

Programación Lógica

Generación y chequeo



Objetivos

- Conocimiento de técnicas de programación en Prolog.
- Generación y chequeo como método de especificación ejecutable.
- Desafíos en cuanto a la eficiencia.
- Desarrollo incremental de 3 ejemplos:
 - Sort
 - Coloración de mapas
 - n-reinas

Generación y chequeo

- Técnica para diseñar y programar algoritmos.
- Un proceso genera soluciones candidatas al problema.
- Otro proceso chequea los candidatos, intentando encontrar aquel o aquellos que resuelvan el problema.

Generación y chequeo

Forma general:

`solucion(X) :- generacion(X),
chequeo(X).`

Método:

`generacion(X)` retorna candidatos `X` tales que :

- `X` tiene el formato de una eventual solución
- `chequeo(X)` testea si `X` satisface todas las condiciones de una solución

SORT

Deseamos ordenar una lista de n números.

Describimos las propiedades de una solución.

SORT

Deseamos ordenar una lista de n números.

Describimos las propiedades de una solución:

$\text{sort}(X) \text{ :- } \text{permutacion}(X, Xp),$
 $\text{ordenada}(Xp).$

SORT

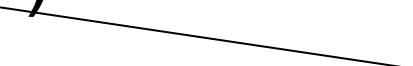
Deseamos ordenar una lista de n números.

Describimos las propiedades de una solución:

`sort(X) :- permutacion(X,Xp),
ordenada(Xp).`



generación



chequeo

SORT: permutación

$\text{permutacion}(+L, ?Lp) \leftarrow$ la lista Lp es una permutación de la lista L

SORT: permutación

$\text{permutacion}(+L, ?Lp) \leftarrow$ la lista Lp es una permutación de la lista L

Idea:

- Seleccionamos un 1er elemento de la solución (cualquier elemento de la lista).
- El resto es una permutación de los restantes elementos de la lista.

SORT: permutación: select

$\text{select}(?X, +L, ?Rx) \leftarrow X$ es un elemento de la lista L y Rx es L sin el elemento X

SORT: permutación: select

$\text{select}(?X,+L,?Rx) \leftarrow X \text{ es un elemento de la lista } L \text{ y } Rx \text{ es } L \text{ sin el elemento } X$

?- select(a,[b,a,c],R).

R = [b, c] ;

false.

4 ?- select(a,[b,a,c,a],R).

R = [b, c, a] ;

R = [b, a, c] ;

false.

5 ?- select(X,[a,b],R).

X = a,

R = [b] ;

X = b,

R = [a] ;

false

SORT: permutación: select

$\text{select}(?X, +L, ?Rx) \leftarrow X$ es un elemento de la lista L y Rx es L sin el elemento X

$\text{select}(X, [X|Xs], Xs).$

$\text{select}(X, [Y|Xs], [Y|R]) :- \text{select}(X, Xs, R).$

SORT: ordenada

ordenada(+L) \leftarrow la lista de números L está ordenada ascendente, $l_i \leq l_{i+1}$, $i=1, \dots, n-1$

SORT: ordenada

$\text{ordenada}(+L) \leftarrow$ la lista de números L está ordenada ascendiente, $l_i \leq l_{i+1}$, $i=1, \dots, n-1$

$\text{ordenada}([_]).$

$\text{ordenada}([X, Y|R]):-$

$X \leq Y,$

$\text{ordenada}([Y|R]).$

SORT

8 ?- sort1([9,8,7,6,5,4,3,2,1],L).

L = [1, 2, 3, 4, 5, 6, 7, 8, 9] .

9 ?- profile(sort1([9,8,7,6,5,4,3,2,1],L)).

L = [1, 2, 3, 4, 5, 6, 7, 8, 9].

0.61 seg., 623.529 =<, 362.880 ordenada

10 ?- profile(sort([9,8,7,6,5,4,3,2,1],L)).

L = [1, 2, 3, 4, 5, 6, 7, 8, 9].

11 ?- profile(sort1([20,19,18,17,16,15,14,9,8,7,6,5,4,3,2,1],L)).

Action (h for help) ? abort

% Execution Aborted

245 seg., 2.517.400 =<, 1.561.417 ordenada

12 ?- profile(sort([20,19,18,17,16,15,14,9,8,7,6,5,4,3,2,1],L)).

L = [1, 2, 3, 4, 5, 6, 7, 8, 9|...].

Generación y chequeo

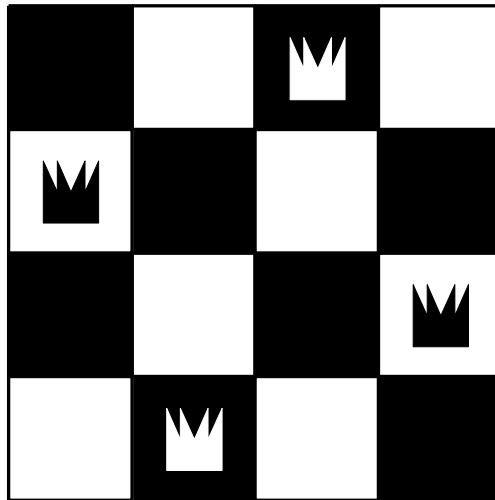
Problema: eficiencia.

Posible solución:

Técnica para optimizar programas consistente en “empujar” el chequeo dentro del generador lo más profundo posible.

Problema de las N-Reinas

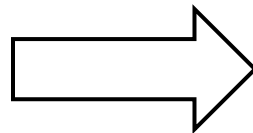
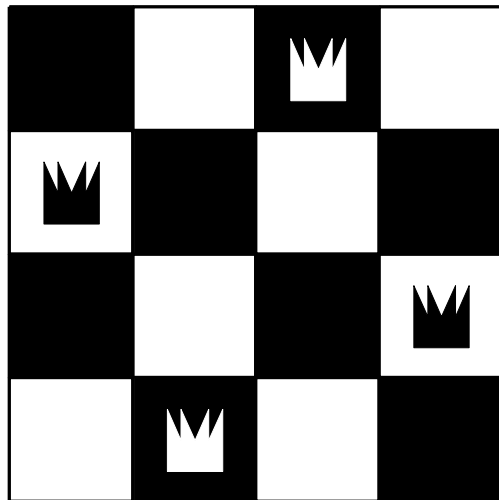
Distribuir N reinas en un tablero de N por N , de forma que toda casilla del tablero quede atacada por una reina, y ninguna reina sea atacada por otra.



Una solución al
problema de
las 4-Reinas

Problema de las N-Reinas

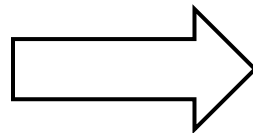
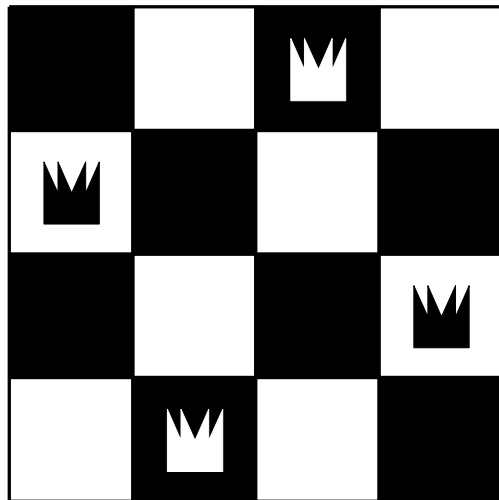
Representación



2	4	1	3
---	---	---	---

Problema de las N-Reinas

Representación



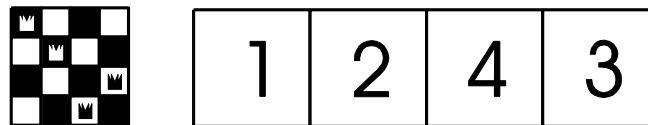
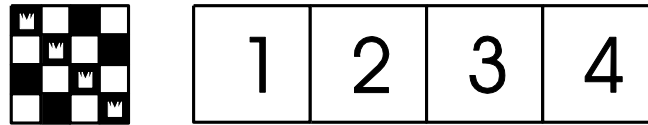
2	4	1	3
---	---	---	---

Lista de N elementos

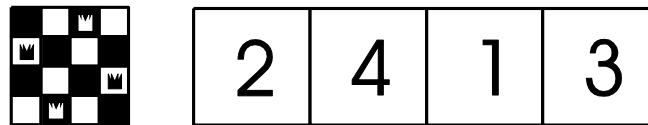
El valor del i ésimo elemento en la lista es la fila donde se coloca la reina de la columna i

Problema de las N-Reinas

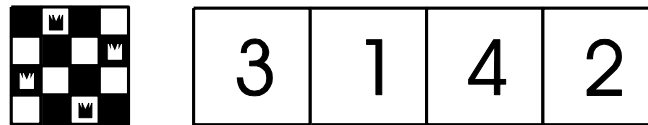
Problema de las N-Reinas: generación



⋮



⋮



Problema de las N-Reinas

```
reinas(N, Qs) :-  
    primeros(N, Ns),  
    permutacion(Ns, Qs),  
    segura(Qs).
```

Problema de las N-Reinas

```
reinas(N, Qs) :-  
    primeros(N, Ns),  
    permutacion(Ns, Qs),  
    segura(Qs).  
  
/*  
primeros(+N, ?Ns) ← inicializa la lista Ns con N  
posiciones diferentes, entonces tenemos una reina  
en cada columna y además todas están en diferentes  
filas.  
*/
```

La representación de datos ya incorpora parte de las restricciones del problema !!

Problema de las N-Reinas

```
reinas(N, Qs) :-  
    primeros(N, Ns),  
    permutation(Ns, Qs),  
    segura(Qs).
```

Problema de las N-Reinas

```
reinas(N, Qs) :-  
    primeros(N, Ns),  
    permutation(Ns, Qs),  
    segura(Qs).  
  
segura([Q|Qs]) :-  
    no_ataca(Q, Qs),  
    segura(Qs).  
segura([]).  
  
no_ataca(X, Xs) :- no_ataca(X, 1, Xs).  
no_ataca(_, _, []).  
no_ataca(X, N, [Y|Ys]) :- X \= Y+N,  
                           X \= Y-N,  
                           N1 is N+1,  
                           no_ataca(X, N1, Ys).
```


Problema de las N-Reinas

Problema:

Chequea todas las permutaciones (muy ineficiente).

Mejora:

Chequear, a medida que se arma la estructura, si la ubicación de la reina es legal.

Problema de las N-Reinas

```
reinas(N, Qs) :-  
    primeros(N, Ns),  
    reinas(Ns, [], Qs).  
  
reinas(SinColocar, Seguras, Qs) :-  
    select(Q, SinColocar, SinColocar1),  
    no_ataca(Q, Seguras),  
    reinas(SinColocar1, [Q|Seguras], Qs).  
  
reinas([], Qs, Qs).  
  
select :- ...  
  
no_ataca :- ...
```

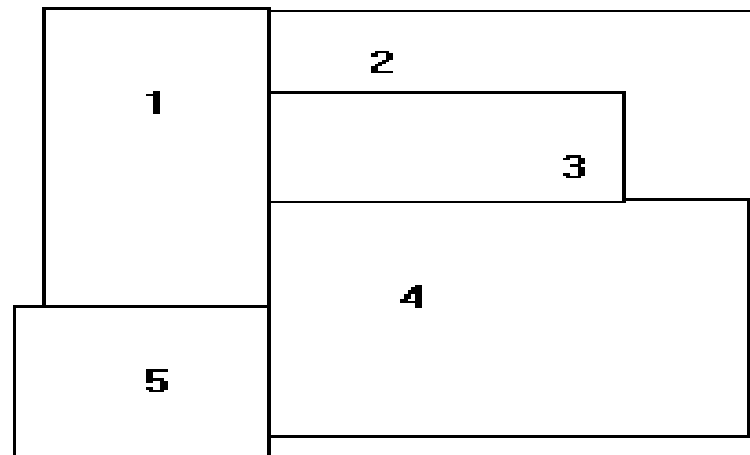
Coloración de mapas

La figura representa un mapa.

Se debe colorear de modo que regiones adyacentes tengan distinto color.

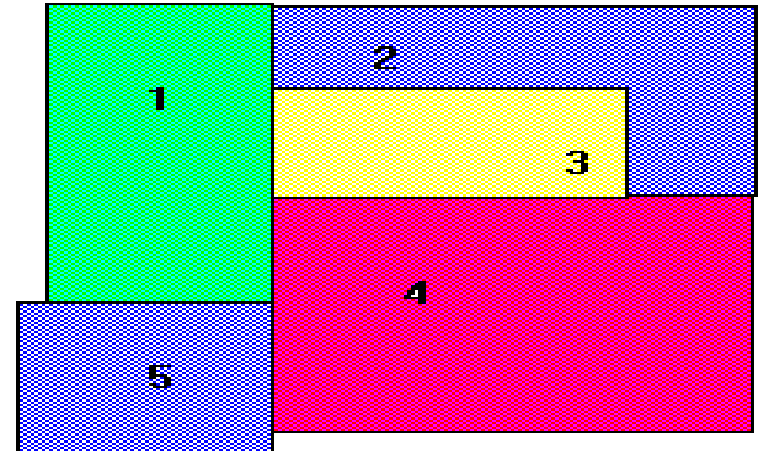
Conjetura : todo mapa (con regiones continuas) se puede colorear con 4 colores.

Fue probada por Appel and Haken (1976)



Coloración de mapas

Coloración con 4 colores.

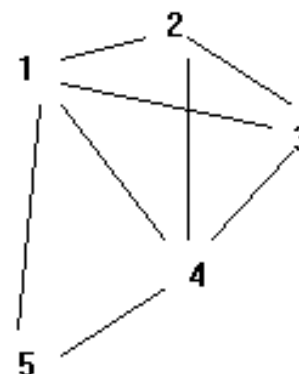
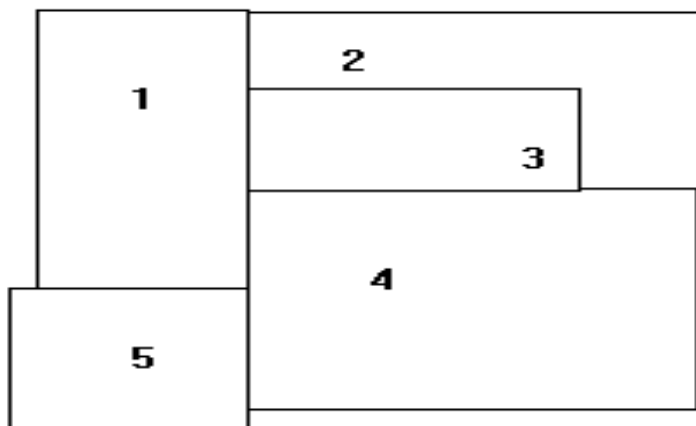


Coloración de mapas

- Seguiremos un esquema de generación y chequeo.
 - Generación: pintamos todas las regiones con algún color
 - Chequeo: no hay regiones vecinas con el mismo color
- Pero primero debemos decidir como vamos a representar los datos.

Coloración de mapas

- representar los datos:



Las relaciones de adyacencia de las regiones determinan un grafo.

Cada región es un vértice, hay arista entre todo par de regiones adyacentes.

Datos

Regiones:

Declaramos las regiones:

En el ejemplo:

`region([1,2,3,4,5]).`

Vecinos

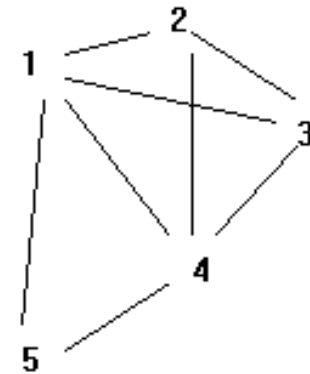
Relación para las aristas:

`vecino(1,2). vecino(2,4).`

`vecino(1,3). ...`

`vecino(1,4).`

`vecino(1,5).`



Datos

Regiones:

Declaramos las regiones:

En el ejemplo:

```
region([1,2,3,4,5]).
```

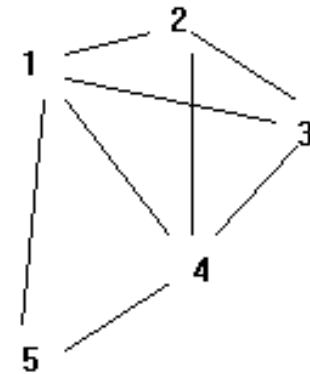
Vecinos (otro modo)

Listas de adyacencias:

```
vecinos(1,[2,3,4,5]).      ...
```

```
vecino(2,[1,3,4]).
```

```
vecino(3,[1,2,4]).
```



Datos

Colores:

Definimos una paleta de colores:

p.ej.,:

colores([rojo,verde,amarillo,azul,violeta,gris]).

Basta con 4 colores !!

Mapa coloreado

Una lista de pares (Región,Color):

[(1,rojo),(2,verde),...]

Coloración de mapas: generación y chequeo

```
mapaColoreado(MapaColor):-  
    genera(MapaColor),  
    sin_conflicto(MapaColor).
```

Coloración de mapas: generación

```
genera(MapaColor) :-  
    regiones(R),  
    colores(C),  
    pintar(R,C,MapaColor).
```

```
pintar([],_,[]).  
pintar([R|Rs], C, [(R,C1)|RM]) :-  
    member(C1,C),  
    pintar(Rs,C,RM).
```

Coloración de mapas: chequeo

```
sin_conflicto([_]).
```

```
sin_conflicto([(R,C)|RM]):-  
    sin_conflicto(R,C,RM),  
    sin_conflicto(RM).
```

```
sin_conflicto(R,C,RM):-  
    vecinos(R,V),  
    todos_color_distinto(C,V,RM).
```

```
todos_color_distinto(_,[],_).  
todos_color_distinto(C,[V|Vr],RM):-  
    member((V,C1),RM),  
    C \= C1,  
    todos_color_distinto(C,Vr,RM).
```

Coloración de mapas: generación y chequeo

- **Se construyó una MALA solución !!**
- **Se puede mejorar mucho, incluyendo chequeo en la generación.**
- **El objetivo es fallar lo antes posible si una asignación de color no sirve.**
- **Ejercicio: Mejorar la solución propuesta (chequear por adyacentes al momento de pintar).**