

# Programació Avançada

## Backtracking

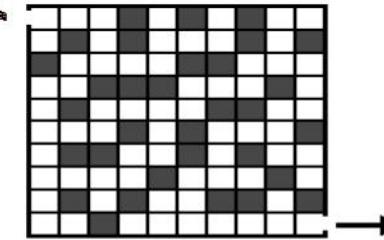
### Exercicis de taulells



# Tècnica de backtracking

## Exercicis que treballen amb un taulell.

Practiquem els diferents esquemes de backtracking



1	2	3	4	5	6	7	8
2							
3							
4							
5							
6							
7							
8							

### ■ Creuar un Laberint

Esquema trobat **1 solució**

### ■ Salt del Cavall

Esquema trobar **TOTES** les solucions

### ■ Rei Coix

Esquema trobar la **MILLOR** solució





Laberint

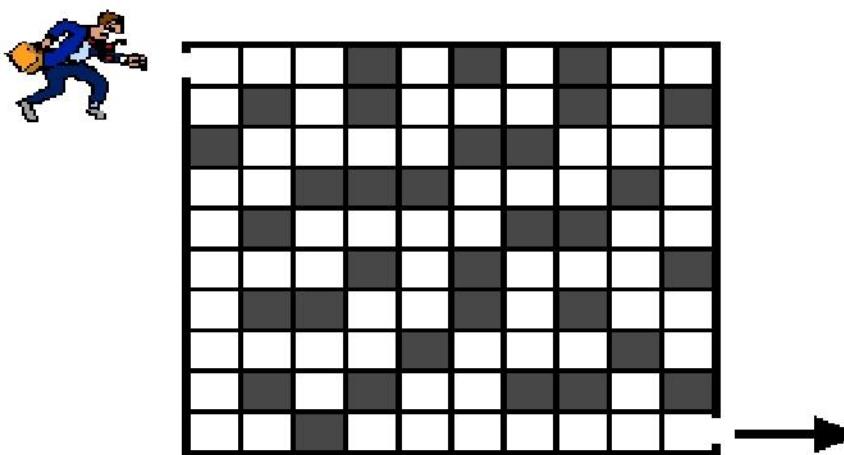
# Tècnica de backtracking

## ■ Enunciat: Creuar un laberint

Estem situats a l'entrada d'un laberint i volem creuar-lo.

El laberint està representat mitjançant una matriu de  $N \times N$  components marcades amb un valor de **Iliure** si s'hi permet l'accés i un **ocupat** si representa una paret. Els moviments possibles: **horizontal ó vertical** per passar d'una casella a altra.

**Objectiu:** anar de la casella  $(1,1)$  a la  $(N,N)$ .





# Tècnica de backtracking

Java

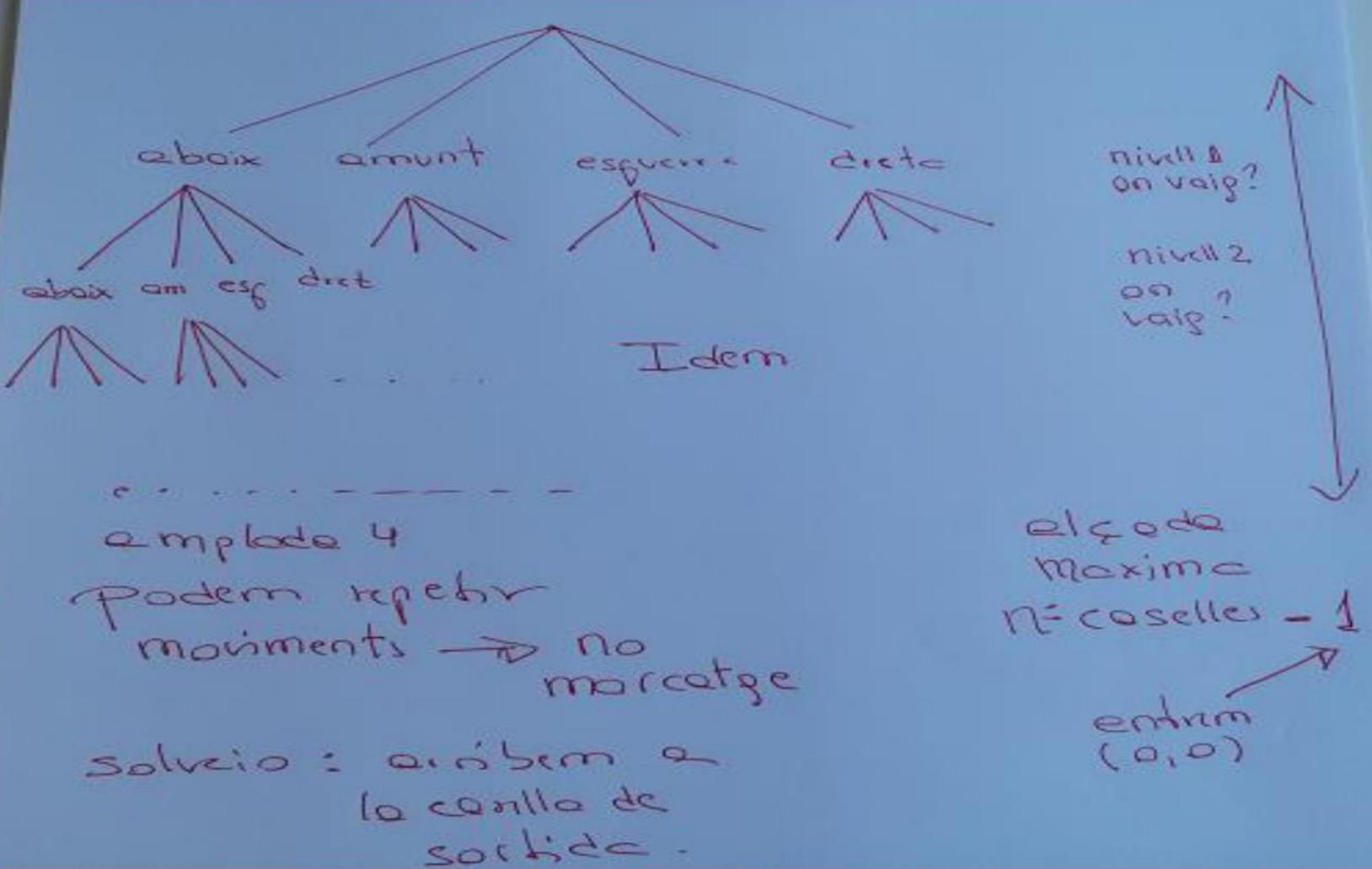
## Anàlisi del problema

- 1.- Podem aplicar backtracking? Si, la solució la podem expressar com una seqüència de decisions, cal decidir quins moviments cal fer per sortir.
- 2.- Quina decisió en cada nivell? Decidir a quina casella ens **deplacem. On vaig?**
- 3.- Quin és el domini de les decisions? **Els 4 moviments: amunt, abaix, dreta i esquerra.**
- 4.- Cal usar marcatge? **NO, els moviments es poden repetir.**
- 5.- Quina és l'amplada de l'arbre? **4.**
- 6.- I l'alçada? És exacta o màxima? **Màxima, el pitjor cas ve donat pel nombre de caselles del taulell menys 1 (la de sortida (0,0))**
- 7.- Condició per saber si és solució, quina? **Estem ubicats a la casella de sortida.**



# Tècnica de backtracking

## Anàlisi del problema. Espai de Cerca





# Tècnica de backtracking

```
public static boolean Back1Solucio( TaulaSolucio TS, int k){  
    boolean trobada=false;  
    inicialitzem_valors_domini_decisio_k  
    agafar_el_primer_valor  
    while (quedin_valors && !trobada){  
        if (valor_acceptable){ //no viola les restriccions  
            anotem_el_valor_a_la_solucio  
            if (solucio_final) trobada=true;  
            else if (solucio_completable)  
                trobada=Back1Solucio(TS, k+1);  
            if (!trobada) desanotem_el_valor  
        } //fi if  
        agafar_seguent_valor  
        //passem al següent germà a la dreta  
    } // fi while  
    return trobada;  
} // fi procediment
```

Esquema de cerca

Opcionalment  
es pot fer el  
tractament  
abans d'acabar  
el procediment

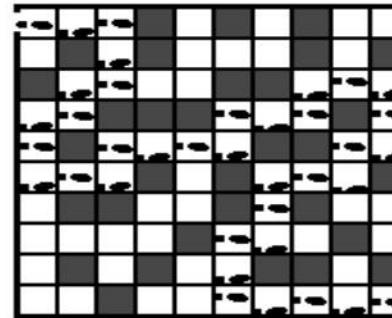


# Tècnica de backtracking

Esquema 1 solució

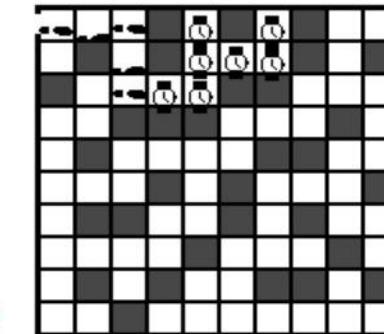
- Solució: aplicar la tècnica del backtracking
- Marcar en la mateixa matriu que representa el laberint el camí a seguir, si existeix

Solució en la  
mateixa taula



- Si seguint un camí s'arriba a una casella de la qual es impossible continuar tornem enrera i busquem una alternativa de camí. S'han de marcar les caselles per a on ja hem passat per evitar de tornar-hi (impossible).

No tècnica del matcatge,  
els moviments es poden repetir.  
Si marcar caselles per no repetir-les





# Tècnica de backtracking

```
import Keyboard;
public class Laberint{
    private static final int lliure=0;
    private static final int paret=1;
    private static final int cami=2;
    private static final int impossible=3;
    private int [][]tau;
    public Laberint(int tamany) throws Exception{
        if (tamany<=0) throw new
            Exception("paràmetre incorrecte");
        tau=new int [tamany][tamany];
        for (int i=0; i<tamany; i++)
            for (int j=0; j<tamany; j++) {
                System.out.println("Paret en la
                    posició"+i+" "+j+"?");
                tau[i][j]=Keyboard.readInt(); //valor 0/1
            }
    } // fi constructor
```

Possible contingut d'una casella del taulell



# Tècnica de backtracking

```
public String toString(){ //redefinició  
    String aux="";  
    for (int i=0; i<tau.length; i++){  
        for (int j=0; j<tau.length; j++) aux=aux+l[i][j];  
        aux = aux+"\n";  
    }  
    return aux;  
}
```

El `main` podria estar en una altra classe

```
public static void main(String args[]) throws Exception{  
    int n=Keyboard.readInt();  
    Laberint l=new Laberint(n);  
    System.out.println(l);  
    if (!l.trobar1cami(0,0))  
        System.out.println(l+"No existeix solució");  
    else System.out.println(l);  
} //fi main
```

Propera casella  
a visitar

// El mètode que fa el back NO és STATIC



JAVA

# Tècnica de backtracking

Esquema alterat

```
public boolean trobar1cami(int x, int y){ // les coord són correctes?  
    boolean trobat=false; int j=0;  
    if (x>=0 && x<tau.length && y>=0 && y<tau.length &&  
        tau[x][y]==lliure){  
        tau[x][y]=cami;  
        if (x==tau.length-1 && y==tau.length-1) trobat= true; //solució  
        while (j<4 && !trobat){  
            switch(j){  
                case 0: trobat=trobar1cami(x+1, y);break;  
                case 1: trobat=trobar1cami(x-1, y);break;  
                case 2: trobat=trobar1cami(x, y-1);break;  
                case 3: trobat=trobar1cami(x, y+1);break;  
            }  
            j++; /*seguent moviment*/  
        } //fi while  
        if(!trobat)tau[x][y]=impossible;//desfer decisió un cop provats 4  
    } /*fi if*/  
    return trobat;  
} // fi mètode
```

Al fer la crida recursiva no ens preocupem de que les coordenades siguin correctes, ho mira el backtracking:

- 0: abax
- 1: amunt
- 2: esquerra
- 3: dreta

**Compte lloc on està el desfer!!!**

Fins que no hem provat els 4 sentits i hem comprovat que no porten a la casella de sortida no desfem la decisió

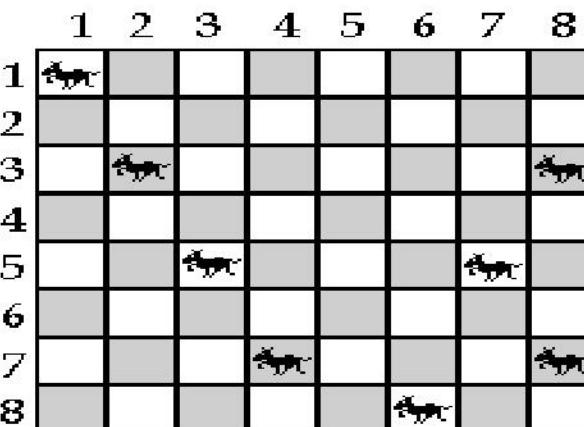


# Salt Cavall

## Tècnica de backtracking

### ■ Enunciat: El salt del cavall

Es disposa d'un taulell d'escacs i d'un cavall, que es mou segons les regles del joc. L'objectiu és trobar **totes** les possibles **maneres de recórrer tot el taulell** partint d'una casella determinada, de tal manera que passi una única vegada per cada casella.



### Sortida

Sobre el mateix taulell marcarem l'ordre de visita, enumerant-los

El taulell contindrà els números del 0 al 63 (taulell de 8x8)

Esquema **TOTES** solució

### Moviments legals





Sant Cugat

# Tècnica de backtracking

## Anàlisi del problema

- 1.- Podem aplicar backtracking? **Si**, la solució la podem expressar com una seqüència de decisions, cal decidir quins moviments cal fer.
- 2.- Quina decisió en cada nivell? **Quin moviment fem?**
- 3.- Quin és el domini de les decisions? **Els 8 moviments del cavall en el joc dels escacs.**
- 4.- Cal usar marcatge? **NO**, els moviments es poden repetir.
- 5.- Quina és l'amplada de l'arbre? **8.**
- 6.- I l'alçada? És exacta o màxima? **Exacta, 63 caselles del taulell (la de sortida no es pot seleccionar).**
- 7.- Condició per saber si és solució, quina? **Hem arribat a una fulla de l'arbre.**



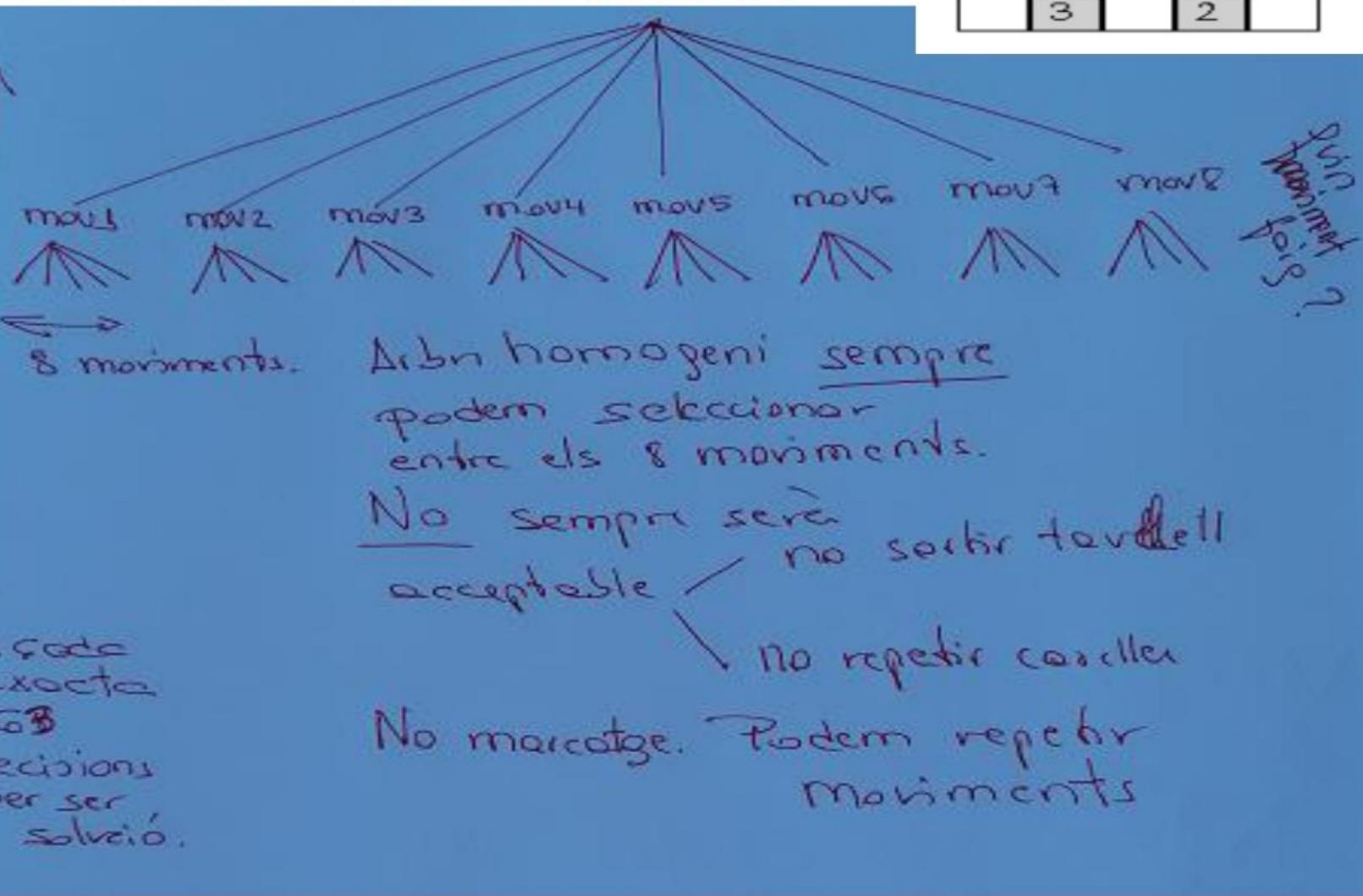
Salt Cavall!

# Moviments legals

	6		7	
5				8
			↗	
4				1
	3		2	

## Tècnica de backtrack

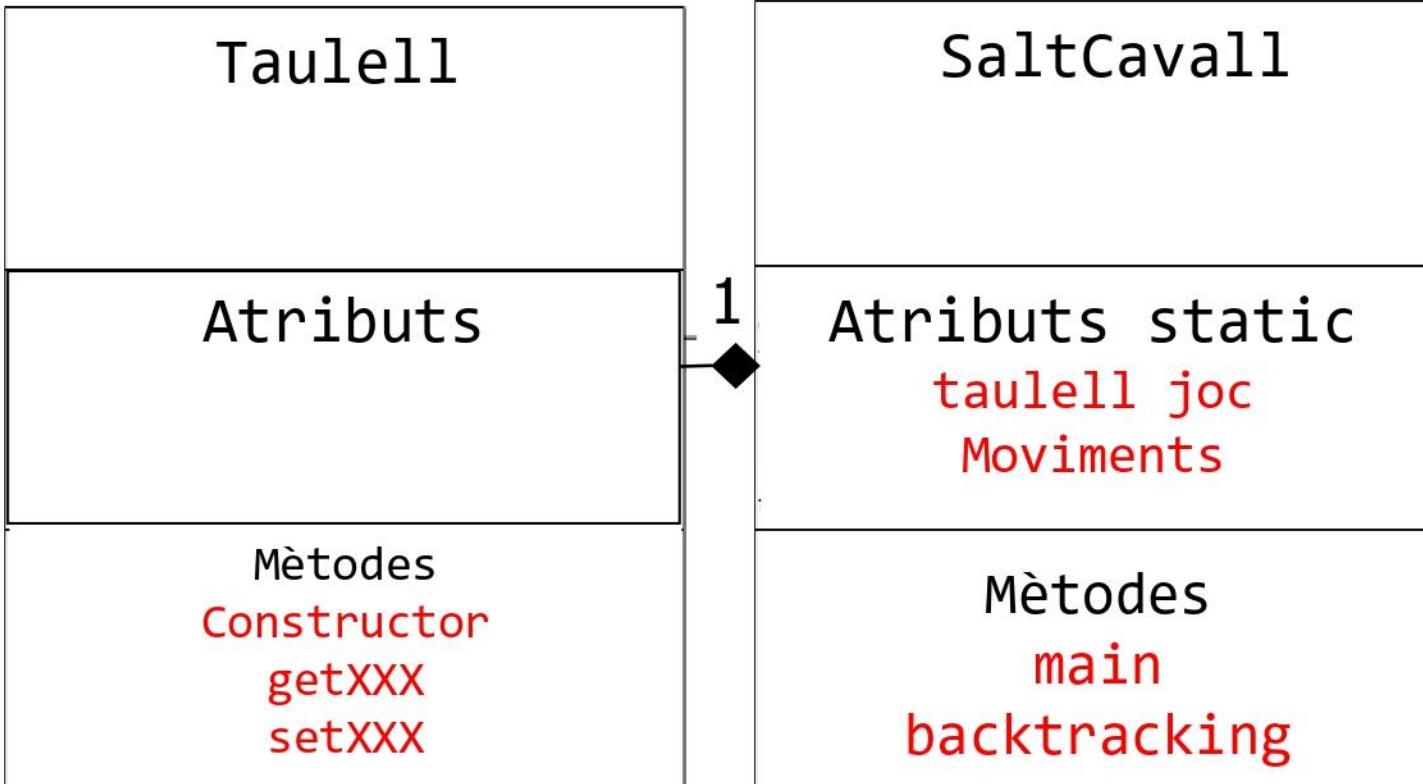
### Anàlisi del problema. Espai de Cerca





# Tècnica de backtracking

## Disseny de classes





Salt Cavall

# Tècnica de backtracking

- Denoteu que des de una casella només es pot avançar una posició en línia recta i una altre en diagonal en la mateixa direcció. Les coordenades **relatives** a on es troba el cavall dels possibles moviments les podem disposar:

Moviments legals

```
CoordenadaX[] = { 1, 2, 2, 1, -1, -2, -2, -1 };
CoordenadaY[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
```

	6		7		8
5					
4					1
	3		2		

```
import Keyboard;
public class Taulell{
    private int x,y; //lloc on es troba el cavall inicialment
    private int [][]taula; //creada en el constructor
```

Atributs



Sant Cavall

# Tècnica de backtracking

```
public Taulell(int x, int y, int N) throws Exception{
    if (N<=0 || x<0 || x>=N || y<0 || y>=N)
        throw new Exception("Parametres incorrectes");
    this.x=x; this.y=y;
    taula = new int[N][N];
    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++) taula[i][j]=-1;
    taula[x][y]=0; // es marca on es troba el cavall
} // fi constructor
public String toString(){
    String aux = "Punt de partida del cavall: " + x + " " + y + "\n";
    for (int i=0; i<taula.length; i++){
        for (int j=0; j<taula.length; j++)
            aux=aux+taula[i][j]+("-");
        aux=aux+"\n";
    }
    return aux;
}
```

Pendent  
de visita



Sant Cavall

# Tècnica de backtracking

```
public void setAnotar(int fil, int col, int i){  
    taula[fil][col]=i;  
}  
public void setDesAnota(int fil, int col){  
    taula[fil][col]=-1;  
}  
public int getNumCaselles(){  
    return taula.length*taula.length;  
}  
public int getDonaValor(int fil, int col){  
    return taula[fil][col];  
}  
public boolean isPosicioOk(int fil, int col){  
    return (fil>=0 && fil<taula.length && col>=0  
        && col<taula.length);  
}  
} // fi classe Taulell
```

Mètodes  
setXX i  
getXXX

No bona idea treure el  
taulell a l'exterior!!!  
~~getTaula~~



# Tècnica de backtracking

```
public class SaltCavall{  
    private static final int CoordenadaX[]={1,2, 2,1,-1,-2,-2,-1 };  
    private static final int CoordenadaY[]={2,1,-1,-2,-2,-1,1, 2 };  
    private static Taulell joc;  
    public static void main(String args[]) throws Exception{  
        int N, x,y;  
        System.out.println("Indica el mida del taulell");  
        N=Keyboard.readInt(); //possibilitar altres mides taulell  
        System.out.println( "Indica on està situat el cavall");  
        x=Keyboard.readInt();y=Keyboard.readInt();  
        joc=new Taulell(x,y,N);  
        trobarMoviments(1,x,y);  
    }  
}
```

1 → nivell on estem, proper numero a guardar en la propera visita.  
x,y → casella on ens trobem.



# ESQUEMA TOTES

## Tècnica de backtracking

```
public static void BackTotesSolucions ( TaulaSolucio TS, int k){  
    inicialitzem_valors_domini_nivell_k  
    agafar_el_primer_valor_decisio_K  
    while (quedin_valors_domini){  
        if (valor_acceptable){ //no viola les restriccions  
            anotem_valor_a_la_solucio  
            if (solucio_final) escriure_solucio  
            else if (solucio_completable)  
                BackTotesSolucions(TS, k+1);  
            desanotem_valor  
        } //fi if  
        agafar_seguent_valor  
        // passem al següent germà a la dreta  
    } // fi while  
} // fi procediment
```

Estem al nivell **k** de l'arbre de cerca



SaintCavall

```
public static void trobarMoviments(int k, int x, int y){  
    int col, fil;  
    for (int i=0; i<8; i++){ // amplada de l'arbre  
        fil=x+CoordenadaX[i];  
        col=y+CoordenadaY[i];  
        if (joc.isPosicioOk(fil, col)){ //Moviment acceptable  
            if (joc.getDonaValor(fil, col)==-1) { //No visitada  
                joc.setAnotar(fil, col, k);  
                if ((k+1)<joc.getNumCaselles()) //encara No  
                    trobarMoviments(k+1,fil,col);  
            } else //trobada una solució  
                System.out.println(joc);  
                joc.setDesAnota(fil, col);  
        } // fi if  
    } //fi if  
} //fi for  
} //fi mètode
```

Important  
que la k+1 no  
superi el  
nombre de  
caselles

Adoneu-vos! No modifiquem  
els paràmetres x,y



X  
I  
O  
R  
E  
I

# Tècnica de backtracking

## Enunciat: Rei Coix.

Limitat a 3 moviments

Objectiu: trobar el millor camí a seguir des de la posició incial a la final, acumulant el màxim de diners. En cada component del taulell hi ha un operador:

+ : sumar 12 euros

- : restar 10 euros

\* : multiplicar \*2

/ : dividir / 2

A l'accendir a la casella apliquem el càlcul sobre la quantitat de diners que es disposa en aquest moment. Inicialment es disposa de 10 euros.



Posició inicial

Posició final



# Tècnica de backtracking

X  
II  
O  
II  
R  
II  
é

## Moviments possibles

- **Moviment 1:** la seva posició canviarà a la posició  $(x+1, y)$  **Abaix**
- **Moviment 2:** canviarà a la posició  $(x, y-1)$  **Esquerra**
- **Moviment 3:** canviarà a la posició  $(x+1, y-1)$  **Diagonal abaix esquerra**

- 1.- **NO marcarem el resultat al taulell perquè no s'ha de perdre l'operador que emmagatzema, quan desfem cal tenir-lo disponible.**
- 2.- **La repetició de caselles visitades no és possible amb els moviments permesos.**



# Tècnica de backtracking

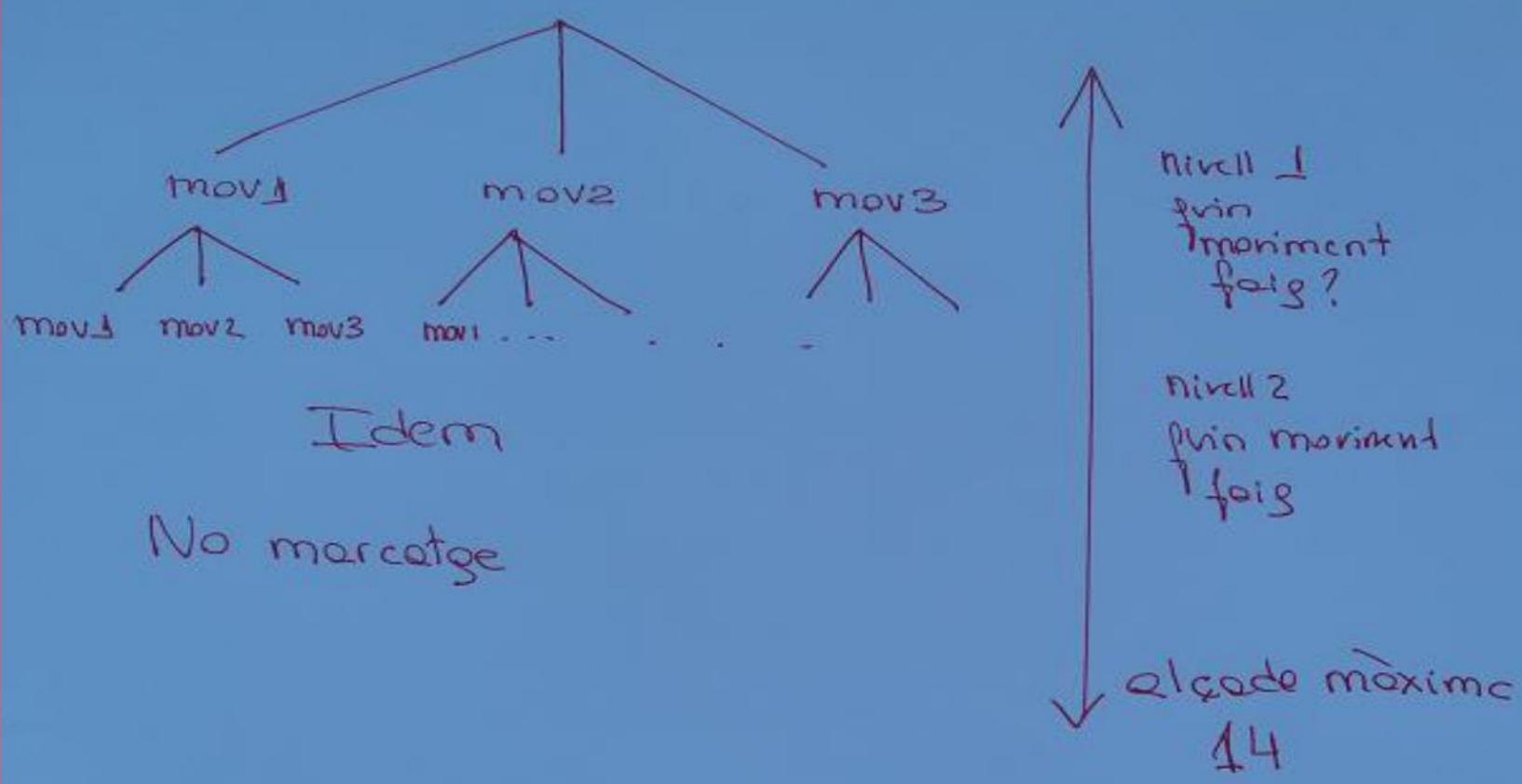
X  
I  
O  
C  
T  
R

## Anàlisi del problema. És d'optimització!

- 1.- Podem aplicar backtracking? Si, la solució la podem expressar com una seqüència de decisions, cal decidir quins moviments per arribar a la casella de sortida.
- 2.- Quina decisió en cada nivell? Decidir a quina casella ens desplaçem. Quin moviment faig?
- 3.- Quin és el domini de les decisions? **Els 3 moviments permesos pel rei coix.**
- 4.- Cal usar marcatge? **NO, els moviments es poden repetir.**
- 5.- Quina és l'amplada de l'arbre? **3.**
- 6.- I l'alçada? És exacta o màxima? **Màxima, el pitjor cas ve donat pel camí més llarg que està format per 14 caselles.**
- 7.- Condició per saber si és solució, quina? **Estem ubicats a la casella destí.**

# Tècnica de backtracking

Rei Coix



## Tècnica backtracking: El Rei Coix



Rei Coix

```
public class Taulell{  
    private class Coordenada{  
        int x,y;  
        public Coordenada(int x, int y){  
            this.x=x; this.y=y;  
        }  
        public String toString(){  
            return new String(x+""+y);}  
        public int getX(){return x;}  
        public int getY(){return y;}  
    } // fi classe interna  
    public static final int N=8; // taulell 8x8  
    private char [][]taulell;  
    public Taulell(){ //constructor  
        // sentències per la creació i omplenat del taulell  
    }
```



X  
C  
O  
I  
R

# Tècnica backtracking: El Rei Coix

```
public static void main(String args[]){
    //1.- creació objectes i variables
    //2.- crida procediment backtracking

    //3.- visualització de les caselles formen el camí
    for (int i=0; ??????; i++)
        System.out.println(?????);
}

public void backMillor(?????){
    /*sentències*/
}
```



Rei Codi

# Tècnica backtracking: Solució

## 1.- Afegim atributs a la classe:

```
private Coordenada solucio[];  
private float dinersActual;
```

Solució en construcció

```
private Coordenada millor[];  
private float dinersMillor;  
private int quantes  
// quantes posicions plenes té la taula millor
```

Millor solució

Problema d'Optimització



X  
I  
O  
C  
R  
E

# Tècnica backtracking: El Rei Coix

```
public Taulell(){ //constructor
    // Creació i omplenat del taulell
    taulell=new char[N][N];
    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            taulell[i][j]=Keyboard.readChar();
    // creació magatzems
    solucio = new Coordenada[14];
    dinersActual = 10.0; //mai pot ser negatiu!!!
    millor=new Coordenada[14];
    dinersMillor = -1.0; quantes=0;
}
```



X  
O  
R

# Tècnica Backtracking: Solució

```
public static void main(String args[]){
    //creació objectes i variables
    Taulell t=new Taulell();           Coordenades on
                                         //està el rei
    t.BackMillor(0,0,7); //dos darrers paràmetres
                          Nivell //on esta el rei!
    for (int i=0; i<t.quantes; i++)
        System.out.println(t.millor[i]);
    // visualització solució a pantalla
}
```



# 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'  
        if (valor_acceptable){ //no viola les re  
            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 Backtracking: Solució

**0: Abaix**  
**1: Esquerra**  
**2: Diagonal**

Ré

```
public void BakMillor(int k, int x, int y ){
    for (int j=0; j<3; j++){
        Coordenada p=acceptable(x,y,j); //retorna on estarà rei
        if (p!= null){
            float Anterior=dinersActual;
            //copiem diners sobre Anterior per si cal desfer
            solucio[k]=p;
            dinersActual=Acumular(taulell[p.getX()][p.getY()]);
            if (p.getX()==7 && p.getY()==0){
                //Solució - cal mirar si és millor
                if (dinersActual > dinersMillor){
                    for(int i=0; i<=k; i++)
                        millor[i]=solucio[i];
                    quantes=k+1; dinersMillor=dinersActual;
                }
            } //fi solució
        else //sempre hi ha solució
            BakMillor(k+1, p.getX(), p.getY());
```



REICO

# Tècnica Backtracking: Solució

```
dinersActual=Anterior; //desfer solució  
                                No cal posar a null  
solutio[k] perquè tenim un atribut que controla  
el nombre de posicions plenes (quantes)  
} //fi if  
} //fi while  
} //fi mètode  
private Coordenada acceptable(int x, int y, int mov){  
    switch(mov){  
        case 0: //Abaix  
            if (x==N-1) return null;  
            else return new Coordenada(x+1, y);  
        case 1: //Esquerra  
            if (y==0) return null;  
            else return new Coordenada(x, y-1);  
        case 2: //Diagonal esquerra  
            if (x==N-1 || y==0) return null;  
            else return new Coordenada(x+1, y-1);  
    }  
    return null;  
}
```



X  
II  
O  
II  
R  
II

# Tècnica Backtracking: Solució

```
private float Acumular(char operador){  
    switch (operador){  
        case '+': return dinersActual+12;  
        case '-': if (dinersActual-10<0)  
                    return 0;  
                    else return dinersActual-10;  
        case '*': return dinersActual*2;  
        case '/': return dinersActual/2;  
    }  
    return 0;  
} // fi mètode
```