Algoritmos Avanzados

Estrategias Algorítmicas
Backtracking

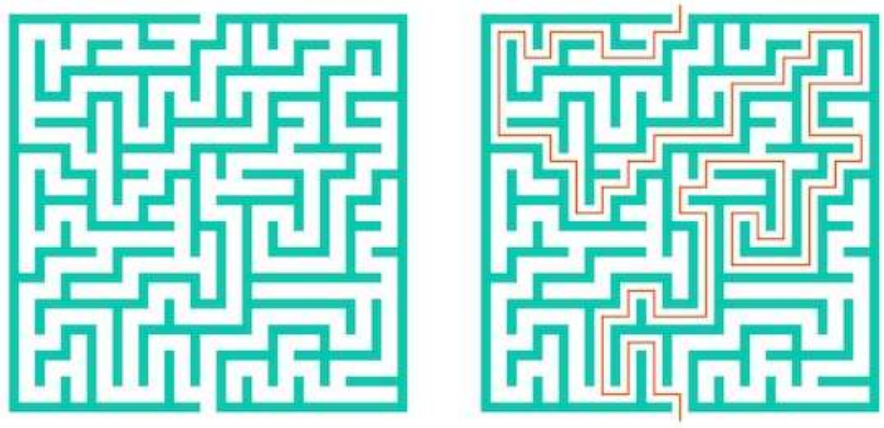
Rony Cueva
Manuel Tupia

Introducción

- Solución a problemas por ensayo y error.
- Viabilidad para resolver problemas radica en descomponer el proceso en ensayo y error en tareas parciales.
- Generalmente, las tareas parciales pueden ser realizadas recursivamente.

Introducción

- Puede iterar a través de todas las combinaciones posibles en el espacio de búsqueda.
- Puede ser adaptada para cada aplicación particular.
- Siempre puede encontrar todas las soluciones existentes.
- El Problema con este método es el tiempo que toma para encontrar las soluciones.



Introducción

- El proceso se basa en ir tomando decisiones (generalmente se evalúan exhaustivamente todas las posibles).
  - Una vez tomada una decisión, se evalúa el resto de la solución, dada la decisión tomada.
  - Si la decisión conduce a la solución total, el algoritmo termina.
  - Si la decisión no conduce a la solución total, se deshace la decisión y se toma otra disponible.
- La idea de deshacer acciones -> Backtracking

Algoritmo General

ALGORITHM *Backtrack*$(X[1..i])$

```
//Gives a template of a generic backtracking algorithm
//Input: X[1..i] specifies first i promising components of a solution
//Output: All the tuples representing the problem's solutions
if X[1..i] is a solution write X[1..i]
else      //see Problem 9 in this section's exercises
    for each element x ∈ S_{i+1} consistent with X[1..i] and the constraints do
        X[i + 1] ← x
        Backtrack(X[1..i + 1])
```

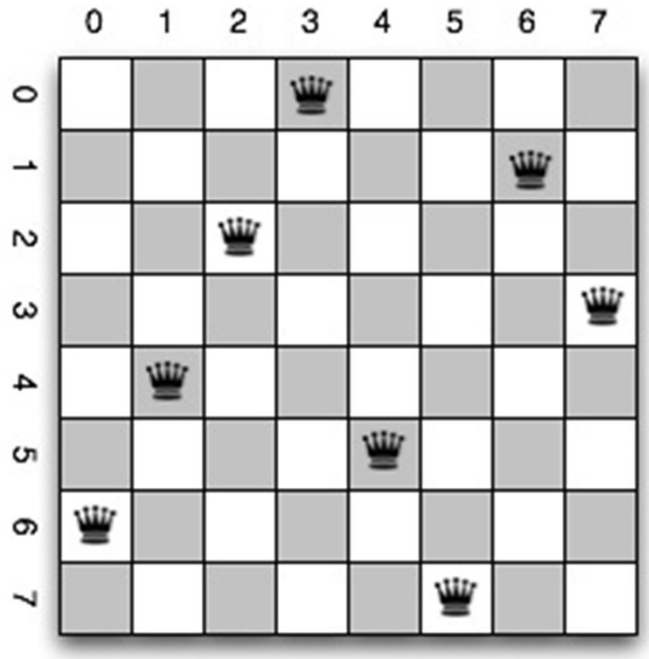
Conclusiones

En conclusión, es necesario decir tres cosas a favor del backtracking:

- Se aplica especialmente en problemas combinatorios difíciles para los cuales no hay algoritmos eficientes para encontrar soluciones exactas.
- A diferencia de la búsqueda exhaustiva que está condenada a ser extremadamente lenta en todos los casos del problema, backtracking alberga la esperanza de resolver el problema con un tiempo aceptable. Esto se aplica especialmente en la optimización, ya que retroceder puede brindar mejoraras en los resultados, evaluando la calidad de las soluciones parcialmente construidas.
- Un beneficio de esta estrategia es que si retrocede no se elimina ningún elemento del espacio de estados del problema y termina revisando todos sus elementos, adaptado el algoritmo para hacerlo.

Ejemplo: n-Reinas

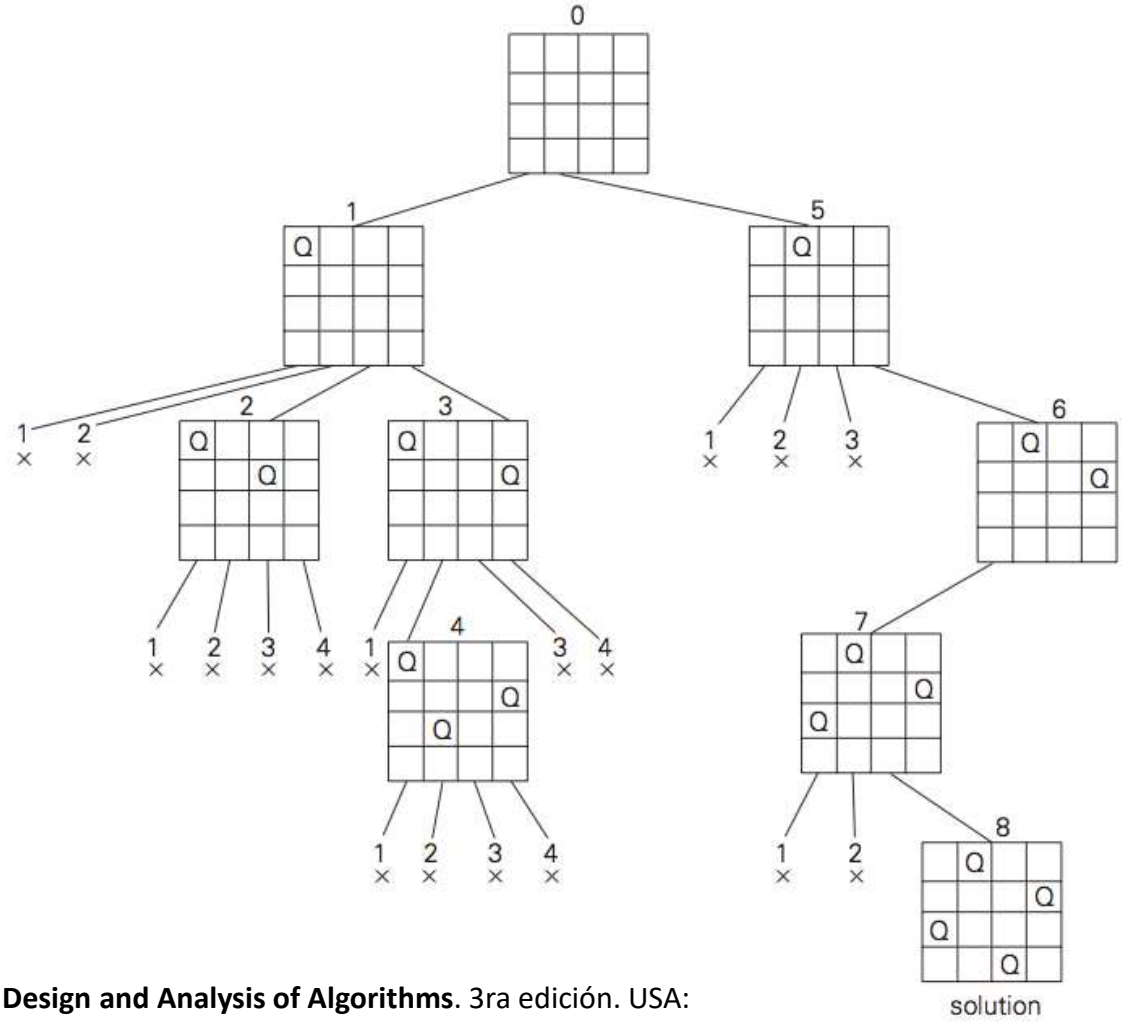
- En un tablero de ajedrez de n x n posiciones, colocar n reinas de tal manera que no se puedan atacar entre sí



n=8

Ejemplo: n-Reinas

- Proceso



Algoritmo: n-Reinas

- Empezar en la columna más a la izquierda
- Si todas las reinas han sido ubicadas, retornar VERDADERO
- Para (posible opción entre las filas de esta columna)
  - Si la reina puede ser colocada de forma segura aquí,
    - Escoger esta opción e intentar colocar las demás reinas recursivamente.
    - Si la recursión es exitosa, retornar VERDADERO
    - Caso contrario, remover la reina e intentar otra fila en la misma columna
- Si todas las filas han sido examinadas y nada funcionó, retornar FALSO (para desencadenar el backtracking)

Ejemplo: Sudoku

- Dada una matriz 9 x 9 parcialmente llena, el objetivo es asignar dígitos del 1 al 9 a las celdas vacías de tal forma que cada fila, columna y submatriz de 3 x 3 contenga exactamente una instancia de los dígitos del 1 al 9

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		