DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Búsqueda sin información

© Fernando Berzal, berzal@acm.org

Búsqueda sin información

- Recorridos sobre grafos
- Búsqueda en anchura (BFS)
- Búsqueda en profundidad (DFS)
- Descenso iterativo (IDS)
- Backtracking ["vuelta atrás"]

Búsqueda sin información

Supongamos que tenemos que tomar una serie de decisiones pero…

- No disponemos de suficiente información como para saber cuál elegir.
- Cada decisión nos lleva a un nuevo conjunto de decisiones.
- Alguna secuencia de decisiones (y puede que más de una) puede solucionar nuestro problema.

Búsqueda sin información

- Realizaremos una **búsqueda sin información**, también conocida como **búsqueda a ciegas** [blind search] cuando no exista información específica sobre el problema que nos ayude a determinar cuál es el mejor operador que se debería aplicar en cada momento o el mejor nodo por el que continuar la búsqueda.

- **Estrategia de búsqueda:**
  Se parte de un nodo dado y se visitan los vértices del grafo de manera ordenada y sistemática, pasando de un vértice a otro a través de las aristas del grafo.

Búsqueda sin información

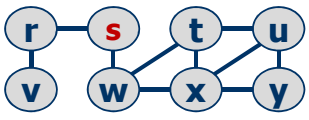
Criterios para la exploración del espacio de búsqueda:

- **Búsqueda primero en profundidad:**
  Estrategia LIFO [Last In First Out]
  Equivalente al recorrido en pre-orden de un árbol.

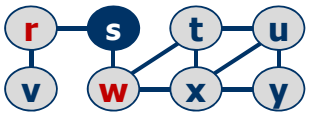
- **Búsqueda primero en anchura:**
  Estrategia FIFO [First In First Out]
  Equivalente al recorrido de un árbol por niveles.

Recorridos sobre grafos

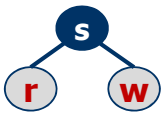
Búsqueda primero en profundidad
[DFS: Depth-First Search]

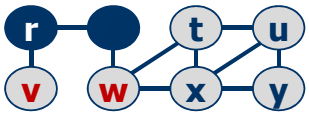


Pila S={}

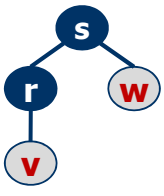


Pila S={s}



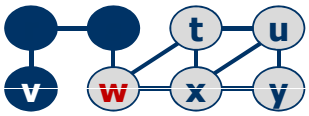


Pila S={r,s}

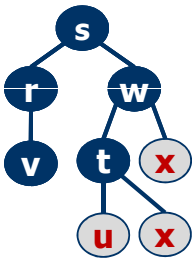


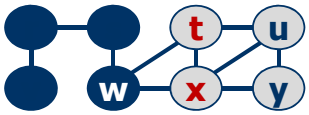
Recorridos sobre grafos

Búsqueda primero en profundidad
[DFS: Depth-First Search]

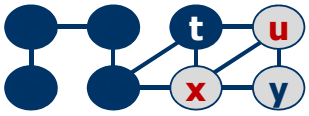


Pila S={v,r,s}





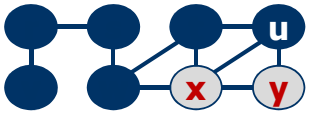
Pila S={w,s}



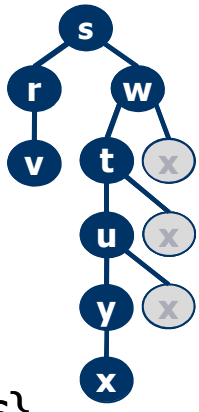
Pila S={t,w,s}

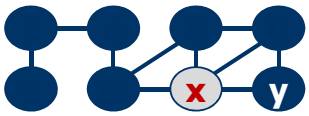
Recorridos sobre grafos

Búsqueda primero en profundidad
[DFS: Depth-First Search]

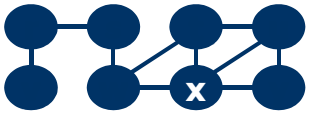


Pila S={u,t,w,s}





Pila S={y,u,t,w,s}



Pila S={x,y,u,t,w,s}

Recorridos sobre grafos

Búsqueda primero en profundidad
[DFS: Depth-First Search]

- Es necesario llevar la cuenta de los nodos visitados.
- El orden de visita de los nodos puede interpretarse como un árbol: **árbol de expansión en profundidad** asociado al grafo.

Recorridos sobre grafos

Búsqueda primero en profundidad
[DFS: Depth-First Search]

```
función DFS (Grafo G(V,E))
{
  for (i=0; i<V.length; i++)
      visitado[i] = false;

  for (i=0; i<V.length; i++)
      if (!visitado[i])
         DFS(G,i);
}
```

Recorridos sobre grafos

Búsqueda primero en profundidad

[DFS: Depth-First Search]

```
función DFS (Grafo G(V,E), int i)
{
  visitado[i] = true;

  foreach (v[j] adyacente a v[i])
          if (!visitado[j])
             DFS(G,j);
}
```

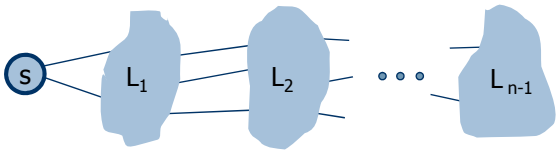
O(|V|+|E|)

si usamos la representación basada en listas de adyacencia.

Recorridos sobre grafos

Búsqueda primero en anchura

[BFS: Breadth-First Search]



Exploración desde s por niveles:

- L_0 = { s }.
- L_1 = Nodos adyacentes a L_0.
- L_2 = Nodos adyacentes a L_1 que no pertenecen ni a L_0 ni a L_1.
- L_{i+1} = Nodos que, sin pertenecer a ningún nivel anterior, están conectados con L_i a través de una arista.

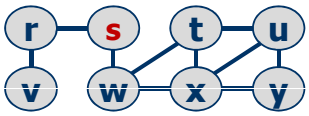
Teorema:

L_i contiene todos los nodos que están a distancia i de s.

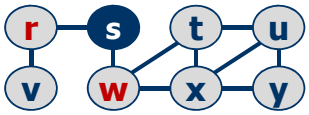
Recorridos sobre grafos

Búsqueda primero en anchura

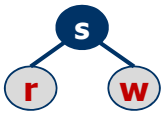
[BFS: Breadth-First Search]

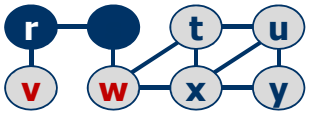


Cola Q={s}

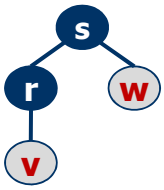


Cola Q={r,w}





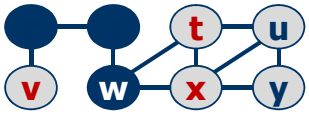
Cola Q={w,v}



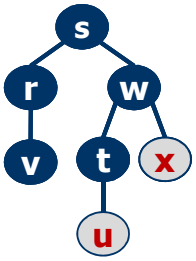
Recorridos sobre grafos

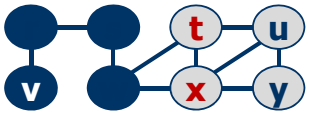
Búsqueda primero en anchura

[BFS: Breadth-First Search]

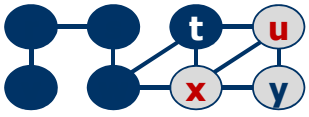


Cola Q={v,t,x}





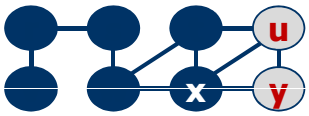
Cola Q={t,x}



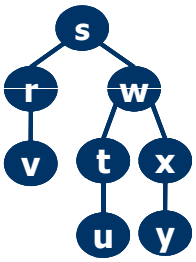
Cola Q={x,u}

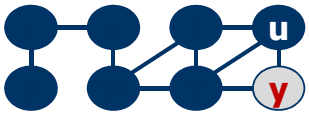
Recorridos sobre grafos

Búsqueda primero en anchura
[BFS: Breadth-First Search]

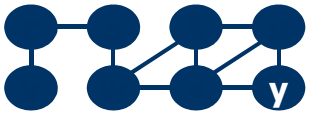


Cola Q={u,y}





Cola Q={y}



Cola Q={}

Recorridos sobre grafos

Búsqueda primero en anchura
[BFS: Breadth-First Search]

- Empezando en un nodo v:
  - Primero se visita v.
  - Luego se visitan todos sus vértices adyacentes.
  - A continuación, los adyacentes a éstos…
    y así sucesivamente.
- El algoritmo utiliza una **cola de vértices.**
- Es necesario conocer los nodos ya visitados.

Recorridos sobre grafos

Búsqueda primero en anchura
[BFS: Breadth-First Search]

```
función BFS (Grafo G(V,E))
{
  for (i=0; i<V.length; i++)
      visitado[i] = false;

  for (i=0; i<V.length; i++)
      if (!visitado[i])
         BFS(G,i);
}
```

Recorridos sobre grafos

Búsqueda primero en anchura
[BFS: Breadth-First Search]

O(|V|+|E|)
si usamos la representación basada en listas de adyacencia

```
función BFS (Grafo G(V,E), int v)
{
  Cola Q;

  visitado[v]=true; Q.add(v);

  while (!Q.empty()) {
        x = Q.extract();
        foreach (v[y] adyacente a v[x])
                if (!visitado[y]) {
                   visitado[y]=true; Q.add(y);
                }
  }
}
```

Búsqueda sin información

Árbol de búsqueda

- Debido a la complejidad del grafo del espacio de estados, sólo se irá generando una porción del grafo conforme avance el proceso de búsqueda.
- Para que el algoritmo termine, deberemos controlar qué nodos hemos visitado ya.

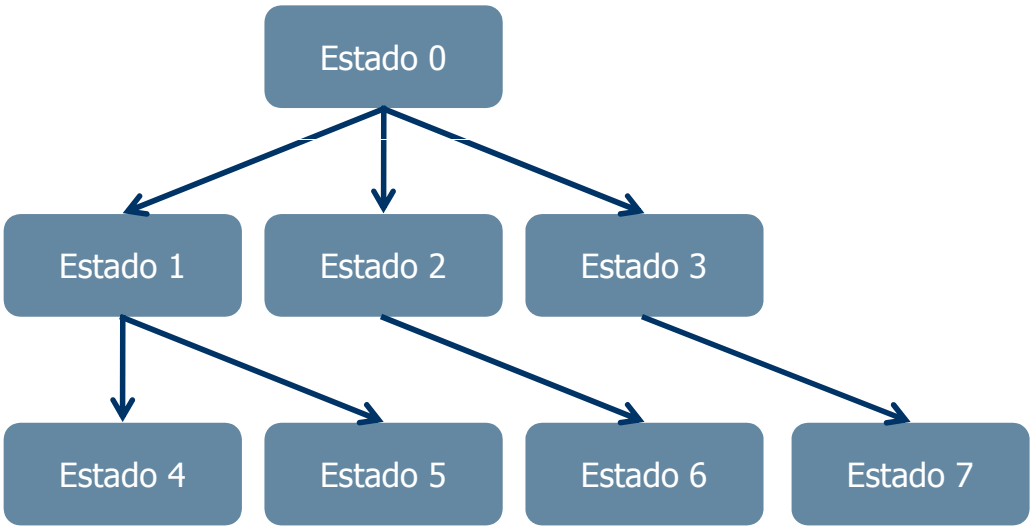
Búsqueda sin información

Condiciones de parada

- Se ha encontrado la solución.
- Se ha acabado el tiempo disponible.
- Se ha llegado a un nivel de profundidad determinado.

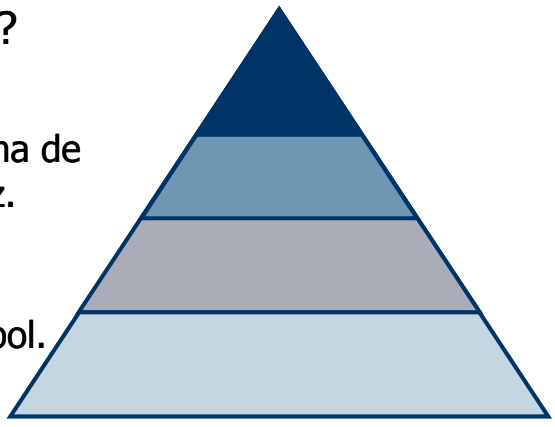
Búsqueda en anchura

Exploración en anchura vista como un árbol



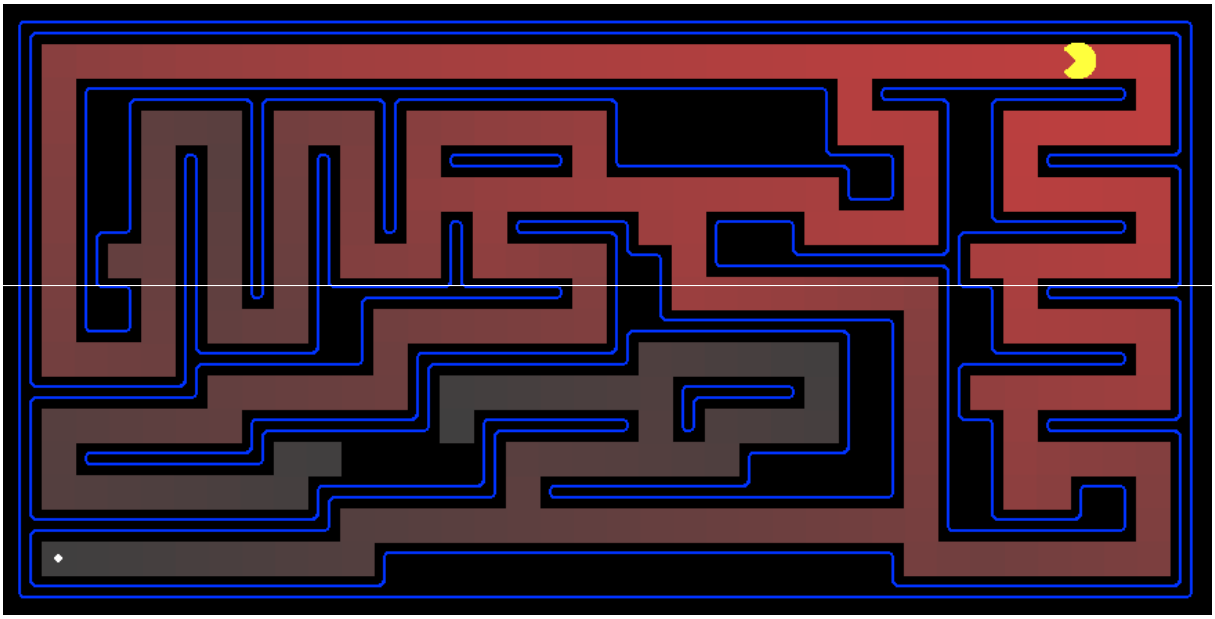
Búsqueda en anchura

- ¿Cuántos nodos expande BFS?
  - Todos los que quedan por encima de la solución más cercana a la raíz.
  - Tiempo **$O(b^s)$**
    b: Factor de ramificación del árbol.
    s: Profundidad de la solución.



- ¿Cuántos nodos puede haber en la frontera de búsqueda?
  - Los del nivel al que se encuentra la solución, **$O(b^s)$**
- Si s es finito, el algoritmo encuentra la solución.

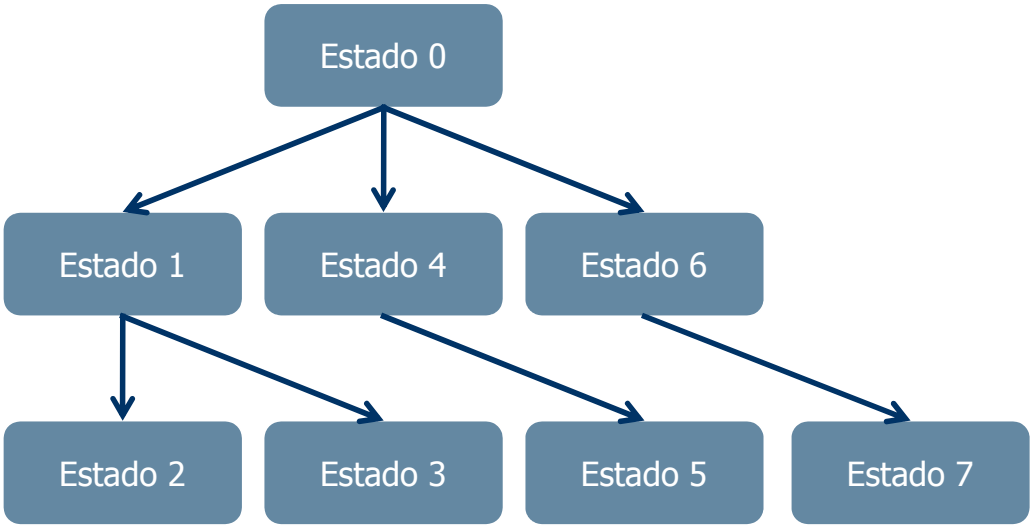
Búsqueda en anchura



Demo

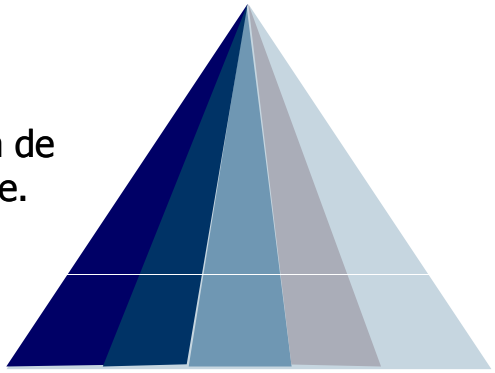
Búsqueda en profundidad

Exploración en profundidad vista como un árbol



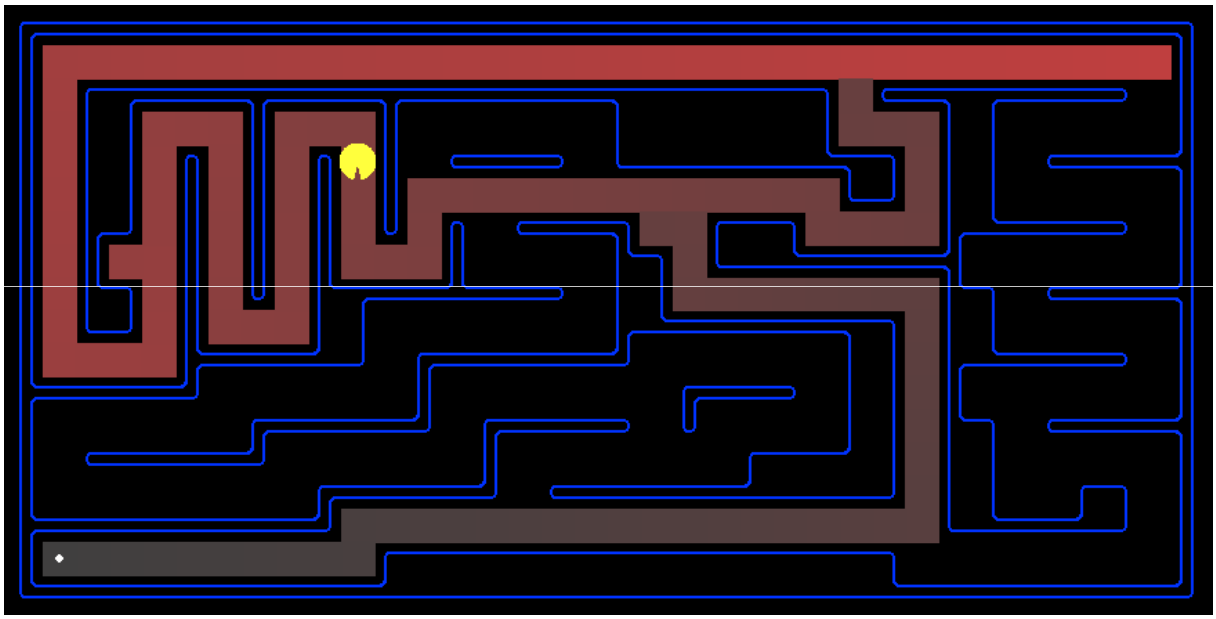
Búsqueda en profundidad

- ¿Cuántos nodos expande DFS?
  - Todos los que quedan a la izquierda de la primera solución que se encuentre.
  - Tiempo **O(b^m)**
    b: Factor de ramificación del árbol.
    m: Profundidad del árbol.



- ¿Cuántos nodos puede haber en la frontera de búsqueda?
  - Los hermanos del camino desde la raíz, **O(bm)**
- El árbol puede ser infinito,
  por lo que puede que no encontremos la solución.

Búsqueda en profundidad



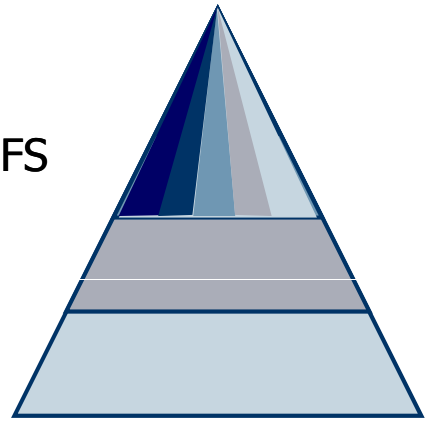
Demo

Descenso iterativo

IDS: Iterative Deepening Search

IDEA

Combinar las ventajas de espacio de DFS y las ventajas de tiempo de BFS:

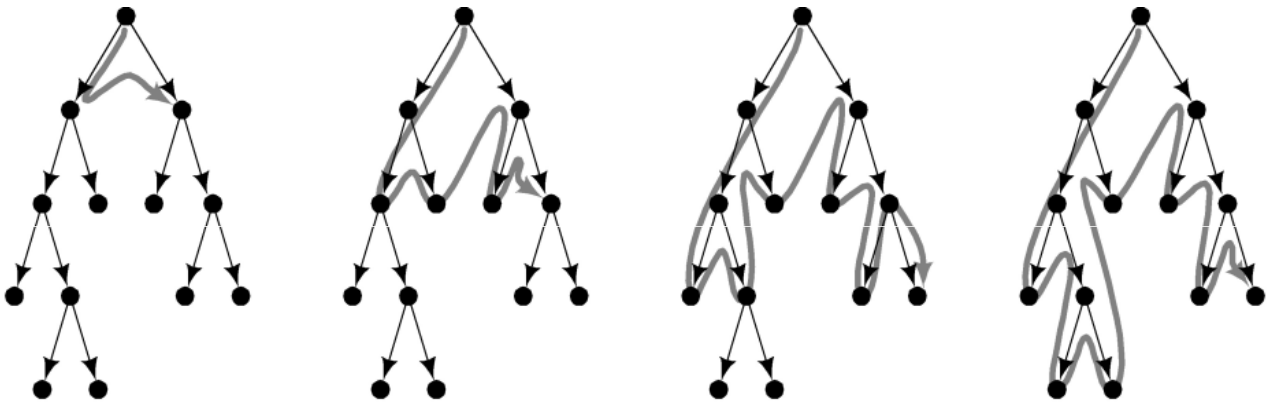


- Ejecutar DFS con límite de profundidad 1
- Ejecutar DFS con límite de profundidad 2
- …
- Ejecutar DFS con límite de profundidad s

¿No resulta demasiado redundante/lento?

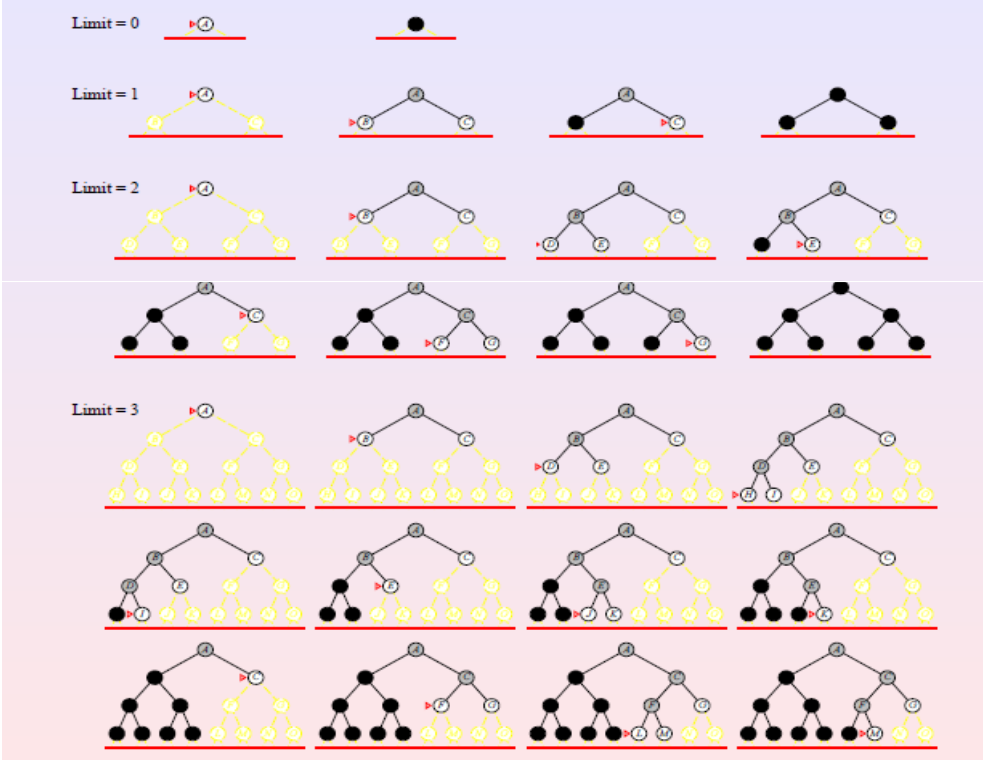
Generalmente, la mayor parte del trabajo se hace en el nivel inferior del árbol, por lo que suele compensar…

Descenso iterativo



Depth bound = 1 | Depth bound = 2 | Depth bound = 3 | Depth bound = 4

Descenso iterativo



Backtracking

¿Se puede mejorar el proceso de exploración?

Sí, siempre y cuando podamos eliminar la necesidad de tener que explorar el árbol completo de búsqueda.

¿Cuándo?

Cuando para un nodo interno del árbol podemos asegurar que desde ahí no alcanzaremos una solución, entonces podemos podar una rama completa del árbol.

¿Cómo?

Realizamos una vuelta atrás ("backtracking").

VENTAJA: Alcanzamos antes la solución.

Backtracking

Estrategia tentativa retroactiva

Backtracking ("vuelta atrás")

- Realizamos una búsqueda en profundidad.

- ¿Cuando termina el proceso de búsqueda?
  - Cuando hemos llegado a una solución (y no deseamos encontrar otras soluciones).
  - Cuando no hay más operadores aplicables al nodo raíz del árbol de búsqueda.

Backtracking

Estrategia tentativa retroactiva

Backtracking ("vuelta atrás")

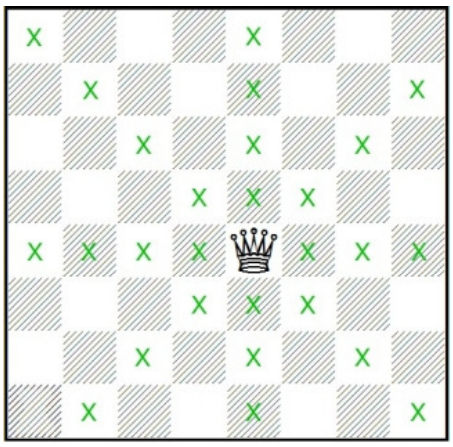
¿Cuando se produce una vuelta atrás o retroceso?

- Cuando se ha encontrado una solución, pero deseamos encontrar otra solución alternativa.
- Cuando se ha llegado a un límite en el nivel de profundidad explorado o el tiempo de exploración en una misma rama.
- Cuando no existen reglas aplicables al último nodo de la lista (último nodo del grafo explícito).

Backtracking
El problema de las 8 reinas

Problema combinatorio clásico:

Colocar ocho reinas en un tablero de ajedrez de modo que no haya dos que se ataquen; (que estén en la misma fila, columna o diagonal).



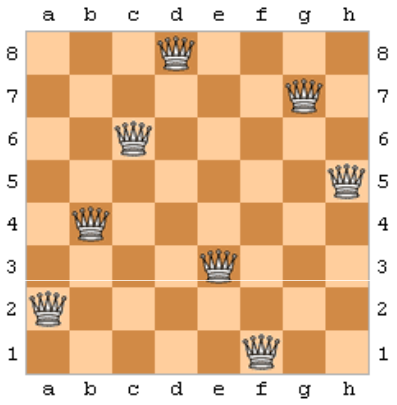
- Como cada reina debe estar en una fila diferente, sin pérdida de generalidad podemos suponer que la reina i se coloca en la fila i.
- Todas las soluciones para este problema pueden representarse como 8-tuplas $(x_1,...,x_8)$ en las que x_i indica la columna en la que se coloca la reina i.

Backtracking
El problema de las 8 reinas

Restricciones explícitas:

$S_i = \{1,2,3,4,5,6,7,8\}$, $1 \le i \le 8$

Espacio de soluciones:
Tamaño $|S_i|^8 = 8^8 = 2^{24} = 16M$



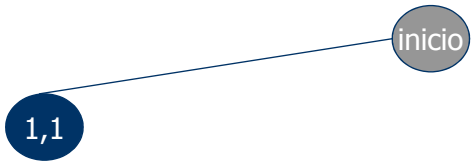
Restricciones implícitas:

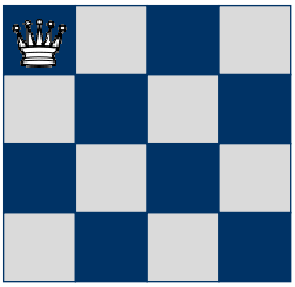
- Ningún par (x_i, x_j) con $x_i=x_j$ (todas las reinas deben estar en columnas diferentes).
- Ningún par (x_i, x_j) con $|j-i|=|x_j-x_i|$ (todas las reinas deben estar en diagonales diferentes).

NOTA:
La primera de las restricciones implícitas implica que todas las soluciones son permutaciones de {1,2,3,4,5,6,7,8}, lo que reduce el espacio de soluciones de 8^8 tuplas a 8! = 40320.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

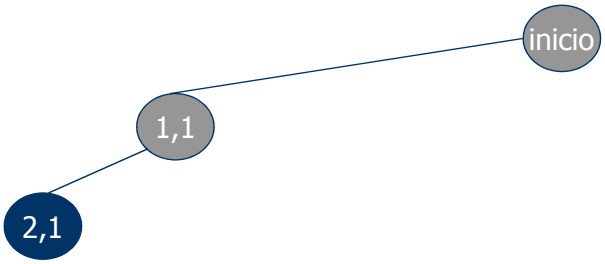


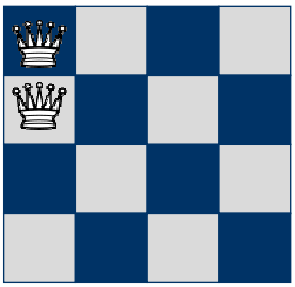


OK! Adelante con la búsqueda…

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

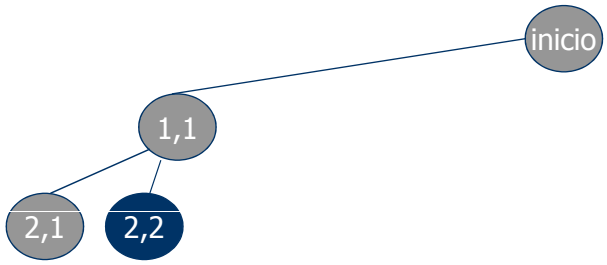


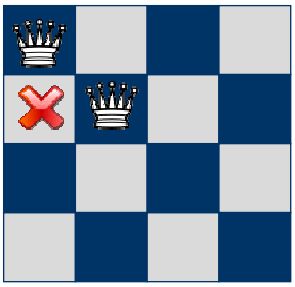


No se cumplen las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

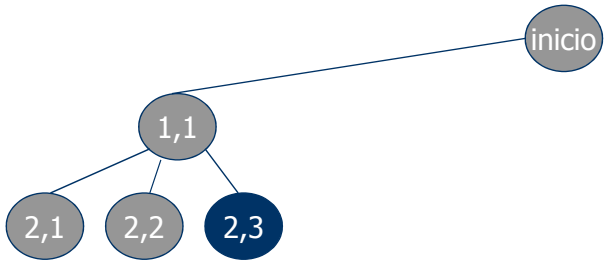


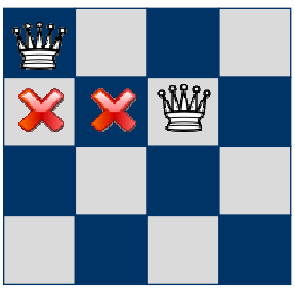


No se cumplen las restricciones: Misma diagonal.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

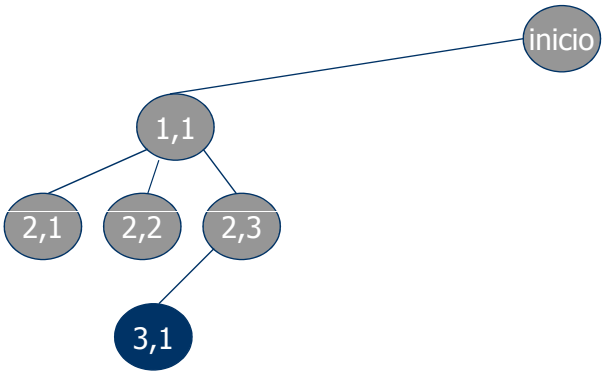


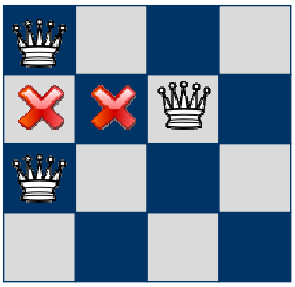


OK! Adelante con la búsqueda…

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

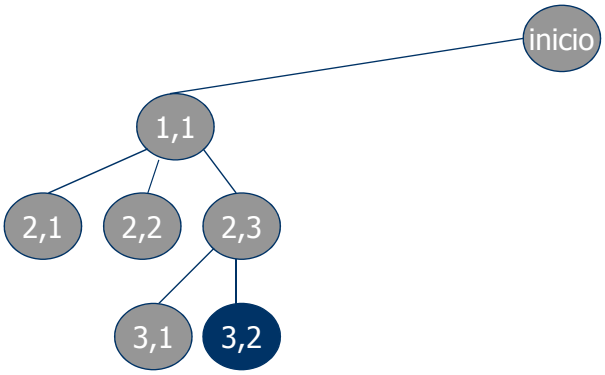


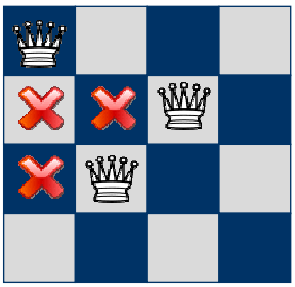


No se cumplen las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

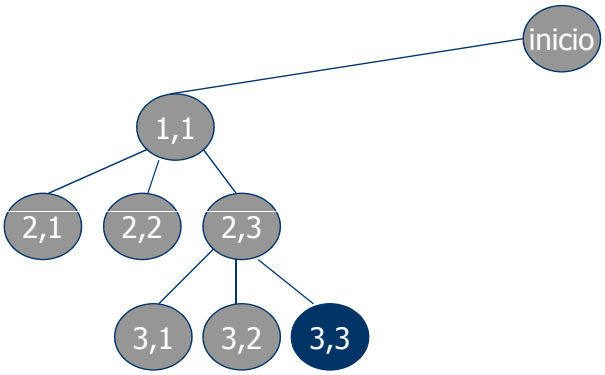


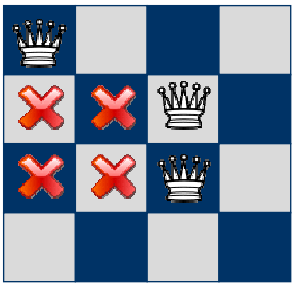


No se cumplen las restricciones: Misma diagonal.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

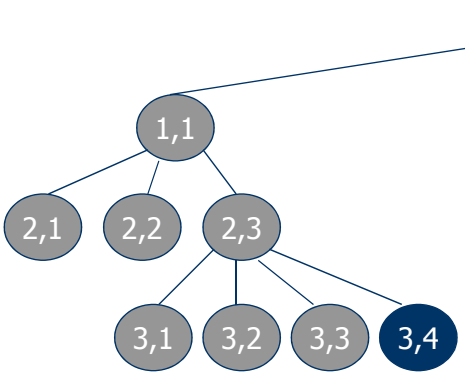


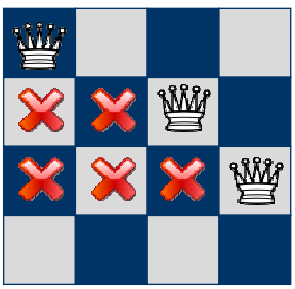


No se cumplen las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

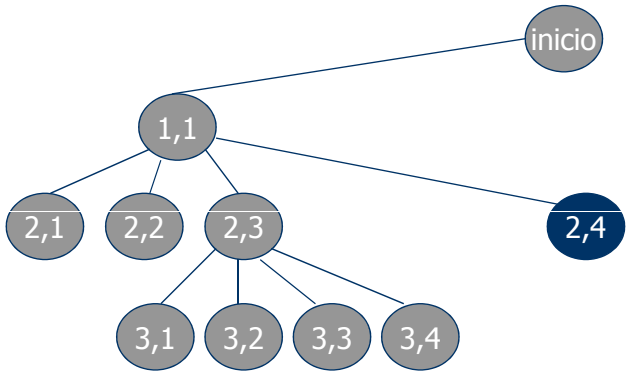


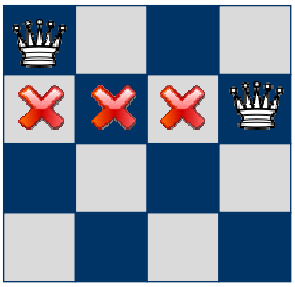


No se cumplen las restricciones: Misma diagonal.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

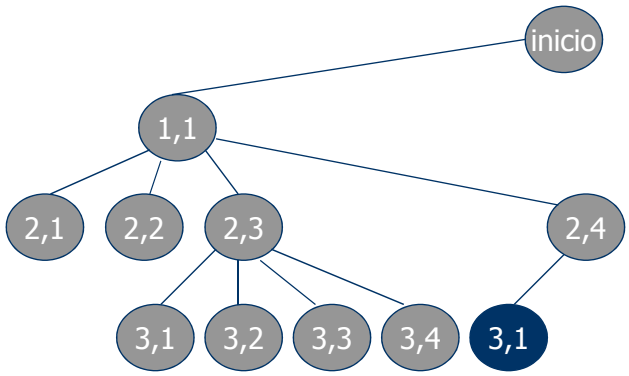


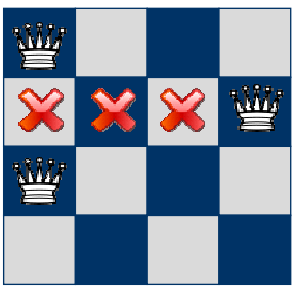


OK! Adelante con la búsqueda…

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

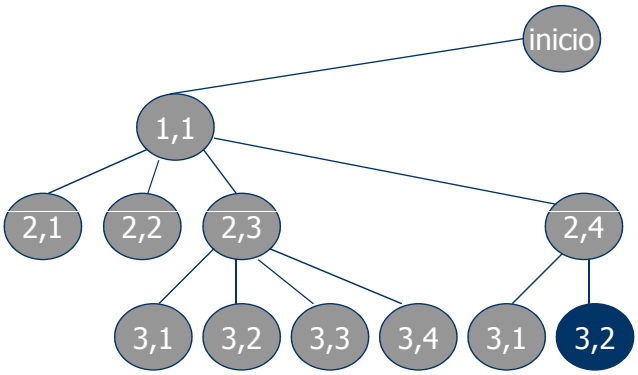


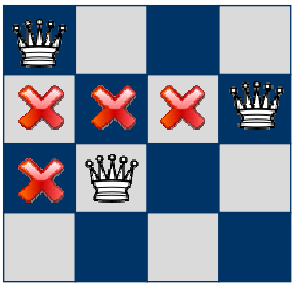


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

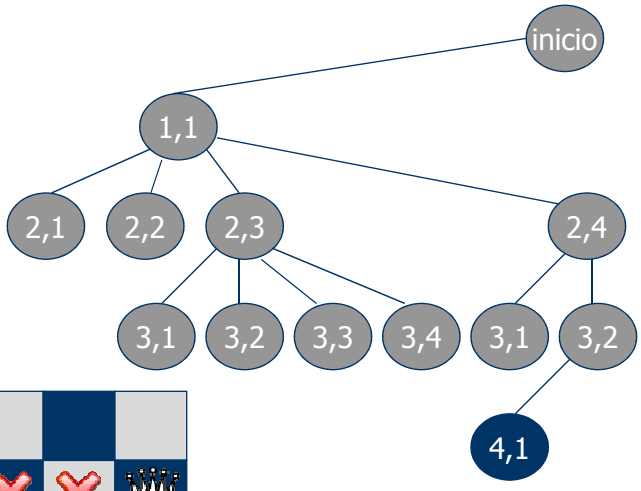


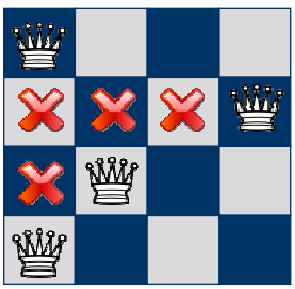


OK! Adelante con la búsqueda…

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

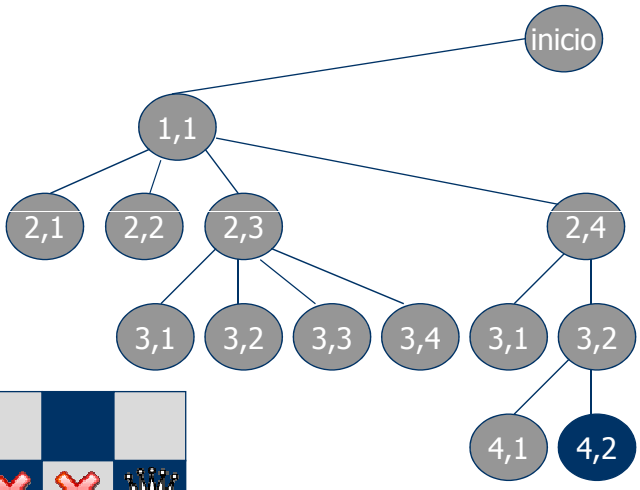


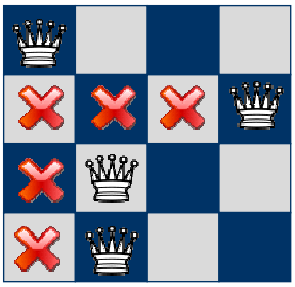


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

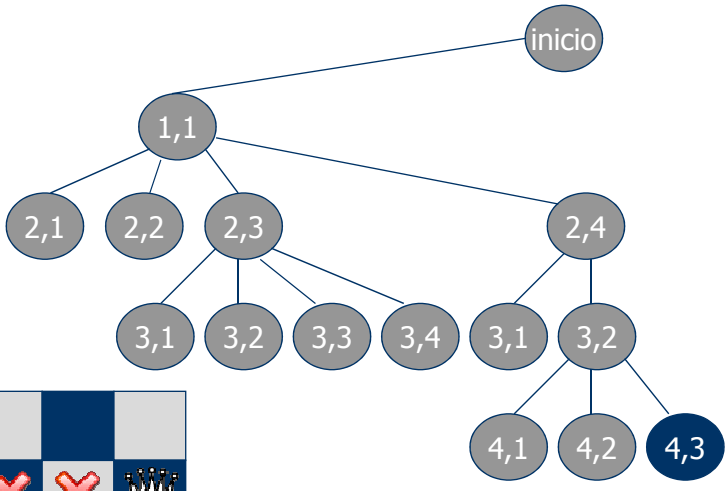


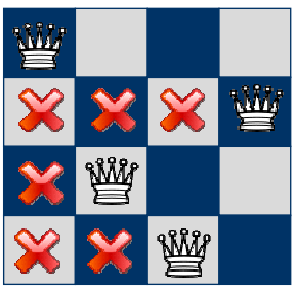


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

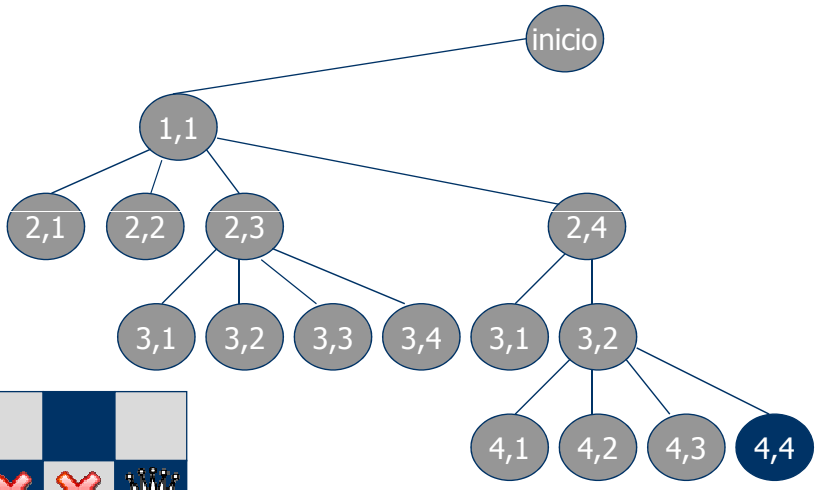


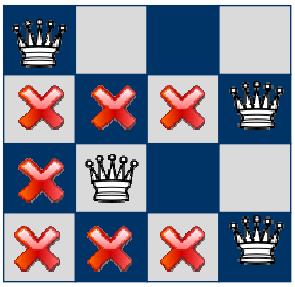


No cumple las restricciones: Misma diagonal.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

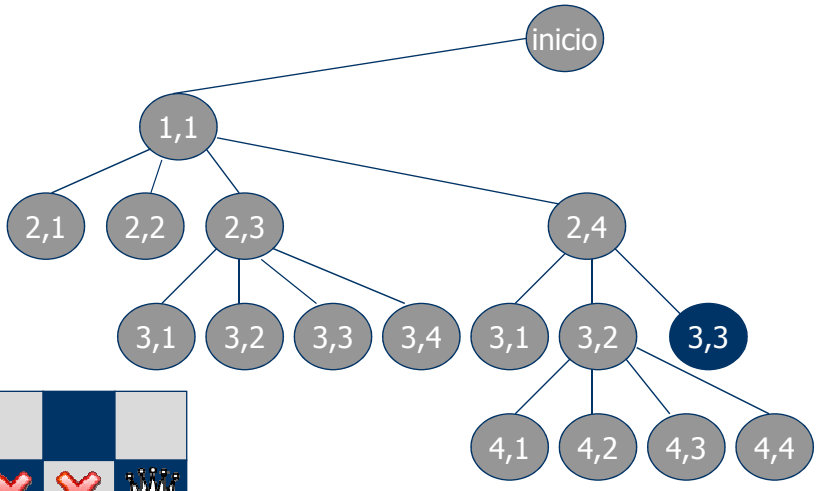


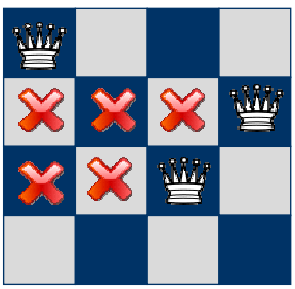


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

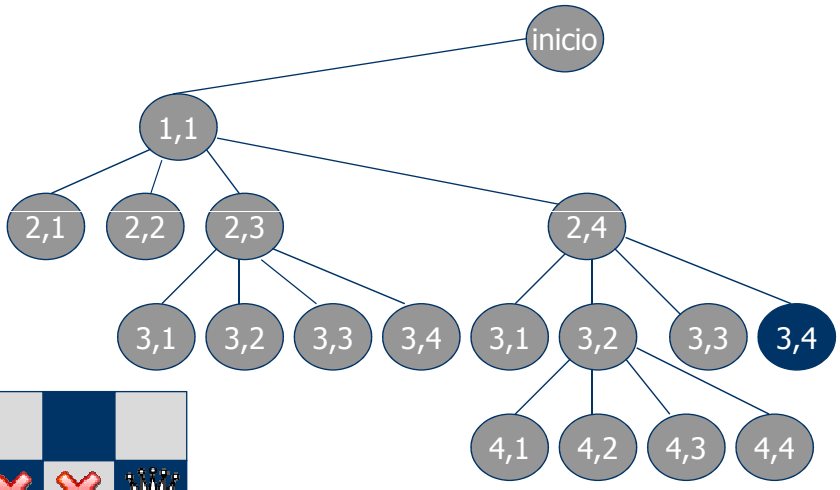


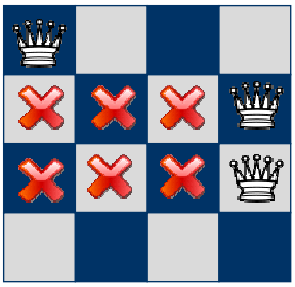


No cumple las restricciones: Misma diagonal.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

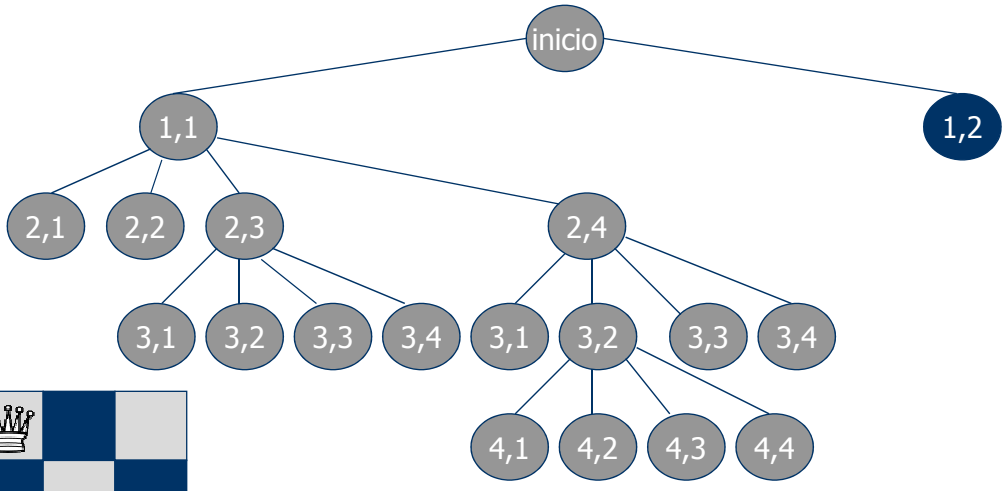


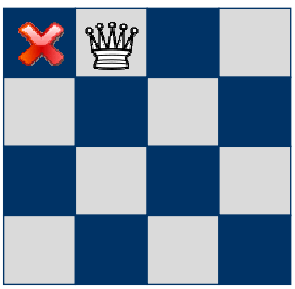


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

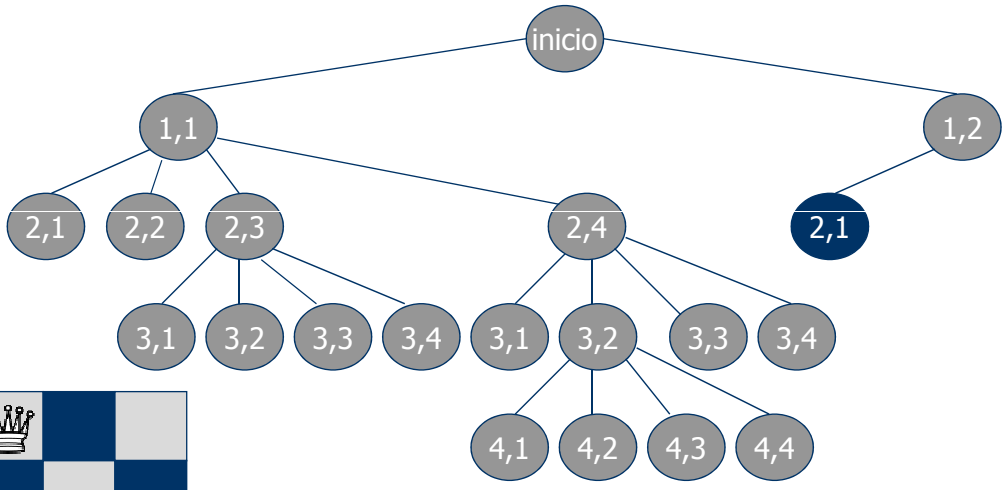


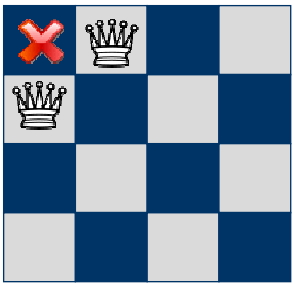


OK! Adelante con la búsqueda…

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

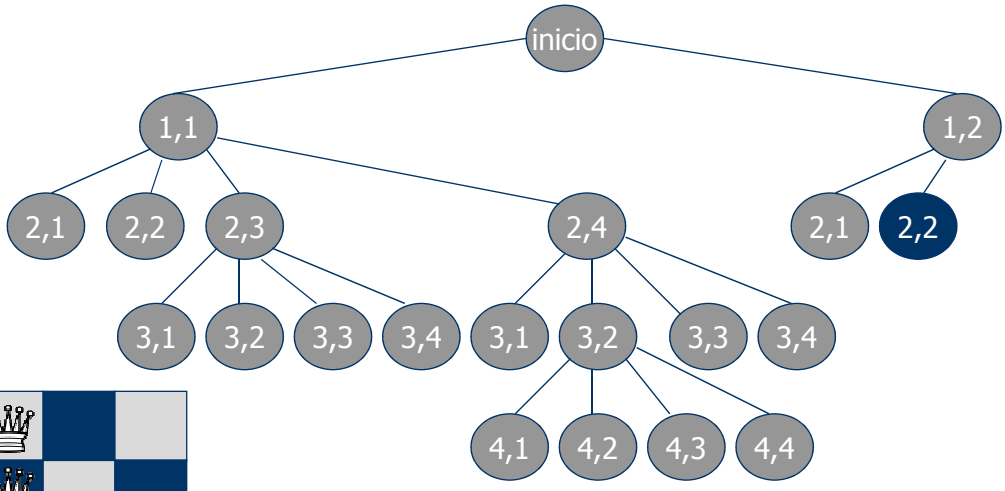


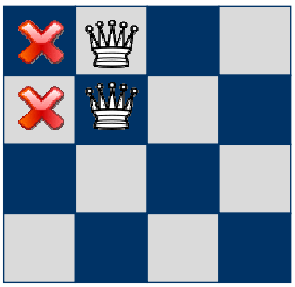


No cumple las restricciones: Misma diagonal.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

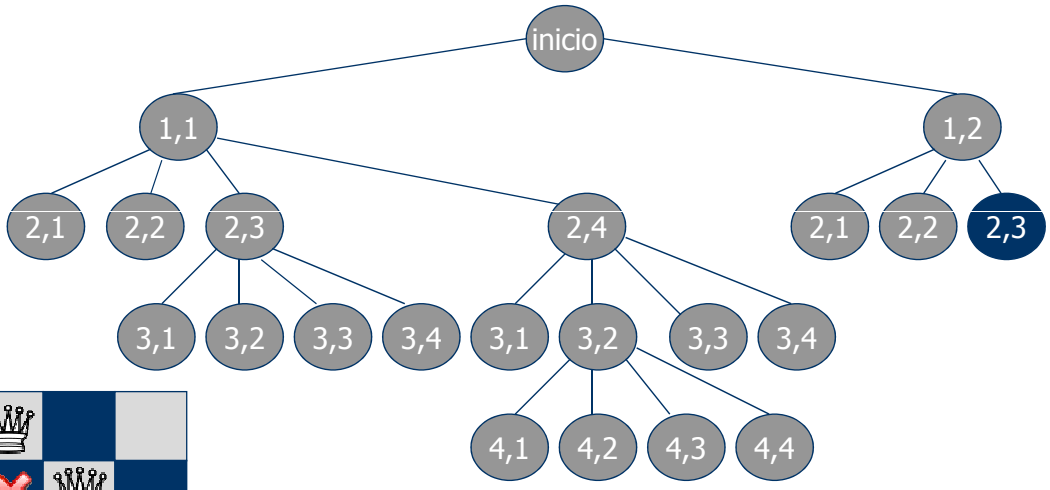


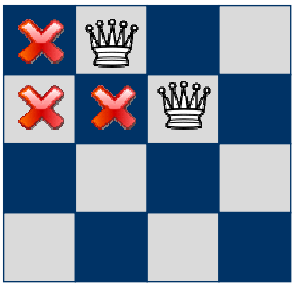


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

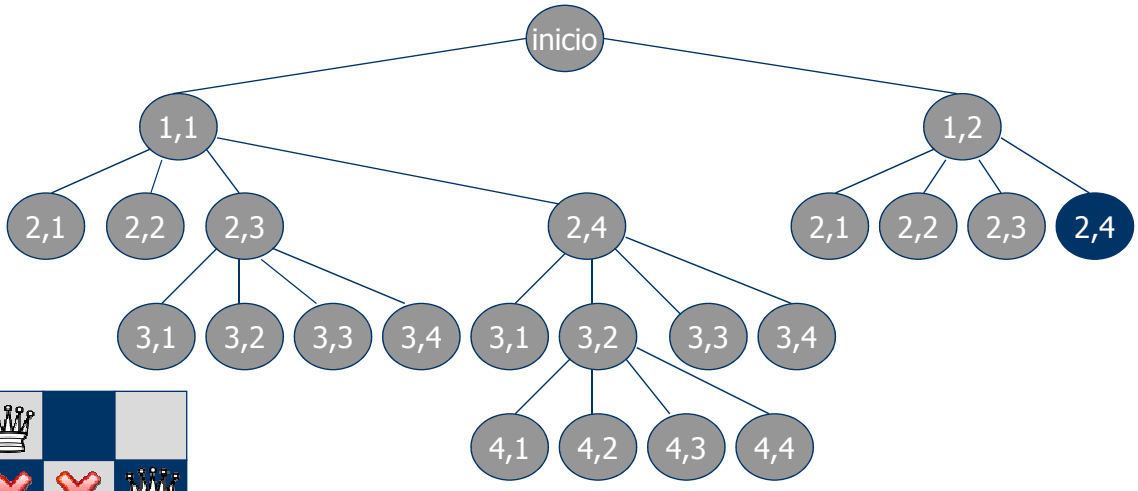


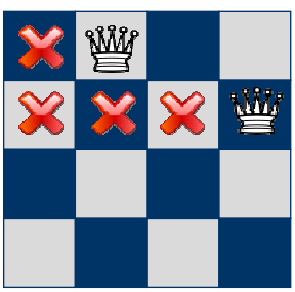


No cumple las restricciones: Misma diagonal.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

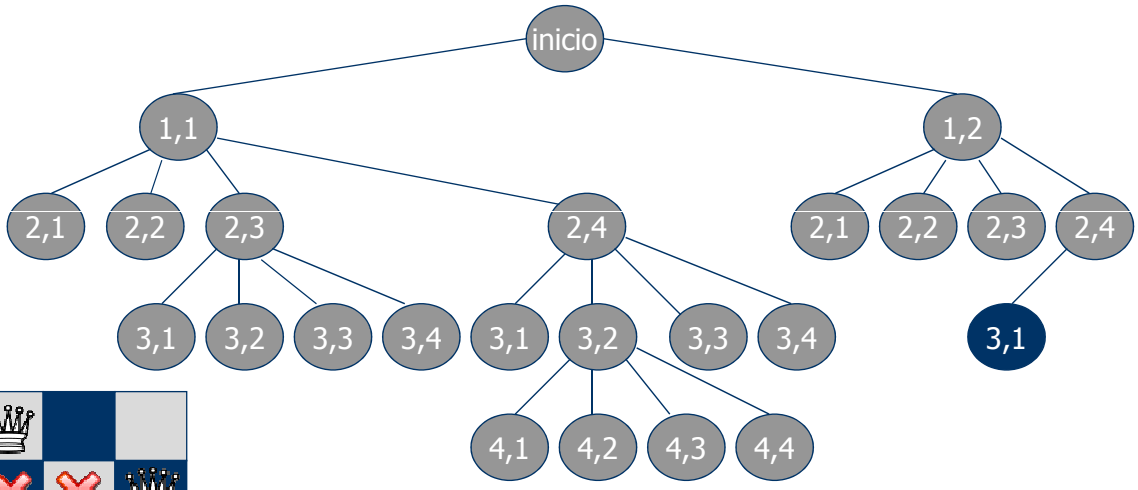


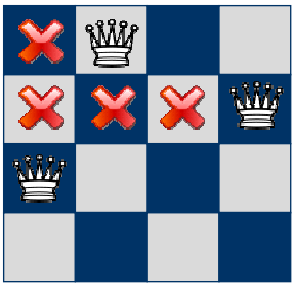


OK! Adelante con la búsqueda

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

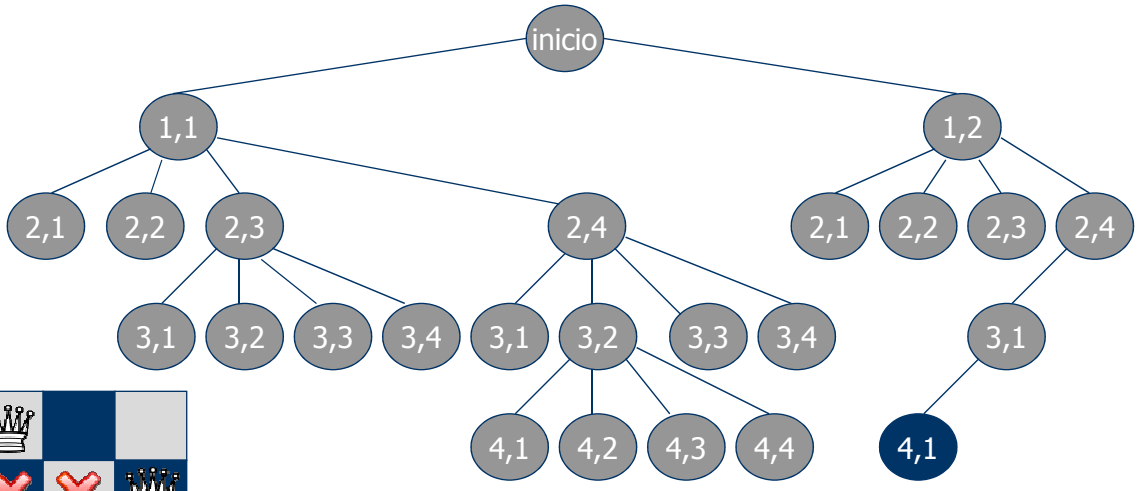


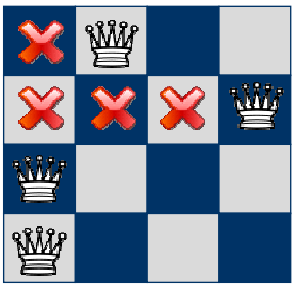


OK! Adelante con la búsqueda

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

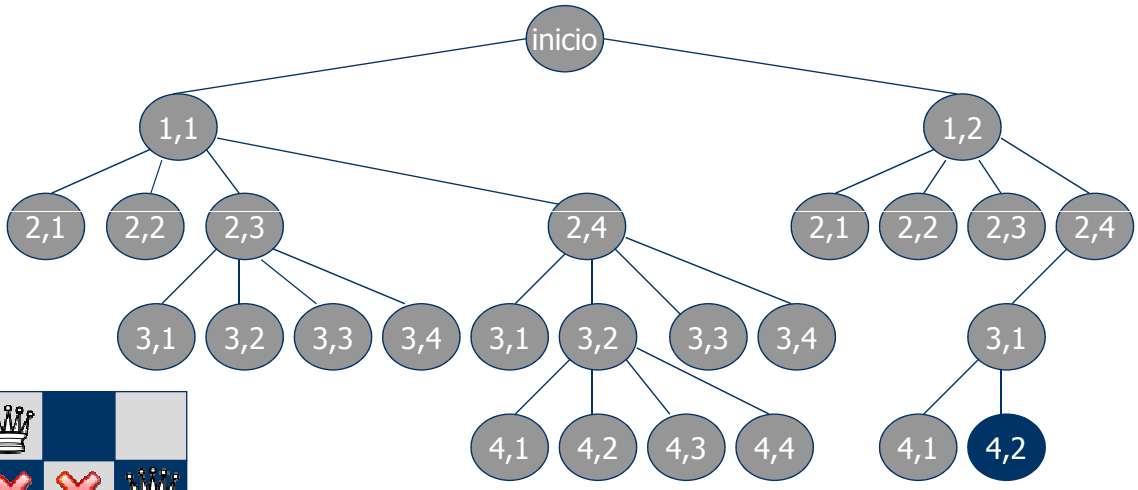


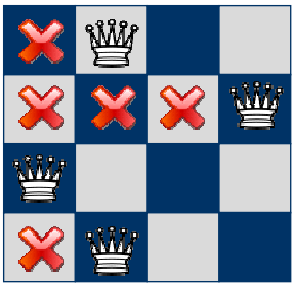


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

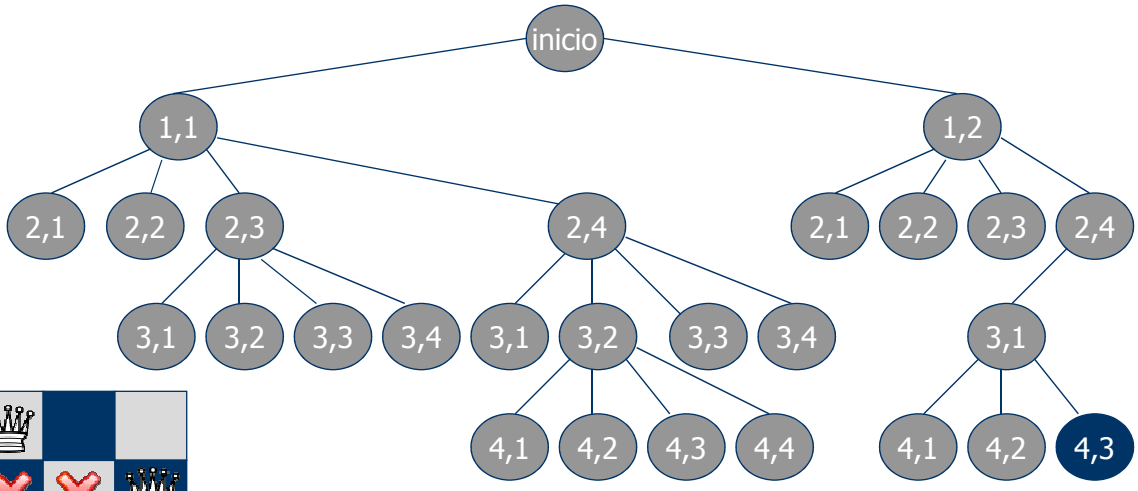


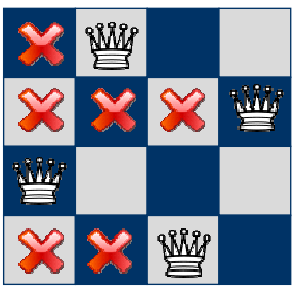


No cumple las restricciones: Misma columna.

Backtracking
El problema de las 4 reinas

Generación de estados usando backtracking

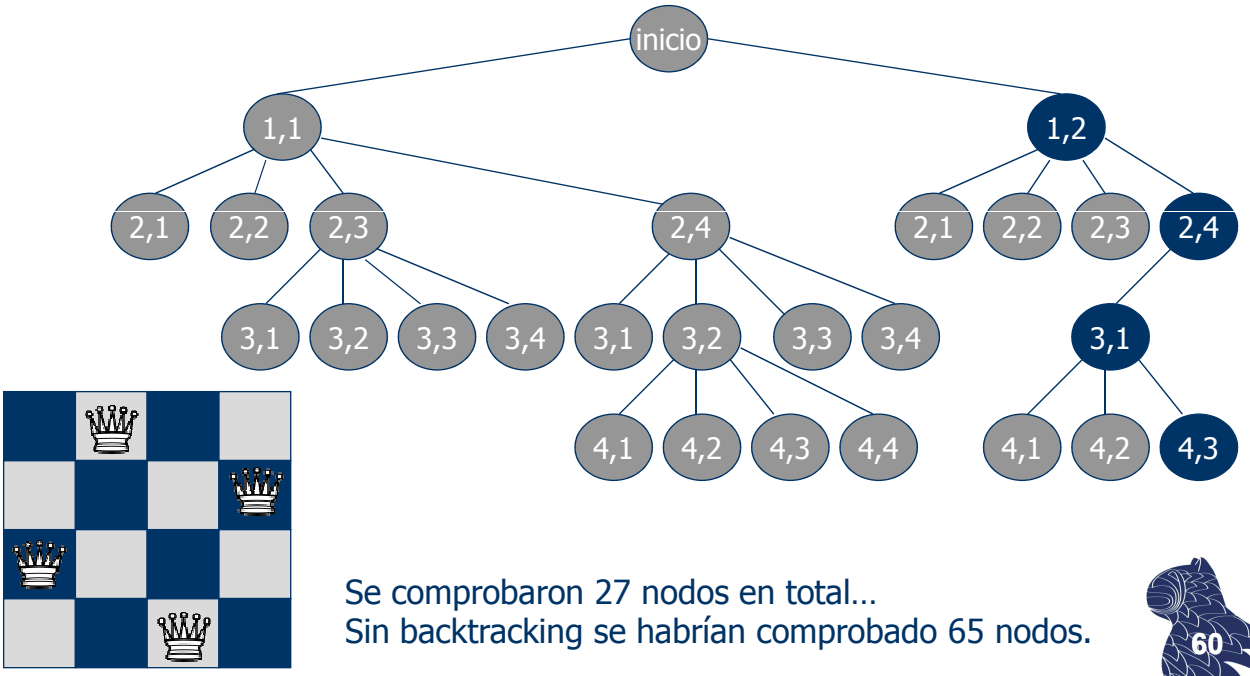




¡OK! Se encontró la solución

Backtracking
El problema de las 4 reinas

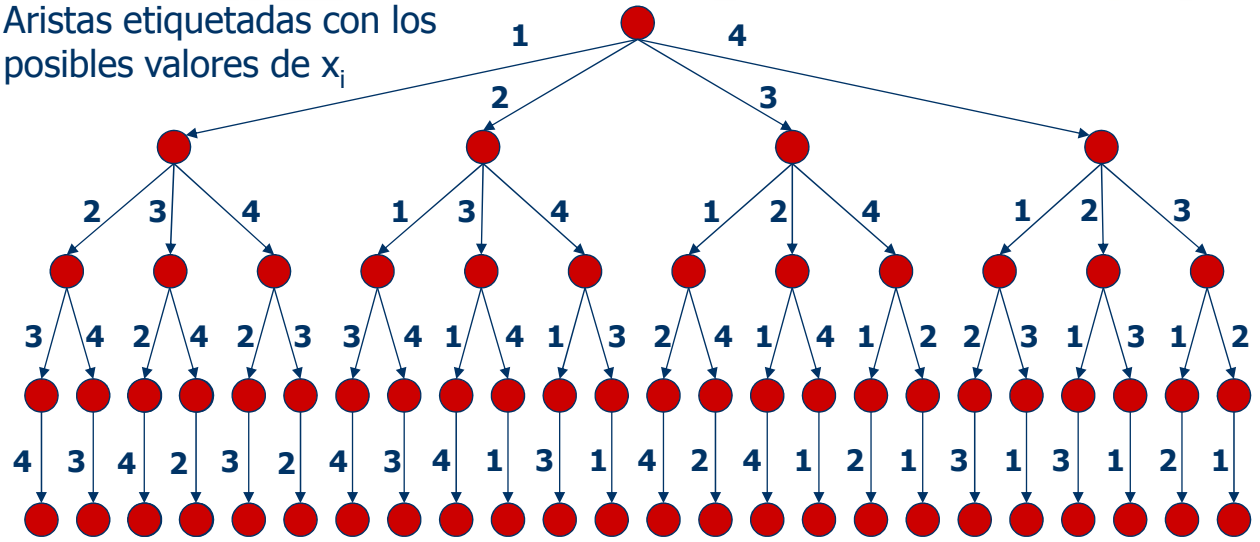
Generación de estados usando backtracking



Se comprobaron 27 nodos en total…
Sin backtracking se habrían comprobado 65 nodos.

Backtracking
El problema de las 4 reinas

Aristas etiquetadas con los posibles valores de x_i

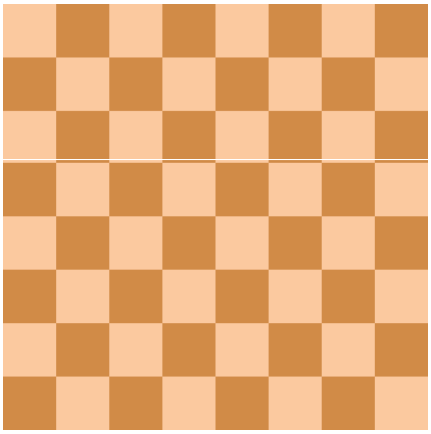


Las aristas desde los nodos del nivel i al i+1 están etiquetadas con los valores de x_i
p.ej. La rama más a la izquierda representa la solución $x_1=1$, $x_2=2$, $x_3=3$ y $x_4=4$.

El espacio de búsqueda viene definido por todos los caminos desde el nodo raíz a un nodo hoja (4! permutaciones ⇒ 4! nodos hoja).

Backtracking
El problema de las 8 reinas

Generación de estados usando backtracking



Backtracking
El problema de las N reinas

Implementación

Tablero NxN en el que colocar N reinas que no se ataquen.

- **Solución**: $(x_0,x_2,x_3,\ldots,x_{n-1})$, donde x_i es la columna de la i-ésima fila en la que se coloca la reina i.
- **Restricciones implícitas**: $x_i \in \{0..n-1\}$.
- **Restricciones explícitas**: No puede haber dos reinas en la misma columna ni en la misma diagonal.

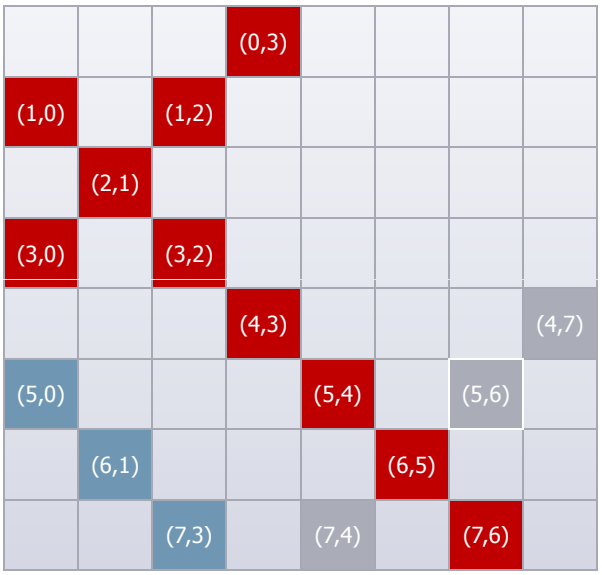
Backtracking
El problema de las N reinas

Implementación

- Distinta columna:
  Todos los x_i diferentes.

- Distinta diagonal:
  Las reinas (i,j) y (k,l) están en la misma diagonal si **i-j=k-l** o bien **i+j=k+l**, lo que se puede resumir en |j-l| = |k-i|.

  En términos de los x_i, tendremos $|x_k-x_i|=|k-i|$.



Backtracking
El problema de las N reinas

Implementación

```
// Comprobar si la reina de la fila k está bien colocada
// (si no está en la misma columna ni en la misma diagonal
// que cualquiera de las reinas de las filas anteriores)
// Eficiencia: O(k-1).

bool comprobar (int reinas[], int n, int k)
{
  int i;

  for (i=0; i<k; i++)
      if (  ( reinas[i]==reinas[k] )
         || ( abs(k-i) == abs(reinas[k]-reinas[i]) ) )
         return false;

  return true;
}
```

Backtracking
El problema de las N reinas

Implementación recursiva mostrando todas las soluciones

```
// Inicialmente, k=0 y reinas[i]=-1.

void NReinas (int reinas[], int n, int k)
{
  if (k==n) {            // Solución (no quedan reinas por colocar)
     print(reinas,n);
  } else {                  // Aún quedan reinas por colocar (k<n)
     for (reinas[k]=0; reinas[k]<n; reinas[k]++)
         if (comprobar(reinas,n,k))
            NReinas (reinas, n, k+1);
  }
}
```

Backtracking
El problema de las N reinas

Implementación recursiva mostrando sólo la primera solución

```
bool NReinas (int reinas[], int n, int k)
{
  bool ok = false;
  if (k==n) {            // Caso base: No quedan reinas por colocar
     ok = true;
  } else {                  // Aún quedan reinas por colocar (k<n)
     while ((reinas[k]<n-1) && !ok) {
           reinas[k]++;
           if (comprobar(reinas,n,k))
              ok = NReinas (reinas, n, k+1);
     }
  }
  return ok;  // La solución está en reinas[] cuando ok==true
}
```

Backtracking
El problema de las N reinas

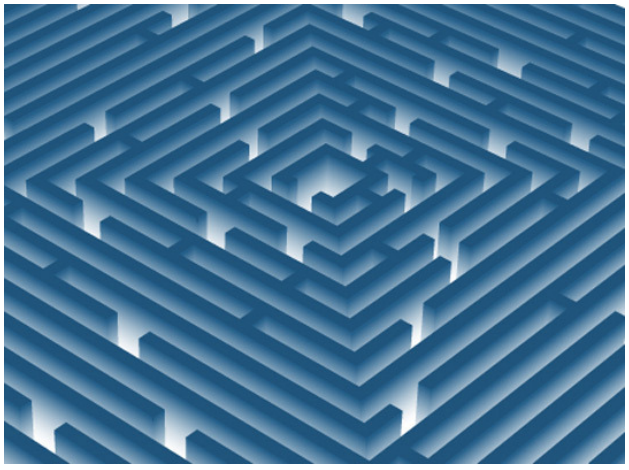
Implementación iterativa mostrando todas las soluciones

```
void NReinas (int reinas[], int n)
{
  int k=0;                                    // Fila actual = k
  for (int i=0; i<n; i++) reinas[i]=-1;  // Configuración inicial

  while (k>=0) {
        reinas[k]++;  // Colocar reina k en la siguiente columna…
        while ((reinas[k]<n) && !comprobar(reinas,n,k))
              reinas[k]++;
        if (k<n) {                                  // Reina colocada
           if (k==n-1) { print(reinas,n);    // Solución
           } else { k++; reinas[k] = -1; }   // Siguiente reina
        } else { k--; }
  }
}
```


Demo

Cómo salir de un laberinto…

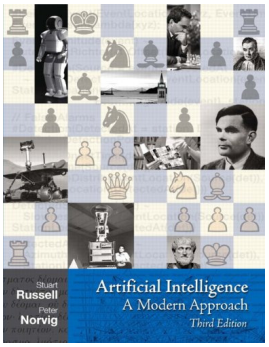


Un laberinto puede modelarse como un grafo:

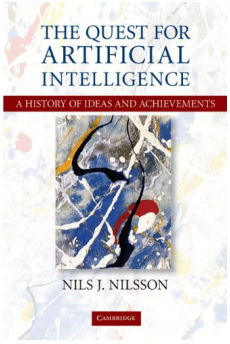
En cada nodo (cruce) hay que tomar una decisión que nos conduce a otros nodos (cruces).

Bibliografía

- Stuart Russell & Peter Norvig:
  Artificial Intelligence:
  A Modern Approach
  Prentice-Hall, 3rd edition, 2009
  ISBN 0136042597



- Nils J. Nilsson
  The Quest for Artificial Intelligence
  Cambridge University Press, 2009
  ISBN 0521122937



Bibliografía

Bibliografía complementaria

- Elaine Rich & Kevin Knight:
  Artificial Intelligence.
  McGraw-Hill, 1991.

- Patrick Henry Winston:
  Artificial Intelligence.
  Addison-Wesley, 1992.

- Nils J. Nilsson:
  Principles of Artificial Intelligence.
  Morgan Kaufmann, 1986.