Algoritmos y Estructuras de Datos II

Backtracking

29 y 31 de mayo de 2017

Clase de hoy

1. Backtracking
   - Forma general de algoritmos voraces
   - ¿Y cuando no hay un buen criterio de selección?
   - Problema de la moneda
   - Problema de la mochila
   - Camino de costo mínimo entre todo par de vértices
2. Conclusiones
3. Backtracking, grafo implícito
4. Ocho reinas

Forma general de algoritmos voraces

fun greedy(C) **ret** S
    {C: conjunto de candidatos, S: solución a construir}
  S:= {}
  do S no es solución $\rightarrow$ c:= seleccionar de C
    C:= C-{c}
    if S$\cup${c} es factible $\rightarrow$ S:= S$\cup${c} **fi**
  od
end fun

- Ser solución y ser factible no tienen en cuenta optimalidad.
- Optimalidad depende totalmente del criterio de selección.

¿Y cuando no hay un buen criterio de selección?

- A veces no hay un criterio de selección que garantice optimalidad.
- Por ejemplo:
  - Problema de la moneda para conjuntos de denominaciones arbitrarios.
  - Problema de la mochila para objetos no fraccionables.
- En este caso, si se elige un fragmento de solución puede ser necesario “volver hacia atrás” (**backtrack**) sobre esa elección e intentar otro fragmento.
- En la práctica, estamos hablando de considerar todas las selecciones posibles e intentar cada una de ellas para saber cuál de ellas conduce a la solución óptima.

Problema de la moneda

- Sean $d_1, d_2, \ldots, d_n$ las denominaciones de las monedas (todas mayores que 0),
- no se asume que estén ordenadas,
- se dispone de una cantidad infinita de monedas de cada denominación,
- se desea pagar un monto k de manera exacta,
- utilizando el **menor número de monedas posibles**.
- Vimos que el algoritmo voraz puede no funcionar para ciertos conjuntos de denominaciones.
- Daremos un algoritmo consistente en considerar todas las combinaciones de monedas posibles.

Simplificación y generalización

- Simplificamos el problema:
  - sólo nos interesa por ahora hallar el menor número de monedas necesario,
  - no nos interesa saber cuáles son esas monedas.
- Generalizamos el problema:
  - Sea *ds* una lista cualquiera de denominaciones y $0 \leq j \leq k$,
  - definimos $cambio(ds, j) =$ “menor número de monedas necesarias para pagar exactamente el monto j con las denominaciones de *ds*.”
  - La solución del problema original se obtiene calculando $cambio([d_1, d_2, \ldots, d_n], k)$.

Definiendo $cambio(ds, j)$

Caso $j = 0$

- Recordemos que $cambio(ds, j) =$ "menor número de monedas necesarias para pagar exactamente el monto j con las denominaciones de ds."
- $cambio(ds, 0) = 0$

Definiendo $cambio(ds, j)$

Caso $j > 0$ y $ds = [\,]$

- Recordemos que $cambio(ds, j) =$ "menor número de monedas necesarias para pagar exactamente el monto j con las denominaciones de ds."
- $cambio(ds, 0) = 0$,
- $j > 0 \Rightarrow cambio([\,], j) = \infty$,
  ya que no hay manera posible de pagar el monto

Definiendo $cambio(ds \triangleleft d, j)$

Caso $d > j > 0$

- Recordemos que $cambio(ds, j) =$ "menor número de monedas necesarias para pagar exactamente el monto j con las denominaciones de ds."
- $cambio(ds, 0) = 0$,
- $j > 0 \Rightarrow cambio([\,], j) = \infty$,
- $d > j > 0 \Rightarrow cambio(ds \triangleleft d, j) = cambio(ds, j)$,
  ya que no se pueden usar monedas de denominación d, es como si no estuvieran disponibles

Definiendo $cambio(ds \lhd d, j)$

Caso $j \geq d$

- Recordemos que $cambio(ds, j) =$ “menor número de monedas necesarias para pagar exactamente el monto j con las denominaciones de ds.”
- $cambio(ds, 0) = 0$,
- $j > 0 \Rightarrow cambio([\,], j) = \infty$,
- $d > j > 0 \Rightarrow cambio(ds \lhd d, j) = cambio(ds, j)$,
- si $j \geq d$ hay dos posibilidades
  - la solución óptima no usa monedas de denominación d
    - $cambio(ds \lhd d, j) = cambio(ds, j)$
  - la solución óptima usa una o más monedas de denominación d
    - $cambio(ds \lhd d, j) = 1 + cambio(ds \lhd d, j - d)$

Definición recursiva (a la Haskell) de $cambio(ds, j)$

Conclusión de estas últimas filminas:

```
cambio ds 0 = 0
cambio [] j = ∞
cambio (ds ◁ d) j | d > j = cambio ds j
                  | otherwise = min (cambio ds j)
                                    (1 + cambio (ds ◁ d) (j - d))
```

Decisión que determina esta definición: ¿usamos monedas de denominación *d* o no?

Definición recursiva de $cambio(i, j)$

Otra notación: definir $cambio(i, j) =$ "menor número de monedas necesarias para pagar exactamente el monto j con denominaciones $d_1, d_2, \ldots, d_i$."

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \wedge i = 0 \\ cambio(i-1, j) & d_i > j > 0 \wedge i > 0 \\ \min(cambio(i-1,j), 1 + cambio(i, j-d_i)) & j \geq d_i > 0 \wedge i > 0 \end{cases}$$

Decisión que determina esta definición: ¿usamos monedas de denominación d_i o no?

Otras posibles definiciones que usan backtracking

Considerando el número exacto de monedas de denominación d_i

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \wedge i = 0 \\ \min_{q \in \{0,1,\ldots,j \div d_i\}} (q + cambio(i-1, j - q * d_i)) & j > 0 \wedge i > 0 \end{cases}$$

Acá estamos considerando la posibilidad de usar 0 monedas ($q = 0$) de denominación d_i, 1 moneda ($q = 1$) de denominación d_i, etc. De todas esas posibilidades se elige la que minimice el número total de monedas.

Decisión que determina esta definición: ¿cuántas monedas de denominación d_i usamos?

Otras posibles definiciones que usan backtracking

Considerando cuál moneda de las disponibles se usa

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ 1 + \min_{i' \in \{1,2,\ldots,i \mid d_{i'} \leq j\}}(cambio(i', j - d_{i'})) & j > 0 \end{cases}$$

Acá estamos considerando la posibilidad de usar 1 moneda de denominación d_i ($i' = i$), 1 moneda de denominación d_{i-1} ($i' = i - 1$), etc. De todas esas posibilidades se elige la que minimice el número total de monedas. Para evitar cálculos repetidos, se restringe la búsqueda a monedas de índice menor o igual a los ya utilizados.

Decisión que determina esta definición: de las monedas de que disponemos, ¿cuál usamos?

Primera definición recursiva en pseudocódigo

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \land i = 0 \\ cambio(i-1,j) & d_i > j > 0 \land i > 0 \\ \min(cambio(i-1,j), 1 + cambio(i, j-d_i)) & j \geq d_i > 0 \land i > 0 \end{cases}$$

fun cambio(d:**array**[1..n] **of nat**, i,j: **nat**) **ret** r: **nat**
  if j=0 **then** r:= 0
  else if i = 0 **then** r:= ∞
  else if d[i] > j **then** r:= cambio(d,i-1,j)
  else r:= min(cambio(d,i-1,j),1+cambio(d,i,j-d[i]))
  fi
end fun

Segunda definición recursiva en pseudocódigo

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \wedge i = 0 \\ \min_{q \in \{0,1,\ldots,j \div d_i\}}(q + cambio(i-1, j - q * d_i)) & j > 0 \wedge i > 0 \end{cases}$$

```
fun cambio(d:array[1..n] of nat, i,j: nat) ret r: nat
    if j=0 then r:= 0
    else if i = 0 then r:= ∞
    else r:= cambio(d,i-1,j)
         for q:= 1 to j ÷ d[i] do
             r:= min(r,q+cambio(d,i-1,j-q*d[i]))
         od
    fi
end fun
```

Tercera definición recursiva en pseudocódigo

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ 1 + \min_{i' \in \{1,2,\ldots,i \mid d_{i'} \leq j\}}(cambio(i', j - d_{i'})) & j > 0 \end{cases}$$

fun cambio(d:**array**[1..n] **of nat**, i,j: **nat**) **ret** r: **nat**
    if j=0 **then** r:= 0
    else r:= ∞
        for i':= 1 **to** i **do**
            if d[i'] $\leq$ j **then** r:= min(r,cambio(d,i',j-d[i'])) **fi**
        od
        r:= r + 1
    fi
end fun

Otras posibilidades

- No son éstas las únicas formas de resolver el problema usando backtracking.
- Podríamos definir, por ejemplo, a la Haskell

$$
\begin{aligned}
&\text{cambio ds } 0 = 0\\
&\text{cambio [] j} = \infty\\
&\text{cambio } (d \triangleright ds)\ j \mid d > j = \text{cambio ds j}\\
&\qquad\qquad\qquad\qquad \mid \text{otherwise} = \min\ (\text{cambio ds j})\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad (1 + \text{cambio } (d \triangleright ds)\ (j - d))
\end{aligned}
$$

Otras posibilidades

- Se correspondería con
  - $cambio(i,j) =$ "menor número de monedas necesarias para pagar exactamente el monto j con denominaciones $d_{i+1}, d_{i+2}, \ldots, d_n$."
- Obtendríamos, entre otras posibles definiciones recursivas,

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \wedge i = n \\ cambio(i+1,j) & d_i > j > 0 \wedge i < n \\ \min(cambio(i+1,j), 1 + cambio(i, j - d_i)) & j \geq d_i > 0 \wedge i < n \end{cases}$$

- Para resolver el problema original se calcula $cambio(0,k)$.

Primera solución, pero ¡Queremos las monedas!

fun cambio(d:**array**[1..n] **of nat**, i,j: **nat**) **ret** r: **list of nat**
  var r_1, r_2 : **list of nat**
  if j=0 **then** r:= [ ]
  else if i = 0 **then** r:= una lista infinita o muy larga
  else if d[i] > j **then** r:= cambio(d,i-1,j)
  else r_1 := cambio(d,i-1,j)
    r_2 := cambio(d,i,j-d[i]) $\lhd$ d[i]
    if $|r_1| \leq |r_2|$ **then** r:= r_1 **else** r:= r_2 **fi**
  fi
end fun

donde |x| es la longitud de x,
y la llamada principal es cambio(d,n,k).

Problema de la mochila

- Tenemos una mochila de capacidad W.
- Tenemos n objetos **no fraccionables** de valor $v_1, v_2, \ldots, v_n$ y peso $w_1, w_2, \ldots, w_n$.
- Se quiere encontrar la mejor selección de objetos para llevar en la mochila.
- Por mejor selección se entiende aquélla que totaliza **el mayor valor posible** sin que su peso exceda la capacidad W de la mochila.

Simplificación y generalización

- Simplificamos el problema:
  - sólo nos interesa por ahora hallar el mayor valor posible sin exceder la capacidad de la mochila,
  - no nos interesa saber cuáles son los objetos que alcanzan ese máximo.
- Generalizamos el problema:
  - Sea *os* una lista cualquiera de pares (valor,peso) y $0 \leq j \leq W$,
  - definimos $mochila(os, j) =$ "mayor valor alcanzable sin exceder la capacidad *j* con objetos de *os*."
  - La solución del problema original se obtiene calculando $mochila([(v_1, w_1), (v_2, w2), \ldots, (v_n, w_n)], W)$.

Definición recursiva (a la Haskell) de $mochila(os, j)$

```
mochila os 0 = 0
mochila [] j = 0
mochila (os ◁ (v,w)) j | w > j = mochila os j
                       | otherwise = max (mochila os j)
                                         (v + mochila os (j - w))
```

Decisión que determina esta definición: ¿colocamos o no el objeto de (valor,peso) = (v, w) en la mochila?

Definición recursiva de *mochila*(i, j)

Otra notación: definir *mochila*$(i, j) =$ “mayor valor alcanzable sin exceder la capacidad j con objetos $1, 2, \ldots, i$.”

$$mochila(i, j) = \begin{cases} 0 & j = 0 \\ 0 & j > 0 \wedge i = 0 \\ mochila(i-1, j) & w_i > j > 0 \wedge i \\ \max(mochila(i-1, j), v_i + mochila(i-1, j-w_i)) & j \geq w_i > 0 \wedge i \end{cases}$$

Decisión que determina esta definición: ¿colocamos o no el objeto i en la mochila?

Otras posibilidades

- Ofrece las mismas variantes que en el problema de la moneda,
- el pasaje a pseudocódigo es similar,
- la incorporación de información con los objetos que van en la mochila es también parecido.

Problema del camino de costo mínimo
Entre todo par de vértices

- Tenemos un grafo dirigido $G = (V, A)$,
- con costos no negativos en las aristas,
- se quiere encontrar, para cada par de vértices, el camino de menor costo que los une.
- Se asume $V = \{1, \ldots, n\}$

Simplificación y generalización

- Simplificamos el problema:
  - sólo nos interesa por ahora hallar el costo de cada uno de los caminos de costo mínimo.
  - no nos interesa saber cuáles son los caminos que alcanzan ese mínimo.
- Generalizamos el problema:
  - Sean $1 \leq i, j \leq n$ y $0 \leq k \leq n$,
  - definimos $camino_k(i, j) =$ "menor costo posible para caminos de i a j cuyos vértices intermedios se encuentran en el conjunto $\{1,\ldots,k\}$."
  - La solución del problema original se obtiene calculando $camino_n(i, j)$ para el par i (origen) y j (destino) que se desea.

Definición recursiva de $camino_k(i,j)$

$$camino_k(i,j) = \begin{cases} L[i,j] & k = 0 \\ \min(camino_{k-1}(i,j), camino_{k-1}(i,k) + camino_{k-1}(k,j)) & k \geq 1 \end{cases}$$

donde $L[i,j]$ es el costo de la arista que va de i a j, o infinito si no hay tal arista.
Decisión que determina esta definición: ¿pasamos por el vértice k o no?

Conclusiones

- Hemos visto soluciones a tres problemas.
- En general, muy ineficiente.
- Por ejemplo, para el problema de la moneda, si queremos pagar el monto 90 con nuestros billetes con denominaciones 1, 5 y 10,
  - cambio(3,90) llama a cambio(2,90) y cambio(3,80),
  - cambio(2,90) llama a cambio(1,90) y cambio(2,85),
  - cambio(2,85) llama a cambio(1,85) y **cambio(2,80)**,
  - cambio(3,80) llama a **cambio(2,80)** y cambio(3,70).
- Se ve que cambio(2,80) se calcula 2 veces.
- y muchos otros llamados se repiten, incluso varias veces.
- Esto vuelve los algoritmos exponenciales en el peor caso.

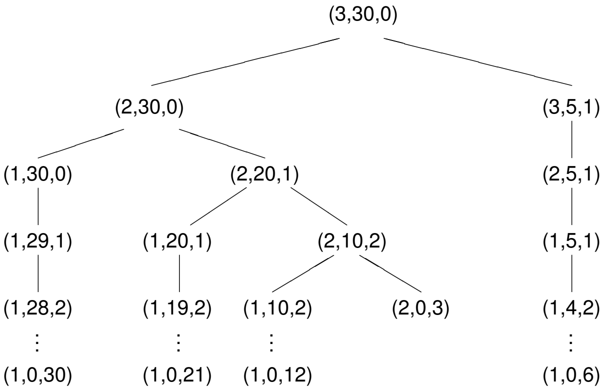
Problema de la moneda
Primera solución que usa backtracking

Recordemos la primera solución al problema de la moneda usando backtracking:

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \wedge i = 0 \\ cambio(i-1,j) & d_i > j > 0 \wedge i > 0 \\ \min(cambio(i-1,j), 1 + cambio(i, j-d_i)) & j \geq d_i > 0 \wedge i > 0 \end{cases}$$

Grafo implícito

Ejemplo $d_1 = 1$, $d_2 = 10$, $d_3 = 25$ y $k = 30$



Grafo implícito
Definición general

- Desde el vértice (i, j, x), si $i, j > 0$ y $d_i < j$ existe una única arista al vértice $(i - 1, j, x)$.
- En cambio si $j \leq d_i$ existen dos aristas:
  - una a $(i - 1, j, x)$
  - y otra a $(i, j - d_i, x + 1)$.
- la raíz es el vértice $(n, k, 0)$.

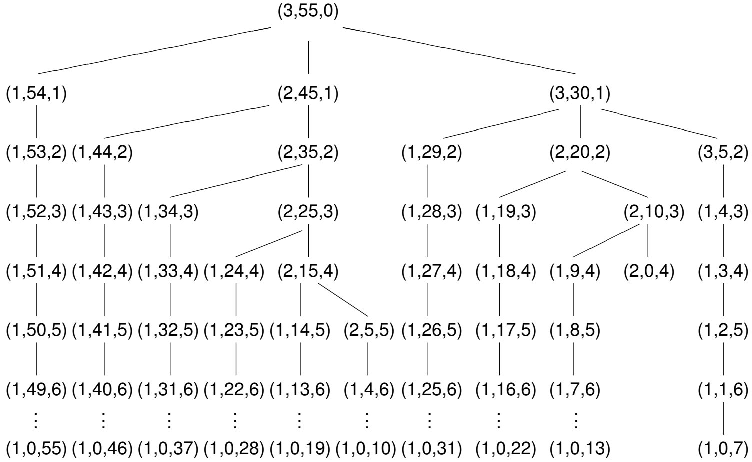
Problema de la moneda
Tercera solución que usa backtracking

Recordemos otra solución al problema de la moneda usando backtracking:

$$cambio(i,j) = \begin{cases} 0 & j = 0 \\ 1 + \min_{i' \in \{1,2,\ldots,i \mid d_{i'} \leq j\}}(cambio(i', j - d_{i'})) & j > 0 \end{cases}$$

Grafo implícito

Ejemplo $d_1 = 1$, $d_2 = 10$, $d_3 = 25$ y $k = 55$



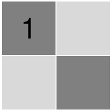
Grafo implícito
Definición general

- La raíz resulta la misma que en el caso anterior,
- pero el vértice (i, j, x) puede tener 0, 1, o varios hijos:
  - todos los vértices de la forma $(i', j - d_{i'}, 1 + x)$ tal que $1 \leq i' \leq i$ y $d_{i'} \leq j$,
  - son hijos de (i, j, x).

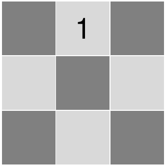
Ocho reinas

- Problema: Encontrar la manera de ubicar 8 reinas en un tablero de 8 filas por 8 columnas de manera tal que ningún par de reinas ocupe la misma fila, la misma columna o la misma diagonal.
- para los que saben ajedrez: de modo de que ninguna reina amenace a otra.
- Es un ejemplo típico de problema que se resuelve usando backtracking, a pesar de no ser de optimización.
- Generalización: ubicar n reinas en un tablero de n filas por n columnas de manera tal que ningún par de reinas ocupe la misma fila, la misma columna o la misma diagonal.
  - 0 reinas, tiene una solución (0 reinas en tablero de 0×0).
  - 1 reina, también (1 reina en tablero de 1×1).

Dos reinas



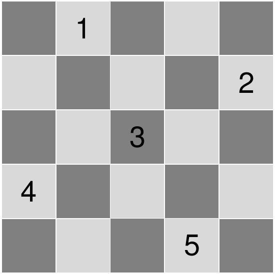
Tres reinas



Cuatro reinas

	1		
			2
3			
		4	

Cinco reinas

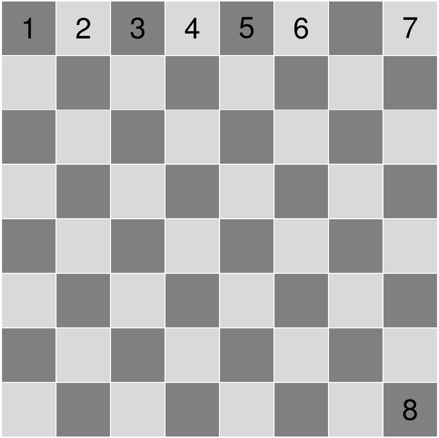


Resumiendo

n reinas:

- $n = 0$ tiene una solución
- $n = 1$ tiene una solución
- $n = 2$ no tiene solución
- $n = 3$ no tiene solución
- $n = 4$ tiene solución
- $n = 5$ varias soluciones
- $n \geq 4$ siempre tiene solución

Ocho reinas, peor algoritmo posible



Ocho reinas, peor algoritmo posible

El algoritmo

Calcula el número de maneras de ubicar 8 reinas sin que se amenacen.

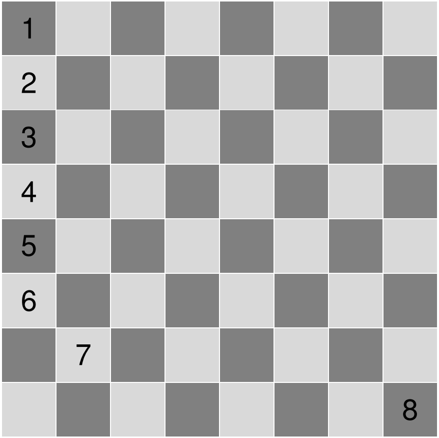
```
fun ocho_reinas_1() ret r: nat
    r:= 0
    for i1:= 1 to 57 do
        for i2:= i1+1 to 58 do
            ...
                for i8:= i7+1 to 64 do
                    if solucion_1([i1,i2,i3,i4,i5,i6,i7,i8]) then r:= r+1 fi
                od
            ...
        od
    od
```

Ocho reinas, peor algoritmo posible
El grafo implícito

$$\begin{aligned} V \;=\; & \{[p_1, p_2, \ldots, p_n] \in \{1, \ldots, 64\}^* \mid \\ & n \leq 8 \wedge p_1 < p_2 < \ldots < p_n \leq 56 + n\} \end{aligned}$$

Dados $p = [p_1, p_2, \ldots, p_n] \in V$ y $q = [q_1, q_2, \ldots, q_m] \in V$ hay una arista de p a q sii $m = n + 1$ y $p_i = q_i$ para todo $1 \leq i \leq n$.

Ocho reinas, un algoritmo menos malo



Ocho reinas, un algoritmo menos malo

El algoritmo

Calcula el número de maneras de ubicar 8 reinas sin que se amenacen.

```
fun ocho_reinas_2() ret r: nat
    r:= 0
    for j1:= 1 to 8 do
        for j2:= 1 to 8 do
            ...
                for j8:= 1 to 8 do
                    if solucion_2([j1,j2,j3,j4,j5,j6,j7,j8]) then r:= r+1 fi
                od
            ...
        od
    od
```

Ocho reinas, un algoritmo menos malo

El grafo implícito

$$V = \{p \in \{1, \ldots, 8\}^* \mid |p| \leq 8\}$$

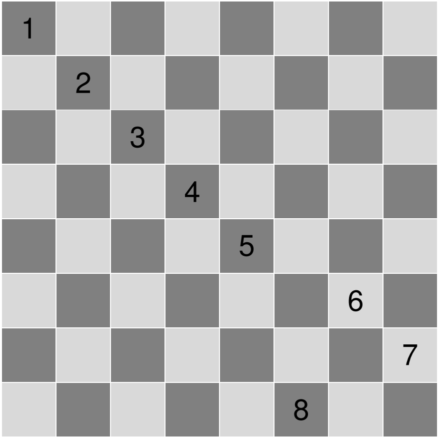
Y las aristas se definen como antes.

Ocho reinas, versión recursiva

```
fun ocho_reinas_2() ret r: nat
    r:= 0
    or_2([ ], r)
end

proc or_2(in sol: list of nat, in/out r: nat)
                {calcula el número de maneras de extender sol}
            {hasta ubicar en total 8 reinas sin que se amenacen}
    if |sol| = 8 then
      if solucion_2(sol) then r:= r+1 fi
    else for j:= 1 to 8 do
                or_2(sol ◁ j, r)
          od
    fi
```

Ocho reinas, un algoritmo mejor



Ocho reinas, un algoritmo mejor
El algoritmo

```
fun ocho_reinas_3() ret r: nat
    r:= 0
    or_3([ ], r)
end

proc or_3(in sol: list of nat, in/out r: nat)
                        {calcula el número de maneras de extender sol}
                {hasta ubicar en total 8 reinas sin que se amenacen}
  if |sol| = 8 then
     if solucion_3(sol) then r:= r+1 fi
  else for j:= 1 to 8 do
             if j ∉ sol then or_3(sol ◁ j, r) fi
       od
```

Ocho reinas, un algoritmo mejor
El grafo implícito

$$V = \{p \in \{1, \ldots, 8\}^* \mid |p| \leq 8 \wedge p \text{ sin repeticiones}\}$$

Y las aristas se definen como antes.

Ocho reinas, un algoritmo optimizado

1							
				2			
							3
					4		
		5					
						6	
	7						
			8				

Ocho reinas, todas las soluciones

Q							
				Q			
							Q
					Q		
		Q					
						Q	
	Q						
			Q				

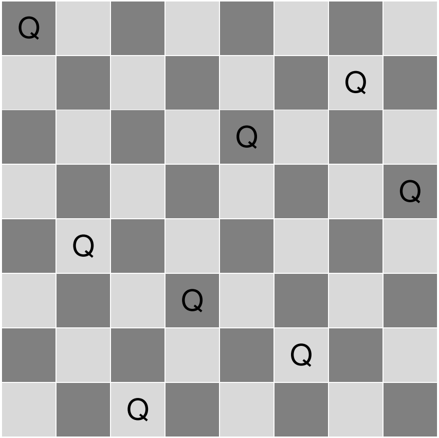
Ocho reinas, todas las soluciones

Q							
					Q		
							Q
		Q					
						Q	
			Q				
	Q						
				Q			

Ocho reinas, todas las soluciones

Q							
						Q	
			Q				
					Q		
							Q
	Q						
				Q			
		Q					

Ocho reinas, todas las soluciones



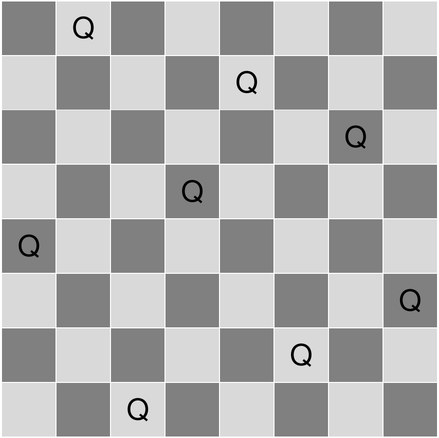
Ocho reinas, todas las soluciones

	Q						
			Q				
					Q		
							Q
		Q					
Q							
						Q	
				Q			

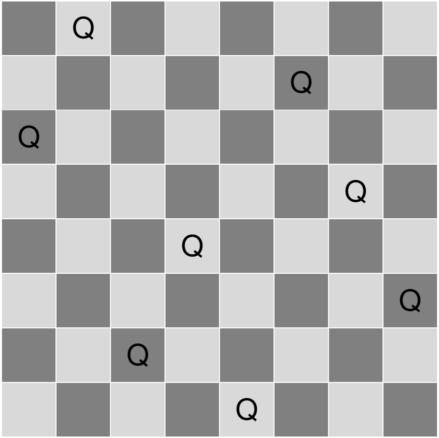
Ocho reinas, todas las soluciones

	Q						
				Q			
						Q	
Q							
		Q					
							Q
					Q		
			Q				

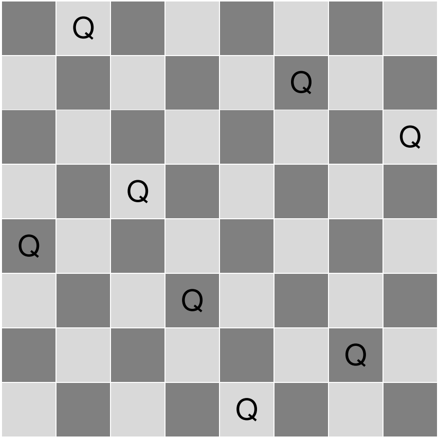
Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

	Q						
						Q	
		Q					
					Q		
							Q
				Q			
Q							
			Q				

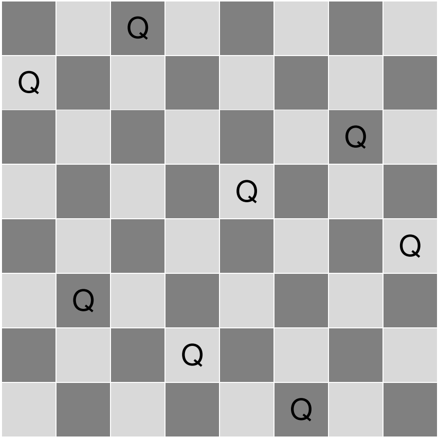
Ocho reinas, todas las soluciones

	Q						
						Q	
				Q			
							Q
Q							
			Q				
					Q		
		Q					

Ocho reinas, todas las soluciones

	Q						
							Q
					Q		
Q							
		Q					
				Q			
						Q	
			Q				

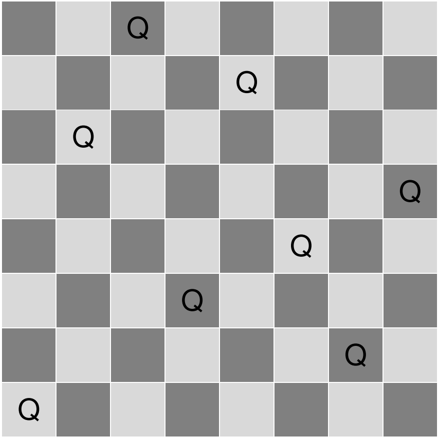
Ocho reinas, todas las soluciones



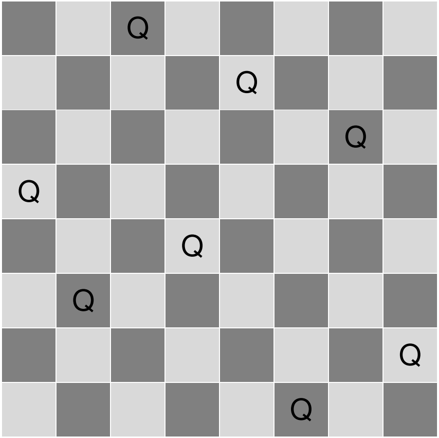
Ocho reinas, todas las soluciones

		Q					
				Q			
	Q						
							Q
Q							
						Q	
			Q				
					Q		

Ocho reinas, todas las soluciones



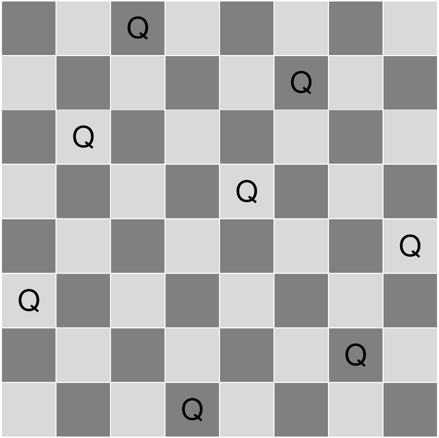
Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

		Q					
				Q			
							Q
			Q				
Q							
						Q	
	Q						
					Q		

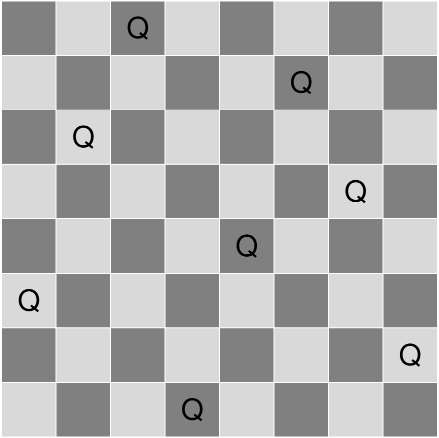
Ocho reinas, todas las soluciones



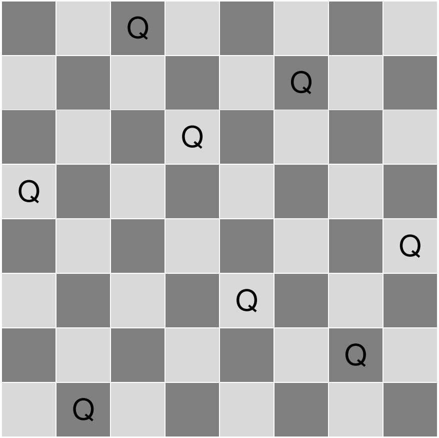
Ocho reinas, todas las soluciones

		Q					
					Q		
	Q						
						Q	
Q							
			Q				
							Q
				Q			

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones



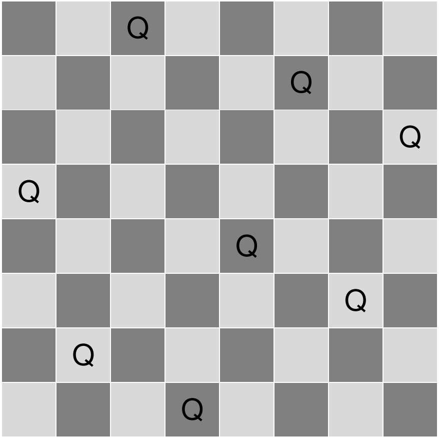
Ocho reinas, todas las soluciones

		Q					
					Q		
			Q				
	Q						
							Q
				Q			
						Q	
Q							

Ocho reinas, todas las soluciones

		Q					
					Q		
							Q
Q							
			Q				
						Q	
				Q			
	Q						

Ocho reinas, todas las soluciones



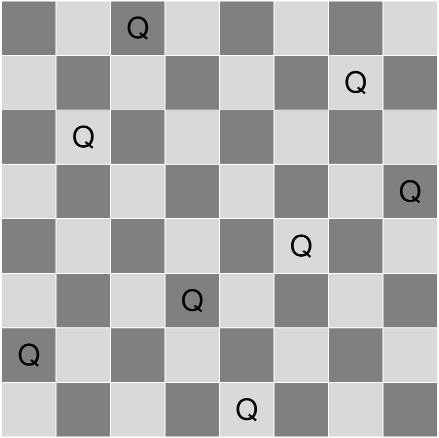
Ocho reinas, todas las soluciones

		Q					
					Q		
							Q
	Q						
			Q				
Q							
						Q	
				Q			

Ocho reinas, todas las soluciones

		Q					
						Q	
	Q						
							Q
				Q			
Q							
			Q				
					Q		

Ocho reinas, todas las soluciones



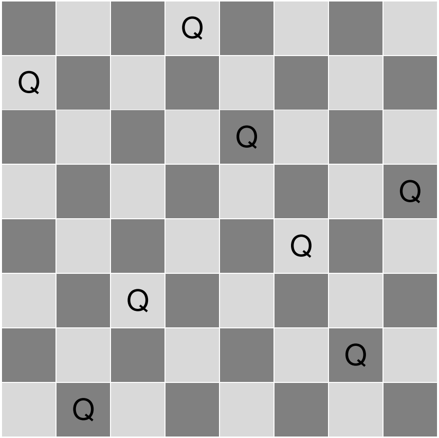
Ocho reinas, todas las soluciones

		Q					
							Q
			Q				
						Q	
Q							
					Q		
	Q						
				Q			

Ocho reinas, todas las soluciones

			Q				
Q							
				Q			
							Q
	Q						
						Q	
		Q					
					Q		

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

			Q				
	Q						
				Q			
							Q
					Q		
Q							
		Q					
						Q	

Ocho reinas, todas las soluciones

			Q				
	Q						
						Q	
		Q					
					Q		
							Q
Q							
				Q			

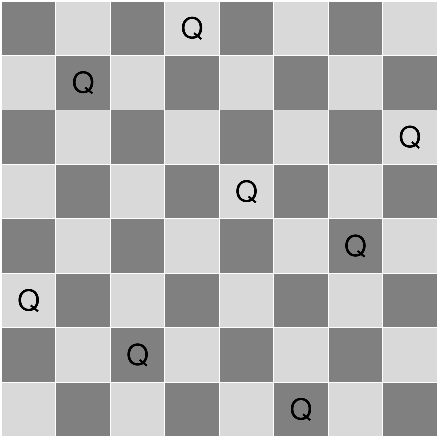
Ocho reinas, todas las soluciones

			Q				
	Q						
						Q	
		Q					
					Q		
							Q
				Q			
Q							

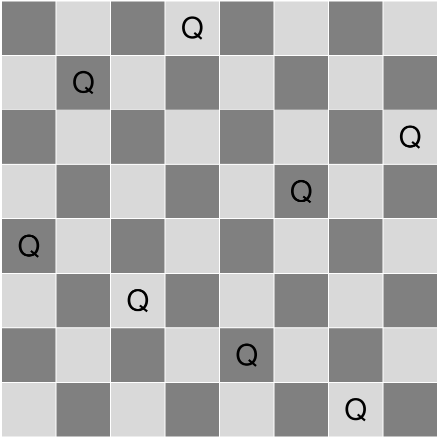
Ocho reinas, todas las soluciones

			Q				
	Q						
						Q	
				Q			
Q							
							Q
					Q		
		Q					

Ocho reinas, todas las soluciones



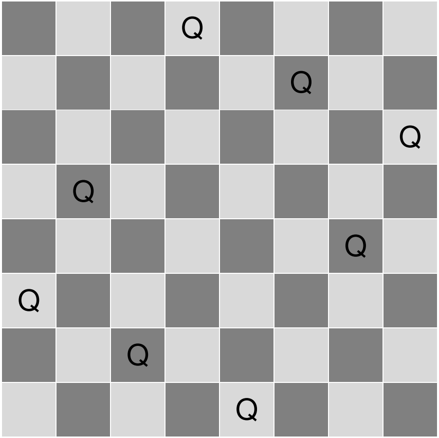
Ocho reinas, todas las soluciones



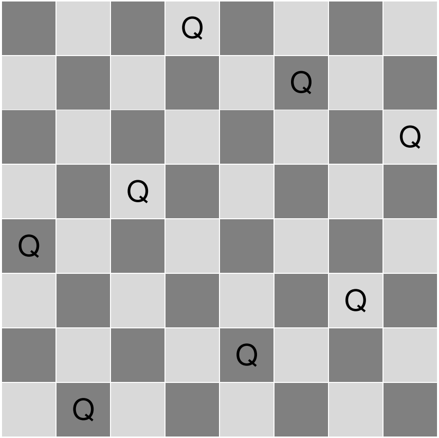
Ocho reinas, todas las soluciones

			Q				
					Q		
Q							
				Q			
	Q						
							Q
		Q					
						Q	

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones



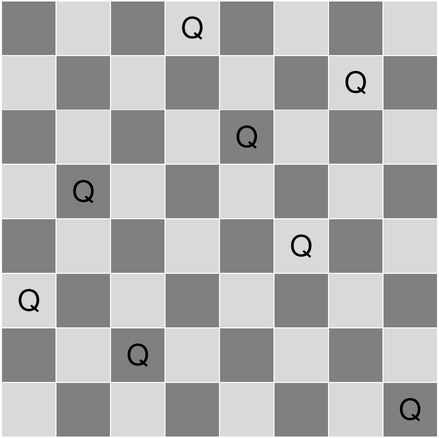
Ocho reinas, todas las soluciones

			Q				
						Q	
Q							
							Q
				Q			
	Q						
					Q		
		Q					

Ocho reinas, todas las soluciones

			Q				
						Q	
		Q					
							Q
	Q						
				Q			
Q							
					Q		

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

			Q				
						Q	
				Q			
		Q					
Q							
					Q		
							Q
	Q						

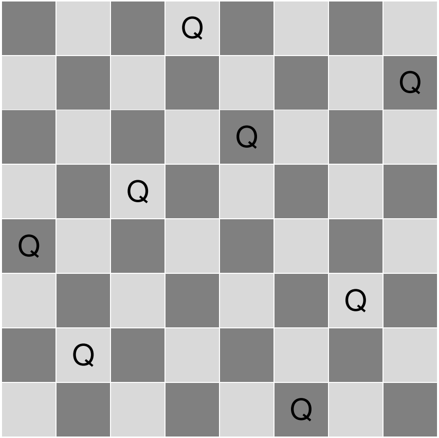
Ocho reinas, todas las soluciones

			Q				
							Q
Q							
		Q					
					Q		
	Q						
						Q	
				Q			

Ocho reinas, todas las soluciones

			Q				
							Q
Q							
				Q			
						Q	
	Q						
					Q		
		Q					

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

				Q			
Q							
			Q				
					Q		
							Q
	Q						
						Q	
		Q					

Ocho reinas, todas las soluciones

				Q			
Q							
							Q
			Q				
	Q						
						Q	
		Q					
					Q		

Ocho reinas, todas las soluciones

				Q			
Q							
							Q
					Q		
		Q					
						Q	
	Q						
			Q				

Ocho reinas, todas las soluciones

				Q			
	Q						
			Q				
					Q		
							Q
		Q					
Q							
						Q	

Ocho reinas, todas las soluciones

				Q			
	Q						
			Q				
						Q	
		Q					
							Q
					Q		
Q							

Ocho reinas, todas las soluciones

				Q			
	Q						
					Q		
Q							
						Q	
			Q				
							Q
		Q					

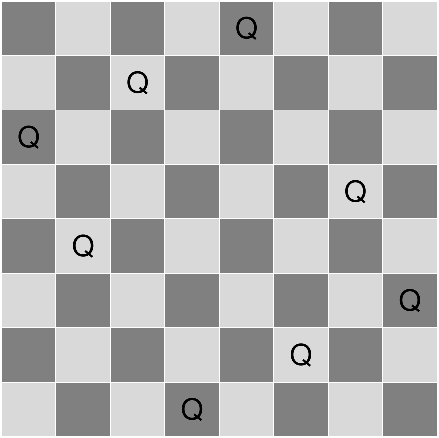
Ocho reinas, todas las soluciones

				Q			
	Q						
							Q
Q							
			Q				
						Q	
		Q					
					Q		

Ocho reinas, todas las soluciones

				Q			
		Q					
Q							
					Q		
							Q
	Q						
			Q				
						Q	

Ocho reinas, todas las soluciones



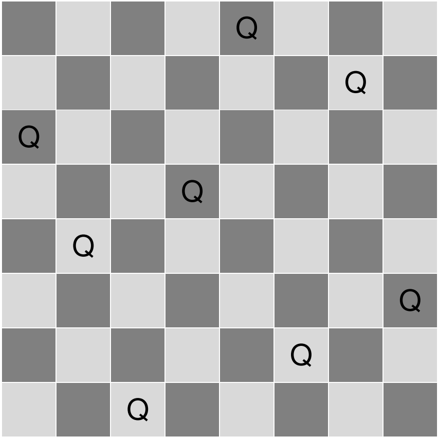
Ocho reinas, todas las soluciones

				Q			
		Q					
							Q
			Q				
						Q	
Q							
					Q		
	Q						

Ocho reinas, todas las soluciones

				Q			
						Q	
Q							
		Q					
							Q
					Q		
			Q				
	Q						

Ocho reinas, todas las soluciones



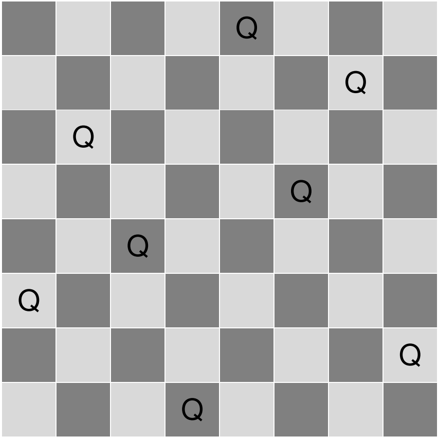
Ocho reinas, todas las soluciones

				Q			
						Q	
	Q						
			Q				
							Q
Q							
		Q					
					Q		

Ocho reinas, todas las soluciones

				Q			
						Q	
	Q						
					Q		
		Q					
Q							
			Q				
							Q

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

				Q			
						Q	
			Q				
Q							
		Q					
							Q
					Q		
	Q						

Ocho reinas, todas las soluciones

				Q			
							Q
			Q				
Q							
		Q					
					Q		
	Q						
						Q	

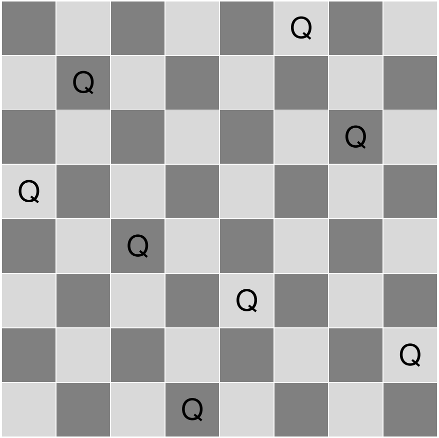
Ocho reinas, todas las soluciones

				Q			
							Q
			Q				
Q							
						Q	
	Q						
					Q		
		Q					

Ocho reinas, todas las soluciones

					Q		
Q							
				Q			
	Q						
							Q
		Q					
						Q	
			Q				

Ocho reinas, todas las soluciones



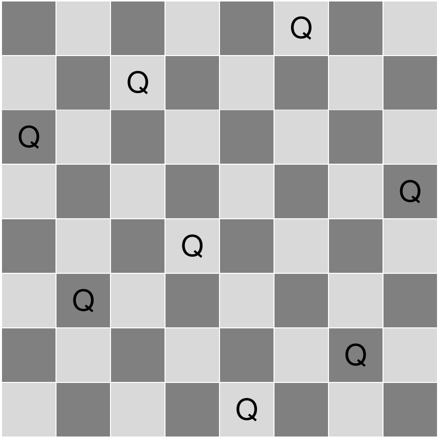
Ocho reinas, todas las soluciones

					Q		
	Q						
						Q	
Q							
			Q				
							Q
				Q			
		Q					

Ocho reinas, todas las soluciones

					Q		
		Q					
Q							
						Q	
				Q			
							Q
	Q						
			Q				

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

					Q		
		Q					
Q							
							Q
				Q			
	Q						
			Q				
						Q	

Ocho reinas, todas las soluciones

					Q		
		Q					
				Q			
						Q	
Q							
			Q				
	Q						
							Q

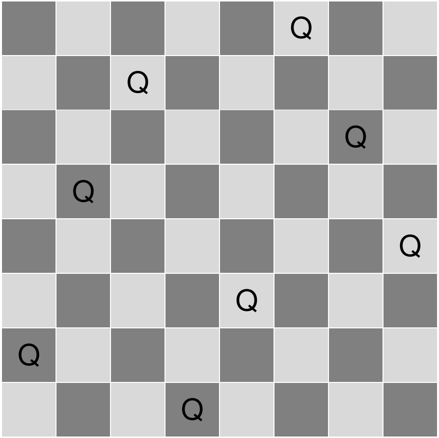
Ocho reinas, todas las soluciones

					Q		
		Q					
				Q			
							Q
Q							
			Q				
	Q						
						Q	

Ocho reinas, todas las soluciones

					Q		
		Q					
						Q	
	Q						
			Q				
							Q
Q							
				Q			

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

					Q		
		Q					
						Q	
			Q				
Q							
							Q
	Q						
				Q			

Ocho reinas, todas las soluciones

					Q		
			Q				
Q							
				Q			
							Q
	Q						
						Q	
		Q					

Ocho reinas, todas las soluciones

					Q		
			Q				
	Q						
							Q
				Q			
						Q	
Q							
		Q					

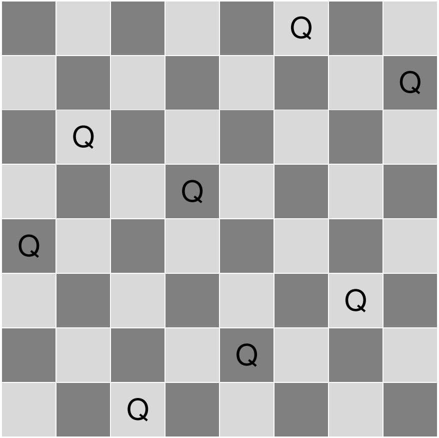
Ocho reinas, todas las soluciones

					Q		
			Q				
						Q	
Q							
		Q					
				Q			
	Q						
							Q

Ocho reinas, todas las soluciones

					Q		
			Q				
						Q	
Q							
							Q
	Q						
				Q			
		Q					

Ocho reinas, todas las soluciones



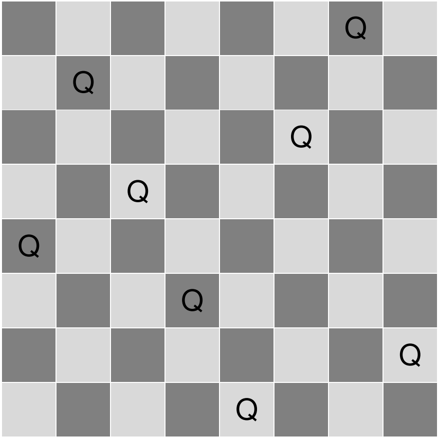
Ocho reinas, todas las soluciones

						Q	
Q							
		Q					
							Q
					Q		
			Q				
	Q						
				Q			

Ocho reinas, todas las soluciones

						Q	
	Q						
			Q				
Q							
							Q
				Q			
		Q					
					Q		

Ocho reinas, todas las soluciones



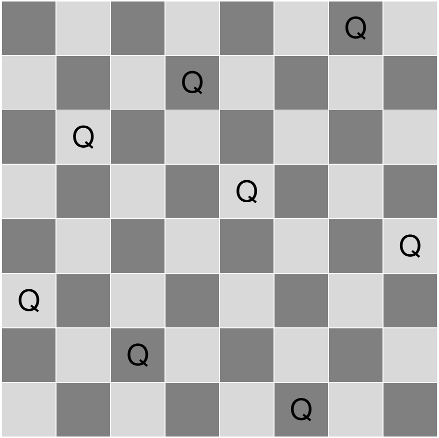
Ocho reinas, todas las soluciones

						Q	
		Q					
Q							
					Q		
							Q
				Q			
	Q						
			Q				

Ocho reinas, todas las soluciones

						Q	
		Q					
							Q
	Q						
				Q			
Q							
					Q		
			Q				

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

						Q	
			Q				
	Q						
							Q
					Q		
Q							
		Q					
				Q			

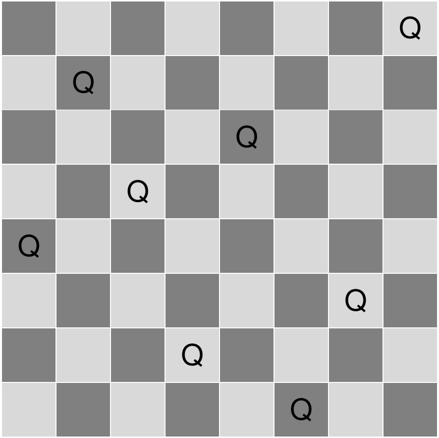
Ocho reinas, todas las soluciones

						Q	
				Q			
		Q					
Q							
					Q		
							Q
	Q						
			Q				

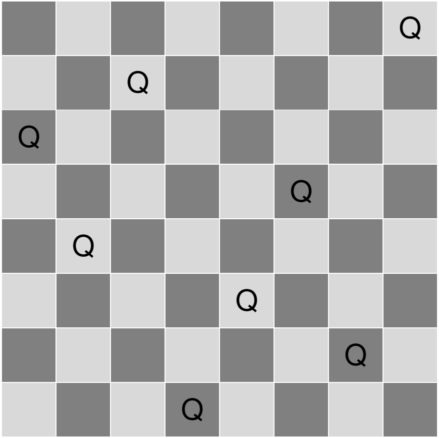
Ocho reinas, todas las soluciones

							Q
	Q						
			Q				
Q							
						Q	
				Q			
		Q					
					Q		

Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones



Ocho reinas, todas las soluciones

							Q
			Q				
Q							
		Q					
					Q		
	Q						
						Q	
				Q			

Ocho reinas, un algoritmo optimizado

El algoritmo

proc or_4(**in** sol, bajadas, subidas: **list of nat**, **in**/**out** r: **nat**)
{calcula el número de maneras de extender sol}
{hasta ubicar en total 8 reinas sin que se amenacen}
{bajadas y subidas son las diagonales ya amenazadas}
    if |sol| = 8 **then** r:= r+1 **fi**
    else i:= |sol|+1
        for j:= 1 **to** 8 **do**
            if j $\notin$ sol $\wedge$ bajada(i,j) $\notin$ bajadas $\wedge$ subida(i,j) $\notin$ subidas
            then or_4(sol $\triangleleft$ j, bajadas $\triangleleft$ bajada(i,j), subidas $\triangleleft$ subida(i,j), r)
            fi
        od
    fi
end

Ocho reinas, un algoritmo optimizado

El algoritmo principal

```
fun ocho_reinas_4() ret r: nat
    r:= 0
    or_4([ ], [ ], [ ], r)
end
```

Sobre bajadas y subidas

Observar que

- Todas las celdas de una bajada tienen en común que la diferencia entre la fila y la columna dan el mismo resultado.
- Todas las celdas de una subida tienen en común que la suma entre la fila y la columna dan el mismo resultado.

Esto sugiere la siguiente idea.

Ocho reinas, un algoritmo optimizado
Algoritmos auxiliares

```
fun bajada(i,j:nat) ret r: nat
    r:= i-j+7
end
fun subida(i,j:nat) ret r: nat
    r:= i+j
end
```

Ocho reinas, un algoritmo mejor

El grafo implícito

Ahora resulta más complicado explicitar el grafo implícito: V es el conjunto de listas $[p_1, \ldots, p_n] \in \{1, \ldots, 8\}^*$ tales que para todo $i \neq j$ las siguientes condiciones se cumplen:

1. $p_i \neq p_j$, es decir que no se repiten columnas,
2. $p_i - i \neq p_j - j$, es decir que no se repiten bajadas, y
3. $p_i + i \neq p_j + j$, es decir que no se repiten subidas.

Las aristas se establecen como en los intentos anteriores.