Backtracking Generalizado

Llamamos $\boldsymbol{X}$ al conjunto de **variables**

Los valores que puede tomar cada variable $\boldsymbol{x} \in \boldsymbol{X}$ son el **dominio** de $\boldsymbol{x}$

Las restricciones las abstraemos en un conjunto $\boldsymbol{R}$

```
is solvable(X, R):
    if is solution(X, R):
        return true
    x = choose unassigned variable(X)
    for v ∈ x.d:
        if is valid(x, v, R):
            assign(x, v)
            if is solvable(X, R):
                return true
            unassign(x, v)
    return false
```

Modelación

Para resolver un problema siempre es necesario:

- Identificar las componentes del problema
- Expresarlas en términos de variables y restricciones
- Preocuparse de que las **operaciones** sean eficientes

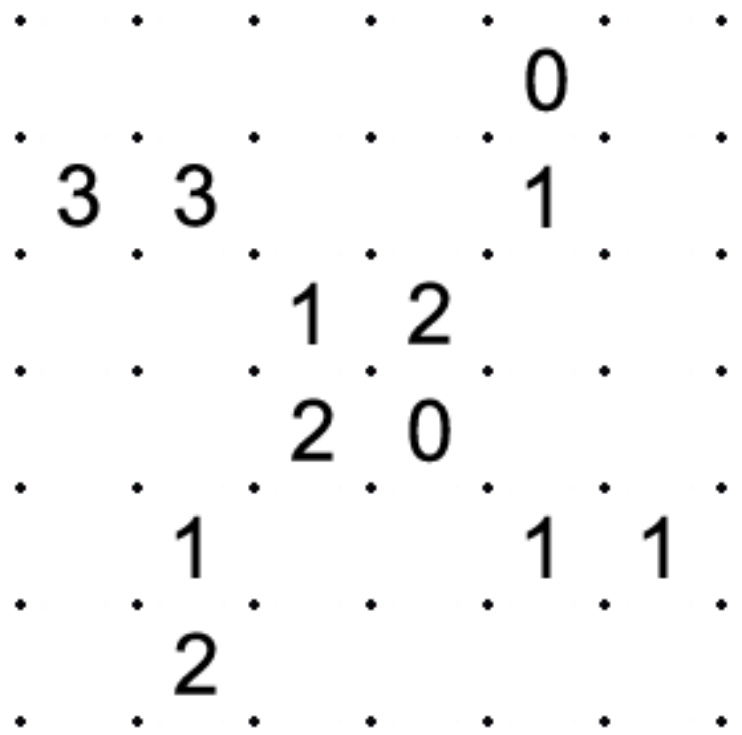
N-Queens

En un tablero de ajedrez de $n \times n$ se quieren poner n reinas…

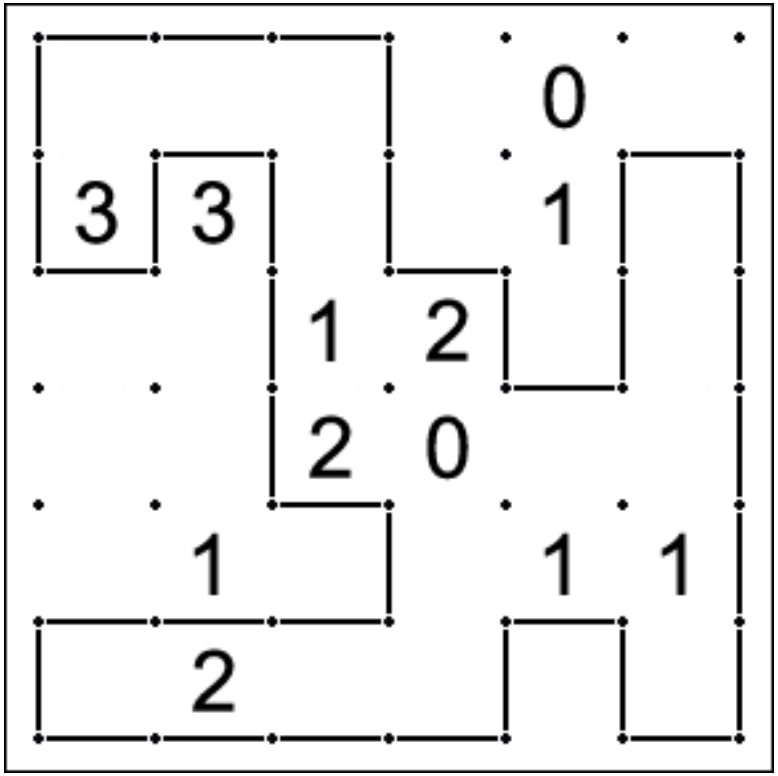
… tal que ninguna reina pueda atacar a otra reina

¿Cómo modelamos esto?

Slitherlink



Slitherlink



Descarte

La idea es **descartar** permutaciones que no llevan a una solución

Una forma de hacer esto es revisar las restricciones

¿Hay alguna otra manera?

```
is solvable(X, R):
    if is solution(X, R):
        return true
    x = choose unassigned variable(X)
    for v ∈ x.d:
        if is valid(x, v, R):
            assign(x, v)
            if is solvable(X, R):
                return true
            unassign(x, v)
    return false
```

Podas

Una **poda** es una restricción adicional

Se **deducen** de las restricciones originales

¿Cuándo nos conviene hacer una poda?

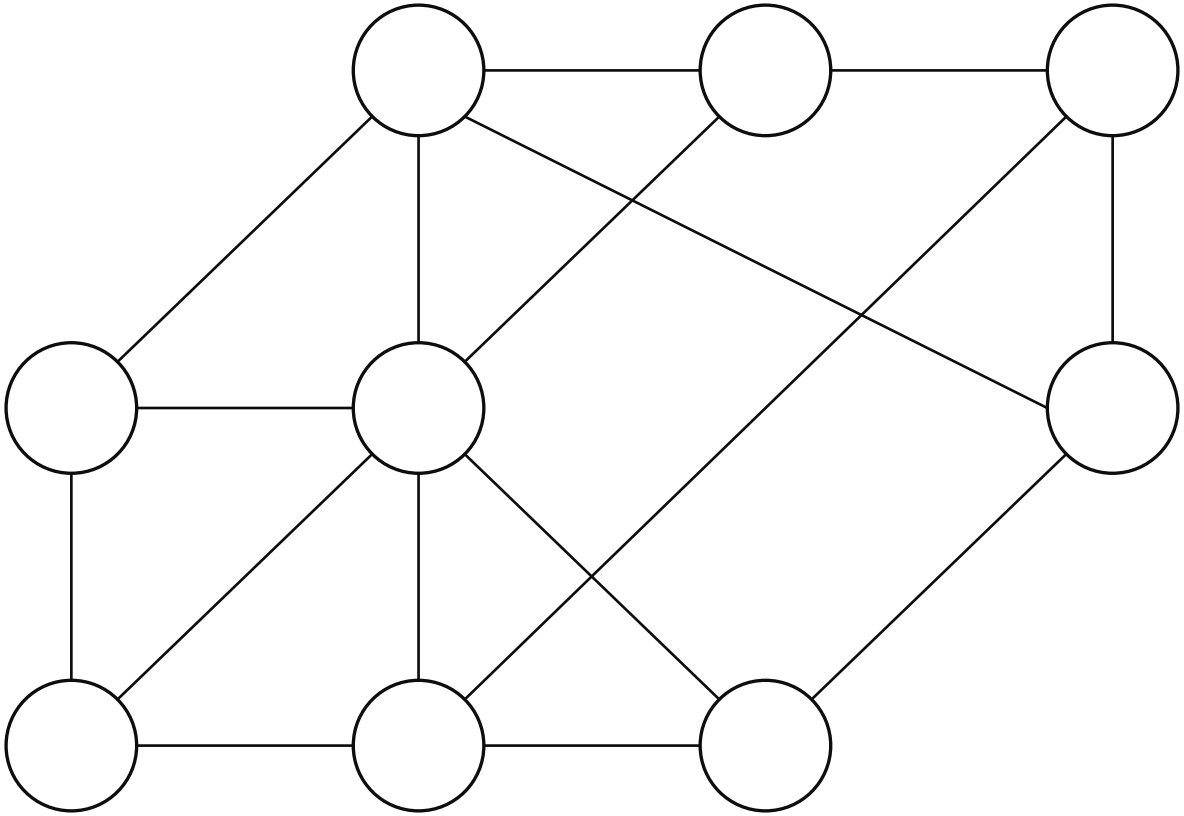
Estrategia

¿Afecta el orden en que asignamos las variables?

¿Afecta el orden en el que probamos los valores?

¿Será posible formular una estrategia para resolver el problema?

Coloración de grafos



¿Qué estrategia podemos usar para colorear este grafo?

```
is solvable(X, R):
    if is solution(X, R):
        return true
    x = choose unassigned variable(X)
    for v ∈ x.d:
        if is valid(x, v, R):
            assign(x, v)
            if is solvable(X, R):
                return true
            unassign(x, v)
    return false
```

Heurísticas

Formalmente, estas estrategias se conocen como **heurísticas**

Las heurísticas nos dicen que opciones nos convienen más

¿Cuándo nos conviene usar heurísticas?

Usos de backtracking

Sirve cuando es necesario probar combinaciones, ya sea

- Problemas de **asignación**
- Problemas de **planificación**
- Problemas de **optimización**

Mejoras adicionales

El mundo de Backtracking tiene muchas más cosas interesantes:

- propagación
- back-jumping
- arc-consistency
- etc…

Para efectos de este curso nos quedaremos con lo visto hoy