



Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos



Escuela Técnica Superior de
Ingeniería de Sistemas Informáticos
Universidad Politécnica de Madrid

Tema 8. Esquema Backtracking

Algorítmica y Complejidad

1

Introducción

Caracterización de los problemas

- **Tipo de problemas:**
 - **Problemas de optimización**, donde la solución es expresable en forma de secuencia de decisiones (con o sin restricciones).
 - **No se conoce a priori un criterio óptimo** de selección en cada decisión.
 - Se aplica a **problemas** en los que **no existe más remedio que buscar**
 - Se presentan de forma natural como **problemas recursivos**.
 - Cuenta con la funcionalidad:
 1. Función **factible** que permite averiguar si la *solución en curso actual*, viola o no las restricciones.
 2. Función **solución**, que permite determinar si una secuencia de decisiones factible es solución al problema planteado.
 3. Función **objetivo**, devuelve la bondad de la solución hallada.

1. Introducción



2

2

Introducción

Tipo de problemas. Ejemplos

- Dado un conjunto de números enteros $\{e_1, e_2, e_3, \dots, e_n\}$, encontrar si existe algún subconjunto cuya suma sea exactamente X .
 - Hay que probar todas las posibles combinaciones sin repetición (por la propiedad conmutativa de la suma) de los n elementos tomados de z en z , con $z=1\dots n$.
 - Por ejemplo, para $\{13, 11, 7\}$ y $X=20$ las combinaciones a probar serían: $13, 11, 7, 13+11, 13+7, 11+7$ y $13+11+7$
- Dado un tablero de ajedrez de tamaño $N \times N$, colocar N reinas sin que se den jaque (dos reinas se dan jaque si comparten fila, columna o diagonal).
 - Probar todas las posiciones posibles y para cada una comprobar si es válida. Para tamaño $N=8$: [4.426.165.368 posibilidades](#).
- Dado un grafo valorado y dos vértices (*origen* y *destino*), encontrar un camino simple de coste exacto C entre esos dos vértices.
 - Probar todas las posibles combinaciones de caminos simples (no se repiten vértices en el camino, excepto quizás origen y destino) entre los vértices *origen* y *destino* hasta encontrar uno con el coste exacto indicado.

1.Introducción

3

Introducción

Características del esquema

- La *vuelta atrás**:
 - Es un **esquema** que, de forma **sistemática** y **organizada**, genera y **recorre un espacio** que **contiene todos los posibles estados** (secuencias de decisiones).
 - Si existe solución, seguro que la encuentra
 - Este espacio se denomina el **espacio de búsqueda del problema**, y se representa como un **árbol** (puede ser un grafo) sobre el que el algoritmo hace un recorrido en profundidad (recursividad) partiendo de la raíz.
 - Se conoce de antemano el orden en que se van a generar y recorrer sus nodos, y se continúa recorriendo el árbol mientras se cumplan las restricciones.
 - Normalmente el árbol de búsqueda crece de forma exponencial y es habitual plantear criterios de poda para evitar explorar caminos que no conducen al éxito.
 - Tiene **tres posibles esquemas**: encontrar **una solución factible**, encontrar **todas las soluciones factibles**, encontrar **la mejor solución factible**.

* también se conocen como *backtracking*, el término "backtrack" fue acuñado por primera vez por el matemático estadounidense D. H. Lehmer en los años 50's

1.Introducción

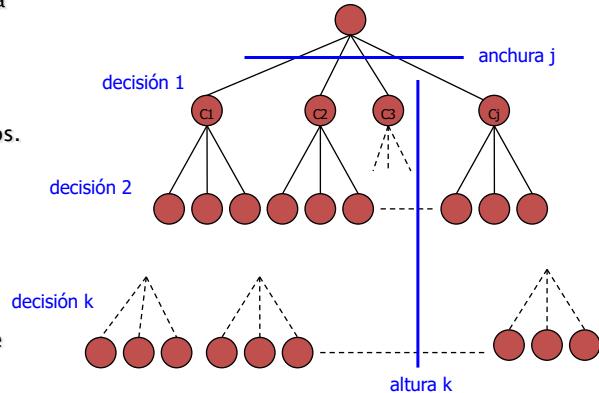
4

Introducción

Características del esquema

- **Espacio de búsqueda:**

- la **altura del espacio**: hay k estados por los que pasar para formar una solución.
- la **anchura del espacio**: cada estado tiene asociado un dominio formado por j decisiones/candidatos distintos.
- j^k hojas
- $n^{\circ} \text{nodos} = \sum_{i=0}^k j^i \in O(j^k)$
- Puede que exista más de un espacio para el mismo problema. Por regla general se elige el más pequeño o el de generación menos costosa.



1. Introducción



5

5

Introducción

Esquema general

- **Método de resolución:**

- La solución se puede expresar como una tupla (x_1, x_2, \dots, x_n) , satisfaciendo unas restricciones $P(x_1, x_2, \dots, x_n)$ y tal vez optimizando una cierta función objetivo.
- En cada momento, el algoritmo se encontrará en un cierto nivel k , con una **solución parcial** (x_1, \dots, x_k) .
 - Si se puede **añadir un nuevo** elemento a la solución x_{k+1} , se genera y se avanza al nivel $k+1$.
 - Si no, se prueban **otros valores de x_k** .
 - Si no existe ningún valor posible por probar, entonces se **retrocede** al nivel anterior $k-1$.
- Se sigue hasta que la solución parcial sea una solución completa del problema, o hasta que no queden más posibilidades.
- El resultado es equivalente a hacer un recorrido en profundidad en el árbol de soluciones. Este árbol es implícito, no se almacena en ningún lugar.

1. Introducción



6

6

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

Árbol de búsqueda implícito generado durante la ejecución del algoritmo (recorrido en profundidad)

1.Introducción

7

Introducción

Esquema general

```

void vueltaAtrs (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtrs(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

```

public class Booleano {
    private boolean valor;
    public Booleano(boolean valor) { this.valor = valor;}
    public boolean getValor() { return this.valor;}
    public void setValor(boolean valor) { this.valor = valor;}
}

```

Estructuras de datos

- Estado:** variables (parámetros) para mantener el estado actual de la búsqueda. Dependen del problema a resolver.
- exito:** Objeto Booleano (clase Java a implementar) inicialmente a *false*.

1.Introducción

8

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while (!exito.getValor() && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

Llamada inicial

Booleano exito = new Booleano(false);
Estado estado -> crear e inicializar variables que definen el estado;
vueltaAtras(estado, exito);
if (!exito.getValor()) <<no existe solución>>
else <<solución encontrada>>

1.Introducción

1 2 3

9

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

Solución

- El estado actual es una solución
- Modifica parámetro éxito. Termina la ejecución del nivel actual y devuelve el control al nivel anterior (vuelta atrás).
- La estructura del esquema hace que cada nivel devuelva el control al nivel anterior hasta el nodo raíz, que también terminará su ejecución.

1.Introducción

1 2 3

10

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

Candidatos

- Declaración e inicialización de una variable o variables (locales) para generar cada uno de los candidatos del nodo actual.
- Cada candidato supone una decisión (paso) a añadir a la solución parcial encontrada hasta el momento.

1.Introducción

11

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

Quedan candidatos y no exito

- Bucle que itera por cada candidato (ramas que surgen del nodo del árbol).
- El bucle se repite mientras no se encuentre una solución (en alguno de los nodos del subárbol que surge de cada candidato) y queden candidatos en el nodo actual.

1.Introducción

12

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

Seleccionar candidato

- En cada iteración del bucle se selecciona un candidato (rama).
- Cuando selecciona candidato el algoritmo ya ha recorrido los subárboles que surgen de los candidatos anteriores (aceptables) sin haber encontrado solución.

1.Introducción

13

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

Aceptable

- Candidato aceptable si cumple todas las restricciones del problema para poder añadirlo como "paso" o "decisión" en la solución parcial actual.
- Si no es aceptable lo desecha y sigue iterando por el bucle. Si no quedan más candidatos devuelve el control al nivel anterior.

1.Introducción

14

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }
        }/*if aceptable*/
    } /*while*/
} /*else*/
}

```

Anotar

- Incluye candidato en la solución parcial actual.
- Da soporte al avance en el espacio de búsqueda que se produce en la llamada recursiva...

1.Introducción

15

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }
        }/*if aceptable*/
    } /*while*/
} /*else*/
}

```

Avance en el espacio de búsqueda

- Llamada recursiva.
- Avanza al nuevo nodo indicando nuevo estado (con el candidato recién incluido en la solución parcial).
- El nivel desde el que se hace la llamada recursiva espera hasta que le devuelvan el control.

1.Introducción

16

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

No éxito: desanotar

- No ha encontrado solución por la rama del candidato actual: elimina candidato en la solución parcial actual. Acción inversa a Anotar.
- Vuelve al estado anterior a anotar para probar otro camino de búsqueda.
- Si no quedan más candidatos devuelve el control al nivel anterior.

1.Introducción

17

17

Introducción

Esquema general

```

void vueltaAtras (Estado estado, Booleano exito, ...){
    if (esSolucion(estado)) {
        exito.setValor(true);
    } else {
        inicializar_conjunto_de_candidatos();
        while ((!exito.getValor()) && (hay_mas_candidatos())){
            seleccionar_un_candidato();
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                vueltaAtras(proximoEstado, exito, ...);
                if (!exito.getValor()){
                    desanotar_candidato();
                }
            }/*if aceptable*/
        } /*while*/
    } /*else*/
}

```

• Debemos poder expresar la **solución del problema** como una **lista de decisiones** (de la misma naturaleza).

• En un momento dado la **solución parcial** actual estará formada por las **decisiones tomadas en la rama actual del árbol de estados**.

*La ejecución del esquema recorre el árbol en profundidad
(árbol de estados implícito generado por la recursividad)*

1.Introducción

18

18

Introducción

Ejemplo. Suma de enteros

Dado un vector de números enteros positivos Y un número: 20

13	11	7
----	----	---

encontrar si existe algún subconjunto de elementos del vector cuya suma sea exactamente el número indicado.

13	7
----	---

Alternativas de árboles de estados:

- A. Árbol binario.
- B. Árbol combinatorio.

1. Introducción



19

19

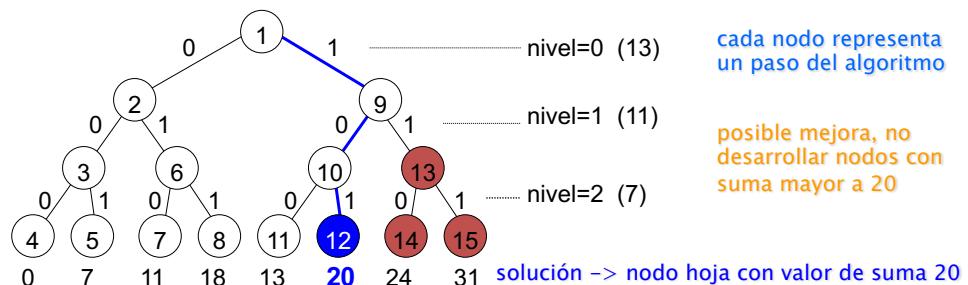
Introducción

Ejemplo. Suma de enteros

Dado un vector de números enteros positivos: 13 11 7 Y un número: 20
encontrar si existe algún subconjunto de elementos del vector cuya suma sea exactamente el número indicado.

A. Árbol de estados binario.

En cada nivel *i* decidir si el elemento *i* forma parte o no de la solución.



1. Introducción



20

20

Introducción

Ejemplo. Suma de enteros

Dado un vector de números enteros positivos:

13	11	7
----	----	---

 Y un número:

20

encontrar si existe algún subconjunto de elementos del vector cuya suma sea exactamente el número indicado.

A. Árbol de estados binario

Estructura datos solución: (x_1, x_2, \dots, x_n) , donde $x_i = (0, 1)$.

1	0	1
---	---	---

Nodo del árbol: trata el elemento de la posición i del vector.

Parámetros estado: vector *solucion* y *nivel*

Llamada inicial: subconjuntoSumaBack(vector, num, solucion, nivel: 0, éxito)

Solución: si $nivel == vector.length ->$ si suma números seleccionados == num

Candidatos: no añadir/ añadir el elemento i -ésimo del vector

Aceptable: si candidato==añadir -> si añadir $vector[i]$ a la solución no supera valor objetivo

Anotar: $solucion[nivel] = 1$ (si candidato == añadir); $nivel = nivel + 1$;

Desanotar: $nivel = nivel - 1$; $solucion[nivel] = 0$;

1.Introducción



21

21

Introducción

Ejemplo. Suma de enteros (árbol binario versión 1)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
                        Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}
```

1.Introducción



22

22

Introducción

Ejemplo. Suma de enteros (*árbol binario versión 1*)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
    Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}

int suma (int[] solucion, int[] vector, int nivel){
    int suma=0;
    for (int i=0; i<nivel; i++)
        if (solucion[i]==1) suma = suma+vector[i];
    return suma;
}
```

1. Introducción



23

23

Introducción

Ejemplo. Suma de enteros (*árbol binario versión 1*)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
    Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}

int suma (int[] solucion, int[] vector, int nivel){
    int suma=0;
    for (int i=0; i<nivel; i++)
        if (solucion[i]==1) suma = suma+vector[i];
    return suma;
}
```

1. Introducción



24

24

Introducción

Ejemplo. Suma de enteros (árbol binario versión 1)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
    Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}
```

Quedan candidatos y no exito

1.Introducción



25

25

Introducción

Ejemplo. Suma de enteros (árbol binario versión 1)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
    Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}
```

Candidato aceptable

```
int suma (int[] solucion, int[] vector, int nivel){
    int suma=0;
    for (int i=0; i<nivel; i++)
        if (solucion[i]==1) suma = suma+vector[i];
    return suma;
}
```

1.Introducción



26

26

Introducción

Ejemplo. Suma de enteros (*árbol binario versión 1*)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
    Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}
```

anotar

1.Introducción



27

27

Introducción

Ejemplo. Suma de enteros (*árbol binario versión 1*)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
    Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}
```

Avance
en el
espacio
de
búsqueda

1.Introducción



28

28

Introducción

Ejemplo. Suma de enteros (árbol binario versión 1)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
                        Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}
```

No éxito,
desanotar

1.Introducción



29

29

Introducción

Ejemplo. Suma de enteros (árbol binario versión 1)

```
void subconjuntoSumaBack(int[] vector, int num, int[] solucion, int nivel,
                        Booleano exito){
    if (nivel==vector.length){
        if (suma(solucion, vector, nivel)==num) exito.setValor(true);
    }else {
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || suma(solucion, vector, nivel)+vector[nivel] <= num){
                solucion[nivel] = c;
                nivel = nivel + 1;
                subconjuntoSumaBack(vector, num, solucion, nivel, exito);
                if (!exito.getValor()){
                    nivel=nivel-1;
                    solucion[nivel]=0;
                }
            }
            c++;
        }
    }
}
```

int[] vector = {13, 11, 7}; int num = 20; Booleano exito = new Booleano(false); int[] solucion = new int[vector.length]; for (int i=0; i<solucion.length; i++) solucion[i]=0; subconjuntoSumaBack(vector, num, solucion, 0, exito); Llamada inicial

1.Introducción



30

30

Introducción

Ejemplo. Suma de enteros (árbol binario versión 2)

```

void subconjuntoSumaBack2(int[] vector, int num, int[] solucion, int nivel, int suma,
                           Booleano exito){
    if (nivel==vector.length){
        if (suma == num) exito.setValor(true);
    } else{
        int c=0; // Candidatos: c=0 no añadir; c=1 añadir
        while (!exito.getValor() &&(c < 2)){
            if (c==0 || ((suma + vector[nivel])<=num)){
                solucion[nivel] = c;
                suma = suma + (vector[nivel]*c);
                nivel = nivel + 1;
                subconjuntoSumaBack2(vector, num, solucion, nivel, suma, exito);
                if (!exito.getValor()){
                    nivel = nivel - 1;
                    solucion[nivel] = 0;
                    suma = suma - (vector[nivel]*c);
                }
            }
            c++;
        }
    }
}

```

int[] vector = {13, 11, 7}; int num = 20; Booleano exito = new Booleano(false); int[] solucion = new int[vector.length]; for (int i=0; i<solucion.length; i++) solucion[i]=0; int suma=0; subconjuntoSumaBack2(vector, num, solucion, 0, suma, exito);

1. Introducción



31

31

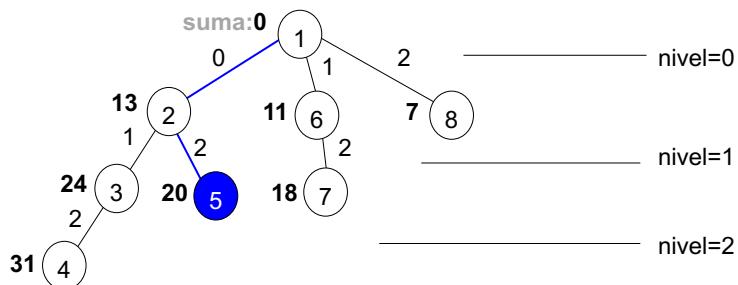
Introducción

Ejemplo. Suma de enteros

Dado un vector de números enteros positivos: 13 11 7 Y un número: 20 encontrar si existe algún subconjunto de elementos del vector cuya suma sea exactamente el número indicado.

B. Árbol de estados combinatorio.

2. En cada nivel *i* decidir qué elemento se añade (0, 1 o 2).



Cada nodo es una posible solución. Será válida si la suma es 20.

1. Introducción



32

32

Introducción

Ejemplo. Suma de enteros

Dado un vector de números enteros positivos:

13	11	7
----	----	---

 Y un número:

20

encontrar si existe algún subconjunto de elementos del vector cuya suma sea exactamente el número indicado.

B. Árbol de estados combinatorio

Estructura datos solución: (x_1, x_2, \dots, x_n) , donde $x_i = (0, 1)$.

1	0	1
---	---	---

Nodo del árbol: intenta añadir un único número del vector a la solución.

Parámetros estado: vector *solucion*, suma acumulada e *id* siguiente candidato

Llamada inicial: subconjuntoSumaBack3(vector, num, solucion, id: 0, suma:0, exito)

Solución: si *suma == num*.

Candidatos: *id .. vector.length-1*

Aceptable: $(\text{suma} + \text{vector}[\text{candidato}]) \leq \text{num}$

Anotar: *solucion[candidato] = 1*

Desanotar: *solucion[candidato] = 0*

1.Introducción



33

33

Introducción

Ejemplo. Suma de enteros (árbol combinatorio)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
            }
            c++;
        }
    }
}
```

1.Introducción



34

34

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true); Es solución
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
            }
            c++;
        }
    }
}
```

1. Introducción



35

35

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length-1 Candidatos
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
            }
            c++; c++;
        }
    }
}
```

1. Introducción



36

36

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
            }
            c++;
        }
    }
}
```

Quedan candidatos y no exito

1. Introducción



37

37

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
            }
            c++;
        }
    }
}
```

Candidato aceptable

1. Introducción



38

38

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
                c++;
            }
        }
    }
}
```

anotar

1. Introducción



39

39

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
                c++;
            }
        }
    }
}
```

Avance
en el
espacio
de
búsqueda

1. Introducción



40

40

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
            }
            c++;
        }
    }
}
```

No éxito,
desanotar

1. Introducción



41

41

Introducción

Ejemplo. Suma de enteros (*árbol combinatorio*)

```
void subconjuntoSumaBack3(int[] vector, int num, int[] solucion, int id, int suma,
                           Booleano exito){
    if (suma == num) exito.setValor(true);
    else{
        int c=id; // Candidatos: c = id..vector.length
        while (!exito.getValor() &&(c < vector.length)){
            if ((suma + vector[c])<=num){
                solucion[c] = 1;
                suma = suma + (vector[c]);
                c = c + 1;
                subconjuntoSumaBack3(vector, num, solucion, c, suma, exito);
                if (!exito.getValor()){
                    c = c - 1;
                    solucion[c]=0;
                    suma = suma - (vector[c]);
                }
            }
            c++;
        }
    }
}
```

```
int[] vector = {13, 11, 7};  int num = 20;      Llamada inicial
Booleano exito = new Booleano(false);
int[] solucion = new int[vector.length];
for (int i=0; i<solucion.length; i++) solucion[i]=0;
int suma=0;
subconjuntoSumaBack2(vector, num, solucion, 0, suma, exito);
```

1. Introducción



42

42

Introducción

Algunos problemas planteados

The diagram illustrates the relationship between specific problems and a general solution framework. It features three main components: a blue cloud-like shape containing the text "El viaje del caballero", another blue cloud-like shape containing "Las 8 reinas", and a dark blue oval labeled "Esquema backtracking". A line connects the "El viaje del caballero" cloud to the "Esquema backtracking" oval, and another line connects the "Las 8 reinas" cloud to the same oval.

1. Introducción

43

43

El viaje del caballero

Algunos problemas planteados

This slide is identical in layout and content to the one above it. It shows the "El viaje del caballero" problem, the "Las 8 reinas" problem, and the "Esquema backtracking" scheme, all interconnected by lines.

2. Viaje del caballero

44

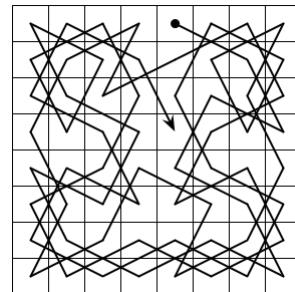
44

El viaje del caballero

Enunciado

- El Problema del viaje del caballero (mal traducido por vuelta del caballo) es un antiguo problema matemático en el que se pide que, teniendo una cuadrícula de $N \times N$ casillas y un caballo de ajedrez colocado en una posición inicial cualquiera (x,y) , el caballo pase por todas las casillas y una sola vez por cada una de ellas.

Simple Knight's Tour
By Dan Thomasson, July 2, 2001
DTHOMASSON@carolina.rr.com



* Este problema es una forma más general del Problema de la ruta Hamiltoniana en la teoría de grafos.

2. Viaje del caballero



45

45

El viaje del caballero

Pasos

Estructura datos solución: tablero NxN.

Nodo del árbol: Añade **1 salto** a la solución.

Parámetros estado: matriz *tablero*, posición actual del caballo (x,y) y el número de movimiento a realizar (*numMov*)

Llamada inicial: viajeCaballero(tablero, numMov:2, x: 0, y: 0, éxito)

Solucion: si *numMov* > (*tablero.length***tablero.length*)

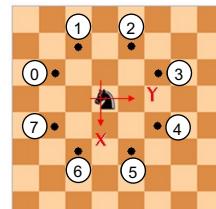
Candidatos: 8 saltos potenciales desde la celda actual (x,y)

```
int [] dx={-1,-2,-2,-1, 1, 2, 2, 1};  
int [] dy={-2,-1, 1, 2, 2, 1,-1,-2};
```

Aceptable: no se sale del tablero y la casilla destino == 0

Anotar: *tablero[xDestino][yDestino]* = *numMov*; *numMov*++

Desanotar: *tablero[xDestino][yDestino]* = 0; *numMov*--;

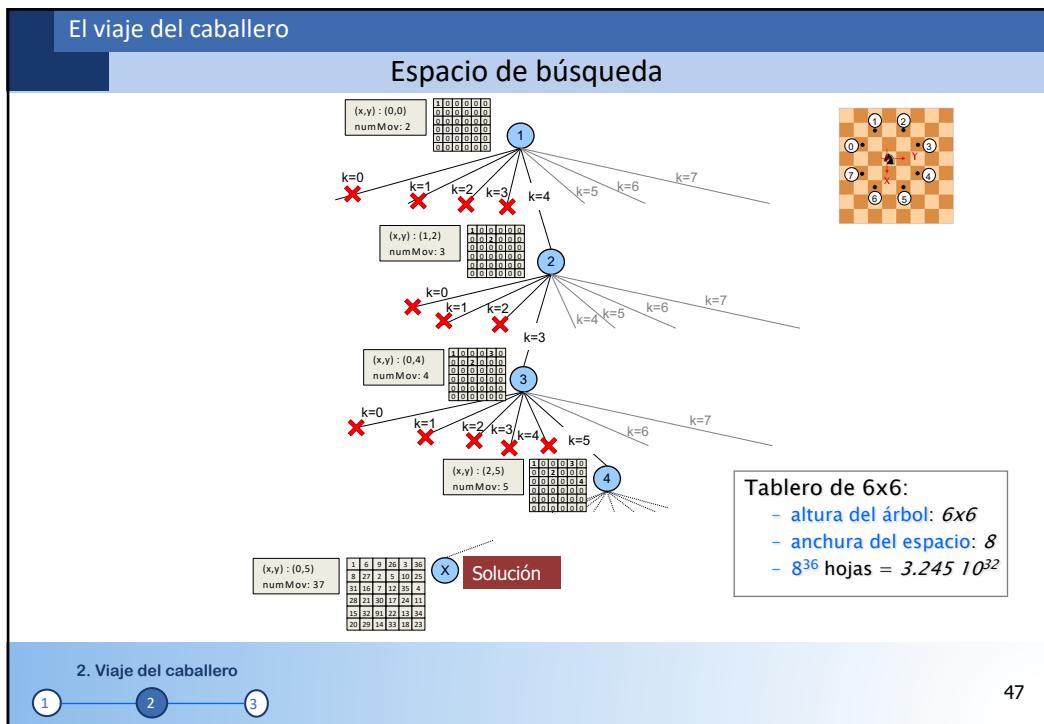


2. Viaje del caballero



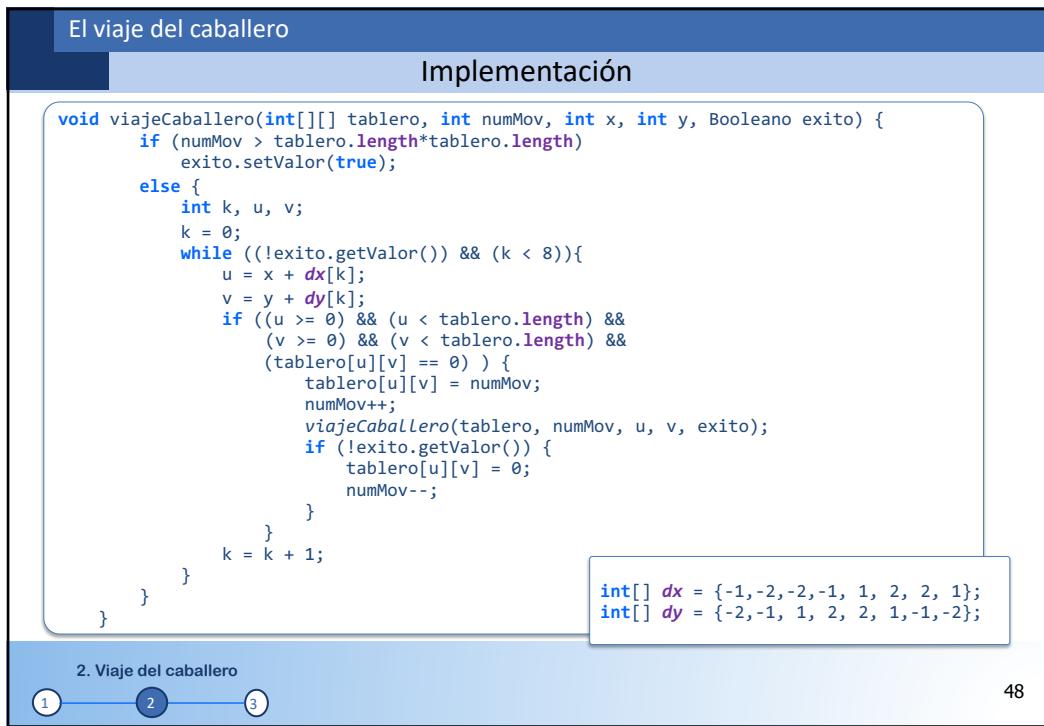
46

46



47

47



48

48

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

Es solución

**int[] dx = {-1,-2,-2,-1, 1, 2, 2, 1};
int[] dy = {-2,-1, 1, 2, 2, 1,-1,-2};**

2. Viaje del caballero

49

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

Candidatos

**int[] dx = {-1,-2,-2,-1, 1, 2, 2, 1};
int[] dy = {-2,-1, 1, 2, 2, 1,-1,-2};**

2. Viaje del caballero

50

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

Quedan candidatos y no exito

`int[] dx = {-1,-2,-2,-1, 1, 2, 2, 1};
int[] dy = {-2,-1, 1, 2, 2, 1,-1,-2};`

2. Viaje del caballero

51

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

Candidato aceptable

`int[] dx = {-1,-2,-2,-1, 1, 2, 2, 1};
int[] dy = {-2,-1, 1, 2, 2, 1,-1,-2};`

2. Viaje del caballero

52

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

anotar

```

int[] dx = {-1,-2,-2,-1, 1, 2, 2, 1};
int[] dy = {-2,-1, 1, 2, 2, 1,-1,-2};

```

2. Viaje del caballero

53

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

Avance en el espacio de búsqueda

```

int[] dx = {-1,-2,-2,-1, 1, 2, 2, 1};
int[] dy = {-2,-1, 1, 2, 2, 1,-1,-2};

```

2. Viaje del caballero

54

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

No éxito,
desanotar

```

int[] dx = {-1,-2,-2,-1, 1, 2, 2, 1};
int[] dy = {-2,-1, 1, 2, 2, 1,-1,-2};

```

2. Viaje del caballero

55

El viaje del caballero

Implementación

```

void viajeCaballero(int[][] tablero, int numMov, int x, int y, Booleano exito) {
    if (numMov > tablero.length*tablero.length)
        exito.setValor(true);
    else {
        int k, u, v;
        k = 0;
        while ((!exito.getValor()) && (k < 8)){
            u = x + dx[k];
            v = y + dy[k];
            if ((u >= 0) && (u < tablero.length) &&
                (v >= 0) && (v < tablero.length) &&
                (tablero[u][v] == 0) ) {
                tablero[u][v] = numMov;
                numMov++;
                viajeCaballero(tablero, numMov, u, v, exito);
                if (!exito.getValor()) {
                    tablero[u][v] = 0;
                    numMov--;
                }
            }
            k = k + 1;
        }
    }
}

```

```

int[][] tablero = new int[6][6];
for (int i=0; i<tablero.length; i++)
    for (int j=0; j< tablero.length; j++)
        tablero[i][j]=0;
tablero[0][0]=1;
Booleano exito = new Booleano(false);
viajeCaballero(tablero, 2, 0, 0, exito);

```

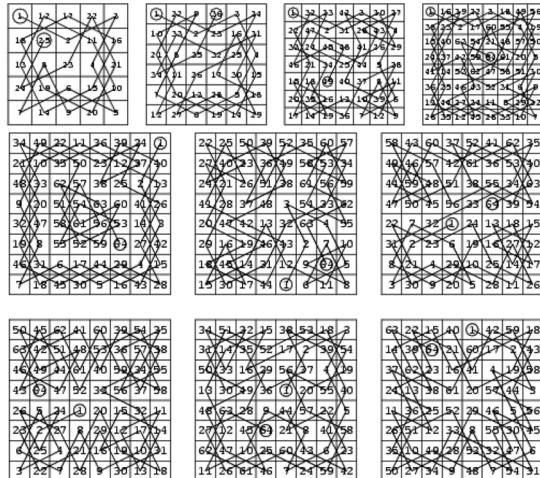
Llamada inicial

2. Viaje del caballero

56

El viaje del caballero

Algunas posibles soluciones



2. Viaje del caballero



57

57

Las 8 reinas

Algunos problemas planteados



3. Las 8 reinas



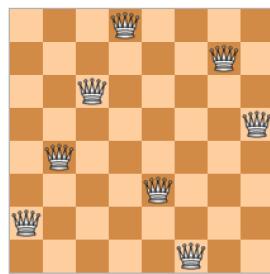
58

58

Las 8 reinas

Enunciado

- El problema consiste en colocar ocho reinas en un tablero de ajedrez sin que se den jaque (dos reinas se dan jaque si comparten fila, columna o diagonal).
- Posibles abordajes con backtracking:
 1. Probar todas las posiciones posibles y para cada una comprobar si es válida. Para tamaño 8: **4.426.165.368 posibilidades.**
 2. Colocar cada reina en cada fila. Una solución será un array de 1..8. Para cada reina se probarán cada una de las 8 columnas. Habrá que probar: $8^8 = 16.777.216$ posibilidades.
 3. No colocar dos reinas en una misma columna. La solución será una permutación de los números (1, 2, ..., 8). El espacio de soluciones se reduce a $8! = 40.320$ posibilidades.



* propuesto por el ajedrecista alemán Max Bezzel en 1848.

3. Las 8 reinas



59

59

Las 8 reinas

Consideraciones

- Solución con vuelta atrás:
 - Puesto que no puede haber más de una reina por fila, podemos replantear el problema como: “**colocar una reina en cada fila del tablero de forma que no se den jaque**”. En este caso, dos reinas se dan jaque si comparten columna o diagonal.
 - La solución puede representarse con una 8-tupla (x_0, \dots, x_7) en la que x_i es la columna en la que se coloca la reina que está en la fila i del tablero.
 - Las posiciones de la misma **diagonal descendente** cumplen que tienen el **mismo valor fila–columna**, mientras que las de la misma **diagonal ascendente** cumplen que tienen el **mismo valor fila + columna**. Así, si tenemos dos reinas colocadas en posiciones (f_1, c_1) y (f_2, c_2) entonces están en la misma diagonal si y solo si cumple que:

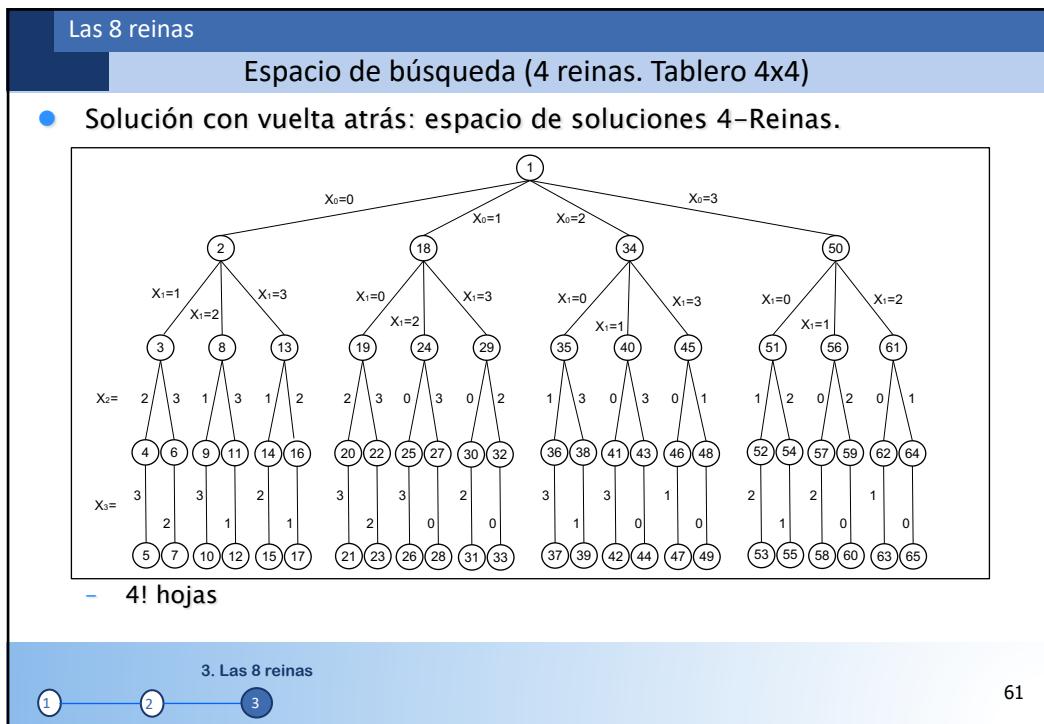
$$\begin{aligned}(f_1 - c_1 = f_2 - c_2) \vee (f_1 + c_1 = f_2 + c_2) \\ \Leftrightarrow (c_1 - c_2 = f_1 - f_2) \vee (c_1 - c_2 = f_2 - f_1) \\ \Leftrightarrow |c_1 - c_2| = |f_1 - f_2|\end{aligned}$$

3. Las 8 reinas

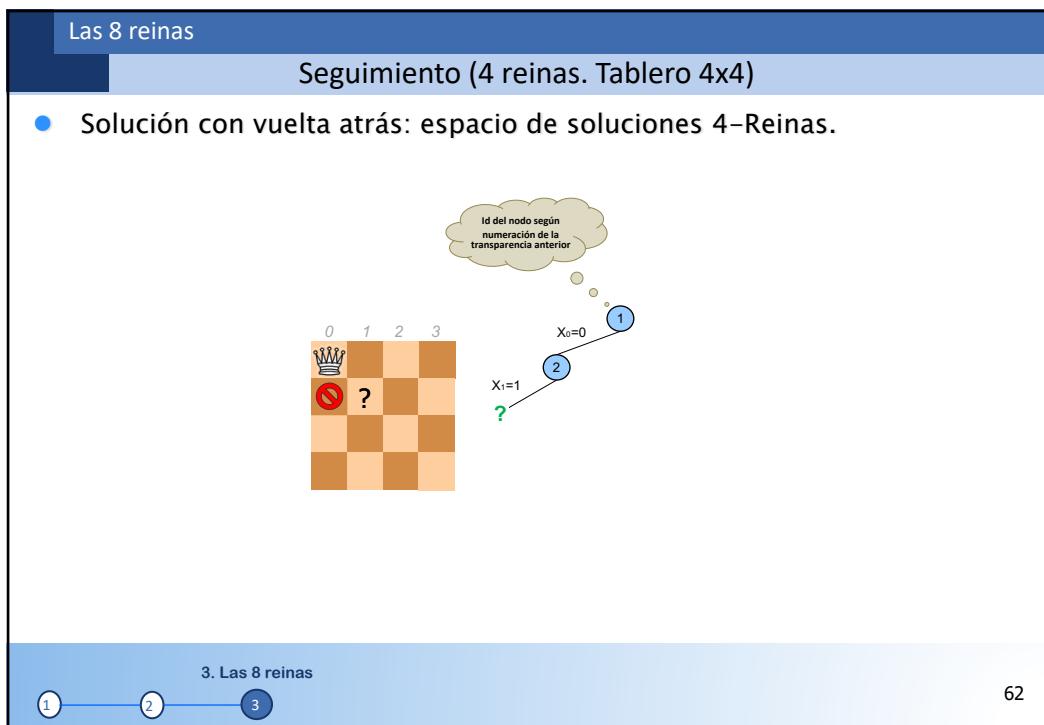


60

60



61

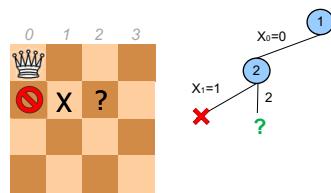


62

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

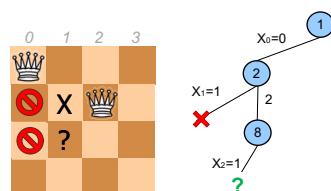
63

63

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

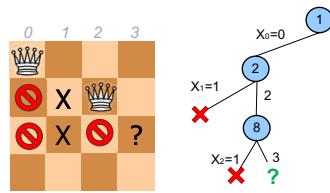
64

64

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

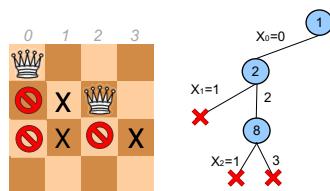
65

65

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

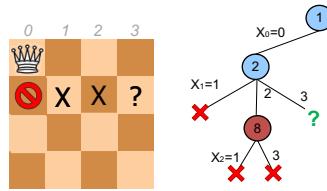
66

66

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

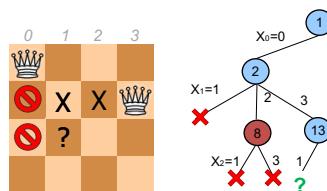
67

67

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

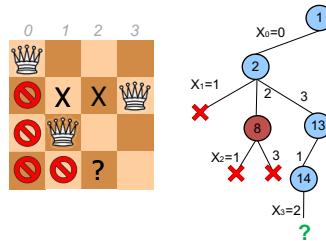
68

68

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

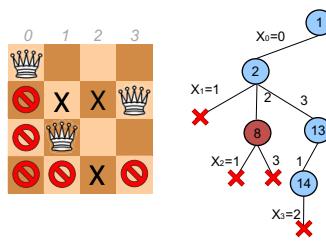
69

69

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

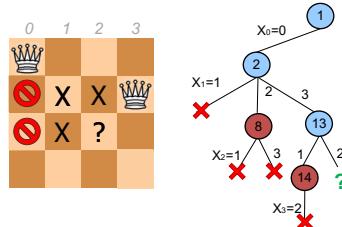
70

70

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

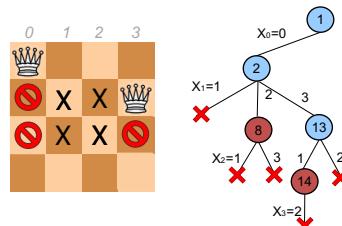
71

71

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

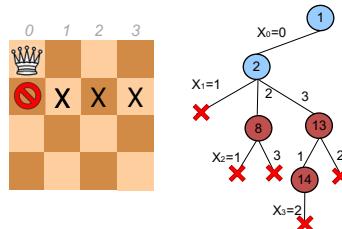
72

72

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

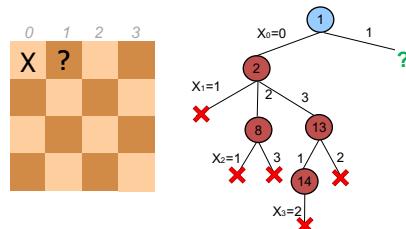
73

73

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

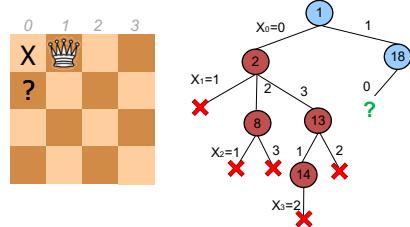
74

74

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

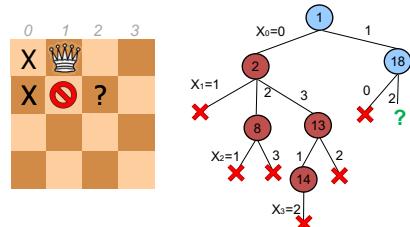
75

75

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

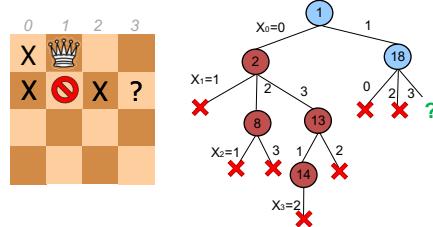
76

76

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

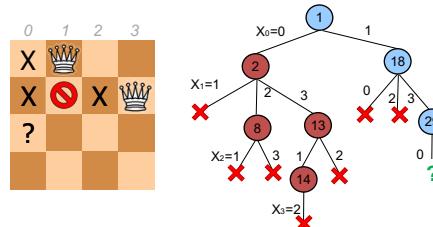
77

77

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

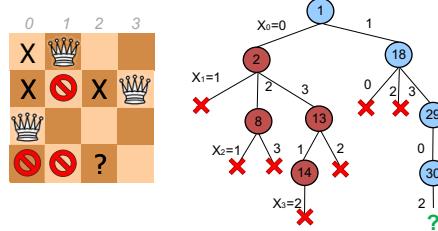
78

78

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.



3. Las 8 reinas

79

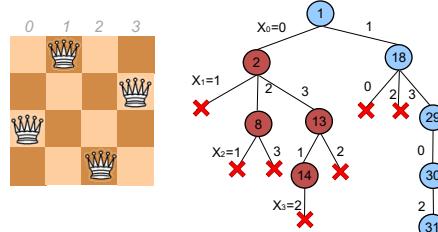
79

Las 8 reinas

Seguimiento (4 reinas. Tablero 4x4)

- Solución con vuelta atrás: espacio de soluciones 4-Reinas.

Solución



3. Las 8 reinas

80

80

Las 8 reinas

Pasos

Estructura datos solución: vector N elementos (*damas*)

1	3	0	2
---	---	---	---

Nodo del árbol: Añade **1 dama** a la solución (*fila actual*).

Parámetros estado: vector *damas* y *fila actual*

Llamada inicial: reinas(damas, fila:0, éxito)

Solución: si *filas == damas.length*

Candidatos: 0..N-1 (columnas en las que colocar la dama de la *fila actual*)

Aceptable: la dama no comparte columna ni diagonal con las damas de las filas previas

Anotar: *damas[fila] = columnaCandidata; fila++*

Desanotar: *fila--;*

3. Las 8 reinas

81

81

Las 8 reinas

Implementación

```

boolean aceptable(int[] damas, int fila, int c){
    boolean correcto = true; int i=0;
    while ((correcto) && (i<fila)) {
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));
        i++;
    }
    return correcto;
}

void reinas(int[] damas, int fila, Booleano exito){
    if (fila==damas.length) exito.setValor(true);
    else {
        int c = 0;
        while ((c < damas.length) && (!exito.getValor())) {
            if (aceptable(damas, fila, c)){
                damas[fila] = c;
                fila=fila + 1;
                reinas(damas, fila, exito);
                if (!exito.getValor()) fila = fila -1;
            }
            c++;
        }
    }
}

```

3. Las 8 reinas

82

82

Las 8 reinas

Implementación

```

boolean aceptable(int[] damas, int fila, int c){
    boolean correcto = true;    int i=0;
    while ((correcto) && (i<fila)) {
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));
        i++;
    }
    return correcto;
}

void reinas(int[] damas, int fila, Booleano exito){
    if (fila==damas.length) exito.setValor(true);
    else {
        int c = 0;
        while ((c < damas.length) && (!exito.getValor())) {
            if (aceptable(damas, fila, c)){
                damas[fila] = c;
                fila=fila + 1;
                reinas(damas, fila, exito);
                if (!exito.getValor()) fila = fila -1;
            }
            c++;
        }
    }
}

```

Es solución

3. Las 8 reinas

83

83

Las 8 reinas

Implementación

```

boolean aceptable(int[] damas, int fila, int c){
    boolean correcto = true;    int i=0;
    while ((correcto) && (i<fila)) {
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));
        i++;
    }
    return correcto;
}

void reinas(int[] damas, int fila, Booleano exito){
    if (fila==damas.length) exito.setValor(true);
    else {
        int c = 0;
        while ((c < damas.length) && (!exito.getValor())) {
            if (aceptable(damas, fila, c)){
                damas[fila] = c;
                fila=fila + 1;
                reinas(damas, fila, exito);
                if (!exito.getValor()) fila = fila -1;
            }
            c++;
        }
    }
}

```

Candidatos

3. Las 8 reinas

84

84

Las 8 reinas

Implementación

```
boolean aceptable(int[] damas, int fila, int c){  
    boolean correcto = true; int i=0;  
    while ((correcto) && (i<fila)) {  
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));  
        i++;  
    }  
    return correcto;  
}  
  
void reinas(int[] damas, int fila, Booleano exito){  
    if (fila==damas.length) exito.setValor(true);  
    else {  
        int c = 0;  
        while ((c < damas.length) && (!exito.getValor())) { Quedan candidatos y no exito  
            if (aceptable(damas, fila, c)){  
                damas[fila] = c;  
                fila=fila + 1;  
                reinas(damas, fila, exito);  
                if (!exito.getValor()) fila = fila -1;  
            }  
            c++;  
        }  
    }  
}
```

3. Las 8 reinas

85

85

Las 8 reinas

Implementación

```
boolean aceptable(int[] damas, int fila, int c){  
    boolean correcto = true; int i=0;  
    while ((correcto) && (i<fila)) {  
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));  
        i++;  
    }  
    return correcto;  
}  
  
void reinas(int[] damas, int fila, Booleano exito){  
    if (fila==damas.length) exito.setValor(true);  
    else {  
        int c = 0;  
        while ((c < damas.length) && (!exito.getValor())) {  
            if (aceptable(damas, fila, c)){  
                damas[fila] = c;  
                fila=fila + 1;  
                reinas(damas, fila, exito);  
                if (!exito.getValor()) fila = fila -1;  
            }  
            c++;  
        }  
    }  
}
```

Candidato aceptable

3. Las 8 reinas

86

86

Las 8 reinas

Implementación

```
boolean aceptable(int[] damas, int fila, int c){  
    boolean correcto = true; int i=0;  
    while ((correcto) && (i<fila)) {  
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));  
        i++;  
    }  
    return correcto;  
}  
  
void reinas(int[] damas, int fila, Booleano exito){  
    if (fila==damas.length) exito.setValor(true);  
    else {  
        int c = 0;  
        while ((c < damas.length) && (!exito.getValor())) {  
            if (aceptable(damas, fila, c)){  
                damas[fila] = c;  
                fila=fila + 1;  
                reinas(damas, fila, exito);  
                if (!exito.getValor()) fila = fila -1;  
            }  
            c++;  
        }  
    }  
}
```

anotar

3. Las 8 reinas

87

87

Las 8 reinas

Implementación

```
boolean aceptable(int[] damas, int fila, int c){  
    boolean correcto = true; int i=0;  
    while ((correcto) && (i<fila)) {  
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));  
        i++;  
    }  
    return correcto;  
}  
  
void reinas(int[] damas, int fila, Booleano exito){  
    if (fila==damas.length) exito.setValor(true);  
    else {  
        int c = 0;  
        while ((c < damas.length) && (!exito.getValor())) {  
            if (aceptable(damas, fila, c)){  
                damas[fila] = c;  
                fila=fila + 1;  
                reinas(damas, fila, exito);  
                if (!exito.getValor()) fila = fila -1;  
            }  
            c++;  
        }  
    }  
}
```

Avance
en el
espacio
de
búsqueda

3. Las 8 reinas

88

88

Las 8 reinas

Implementación

```
boolean aceptable(int[] damas, int fila, int c){  
    boolean correcto = true; int i=0;  
    while ((correcto) && (i<fila)) {  
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));  
        i++;  
    }  
    return correcto;  
}  
  
void reinas(int[] damas, int fila, Booleano exito){  
    if (fila==damas.length) exito.setValor(true);  
    else {  
        int c = 0;  
        while ((c < damas.length) && (!exito.getValor())) {  
            if (aceptable(damas, fila, c)){  
                damas[fila] = c;  
                fila=fila + 1;  
                reinas(damas, fila, exito);  
                if (!exito.getValor()) fila = fila -1;  
            }  
            c++;  
        }  
    }  
}
```

No éxito,
desanotar

3. Las 8 reinas

89

89

Las 8 reinas

Implementación

```
boolean aceptable(int[] damas, int fila, int c){  
    boolean correcto = true; int i=0;  
    while ((correcto) && (i<fila)) {  
        correcto = (damas[i] != c) && (Math.abs(damas[i]-c) != Math.abs(i-fila));  
        i++;  
    }  
    return correcto;  
}  
  
void reinas(int[] damas, int fila, Booleano exito){  
    if (fila==damas.length) exito.setValor(true);  
    else {  
        int c = 0;  
        while ((c < damas.length) && (!exito.getValor())) {  
            if (aceptable(damas, fila, c)){  
                damas[fila] = c;  
                fila=fila + 1;  
                reinas(damas, fila, exito);  
                if (!exito.getValor()) fila = fila -1;  
            }  
            c++;  
        }  
    }  
}
```

Llamada inicial
int[] damas = new int[4];
Booleano exito = new Booleano(false);
reinas(damas, 0, exito);

3. Las 8 reinas

90

90