Backtracking – I

Clase #25

IIC1103 – Introducción a la Programación

Marcos Sepúlveda (marcos@ing.puc.cl)

Veremos hoy …

- Backtracking – motivación
- Backtracking – colorear mapa
- Backtracking – 8 reinas

Motivación

- Supongamos que hay que tomar una serie de ***decisiones***, entre varias ***opciones***, donde:
  - No hay suficiente información para saber qué elegir
  - Cada decisión conduce a un nuevo conjunto de opciones
  - Alguna secuencia de opciones (posiblemente más de una) puede ser una solución al problema

- Backtracking es una manera metódica de probar varias secuencias de opciones, hasta encontrar una que “funcione”

Fuente: https://www.cis.upenn.edu/~matuszek/cit594-2009/Lectures/35-backtracking.ppt

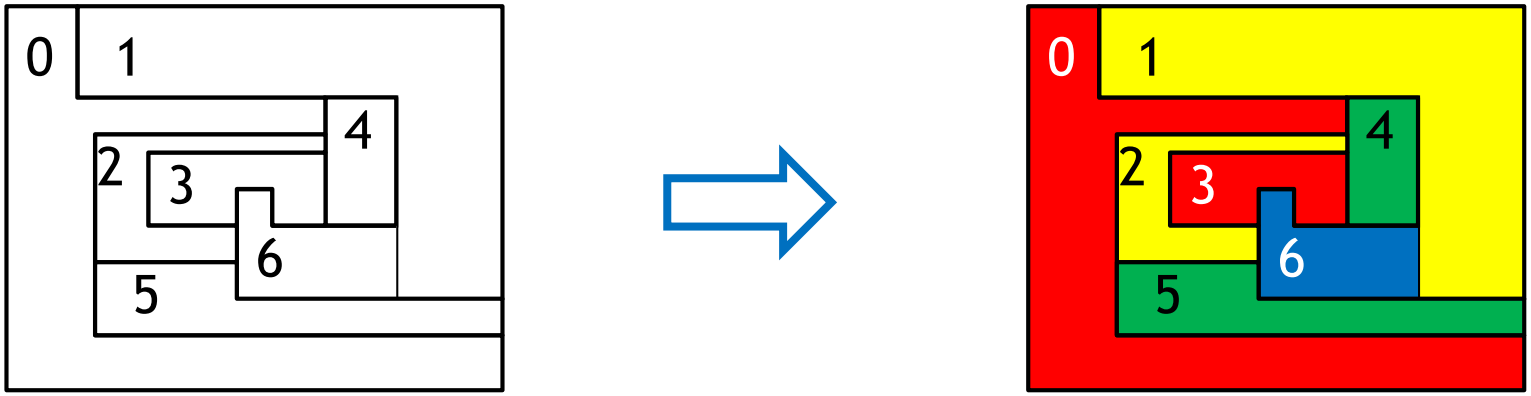
Salir de un laberinto

- Dado un laberinto, encontrar un camino desde el inicio hasta el fin
- En cada intersección, hay que decidir entre tres o menos opciones:
  - Seguir derecho
  - Doblar a la izquierda
  - Doblar a la derecha

- No hay suficiente información para elegir correctamente en cada caso
- Cada decisión conduce a otro conjunto de opciones
- Una o más secuencias de opciones pueden conducir a una solución

Colorear un mapa

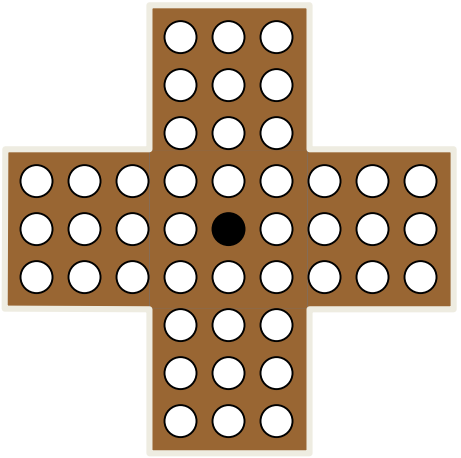
- Se desea colorear un mapa con no más de cuatro colores
  - rojo, amarillo, verde, azul
- Los países adyacentes deben estar en diferentes colores



- No hay suficiente información para elegir los colores
- Cada decisión conduce a otro conjunto de opciones
- Una o más secuencias de opciones pueden conducir a una solución

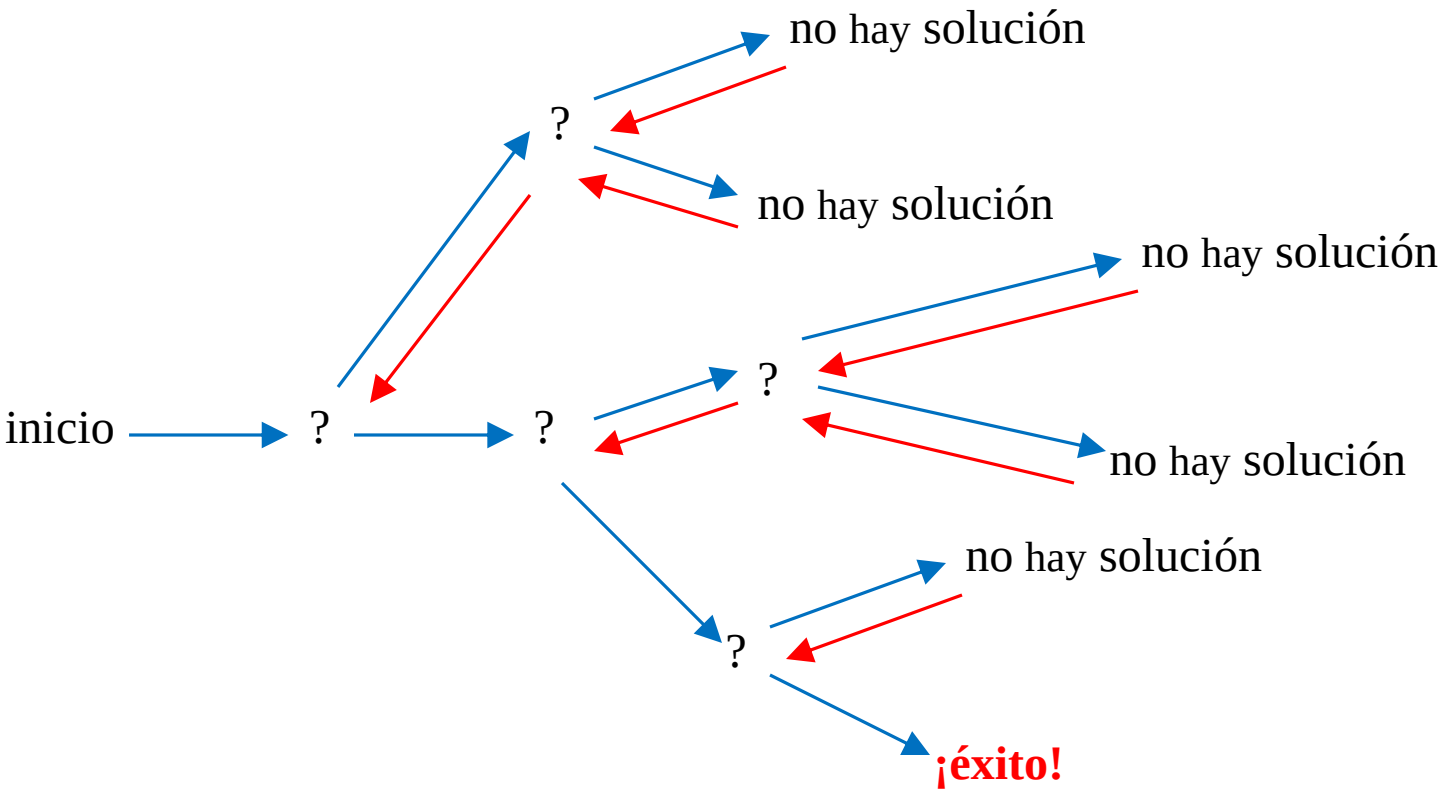
Resolver el puzzle del solitario

- En este puzzle, todos los agujeros, salvo uno, contienen piezas blancas
- Se puede saltar con una pieza sobre una adyacente, siempre que este disponible el agujero donde caerá
  - La pieza saltada se quita
- El objetivo es eliminar todas, quedando sólo una



- No hay suficiente información para saltar correctamente
- Cada decisión conduce a otro conjunto de opciones
- Una o más secuencias de opciones pueden conducir a una solución

Backtracking – esquema de solución



Fuente: https://www.cis.upenn.edu/~matuszek/cit594-2009/Lectures/35-backtracking.ppt

Backtracking

- El *backtracking* (método de retroceso o vuelta atrás) es un algoritmo general para la búsqueda exhaustiva y sistemática de una (o todas) las soluciones a algunos problemas computacionales, en especial problemas de satisfacción de restricciones, que construye incrementalmente soluciones candidatas, y abandona cada solución candidata parcial **c** (hace un "*backtrack*") tan pronto como se determina que **c** no puede derivar en una solución completa válida.

- Sólo puede aplicarse a problemas que admiten el concepto de una "solución parcial candidata" y para las cuales hay una prueba relativamente rápida de si puede completarse para crear una solución completa válida.
  - Es inútil, por ejemplo, para buscar si un valor dado está en una lista no ordenada.

- Sin embargo, cuando es aplicable, el *backtracking* es a menudo mucho más rápido que la enumeración de todas las posibles soluciones completas por fuerza bruta, ya que puede eliminar un gran número de soluciones candidatas con una sola prueba.

- Se pueden usar heurísticas (estrategias para buscar mejores soluciones candidatas) para acelerar el proceso.

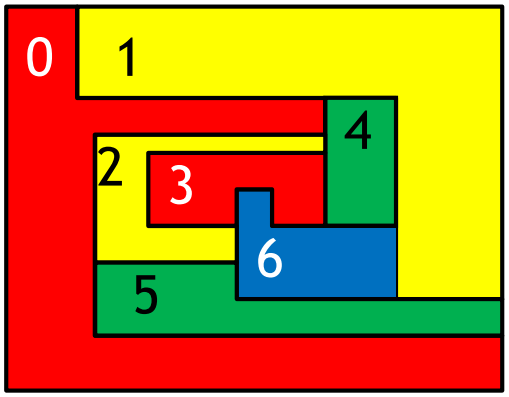
Fuente: http://en.wikipedia.org/wiki/Backtracking

Backtracking – estructura genérica

```
# Retorna True si vale la pena explorar la solución candidata
def SirveComoSolucionParcial(soln_c):
    pass

# Retorna True si la solución candidata resuelve el problema
def SirveComoSolucionFinal(soln_c):
    pass

# Usa la solución candidata como una solución valida al problema
def MostrarSolucionFinal(soln_c):
    pass

# Genera una lista de soluciones extendidas, derivadas de la solución candidata
def ObtenerSolucionesCandidatas(soln_c):
    pass

# Retorna una solución candidata inicial
def SolucionInicial():
    pass

# Backtracking genérico
def Backtracking(soln_c):
    if not SirveComoSolucionParcial(soln_c):
        return False
    if SirveComoSolucionFinal(soln_c):
        MostrarSolucionFinal(soln_c)
        return True
    lista_candidatas = ObtenerSolucionesCandidatas(soln_c)
    for solucion in lista_candidatas:
        if Backtracking(solucion):
            return True
    return False

Backtracking(SolucionInicial())
```

Backtracking – colorear mapa

- El teorema de los Cuatro Colores señala que cualquier mapa en un plano puede ser coloreado con no más de cuatro colores, de modo que no haya dos países con un borde común que sean del mismo color.
- Para un mapa pequeño, encontrar una forma de colorearlo válida es fácil
- Para otros, puede ser bastante difícil



- Para saber más:
  - https://en.wikipedia.org/wiki/Four_color_theorem

Backtracking – colorear mapa

Idea

- Recorrer todos los países en orden, pintando cada uno de un color

- Una **solución parcial** es una en que tenemos coloreados solo algunos países
- Una **solución parcial útil** es una en que todos los países coloreados, están coloreados de un color distinto al de sus vecinos
- Para **extender una solución parcial**, escogemos un siguiente país, y tratamos de colorearlo de los cuatro colores posibles
- Una **solución final** es aquella en que todos los países están coloreados, y todos tienen un color distinto al de sus vecinos

Backtracking – colorear mapa

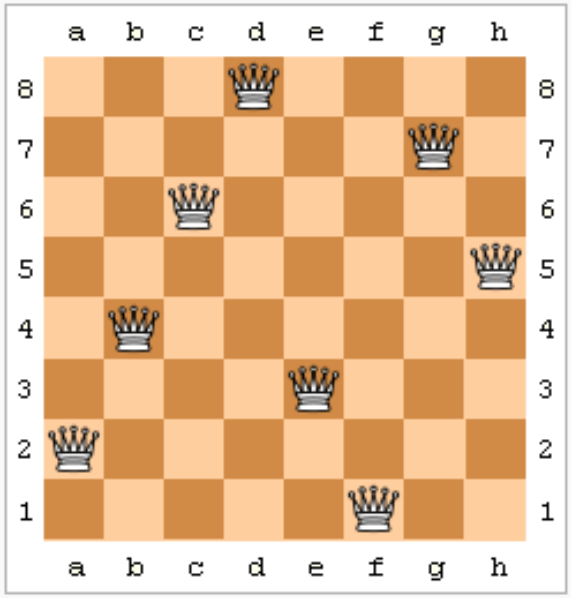
```
# soln_c: solución parcial; es una lista que contiene un
#         color para cada uno de los primeros n países

# Retorna True si vale la pena explorar la solución candidata
def SirveComoSolucionParcial(soln_c):
  # True si no hay conflicto entre los países pintados

# Retorna True si la solución candidata resuelve el problema
def SirveComoSolucionFinal(soln_c):
  # True si hemos pintado todos los países

# Usa la solución candidata como una solución valida al problema
def MostrarSolucionFinal(soln_c):
  # mostramos con qué color pintamos cada país

# Genera una lista de soluciones extendidas, derivadas de la solución candidata
def ObtenerSolucionesCandidatas(soln_c):
    # probamos pintar un país más con distintos colores

# Retorna una solución candidata inicial
def SolucionInicial():
  # retornamos una lista vacía; aún no hemos coloreado
  # ningún país
```

Backtracking – estructura genérica

```
# Backtracking genérico
def Backtracking(soln_c):

    # hay conflicto entre los colores con que
    # hemos pintado los países
    if not SirveComoSolucionParcial(soln_c):
        return False

    # hemos pintado todos los países
    if SirveComoSolucionFinal(soln_c):
        MostrarSolucionFinal(soln_c)
        return True

    # no hemos pintado todos los países, así es que
    # vemos opciones para pintar un país adicional
    lista_candidatas = ObtenerSolucionesCandidatas(soln_c)

    # para cada color con que pintamos el país adicional,
    # vemos si nos conduce a una solución final
    for solucion in lista_candidatas:
        if Backtracking(solucion):
            return True

    return False

# llamado inicial al Backtracking
Backtracking(SolucionInicial())
```

Ocho reinas

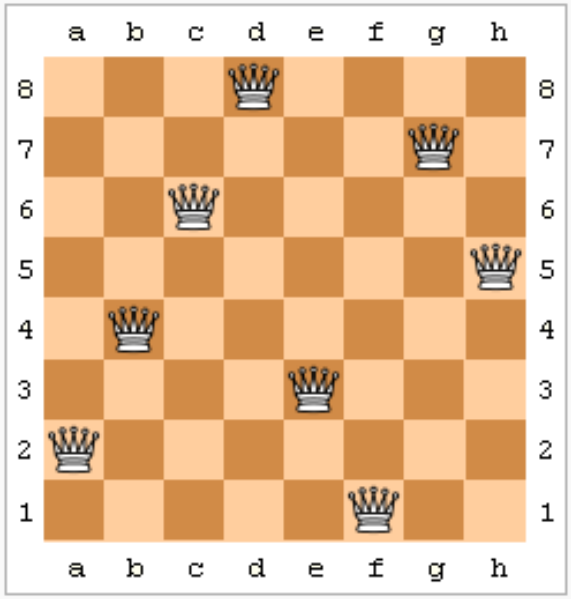
- En el juego del ajedrez, la reina amenaza a aquellas piezas que se encuentren en su misma fila, columna o diagonal. El problema de las ocho reinas consiste en buscar cómo colocar ocho reinas sobre un tablero de ajedrez sin que se amenacen entre ellas.
  - Fue propuesto por el ajedrecista alemán Max Bezzel en 1848.



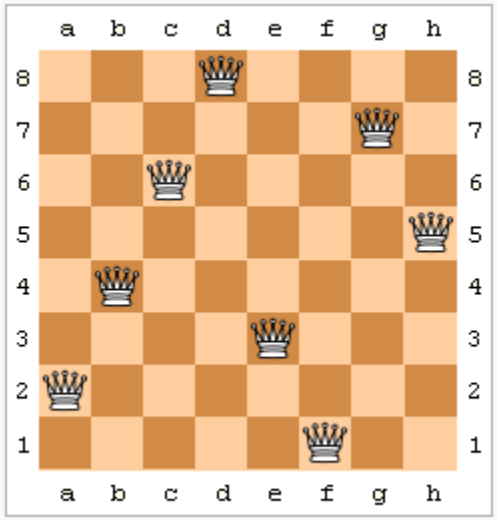
Fuente: http://es.wikipedia.org/wiki/Problema_de_las_ocho_reinas

Soluciones al problema de las ocho reinas

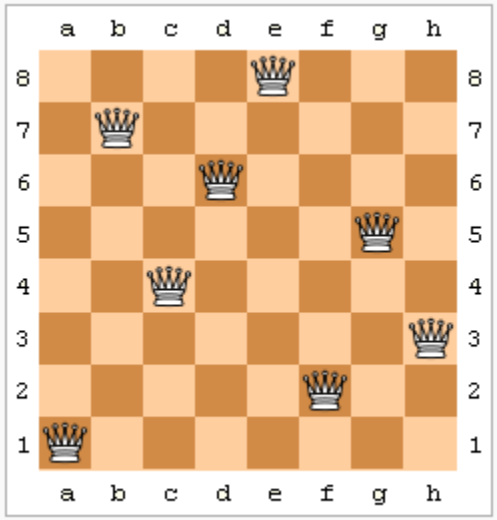
- El problema de las ocho reinas tiene 92 soluciones, de las cuales 12 son **esencialmente distintas**, es decir las 92 soluciones existentes se pueden obtener a partir de simetrías, rotaciones y traslaciones de las 12 soluciones únicas, que se muestran a continuación:



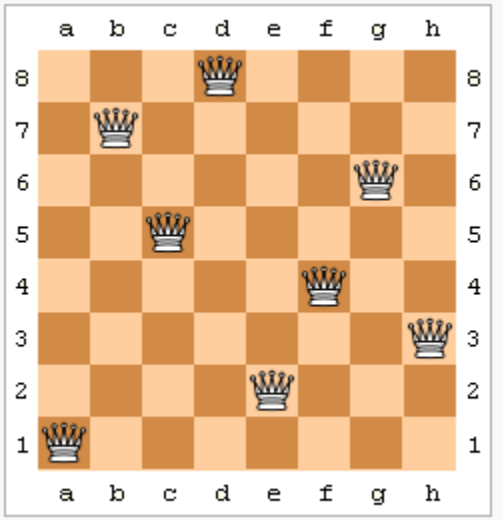
Fuente: http://es.wikipedia.org/wiki/Problema_de_las_ocho_reinas



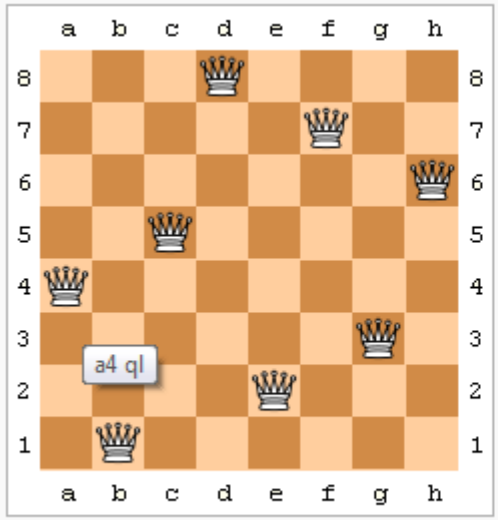
Solución única 1



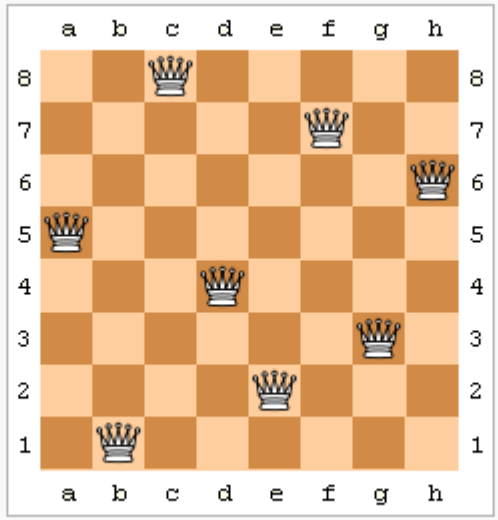
Solución única 2



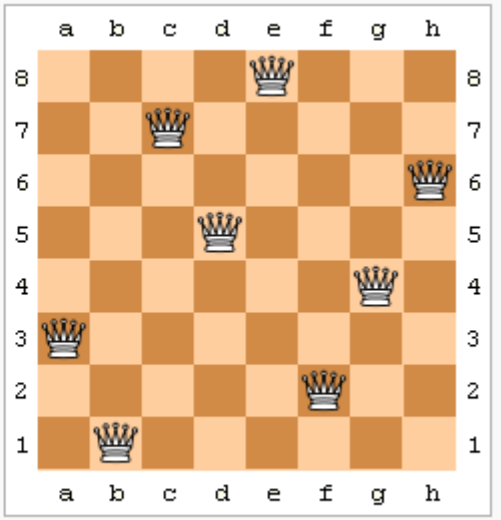
Solución única 3



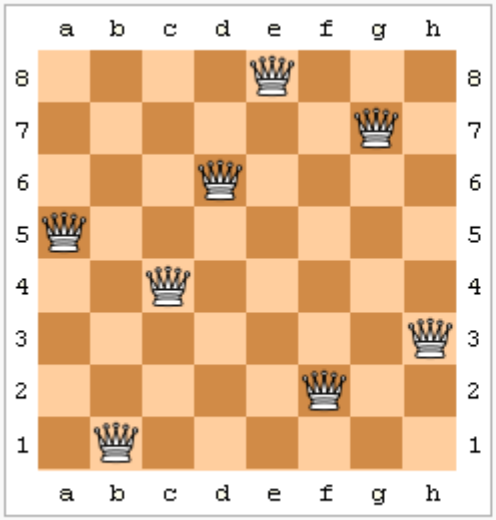
Solución única 4



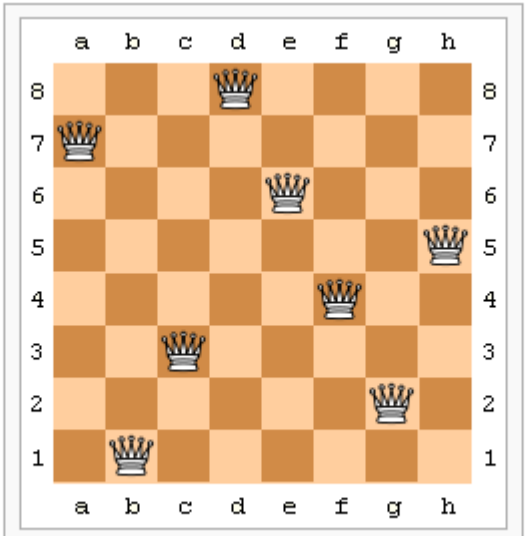
Solución única 5



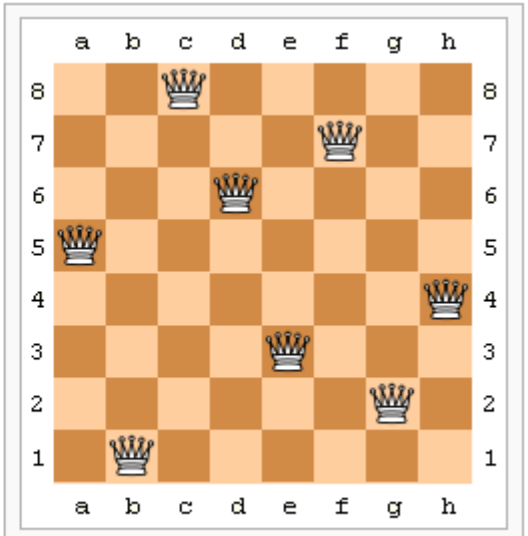
Solución única 6



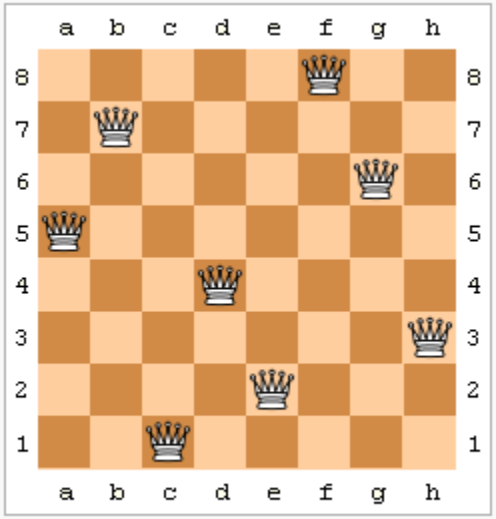
Solución única 7



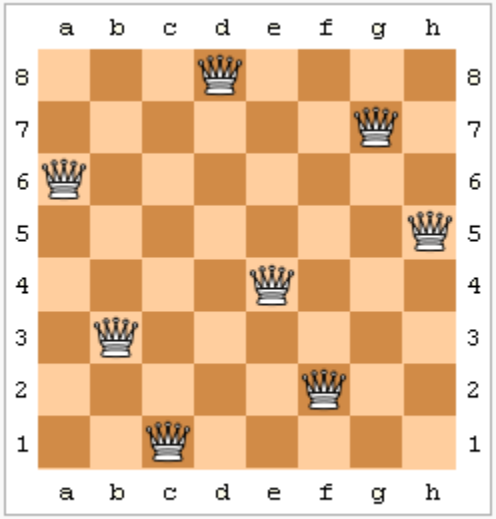
Solución única 8



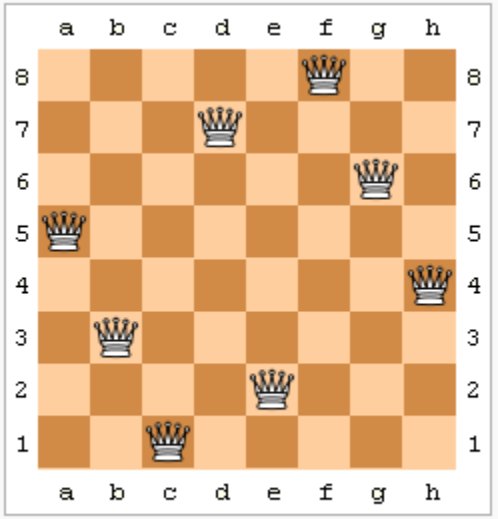
Solución única 9



Solución única 10



Solución única 11



Solución única 12

Backtracking – n reinas

- El problema de las ocho reinas se puede plantear de modo general como problema de las ***n*** reinas. El problema consistiría en colocar ***n*** reinas en un tablero de ajedrez de ***n* * *n***, de tal manera que ninguna de las reinas quede atacando a otra.

- El problema general tiene solución para **n > 3**.

Fuente: http://es.wikipedia.org/wiki/Problema_de_las_ocho_reinas

Backtracking – n reinas

- Para resolver este problema emplearemos un esquema de *backtracking*.

- Veamos el siguiente esquema (páginas 11 a 36), ilustrado con 4 reinas:
  - http://es.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking

Backtracking – n reinas

```
class Tablero:
    def __init__(self, dimension=8):
        self.dimension=dimension
        self.tablero = []
        for i in range(0,self.dimension):
            f = []
            for j in range(0,self.dimension):
                f.append(' ')
            self.tablero.append(f)
```

Backtracking – n reinas

```
# Retorna True si vale la pena explorar la solución candidata
def SirveComoSolucionParcial(self, soln_c):
    marcado = []
    for fila in self.tablero:
        marcado.append(fila.copy())

    for coord in soln_c:
        f = coord[0]; c = coord[1]
        marcado[f][c] = 'R'

    for coord in soln_c:
        f = coord[0]; c = coord[1]

        for iC in range(0,self.dimension): # revisa fila
            if marcado[f][iC] == 'R' and iC != c:
                return False

        for iF in range(0,self.dimension): # revisa columna
            if marcado[iF][c] == 'R' and iF != f:
                return False

        for iD in range(1,self.dimension+1): #arriba-izquierda
            if f-iD>=0 and c-iD>=0:
                if marcado[f-iD][c-iD] == 'R':
                    return False
        for iD in range(1,self.dimension+1): #arriba-derecha
            if f-iD>=0 and c+iD<self.dimension:
                if marcado[f-iD][c+iD] == 'R':
                    return False
        for iD in range(1,self.dimension+1): #abajo-izquierda
            if f+iD<self.dimension and c-iD>=0:
                if marcado[f+iD][c-iD] == 'R':
                    return False
        for iD in range(1,self.dimension+1): #abajo-derecha
            if f+iD<self.dimension and c+iD<self.dimension:
                if marcado[f+iD][c+iD] == 'R':
                    return False

    return True
```

Backtracking – n reinas

```
# Retorna True si la solución candidata resuelve el problema
def SirveComoSolucionFinal(self, soln_c):
    if len(soln_c) < self.dimension:
        return False
    else:
        return self.SirveComoSolucionParcial(soln_c)

# Usa la solución candidata como una solución válida al problema
def MostrarSolucionFinal(self, soln_c):
    print("Solucion:")
    for coord in soln_c:
        f = coord[0]; c = coord[1]
        self.tablero[f][c] = 'R'
    for f in self.tablero:
        print(f)

# Genera una lista de soluciones extendidas, derivadas de solución candidata
def ObtenerSolucionesCandidatas(self, soln_c):
    col = len(soln_c)
    listaCandidatas = []
    for fil in range(0,self.dimension):
        listaCandidatas.append(soln_c + [[fil,col]])
    return listaCandidatas
```

Backtracking – n reinas

```
    # Retorna una solución candidata inicial
    def SolucionInicial(self):
        return []

    # Backtracking genérico
    def Backtracking(self, soln_c):
        if not self.SirveComoSolucionParcial(soln_c):
            return False
        if self.SirveComoSolucionFinal(soln_c):
            self.MostrarSolucionFinal(soln_c)
            return True
        lista_candidatas = self.ObtenerSolucionesCandidatas(soln_c)
        for solucion in lista_candidatas:
            if self.Backtracking(solucion):
                return True
        return False

# Muestra una solución para tableros de tamaño 4 a 16
for n in range(4,17):
    print("----", n)
    t = Tablero(n)
    t.Backtracking(t.SolucionInicial())
```

Backtracking – 4, 8 y 12 reinas

```
>>>
---- 4
Solucion:
[' ', ' ', 'R', ' ']
['R', ' ', ' ', ' ']
[' ', ' ', ' ', 'R']
[' ', 'R', ' ', ' ']

---- 8
Solucion:
['R', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', 'R', ' ']
[' ', ' ', ' ', ' ', 'R', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', 'R']
[' ', 'R', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', 'R', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', 'R', ' ', ' ']
[' ', ' ', 'R', ' ', ' ', ' ', ' ', ' ']

---- 12
Solucion:
['R', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'R', ' ', ' ', ' ']
[' ', 'R', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'R']
[' ', ' ', 'R', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', 'R', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'R', ' ', ' ']
[' ', ' ', ' ', 'R', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'R', ' ']
[' ', ' ', ' ', ' ', 'R', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', 'R', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', 'R', ' ', ' ', ' ', ' ', ' ', ' ']
```

Backtracking – ocho reinas, todas las soluciones

- Para calcular todas las soluciones posibles, basta con cambiar las líneas marcadas a continuación:

```
    # Backtracking genérico
    def Backtracking(self, soln_c):
        if not self.SirveComoSolucionParcial(soln_c):
            return False
        if self.SirveComoSolucionFinal(soln_c):
            self.MostrarSolucionFinal(soln_c)
            return True
        lista_candidatas = self.ObtenerSolucionesCandidatas(soln_c)
        for solucion in lista_candidatas:
            if self.Backtracking(solucion):
                # calculamos todas las soluciones
                pass
                # return True
        return False

# Muestra todas las soluciones posibles para un tablero de tamaño 8
t = Tablero(8)
t.Backtracking(t.SolucionInicial())
```

Backtracking – ocho reinas, todas las soluciones

- Y redefinir el método que muestra la solución final, para que despliegue solo una copia del tablero:

```
# Usa la solución candidata como una solución valida al problema
def MostrarSolucionFinal(self, soln_c):
    self.nSoluciones += 1
    print("Solucion #", self.nSoluciones)
    marcado = []
    for fila in self.tablero:
        marcado.append(fila.copy())
    for coord in soln_c:
        f = coord[0]; c = coord[1]
        marcado[f][c] = 'R'
    for f in marcado:
        print(f)
```