorar però, s'ha de seleccionar aquella proposta que la deixi **lo més plena possible. Si dues propostes acumulen el mateix prestigi se seleccionarà la que ompli més la paret i si ambdues l'emplenen per igual la que acumuli més quadres.**


# Tècnica: Backtracking. Enunciat

Les dades que han d'entrar al programa són:

- La longitud de la paret a decorar.
- Oferta de quadres, quants, prestigi, preu i dimensions de cadascun d'ells.

Nota:

- un quadre no es pot trossejar, si forma part de la solució s'ha de penjar "tot", en amplada o en alçada.
- de cada quadre només en hi ha un exemplar, si es selecciona, o es penja en amplada o en alçada. El backtracking haurà de considerar per cada quadre les dues possibilitats, penjar-lo en amplada o en alçada.

# Tècnica: Backtracking. Enunciat

Classe Quadre

```
public class Quadre {
	private double amplada, alçada, prestigi, preu;
	private int identificador;

	public Quadre(double amplada, double alçada, double prestigi, double preu, int iden) {
		this.amplada = amplada;
		this.alçada = alçada;
		this.prestigi = prestigi;
		this.preu = preu;
		this.identificador = iden;
	}
	public double getAmplada() {
		return this.amplada;
	}
	public int getIdentificador() {
		return this.identificador;
	}
	public double getAlçada() {
		return this.alçada;
	}
	public double getPrestigi() {
		return this.prestigi;
	}
	public double getPreu() {
		return this.preu;
	}
	public String toString() {
		return "Quadre{" + identificador + "Amplada: " + amplada + ", Alçada: "
		+ alçada + ", Prestigi: " + prestigi + ", Preu: " + preu + " }";
	}
```

# Classe Solució

```
public class Solucio {

        private Quadre[] quadres; // Candidats a estriar
        private int quants; //Dimensió real de la taula quadres
        private double paret; // llargària paret a emplenar

        // Afegir atributs Exercici 1

        private static void ompleDades(Quadre[] q) {
               // Obté totes les dades dels objectes quadres i els deixa emmagat
               // en el paràmetre del mètode
               // NO s'ha d'implementar
        }

         public Solucio(double paret, int quants) {
                this.quadres = new Quadre[quants];
                this.quants=quants;
                this.paret=paret;
                ompleDades(this.quadres);
               //Completar Exercici 2
         }
          public void BackMillorSolucio(???) {
                 //Exercici 3. Implementació del backtracking
                 //Imprescindible invocar a mètodes privats:
                                  esAcceptable, esMillor i esSolucio
          }
          private boolean esAcceptable(????) {
                //Exercici 3. Mètodes privats
          }
          private boolean esSolucio(????) {
                //Exercici 3. Mètodes privats
          }
          private boolean esMillor(????) {
                //Exercici 3. Mètodes privats
          }
          public String toString() {
                // Exercici 4. Obté una representació en format String de la
                //solució trobada, si existeix
          }
}
```

# Tècnica: Backtracking. Enunciat

Classe Main

```
public class main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double llargaria; int quants;
		System.out.println("Indica la llargària de la paret a emplenar");
		do{
			llargaria=Keyboard.readDouble();
		}while (llargaria<=0);
		System.out.println("Indica quants quadres tens per triar");
		do{
			quants=Keyboard.readInt();
		}while (quants<=0);
		Solucio s=new Solucio(llargaria, quants);
		s.BackMillorSolucio(/*depèn dels paràmetres que té el vostre mètode*/);
		System.out.println(s);
	}
}
```


# Tècnica: Backtracking. Enunciat

→ 1.- (**1 punt**) Contesteu, sempre justificant la resposta:

- a.- Per què l'esquema de backtracking és aplicable per a resoldre aquest enunciat. **Determina quines decisions ha de prendre el programa**. Indica molt clarament quina pregunta ets fas en cada nivell de l'arbre i contesta les següents preguntes tenint en compte la teva resposta que m'acabes de donar. Quan serà acceptable una decisió?
- b.- Quin és el criteri per determinar si un conjunt de decisions és o no completable. Quin és el criteri per determinar si un conjunt de decisions és o no solució. Dibuixa l'espai de cerca del problema, és a dir, l'arbre que recorrerà la tècnica del backtracking, especificant quina serà l'alçada i l'amplada, indicant si són valors exactes o valors màxims. Amb el teu plantejament cal usar la tècnica del marcatge?
- c.- Indica els atributs que s'han d'**afegir** a la classe *Solucio* per a fer la implementació que has descrit en l'apartat anterior, **l'objectiu és que el mètode que implementa el backtracking tingui el menor número possible de paràmetres**. Per cada atribut afegit indica el seu ús.

→ 2.- (**0.5 punts**) Implementa el mètode constructor, cal crear i inicialitzar els atributs que has afegit a la classe.

→3.- (**4.5 punts**) Implementa el mètode **public void** BackMillorSolucio(???). Has de determinar el(s) paràmetre(s) necessaris, minimitzant-los. **Es valorarà l'anàlisi descendent aplicat**, fes mètodes privats per les diferents parts de l'esquema: esAcceptable(???), esMillor(???), ... Es valorarà l'eficiència. Indica com serà la crida principal a aquest mètode.

→4.- (**1 punt**) Redefineix el mètode toString() per mostrar la solució trobada. Ha de mostrar la llista de quadres seleccionats, per cadascun indicar si es penjarà en amplada o en alçada, i finalment, el prestigi total de la solució i el preu de la seva adquisició. Si no hi ha solució ho ha d'indicar mitjançant un missatge.


# Tècnica: Backtracking. Analisi

## Plantejament 1

1.- L'exercici es pot resoldre aplicant la tècnica del backtracking, és un problema d'optimització, per això cal generar tots els subconjunts de quadres possibles. La solució al problema la podem expressar mitjançant una seqüència de decisions que en aquest problema vindran determinades pels quadres escollits, per tant en cada nivell de l'arbre s'haurà d'escollir un dels quadres. La pregunta de cada nivell serà: "Quin quadre escollim? ". Una opció de decisió serà **acceptable** si el quadre seleccionat no ha estat ja estriat prèviament i té cabuda a la paret.

Un conjunt de decisions serà **completable** si queden quadres i hi ha espai en la paret.

El conjunt és **solució** si no sobrepassa la llargària de la paret, per tant després de cada selecció de quadre, si hi cap (sempre, sinó no hauria estat acceptable), **sempre serà nova solució.** Una solució serà **millor** que una altre si acumula més prestigi. A igual prestigi la que empleni més la paret i a igualtat, la que tingui més quadres.


# Tècnica: Backtracking. Analisi

Arbre: **alçada màxima** nombre **total de quadres**. **Amplada** vindrà determinada pel nombre **màxim de quadres, però cal tenir present que d'un mateix quadre tinc la opció de seleccionar-lo en vertical o horitzontal per tant l'amplada serà dos cops el nombre de quadres. Un mateix quadre l'hem de considerar dos cops, un per cada dimensió, per això hem de considerar la taula de quadres on estriar duplicada encara que sigui fictícia, consideraré que (N denota el número de quadres):**

- Un valor d'índex **i** entre [0, N-1] estria el quadre ièssim agafant la mida indicada per l'atribut amplada, i
- Un valor d'índex **i** entre [N, 2N-1] estria el quadre i-N agafant la mida indicada per l'atribut alçada.

S'haurà **d'usar la tècnica de marcatge** per la no repetició de quadres. Si un quadre es tria per penjar en alguna de les dues orientacions aquest mateix quadre, amb la proposta alternativa de penjat també s'ha de marcar com a usat. Si es selecciona el quadre ièssim s'haurà de marcar també el quadre ièssim+N si ièssim < N, en altre cas marcar el N-ièssim.

Afegim atributs a la classe Solucio per a optimitzar la implementació. I també una classe privada Parell per emmagatzemar la solució, per cada quadre estriat es voldrà saber en quina orientació es penjarà.


# Tècnica: Backtracking. Solució

```
//Afegim una classe nova, seran objectes de la taula solució
private class Parell {
        private Quadre quadre;
        private boolean amplada; //True si es penja en amplada. Fals en cas contrari
        public Parell(Quadre quin, boolean amplada) {
                this.quadre = quin;
                this.amplada = amplada;
        }

}
```


# Tècnica: Backtracking. Solució

```
private Parell solucio[]; //solució en construcció
private int accPrestigi; // acumulador prestigi solució en construcció
private double accLlargariaParet; // acumulador paret usada de la solució en construcció

private Parell millor[]; //millor solució
private int accPrestigiMillor; // acumulador prestigi millor solució
private int quantsMillor; // quants quadres té la millor solució
private double accLlargariaParetMillor; // paret ocupada per la millor
// MARCATGE
private boolean marcats[]; //marcatge quadres estriats en la solució en construcció
```

# Tècnica: Backtracking. Solució

```
public Solucio(double paret, int qQuadres) {
        this.quants=qQuadres;
        this.quadres = new Quadre[quants];

        this.paret=paret;
        ompleDades(this.quadres);
       ////////////////////////////////////////////////////
       // marcatge
       marcats = new boolean[quants * 2];
       for (int i = 0; i < marcats.length; i++)
               marcats[i] = false;
       ////////////////////////AFEGIM ////////////////////////
       // dupliquem els quadres. No és real, només les referències per considerar-lo en
       // ambdues orientacions
       Quadre aux[] = new Quadre[qQuadres * 2];
       for (int i = 0; i < qQuadres; i++) {
               aux[i] = quadres[i];
               aux[i + quants] = quadres[i];
       }
       quadres = aux;
       ////////////////////////////////////////////////////
       solucio = new Parell[qQuadres]; //màxim agafem tots els quadres
       millor = new Parell[qQuadres];
       accPrestigi = 0; // acumulador prestigi solució en construcció
       accLlargariaParet = 0; // acc paret usada en la solució en construcció

       accPrestigiMillor = -1; // acumulador prestigi millor solució
       quantsMillor = -1; // quants quadres té la millor solució
       accLlargariaParetMillor = -1; // paret ocupada per la millor solució
} // fi constructor
```


# Tècnica: Backtracking. Solució

```
public String toString() {
        String r = "";
        if (accPrestigiMillor == -1)
                r = "No hi ha cap solució. Cap quadre cap a la paret";
        else {
                double importe = 0.;
                for (int i = 0; i < quantsMillor; i++) {
                        r += millor[i].quadre.toString();
                        importe += millor[i].quadre.getPreu();
                        if (millor[i].amplada)
                                r += " en amplada";
                        else
                                r += "en alçada";
                }
                r += "/nEl cost és de  " + importe + "\nel prestigi total és de :" +
                     accPrestigiMillor;
        }
        return r;
}
```

# Tècnica de backtracking

Esquema MILLOR

```
public static void BackMillorSolucio( TaulaSolucio  TS, int k,
TaulaSolucio Millor){
    inicialitzem_valors_domini_decisio_nivell_k
    agafar_el_primer_valor
    while (quedin_valors){ //Recorregut de tot l'arbre
        if (valor_acceptable){  //no viola les restriccions
            anotem_el_valor_a_la_solucio
            if (solucio_final)
                    if (millor_solucio) Millor=TS; //else res
            else if (solucio_completable)
                                BackMillorSolucio(TS, k+1, Millor);
            desanotem_el_valor
        } //fi if
        agafar_seguent_valor
        //passem al següent germà a la dreta
    } // fi while
} // fi procediment
```

Sovint s'han d'afegir paràmetres per poder determinar si una solució és millor

# Tècnica de backtracking

Esquema MILLOR

```
public static void BackMillorSolucio( TaulaSolucio  TS, int k,
TaulaSolucio Millor){
    inicialitzem_valors_domini_decisio_nivell_k
    agafar_el_primer_valor
    while (quedin_valors){ //Recorregut de tot l'arbre
        if (valor_acceptable){  //no viola les restriccions
            anotem_el_valor_a_la_solucio
            if (solucio_final)
                if (millor_solucio) Millor=TS; //else res
            else if (solucio_completable)
                        BackMillorSolucio(TS, k+1, Millor);
            desanotem_el_valor
        } //fi if
        agafar_seguent_valor
        //passem al següent germà a la dreta
    } // fi while
} // fi procediment
```

Sovint s'han d'afegir paràmetres per poder determinar si una solució és millor

Esquema alterat

Crida principal: s.BackMillorSolucio(0);

```
public void BackMillorSolucio(int k) {
        // Esquema Millor
        int i = 0; boolean amplada;
        while (i < quants * 2) {
                if (esAcceptable(i)) { //Acceptem quadre ièssim?
                        amplada = true;
                        double increment = quadres[i].getAmplada();
                        if (i >= quants) { //l'agafem en alçada
                                amplada = false;
                                increment = quadres[i].getAlçada();
                                marcats[i] = true;
                                marcats[quants - i] = true;
                        } else {
                                marcats[i] = true;
                                marcats[i + quants] = true;
                        }
                        accPrestigi += quadres[i].getPrestigi();
                        accLlargariaParet += increment;

                        solucio[k] = new Parell(quadres[i], amplada);
                        // Sempre és solució
                        // Millor solució
                        if (esMillor(k)) {
                                for (int j = 0; j <= k; j++)
                                        millor[j] = solucio[j];
                                accPrestigiMillor = accPrestigi;
                                accLlargariaParetMillor = accLlargariaParet;
                                quantsMillor = k + 1;
                        } // fi millor
                        if (k < quants && accLlargariaParet < paret)
                                //podem ampliar la solució, queda paret buida i quadres
                                BackMillorSolucio(k + 1);
                        // desfer
                        accPrestigi -= quadres[i].getPrestigi();
                        if (i >= quants) {

                                marcats[i] = false;
                                marcats[quants - i] = false;
                        } else {

                                marcats[i] = false;
                                marcats[quants + i] = false;
                        }
                        accLlargariaParet -= increment;
                        solucio[k]=null;

                } // fi acceptable
                i++; // següent
        } // fi while
} // fi procediment
```

No hi ha condició de si és solució

Crida principal: s.BackMillorSolucio(0);

```
public void BackMillorSolucio(int k) {
        // Esquema Millor
        int i = 0; boolean amplada;
        while (i < quants * 2) {
                if (esAcceptable(i)) { //Acceptem quadre ièssim?
                        amplada = true;
                        double increment = quadres[i].getAmplada();
                        if (i >= quants) { //l'agafem en alçada
                                amplada = false;
                                increment = quadres[i].getAlçada();
                                marcats[i] = true;
                                marcats[quants - i] = true;
                        } else {
                                marcats[i] = true;
                                marcats[i + quants] = true;
                        }
                        accPrestigi += quadres[i].getPrestigi();
                        accLlargariaParet += increment;

                        solucio[k] = new Parell(quadres[i], amplada);
                        // Sempre és solució
                        // Millor solució
                        if (esMillor(k)) {
                                for (int j = 0; j <= k; j++)
                                        millor[j] = solucio[j];
                                accPrestigiMillor = accPrestigi;
                                accLlargariaParetMillor = accLlargariaParet;
                                quantsMillor = k + 1;
                        } // fi millor
                        if (k < quants && accLlargariaParet < paret)
                                //podem ampliar la solució, queda paret buida i quadres
                                BackMillorSolucio(k + 1);
                        // desfer
                        accPrestigi -= quadres[i].getPrestigi();
                        if (i >= quants) {

                                marcats[i] = false;
                                marcats[quants - i] = false;
                        } else {

                                marcats[i] = false;
                                marcats[quants + i] = false;
                        }
                        accLlargariaParet -= increment;
                        solucio[k]=null;

                } // fi acceptable
                i++; // següent
        } // fi while
} // fi procediment
```

**Quan acceptem?**
No repe i hi cap

**Quan és millor?**
Més prestigi o
Igual prestigi però més plena o
Igual prestigi i igual ple però més número de quadres

Crida principal: s.BackMillorSolucio(0);

```
public void BackMillorSolucio(int k) {
        // Esquema Millor
        int i = 0; boolean amplada;
        while (i < quants * 2) {
                if (esAcceptable(i)) { //Acceptem quadre ièssim?
                        amplada = true;
                        double increment = quadres[i].getAmplada();
                        if (i >= quants) { //l'agafem en alçada
                                amplada = false;
                                increment = quadres[i].getAlçada();
                                marcats[i] = true;
                                marcats[quants - i] = true;
                        } else {
                                marcats[i] = true;
                                marcats[i + quants] = true;
                        }
                        accPrestigi += quadres[i].getPrestigi();
                        accLlargariaParet += increment;

                        solucio[k] = new Parell(quadres[i], amplada);
                        // Sempre es solució
                        // Millor solució
                        if (esMillor(k)) {
                                for (int j = 0; j <= k; j++)
                                        millor[j] = solucio[j];
                                accPrestigiMillor = accPrestigi;
                                accLlargariaParetMillor = accLlargariaParet;
                                quantsMillor = k + 1;
                        } // fi millor
                        if (k < quants && accLlargariaParet < paret)
                                //podem ampliar la solució, queda paret buida i quadres
                                BackMillorSolucio(k + 1);
                        // desfer
                        accPrestigi -= quadres[i].getPrestigi();
                        if (i >= quants) {

                                marcats[i] = false;
                                marcats[quants - i] = false;
                        } else {

                                marcats[i] = false;
                                marcats[quants + i] = false;
                        }
                        accLlargariaParet -= increment;
                        solucio[k]=null;

                } // fi acceptable
                i++; // següent
        } // fi while
} // fi procediment
```

fer

desfer

Crida principal: s.BackMillorSolucio(0);

```
public void BackMillorSolucio(int k) {
        // Esquema Millor
        int i = 0; boolean amplada;
        while (i < quants * 2) {
                if (esAcceptable(i)) { //Acceptem quadre iéssim?
                        amplada = true;
                        double increment = quadres[i].getAmplada();
                        if (i >= quants) { //l'agafem en alçada
                                amplada = false;
                                increment = quadres[i].getAlçada();
                                marcats[i] = true;
                                marcats[quants - i] = true;
                        } else {
                                marcats[i] = true;
                                marcats[i + quants] = true;
                        }
                        accPrestigi += quadres[i].getPrestigi();
                        accLlargariaParet += increment;

                        solucio[k] = new Parell(quadres[i], amplada);
                        // Sempre és solució
                        // Millor solució
                        if (esMillor(k)) {
                                for (int j = 0; j <= k; j++)
                                        millor[j] = solucio[j];
                                accPrestigiMillor = accPrestigi;
                                accLlargariaParetMillor = accLlargariaParet;
                                quantsMillor = k + 1;
                        } // fi millor
                        if (k < quants && accLlargariaParet < paret)
                                //podem ampliar la solució, queda paret buida i quadres
                                BackMillorSolucio(k + 1);
                        // desfer
                        accPrestigi -= quadres[i].getPrestigi();
                        if (i >= quants) {

                                marcats[i] = false;
                                marcats[quants - i] = false;
                        } else {

                                marcats[i] = false;
                                marcats[quants + i] = false;
                        }
                        accLlargariaParet -= increment;
                        solucio[k]=null;

                } // fi acceptable
                i++; // següent
        } // fi while
} // fi procediment
```

**Quan és completable?**
Queden quadres i queda lloc

# Tècnica: Backtracking. Solució

```
private boolean esAcceptable(int i) {
        return !marcats[i] &&
        (i < quants && accLlargariaParet + quadres[i].getAmplada() <= paret)
        || (i >= quants && accLlargariaParet + quadres[i].getAlçada() <= paret);
}

private boolean esMillor(int k) {
        return accPrestigi > accPrestigiMillor
        || accPrestigi == accPrestigiMillor &&
           accLlargariaParet > accLlargariaParetMillor
        || accPrestigi == accPrestigiMillor &&
           accLlargariaParet == accLlargariaParetMillor &&
           (k+1) > quantsMillor;
}
```


# Tècnica: Backtracking. Analisi alternatiu

## Plantejament alternatiu

En cada nivell de l'arbre ens preguntem: el quadre X el penjo? Com tenim 2 opcions de penjat las possibles respostes a aquesta pregunta són 3: no el penjo, el penjo en horitzontal, el penjo en vertical.

Llavors l'arbre serà ternari, d'amplada 3. En cada nivell prenem una decisió per cadascun dels quadres.

Serà solució quan haguem decidit sobre cada quadre, per tant quan arribem a una fulla, l'arbre tindrà una alçada exacta coincidint amb el nombre de quadres. NO necessitem marcatge, les tres opcions de resposta es poden repetir en cada nivell.

Una decisió serà acceptable si en cas de penjar-lo hi cap.

Una col·lecció de decisions serà completable sinó hem arribat a una fulla de l'arbre.

La condició de millor es la mateixa que el plantejament anterior.

La implementació amb aquest plantejament és més senzilla.


# Tècnica: Backtracking. Analisi alternatiu

## Plantejament alternatiu

En cada nivell de l'arbre ens preguntem: el quadre X el penjo? Com tenim 2 opcions de penjat las possibles respostes a aquesta pregunta són 3: no el penjo, el penjo en horitzontal, el penjo en vertical. Llavors l'arbre serà ternari, d'amplada 3. En cada nivell prenem una decisió per cadascun dels quadres. Serà solució quan haguem decidit sobre cada quadre, per tant quan arribem a una fulla, l'arbre tindrà una alçada exacta coincidint amb el nombre de quadres. NO necessitem marcatge, les tres opcions de resposta es poden repetir en cada nivell.

Una decisió serà acceptable si en cas de penjar-hi [illegible]

Una col·lecció de decisions serà completable sinó hem arribat a una fulla de l'arbre.

La condició de millor es la mateixa que el plantejament anterior.

La implementació amb aquest plantejament és més senzilla.

**Més fàcil!**

**Pensem!!!!**

# Tècnica: Backtracking. Solució

```
public class Solucio {

private Quadre[] quadres; // Candidats a estriar
private int quants; // Dimensió real de la taula quadres
private double paret; // llargaria paret a omplenar


private int solucio[]; // solució en construcció
private int accPrestigi; // acumulador prestigi solució en construcció
private double accLlargariaParet; // acumulador paret usada de la solució en
private int agafats; // quants seleccionats en la solució en construcción, ara
                     // no coincideix amb el nivell

private int millor[]; // millor solució
private int accPrestigiMillor; // acumulador prestigi millor solució
private int quantsMillor; // quants quadres té la millor solució
private double accLlargariaParetMillor; // paret ocupada per la millor
```

# Tècnica: Backtracking. Solució

```
public Solucio(double paret, int qQuadres) {
// creació i inicialització de magatzems
        this.quadres = new Quadre[ qQuadres];
         this.quants=qQuadres;
        this.paret = paret;
        ompleDades(this.quadres);

        solucio = new int[qQuadres];
        millor = new int[qQuadres];
        accPrestigi = 0; // acumulador prestigi solució en construcció
        accLlargariaParet = 0; // acc paret usada en la solució en construcció
        agafats = 0; // inicialment 0 seleccionats

        accPrestigiMillor = -1; // acumulador prestigi millor solució
        quantsMillor = -1; // quants quadres té la millor solució
        accLlargariaParetMillor = -1; // paret ocupada per la millor solució
} // fi constructor
```

# Tècnica: Backtracking. Solució

```
public String toString() {
    String r = "";
    if (accPrestigiMillor == -1)
        r = "No hi ha cap solució. Cap quadre cap a la paret";
    else {
        double importe = 0.;
        for (int i = 0; i < quantsMillor; i++) {
            if (millor[i] != 0) { // quadre seleccionat
                r += quadres[i].toString();
                importe += quadres[i].getPreu();
                if (millor[i] == 1)
                    r += " en amplada";
                else
                    r += "en alçada";
            }
        }
        r += "/nEl cost és de  " + importe + "\nel prestigi total és de :" + accPrestigiMillor;
    }
    return r;
}
```

0→no agafem
1→ en amplada
2→ en alçada

```
public void BackMillorSolucio(int k) {
    double increment; int i = 0;
    while (i < 3) {
        if (esAcceptable(i)) { // Acceptem quadre ièssim?
            if (i != 0) {
                increment = quadres[i].getAmplada();
                if (i == 2) increment = quadres[i].getAlçada();
                accPrestigi += quadres[i].getPrestigi();
                accLlargariaParet += increment;
                agafats++;
            }
            solucio[k] = i;
            if (k == quants - 1) {
                if (esMillor()) {
                    for (int j = 0; j < quants; j++)
                        millor[j] = solucio[j];
                    accPrestigiMillor = accPrestigi;
                    accLlargariaParetMillor = accLlargariaParet;
                    quantsMillor = agafats;
                }
            }
            else if (accLlargariaParet < paret) // queda paret buida i quadres
                BackMillorSolucio(k + 1);
            // desfer
            if (i != 0) {


                accPrestigi -= quadres[i].getPrestigi();
                accLlargariaParet -= increment;
                agafats--;
            }
            // No caL BUIDAR LA TAULA
        } // fi acceptable
        i++; // següent
    } // fi while
} // fi procediment
```

Crida principal: s.BackMillorSolucio(0);

0→no agafem
1→ en amplada
2→ en alçada

# Tècnica: Backtracking. Solució

```
    private boolean esAcceptable(int i) {
        return (i == 0 || (i == 1 && accLlargariaParet + quadres[i].getAmplada() <= paret)
                || (i == 2 && accLlargariaParet + quadres[i].getAlçada() <= paret));
    }

    private boolean esMillor() {
        return accPrestigi > accPrestigiMillor
                || accPrestigi == accPrestigiMillor && accLlargariaParet > accLlargariaParetMillor
                || accPrestigi == accPrestigiMillor && accLlargariaParet == accLlargariaParetMillor
                        && agafats > quantsMillor;
    }

}   // fi classe
```


# Tècnica: Backtracking. Enunciat

S'ha rebut un missatge de text codificat mitjançant un valor numèric: 903900439651484. Es sap que **cada dígit** del missatge es correspon amb **un caràcter diferent**, però no es sap quin. Òbviament, per poder descodificar-lo es necessita la taula de conversió entre caràcters i dígits. Per obtenir aquesta relació existent entre caràcters i números cal resoldre la següent suma:

```
  HARRY
+POTTER
--------
TROLLS
```

Sabent que:

→Cada caràcter és un únic dígit per tot el problema. Per exemple, si la lletra "R" és el número "3" aquest valdria per totes les "R" de les paraules HARRY, POTTER i TROLLS , i a més el "3" del missatge se substituiria por la lletra "R".

→Quan els caràcters de la suma se substitueixen pel seu dígit, l'operació aritmètica ha de ser correcta. És a dir, no és possible que "Y" valgui 1, "R" valgui 2 y "S" valgui 8.

→ Els números que apareixen en la suma no poden començar per "0". És una restricció del problema. Per tant, ni el caràcter "H" ni "P" ni "T" poden ser el dígit 0.

→En cas d'haver solució, només n'existeix una.


# Tècnica: Backtracking. Enunciat

Dissenyar un algorisme que usant la tècnica del Backtracking trobi la taula de conversió entre caràcters i dígits necessària per poder descodificar aquest missatge rebut. En aquest problema no s'ha de optimitzar res, per tant la variant d'esquema de backtracking a aplicar és el de trobar **UNA solució.**

Completa els mètodes de la següent classe per realitzar la descodificació del missatge rebut:


# Tècnica: Backtracking. Enunciat

```
public class Descodificador{
        //Exercici 1 – Posar atributs
    public static void main(String args[]){
            int []missatge={9,0,3,9,0,0,4,3,9,6,5,1,4,8,4};
            char []operand1={'H','A','R','R','Y'};
            char []operand2={'P','O','T','T','E','R'};
            char []suma={'T','R','O','L','L','S'};
            Descodificador m=new Descodificador(missatge, operand1, operand2, suma);
            if (m.back1Solucio(??)) System.out.println(m);
            else System.out.println("No hi ha solució");
    }
    public Descodificador(int[] missatge, char[] op1, char[] op2, char[] suma){
            //Exercici 2
    }
    public boolean back1Sol(???){
            //Exercici 3
    }
    public String toString(){
            //Exercici 4
    }
}
```


# Tècnica: Backtracking. Enunciat

**Exercicis**

→ 1.- (**1 punt**) Contesteu, sempre justificant la resposta:

- a.- Per què l'esquema de backtracking és aplicable per a resoldre aquest enunciat.
- b.- Determina quines decisions ha de prendre el programa. Quin és el criteri per determinar si un conjunt de decisions és o no completable. Quin és el criteri per determinar si un conjunt de decisions és o no solució. Dibuixa l'espai de cerca del problema, és a dir, l'arbre que recorrerà la tècnica del backtracking, especificant quina serà l'alçada i l'amplada, indicant si són valors exactes o valors màxims. Amb el teu plantejament cal usar la tècnica del marcatge?
- c.- Indica els atributs que s'han d'**afegir** a la classe *Descodificador* per a fer la implementació que has descrit en l'apartat anterior, **l'objectiu és que el mètode que implementa el backtracking tingui el menor número possible de paràmetres**. Per cada atribut afegit indica el seu ús.

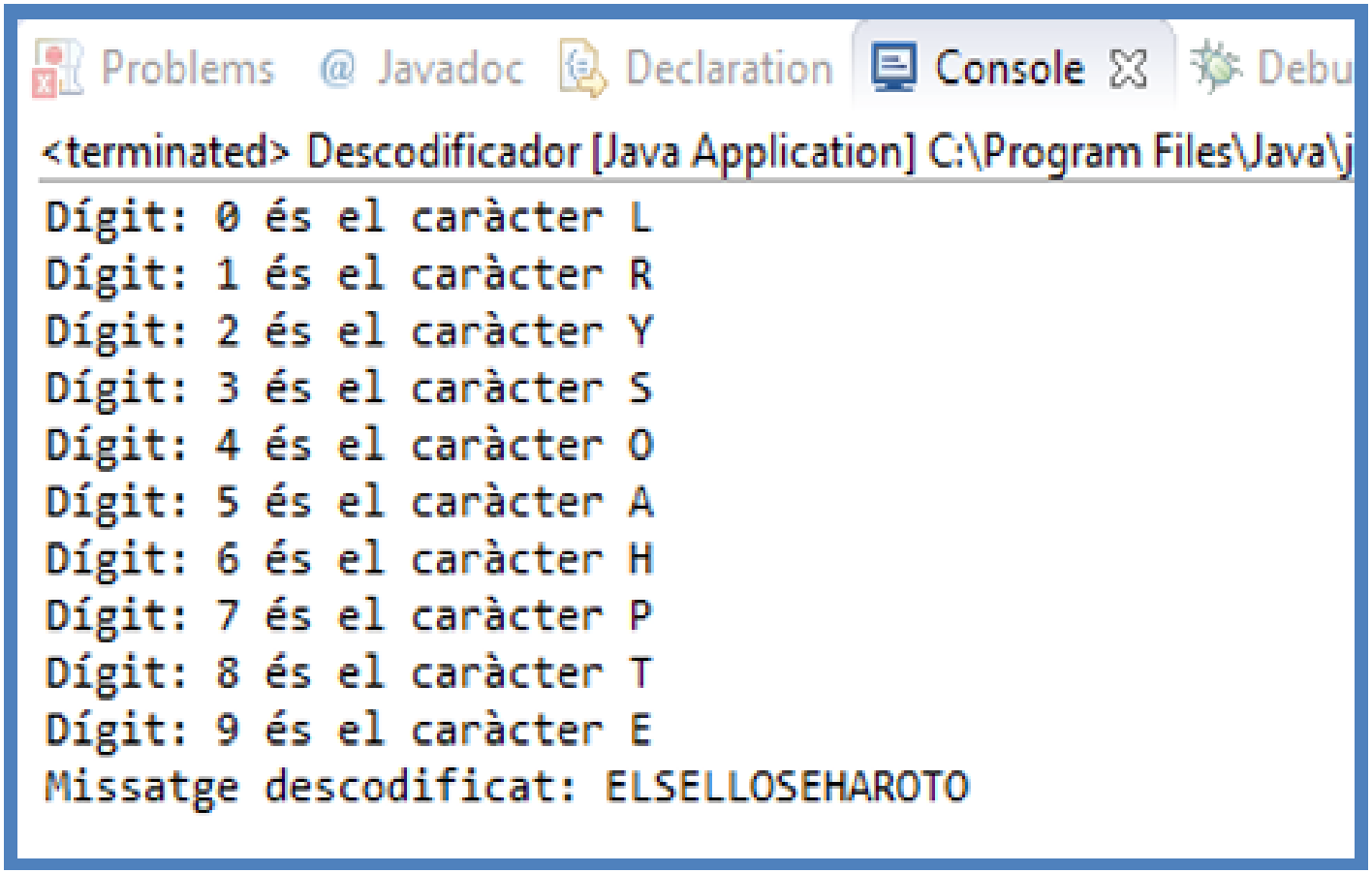
→ 2.- (**0.5 punts**) Implementa el mètode constructor, cal crear i inicialitzar els atributs que has afegit a la classe.

→3.- (**3 punts**) Implementa el mètode **public boolean** back1Solucio(???). Has de determinar el(s) paràmetre(s) necessaris, minimitzant-los. **Es valorarà l'anàlisi descendent aplicat**, fes mètodes privats per les diferents parts de l'esquema. Es valorarà l'eficiència.

→4.- (**0.5 punt**) Redefineix el mètode toString() per mostrar la solució trobada. Ha de mostrar la taula de conversió caràcter/dígit i el missatge descodificat.


# Tècnica: Backtracking. Enunciat

**Exemple d'execució. Solució del problema**



# Tècnica: Backtracking. Solució

**Apartat 1**

a- És un problema en el que la solució pot expressar-se com una seqüència de decisions; aquestes venen donades per decidir en cada nivell quin caràcter assignem a cadascun dels dígits a descodificar. S'ha de trobar una, la que compleixi la restricció i resolgui la suma correctament.

Una col·lecció de decisions és completable si encara no s'ha decidit per cada dígit a quin caràcter es correspon. I és solució quan tots els dígits tenen caràcter assignat complint les restriccions de l'enunciat.

b.- Arbre: en cada nivell s'ha de prendre la decisió de dígitX a quin caràcter es correspon? (dígitX varia del 0 al 9). Generarem un arbre d'amplada 10 amb marcatge doncs cada caràcter només pot esser assignat a un dígit. L'alçada és exacta també de 10, a tots els dígits de l'intèrval [0,9] se'ls ha de buscar la seva codificació.

Les solucions al problema són els camins que van des de l'arrel de l'arbre fins a una fulla, i evidentment sense incompatibilitats. Només volem una solució. S'han de prendre exactament 10 decisions.

**Podríem optimitzar el plantejament**, deduïm de la operació matemàtica que s'ha de complir que els caràcters 'T' i 'E' han de tenir dígits consecutius, que el caràcter 'T' té el dígit següent al del caràcter 'P' i que la suma dels dígits corresponents als caràcters 'H'+'O' >9. La solució mostrada a continuació **no** implementa aquestes optimitzacions.

Aquest exercici també pot esser solucionat amb un planteig totalment contrari, l'amplada de l'arbre que siguin els 9 dígits a descodificar i l'alçada els caràcters, així doncs la pregunta en cada nivell seria el caràcter X amb quin dígit està codificat?. La solució seria totalment simètrica a la que es mostra a continuació.

# Tècnica: Backtracking. Solució

c.- Atributs a afegir:

```
        private char lletres[]={'H','A','R','Y','P','O','T','E','S','L'};
        //diferents caràcters a codificar. Domini decisions. Es pot omplenar al constructor
        private int missatge[];
        //missatge a descodificar
        private char solucio[];
        // la posició 0 emmagatzema el caràcter amb codi  0 i així ...
        private boolean marcats[];
        // marcatge per no repetició

        private char operand1[];
        private char operand2[];
        private char suma[];
```

**Apartat 2**

```
        public Descodificador(int[] missatge, char[] r1, char[] r2, char[] r3){
                //Exercici 2
                this.missatge=missatge;
                this.operand1=r1;
                this.operand2=r2;
                this.suma=r3;
                solucio=new char[10];
                //índex es correspon amb el codi:  solucio[0] hi ha el caràcter del dígit 0
                marcats=new boolean[10];
                for(int i=0; i<10; i++)marcats[i]=false;
        }
```

# Tècnica: Backtracking. Solució

**Apartat 4**

```
public String toString(){
        String des=taulaDescodificar()+"Missatge descodificat: ";
        for (int i=0; i<missatge.length; i++){
                des+=solucio[missatge[i]];
        }
        return des;
 }
 private String taulaDescodificar(){
        String resultat=" ";
        for (int i=0; i<solucio.length; i++)
                resultat+="Dígit: "+i+" és el caràcter "+solucio[i]+"\n" ;
        return resultat;
}
```

# Tècnica: Backtracking. Solució

```
public boolean back1Solucio(int k){ //crida principal back1Solucio(0)
        //k és correspon amb el digit
        int i=0; //index de la taula de caracters. Domini del problema
        boolean trobat=false;
        while (i<10 && !trobat){ //amplada caracters
                if (!marcats[i] && acceptable(lletres[i], k)){ //acceptable. Assignar al digit k el
                                                               //caràcter lletres[i]
                        marcats[i]=true;
                        solucio[k]=lletres[i];
                        if (k==9 && sumaOK()){
                                trobat=true;
                        }
                        else if (k<9)trobat=back1Solucio(k+1);
                        if (!trobat)marcats[i]=false;
                }
                i++;
        }
        return trobat;
}
private boolean acceptable(char car, int valor){
        // restricció: els caràcters H, P, T no poden ser el 0
        return  ((valor==0 && car!='H' && car!='P' && car!='T') || (valor!=0));
 }
```

# Tècnica: Backtracking. Solució

```
private boolean sumaOK(){
        int h=descodificar(operand1);
        int p=descodificar(operand2);
        int t=descodificar(suma);
        return h+p==t;
  }
private int descodificar(char m[]){
        int i=0, codi;
        int valor=0;
        while (i<m.length){
                codi=donaCodi(m[i]);
                valor=10*valor+codi;
                i++;
        }
        return valor;
  }
  private int donaCodi(char m){
        //sabem segur hi és
        int i=0;
        while (solucio[i]!=m)i++;
        return i;
}
```