INF221 – Algoritmos y Complejidad

Clase #14
Backtracking

Aldo Berrios Valenzuela Horst H. von Brand

Miércoles 21 de septiembre de 2016

1. Backtracking

Otra estrategia recursiva... La idea es ir construyendo la solución incrementalmente, explorando distintas ramas y volviendo atrás (backtrack) si resulta que un camino es sin salida.

Ejemplo 1.1 (Un clásico)**.** En el ajedrez la reina es la pieza más poderosa. Amenaza los casilleros en su fila y columna, y los ubicados en diagonal. La figura 1 muestra los casilleros que amenaza una reina en el ajedrez. Un problema

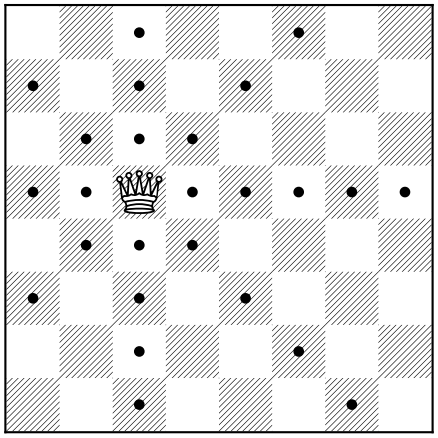

Figura 1: Los casilleros amenazados por una reina

clásico es determinar si se pueden ubicar ocho reinas en el tablero de manera que ninguna pueda amenzar a otra. Claramente no pueden ser más de ocho, puede haber a lo más una reina por columna.

Pasos:

- Reducir espacio de búsqueda.

  $\rightsquigarrow$ Si son 8 reinas, hay una por columna.

  Por lo tanto, llenar por columnas, solución (parcial) indica las filas de las reinas ya ubicadas.

- Ordenar avance.
- Subproblemas similares.

En este caso:

- Ubicar reina en columna 1, 2, ...
- Registrar filas libres (por omitir ocupadas al ubicar la siguiente reina)
- Registrar diagonales libres.
  Reina en i, j:

$$y - j = 1 \cdot (x - i)$$
$$i - j = x - y \rightsquigarrow x - ycte$$
$$y - j = -1 \cdot (x - i) \rightsquigarrow x + ycte$$

Por ejemplo, luego de ubicadas las primeras tres reinas en las filas 1, 3 y 5 la configuración resultante es la de la figura 2. Vemos que las filas y diagonales amenazadas por estas tres restringen muchísimo las posiciones viables

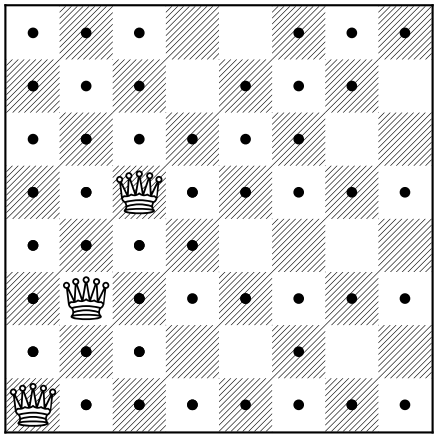

Figura 2: Configuración con tres reinas

para la cuarta y siguientes. Con estas tres reinas, para la cuarta reina quedan solo 3 posibilidades.

Elegimos Python (!), en Python los arreglos tienen índices desde 0, rango de i, j es $0 \ldots 7$. Interesan los rangos de:

- $i - j$: $-7 \ldots 7 \rightsquigarrow$ sumar 7 para llevar al rango $0 \ldots 14$.
- $i + j$: $0 \ldots 14$

El programa final es el siguiente:

```
#!/usr/bin/python3

queen = [0 for i in range(8)]    # Row of queen in column i
rfree = [True for i in range(8)]  # Row i is free
dufree = [True for i in range(15)]  # Up diagonal through i + 7 - j free
ddfree = [True for i in range(15)]  # Down diagonal through i + j free

def solve(n):
  global solutions

  if n == 8:
    solutions += 1
    print(solutions, end = ": ")
    for i in range(8):
      print(queen[i] + 1, end = " ")
```

```
    print()
  else:
    for i in range(8):
      if rfree[i] and ddfree[n + i] and dufree[n + 7 - i]:
        queen[n] = i
        rfree[i] = ddfree[n + i] = dufree[n + 7 - i] = False
        solve(n + 1)
        rfree[i] = ddfree[n + i] = dufree[n + 7 - i] = True

solutions = 0
solve(0)

print()
print(solutions, "solutions")
```

Una de las 92 soluciones posibles se muestra en la figura 3. ■

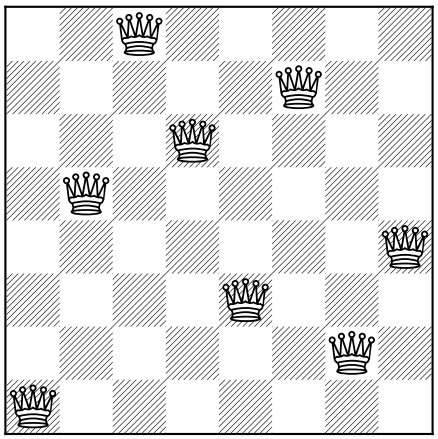

Figura 3: Una solución para el problema de 8 reinas.