Teoría de Algoritmos

Capitulo 5: Algoritmos para la Exploración de Grafos.

Tema 14: Backtracking y Branch and Bound

- Árbol de estados.
- Terminología.
- Algoritmo general Backtracking.
- Resolución de problemas concretos

Terminología

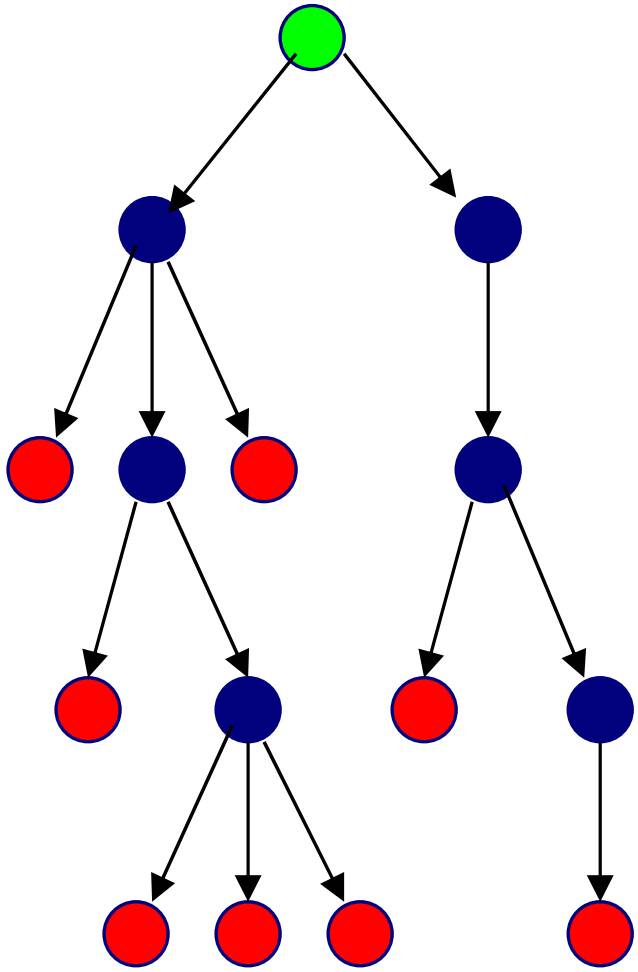
- La organización en árbol del espacio solución se llama **árbol de estados** y en él cada nodo define un **estado** del problema.
- Todos los caminos desde la raíz a otros nodos definen el **espacio de estados** del problema.
- **Estados solución** son aquellos estados del problema S para los que el camino desde la raíz a S define una n-tupla en el espacio solución.
  - En el primero de los árboles anteriores, todos los nodos son estados solución, mientras que en el segundo solo los nodos hoja son estados solución.
- **Estados respuesta** son aquellos estados solución S para los que el camino desde la raíz hasta S define una tupla que es miembro del conjunto de soluciones (es decir satisfacen las restricciones implícitas) del problema.

Terminología

- Las organizaciones en árbol del problema de las 8 reinas se llaman **árboles estáticos**, porque son independientes del caso del problema que se vaya a resolver.
- En algunos problemas es ventajoso usar diferentes organizaciones de árbol para diferentes casos del problema.
- Entonces la organización en árbol se determina dinámicamente como el espacio solución que esta siendo buscado.
- Las organizaciones en árbol que son dependientes del caso concreto del problema a resolver se llaman **árboles dinámicos**.

Generación de estados de un problema

- Cuando se ha concebido un árbol de estados para algún problema, podemos resolver este problema generando sistemáticamente sus estados, determinando cuales de estos son estados solución, y finalmente determinando que estados solución son estados respuesta.



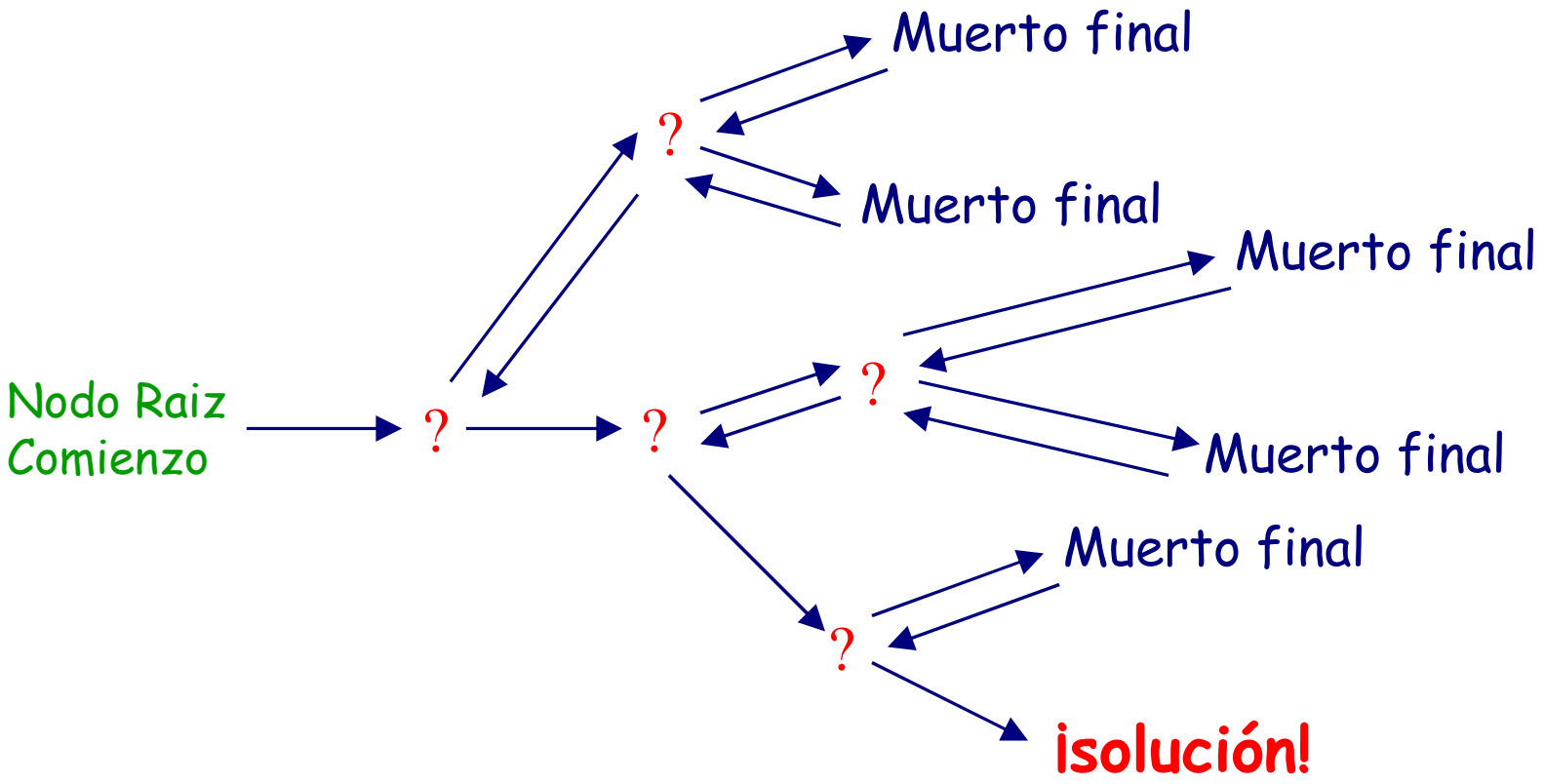
Generación de estados de un problema

- Hay dos formas diferentes de generar los estados del problema.
- Las dos comienzan con el nodo raíz y generan otros nodos.
- Un nodo que ha sido generado, pero para el que no se han generado aun todos sus nodos hijos, se llama un **nodo vivo**.
- El nodo vivo cuyos hijos están siendo generados en ese momento se llama un **E-nodo** (nodo en expansión).
- Un **nodo muerto** es un nodo generado que o no se va a expandir o que todos sus hijos ya han sido generados.
- En ambos métodos de generar estados del problema tendremos una lista de nodos vivos.

Generación de estados de un problema

- En el primer método, tan pronto como un nuevo hijo C, del E-nodo en curso R, ha sido generado, este hijo se convierte en un nuevo E-nodo.
- R se convertirá de nuevo en E-nodo cuando el subarbol C haya sido explorado completamente.
- Esto corresponde a una generación **primero en profundidad** de los estados del problema.
- Adicionalmente se usan funciones de acotación para matar nodos vivos sin tener que generar todos sus nodos hijos.
- Esto habrá de hacerse cuidadosamente para que a la conclusión del proceso al menos se haya generado un nodo respuesta, o se hayan generado todos los nodos respuesta si el problema requiriera encontrar todas las soluciones.
- A esta forma de generación (exploración) se le llama **Backtracking**

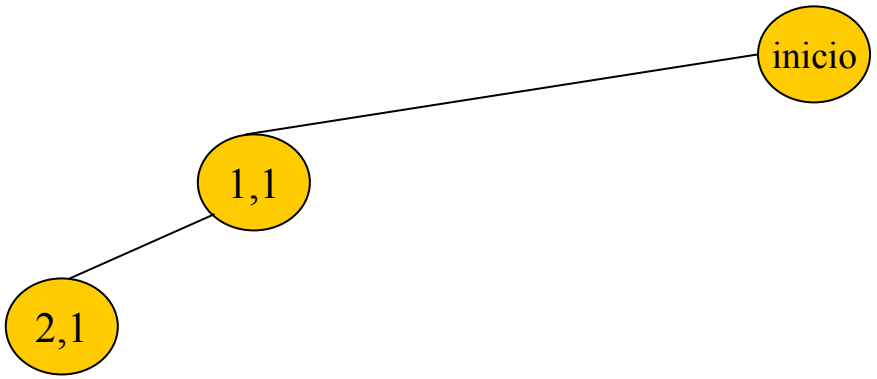
Como procede el Backtracking



Generación de estados de un problema

- En el segundo método, el E-nodo permanece como E-nodo hasta que se hace nodo muerto
- Como en el otro método, también se usan funciones de acotación para detener la exploración en un sub-árbol.
- El método se adapta muy bien a la resolución de problemas de optimización combinatoria (espacio de soluciones discreto): Problema de la Mochila, Soluciones enteras, etc.
- En este método, la construcción de las funciones de acotación es (casi) mas importante que los mecanismos de exploración en si mismos.
- A esta segunda forma de generación (exploración) se le llama **Branch and Bound**

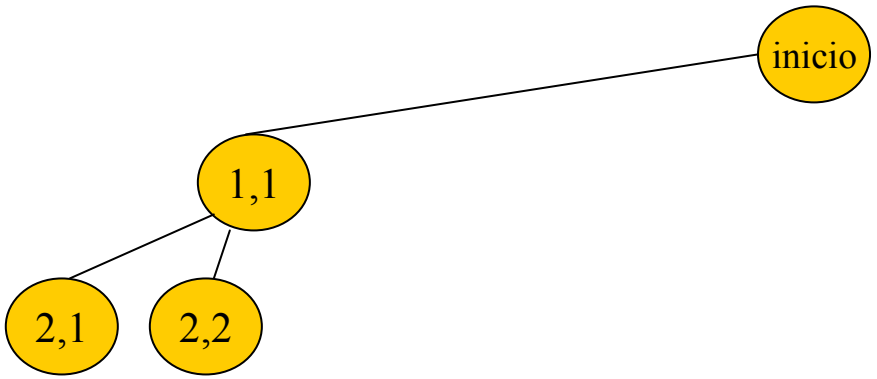
Ejemplo… para *n* = 4



La estrategia ASEGURA
no ocupar el mismo renglón

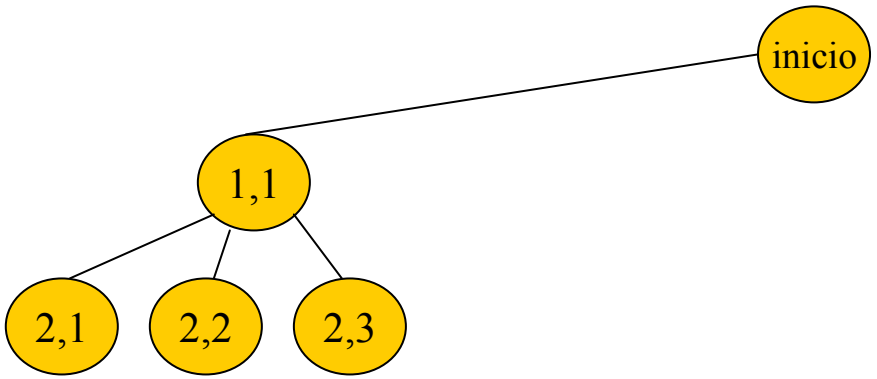
NO se cumple el criterio
(misma columna)

Ejemplo… para *n* = 4



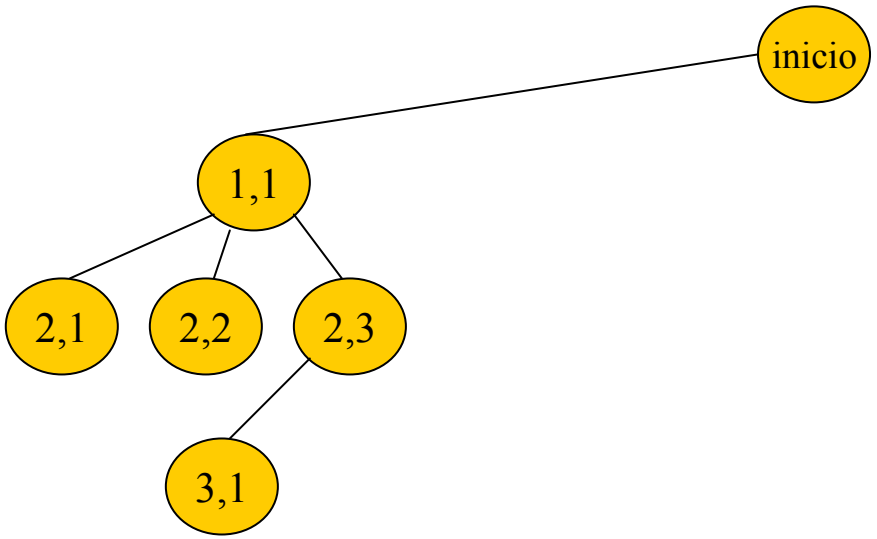
NO se cumple el criterio (misma diagonal)

Ejemplo… para *n* = 4



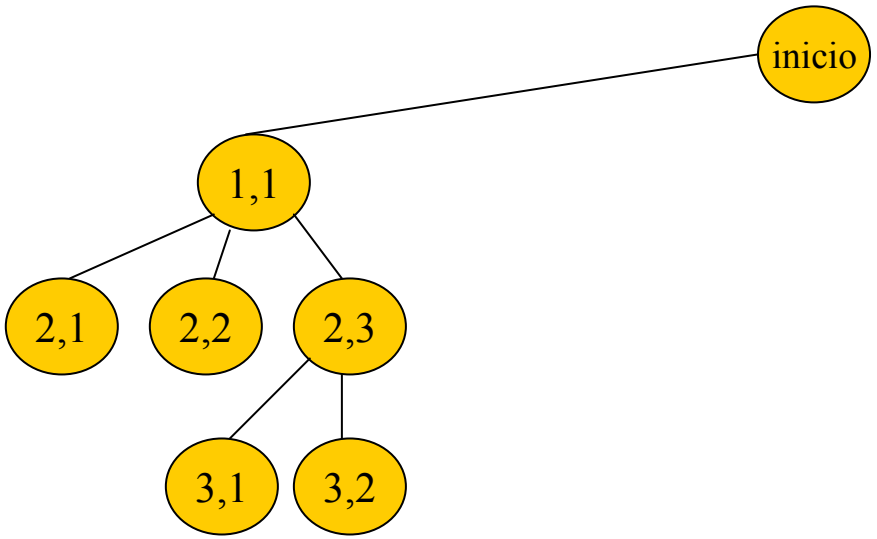
OK… adelante en la búsqueda !

Ejemplo… para *n* = 4



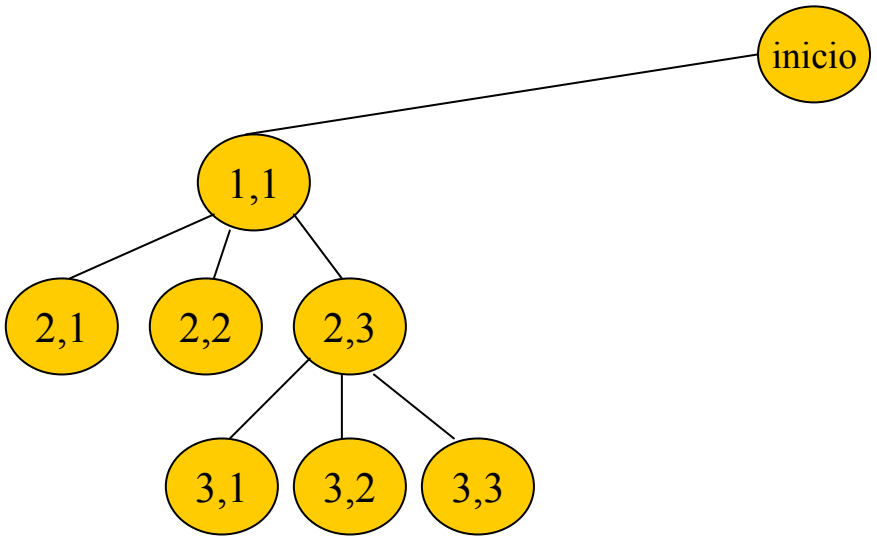
NO se cumple el criterio (misma columna)

Ejemplo… para *n* = 4



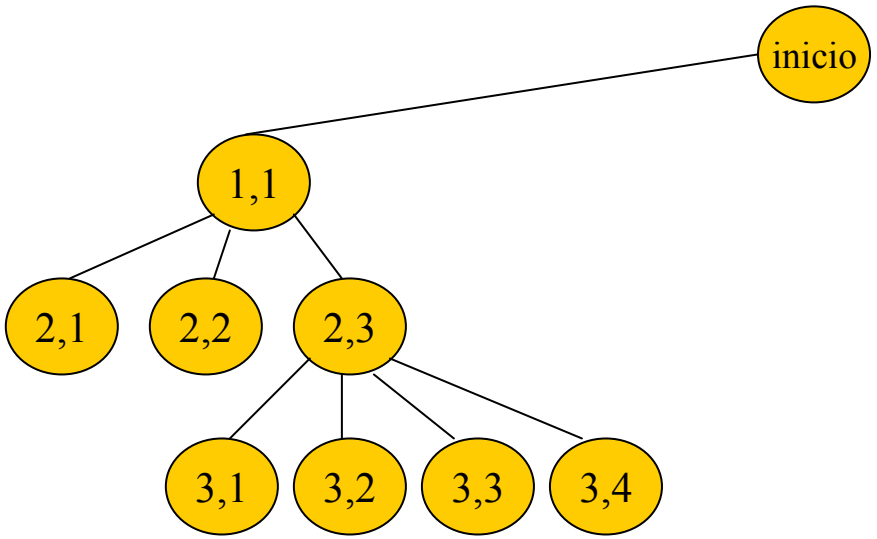
NO se cumple el criterio (misma diagonal que 2,3)

Ejemplo… para *n* = 4



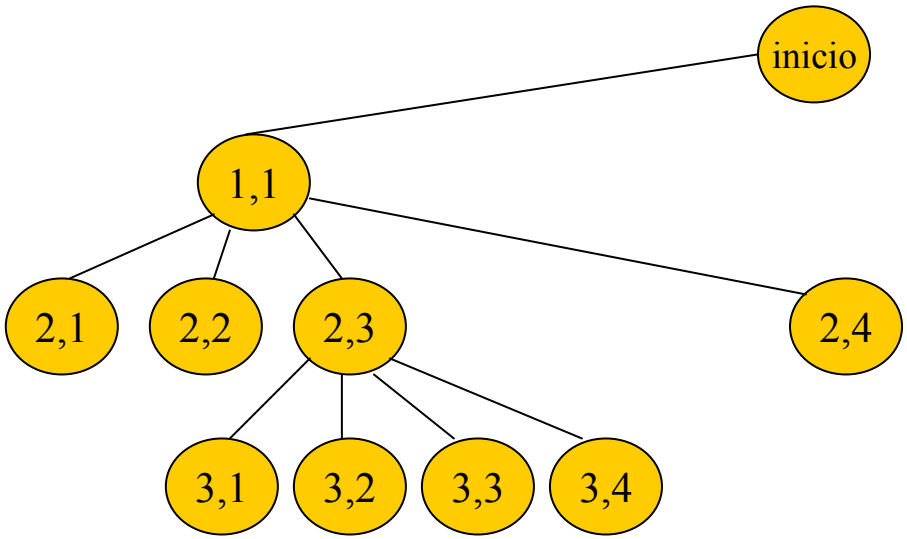
NO se cumple el criterio
(misma diagonal que 1,1)

Ejemplo… para *n* = 4



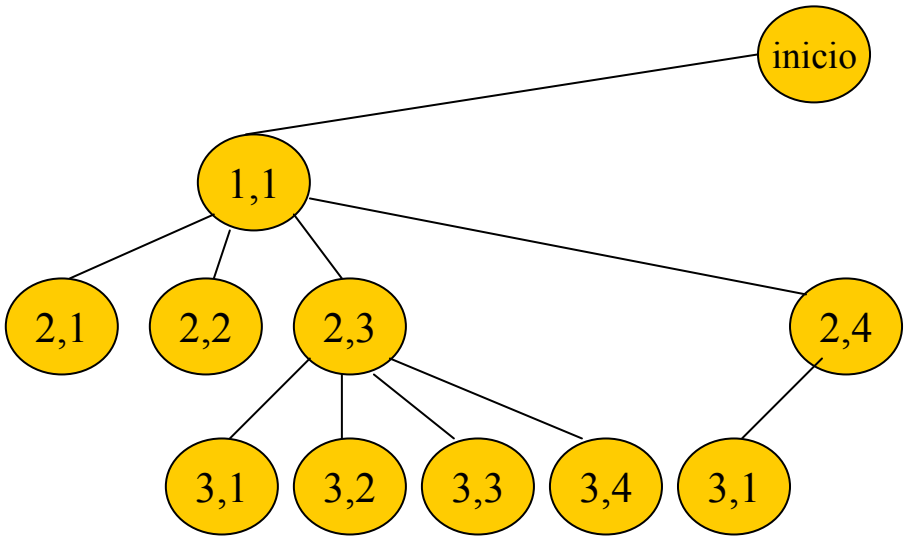
NO se cumple el criterio
(misma diagonal que 2,3)

Ejemplo… para *n* = 4



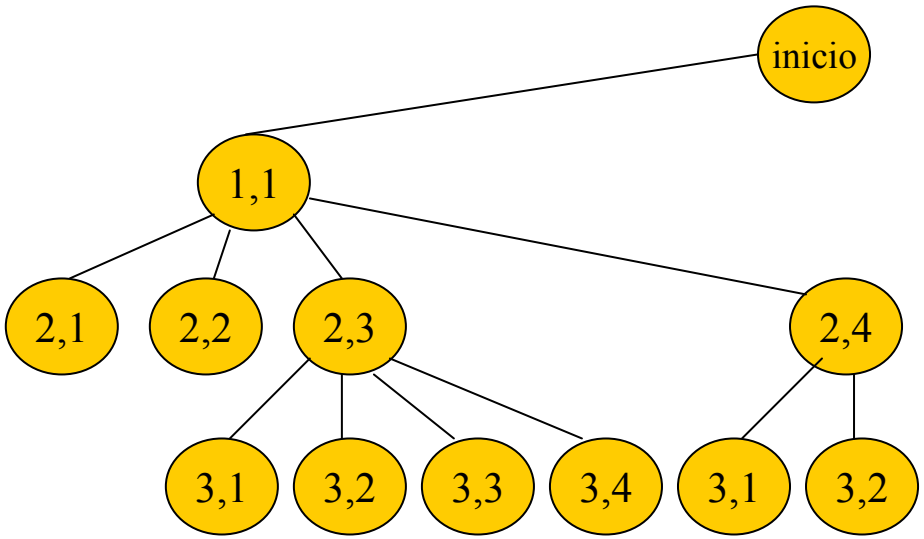
OK… adelante con la búsqueda!

Ejemplo… para *n* = 4



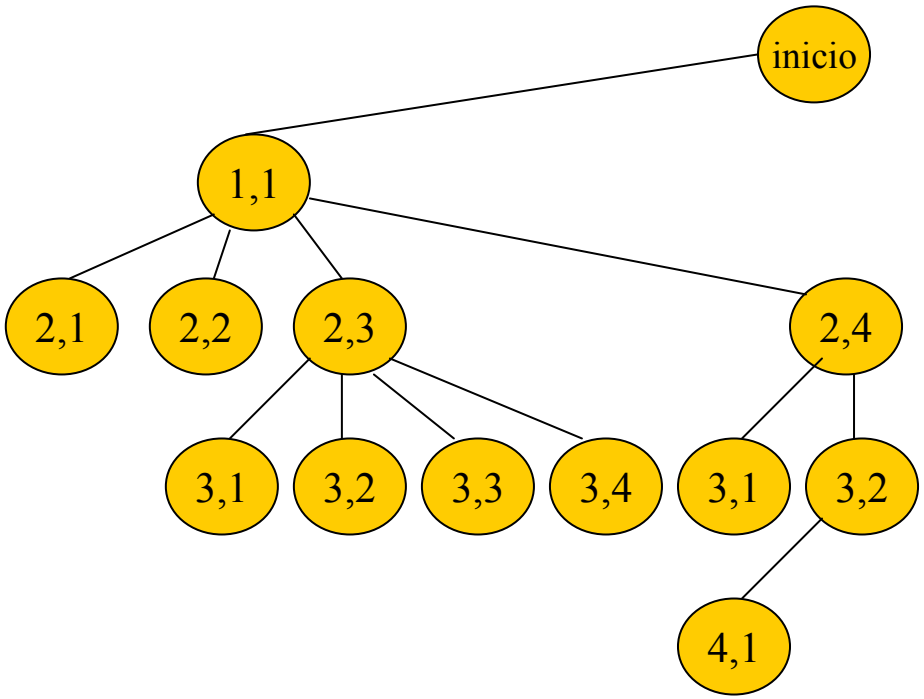
NO se cumple criterio (misma columna)

Ejemplo… para *n* = 4



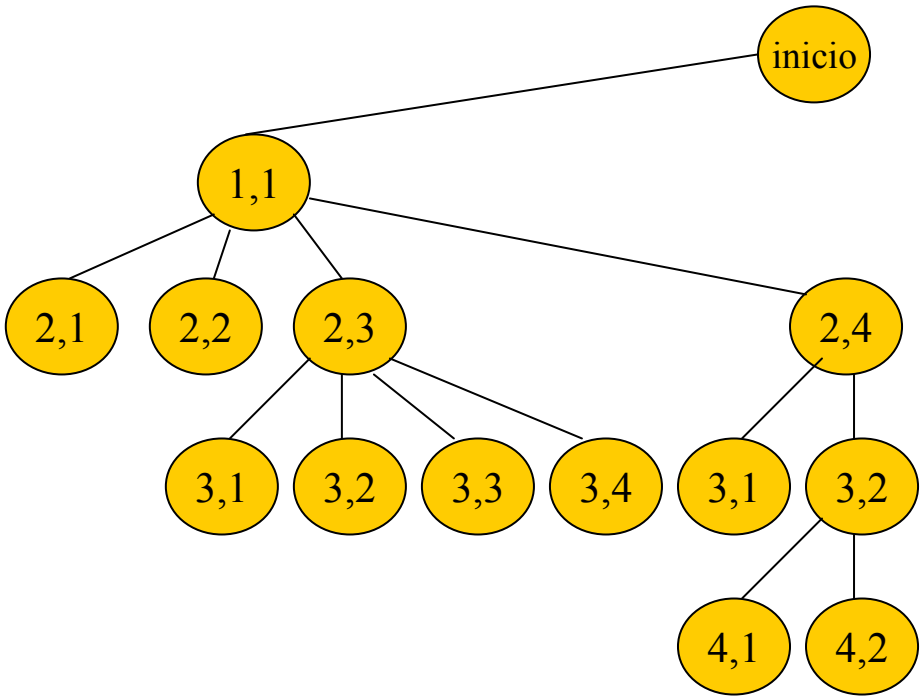
OK… adelante con la búsqueda!

Ejemplo… para *n* = 4



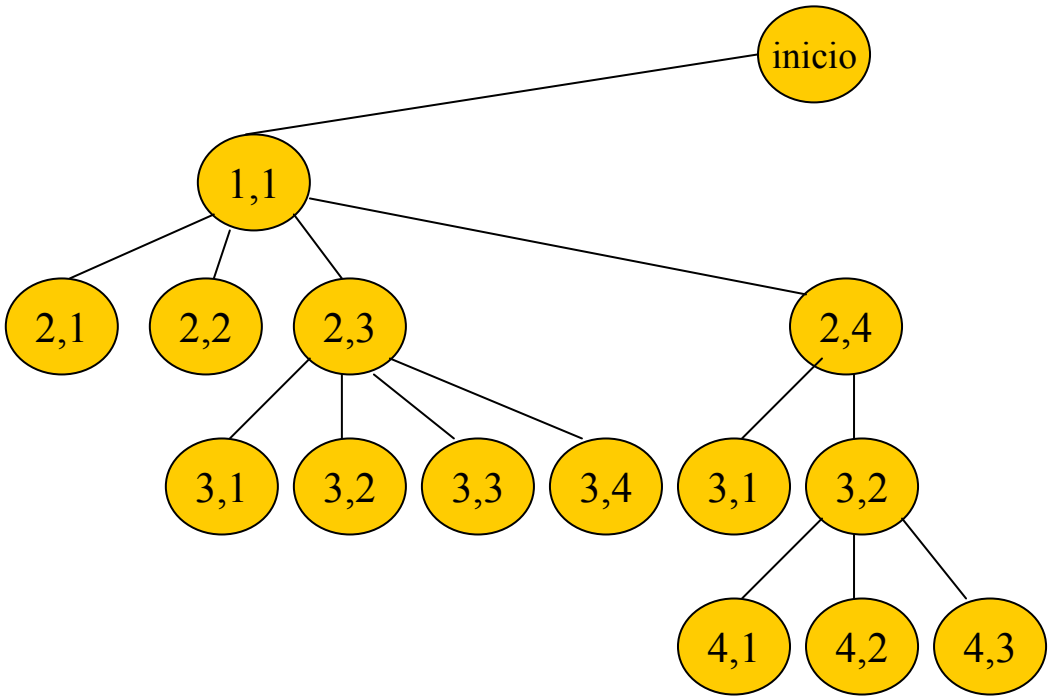
NO se cumple criterio (misma columna)

Ejemplo… para *n* = 4



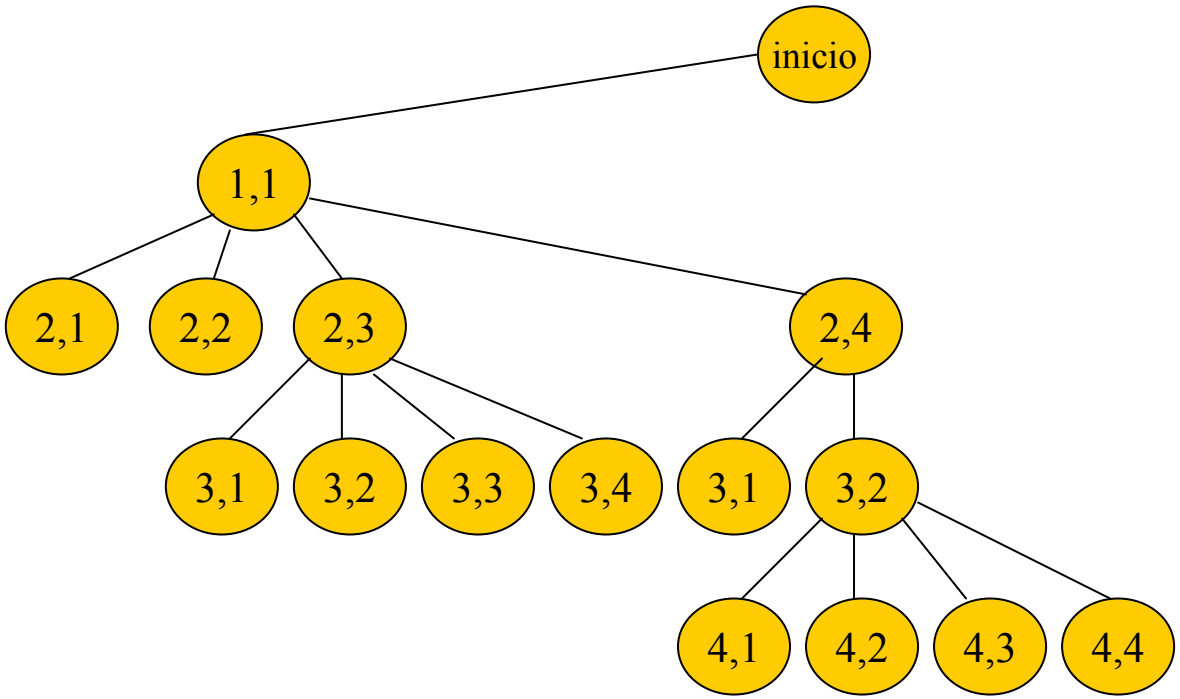
NO se cumple criterio (misma columna)

Ejemplo… para *n* = 4



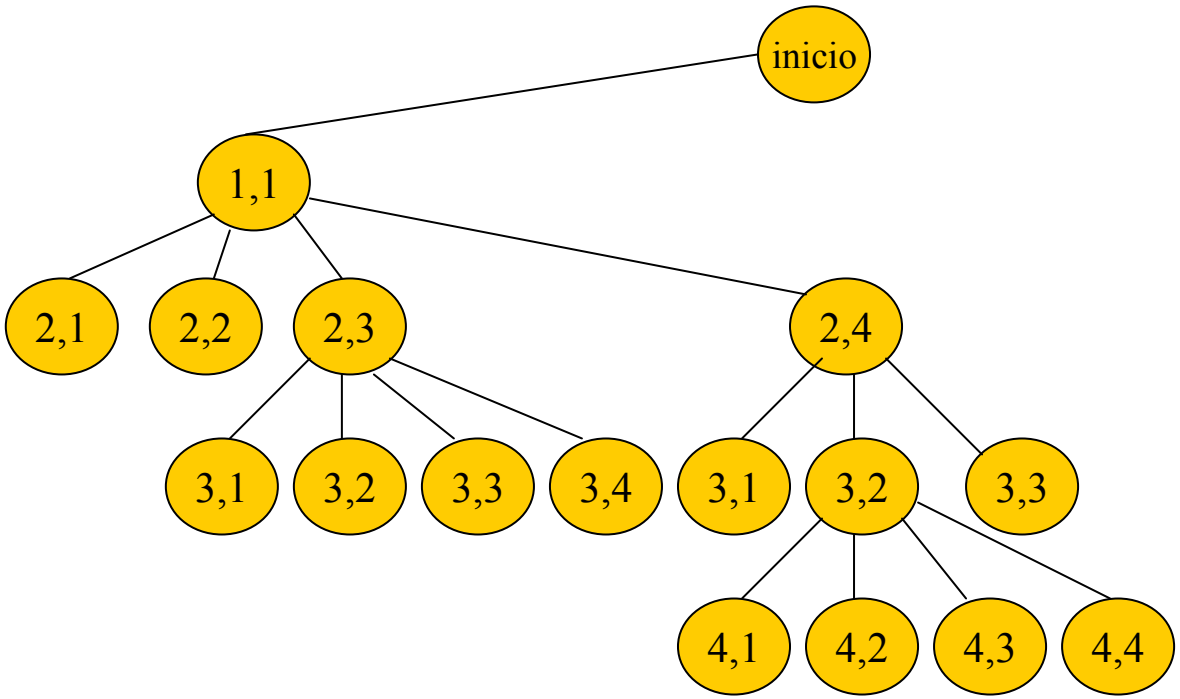
NO se cumple criterio
(misma diagonal 3,2)

Ejemplo… para *n* = 4



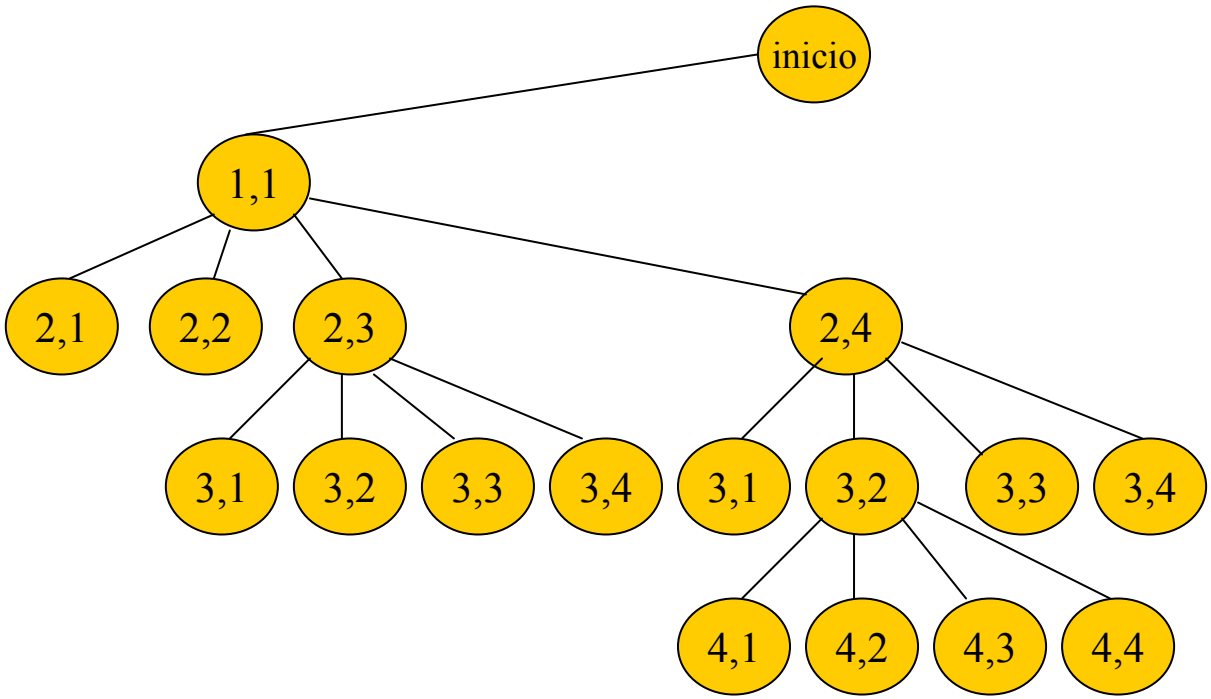
NO se cumple criterio (misma columna)

Ejemplo… para *n* = 4



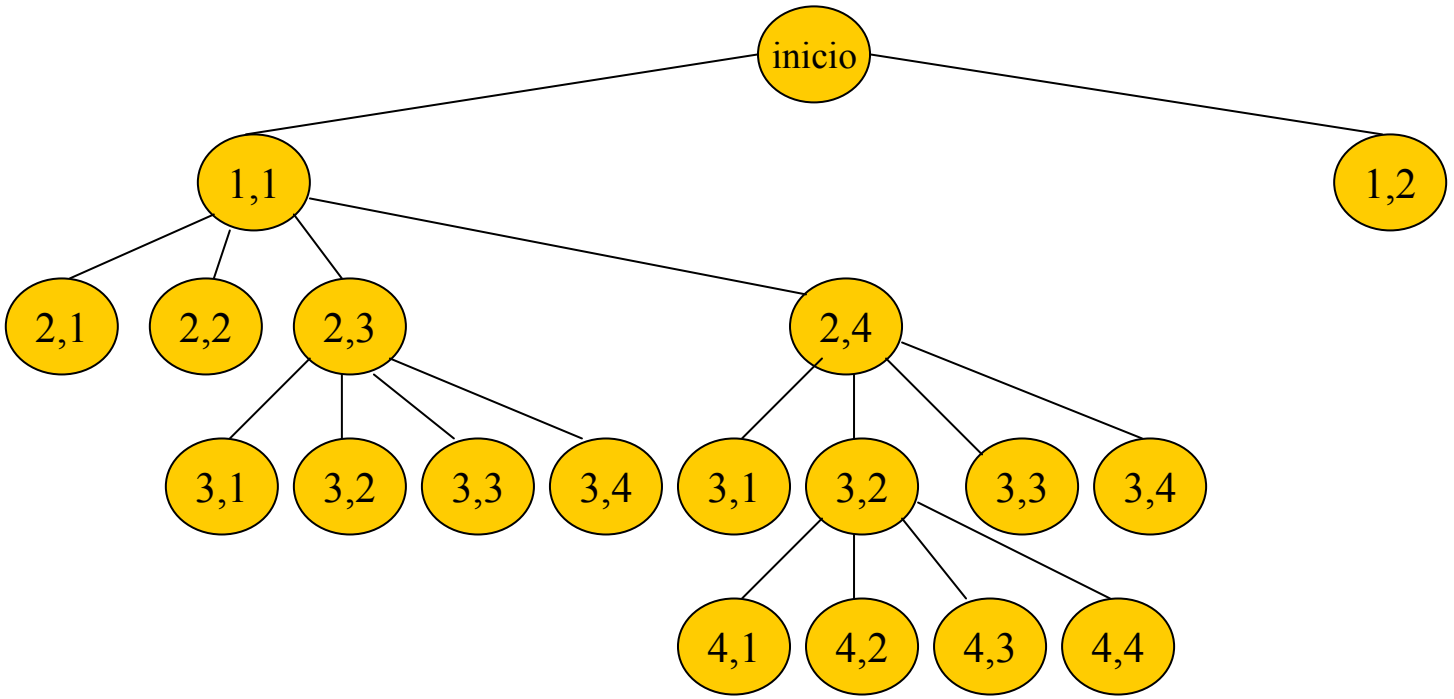
NO se cumple criterio (misma diagonal)

Ejemplo… para *n* = 4



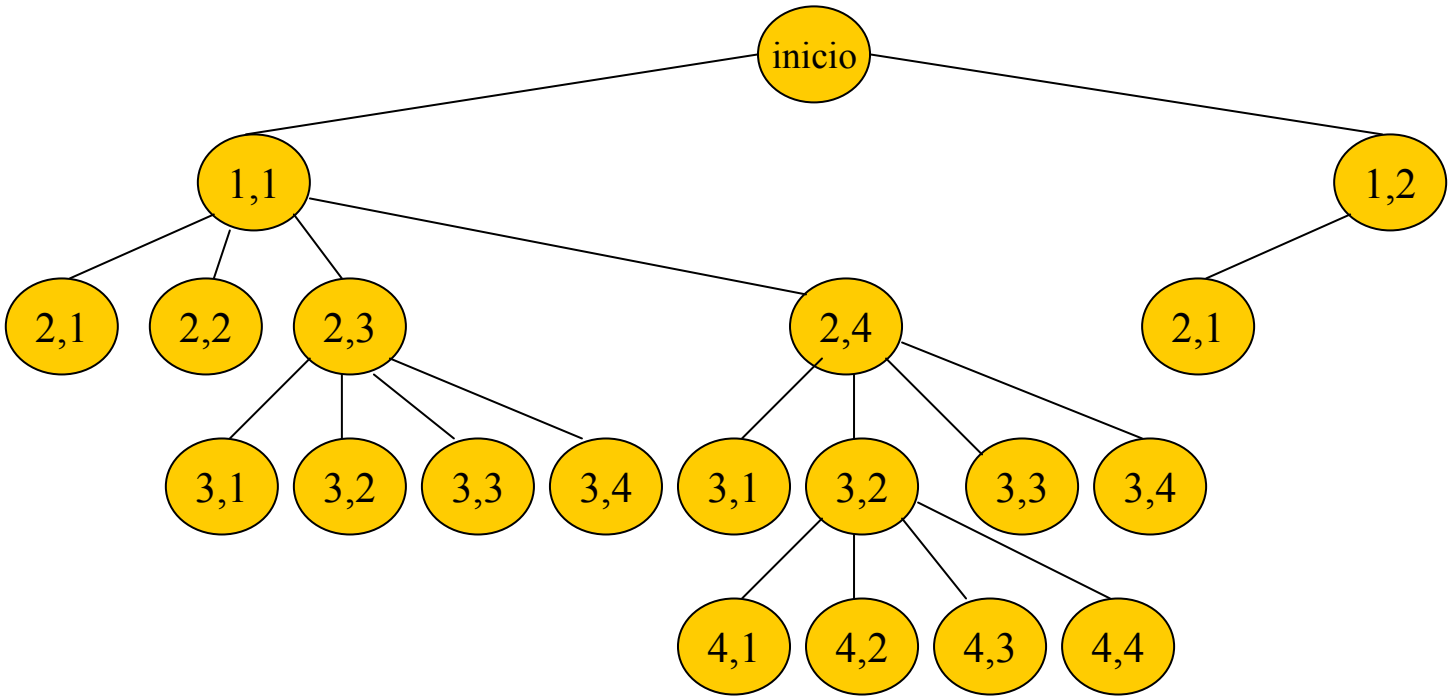
NO se cumple criterio (misma columna)

Ejemplo… para *n* = 4



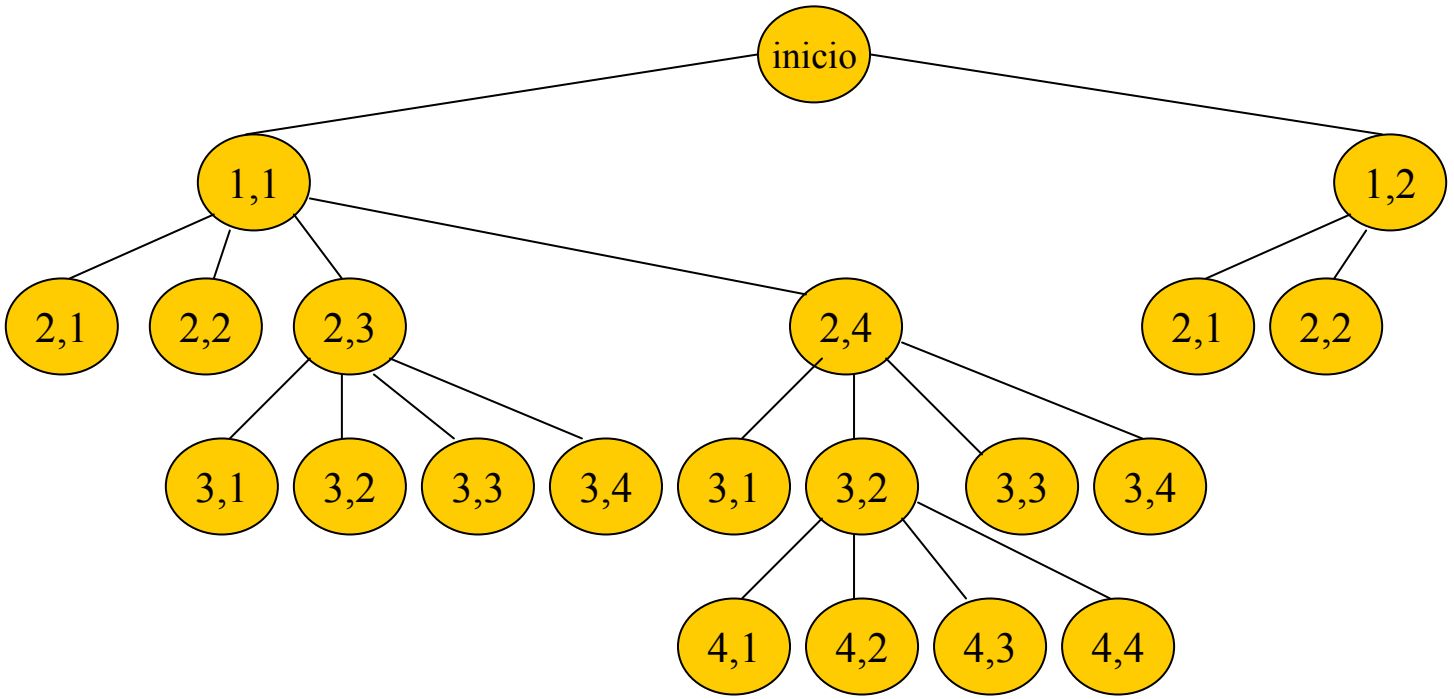
OK… adelante con la búsqueda!

Ejemplo… para *n* = 4



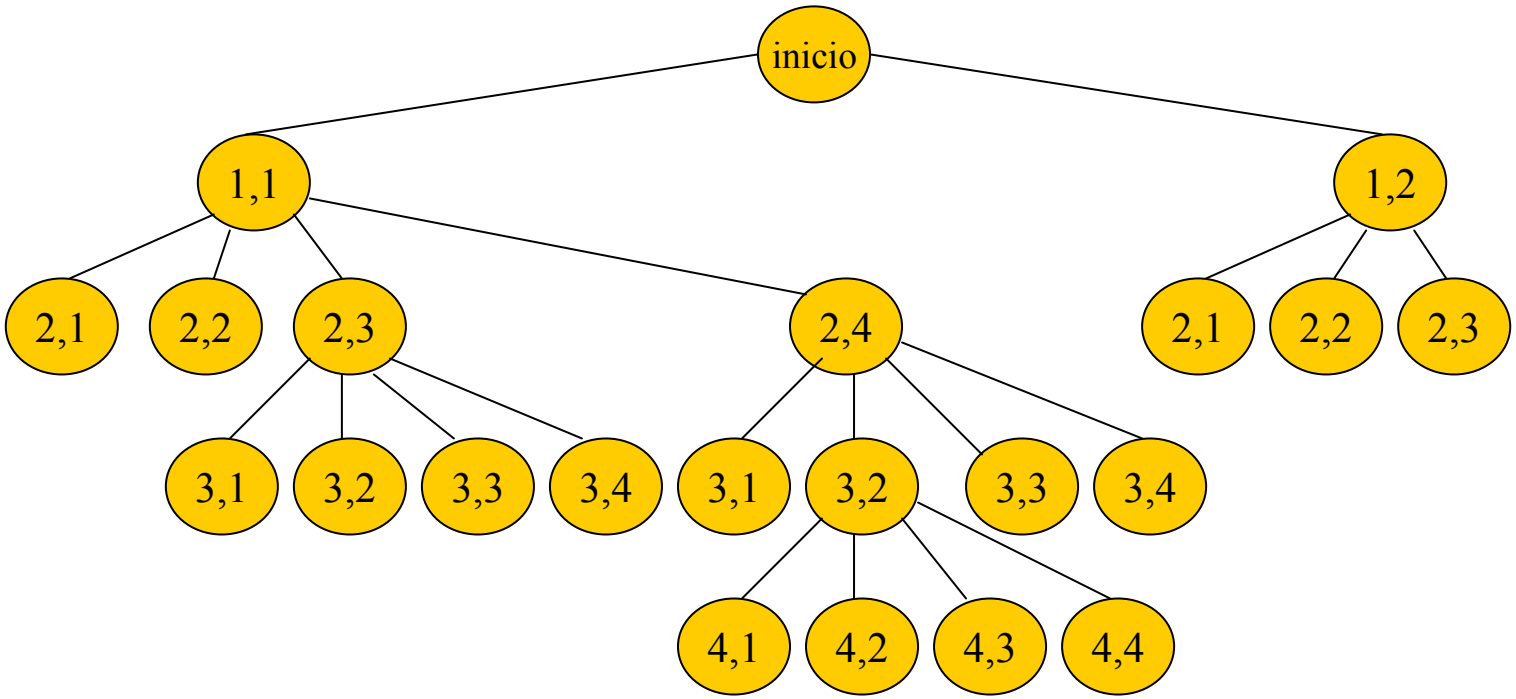
NO cumple criterio
(misma diagonal)

Ejemplo… para *n* = 4



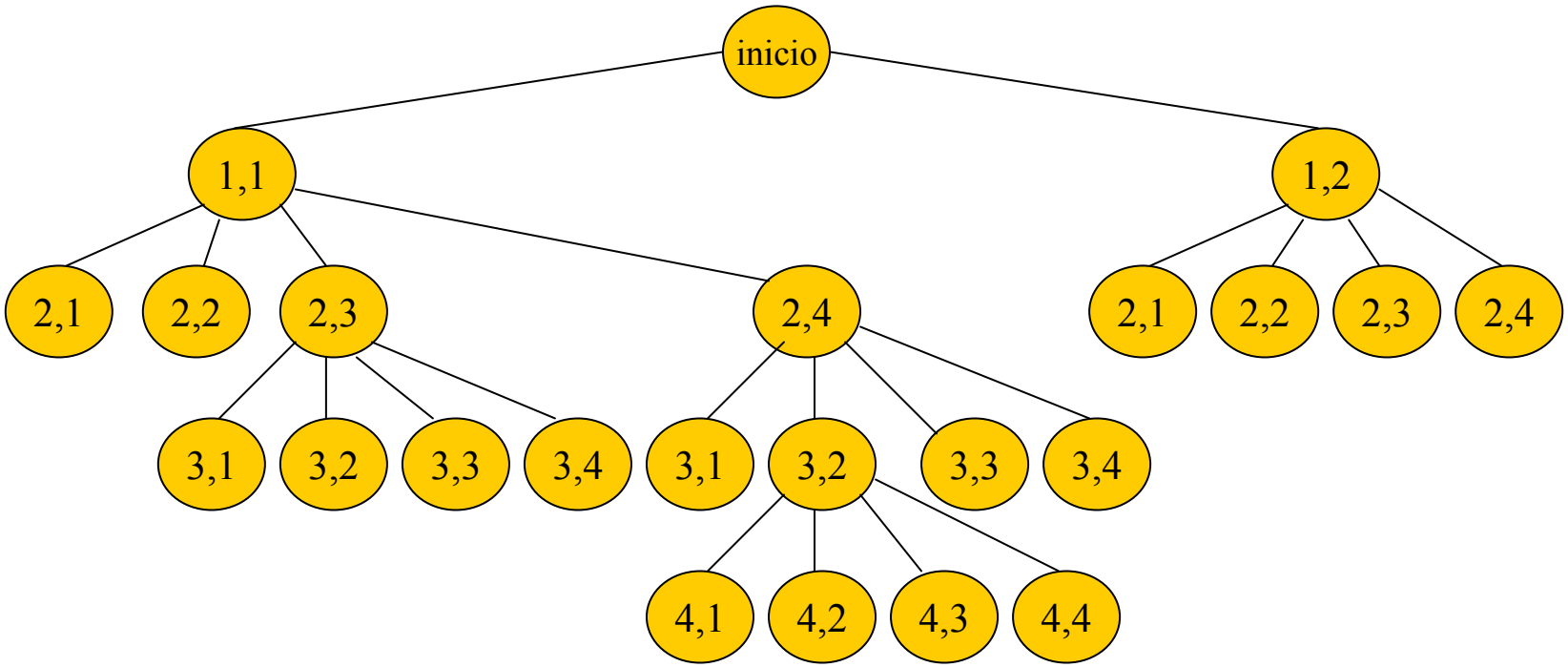
NO cumple criterio
(misma columna)

Ejemplo… para *n* = 4



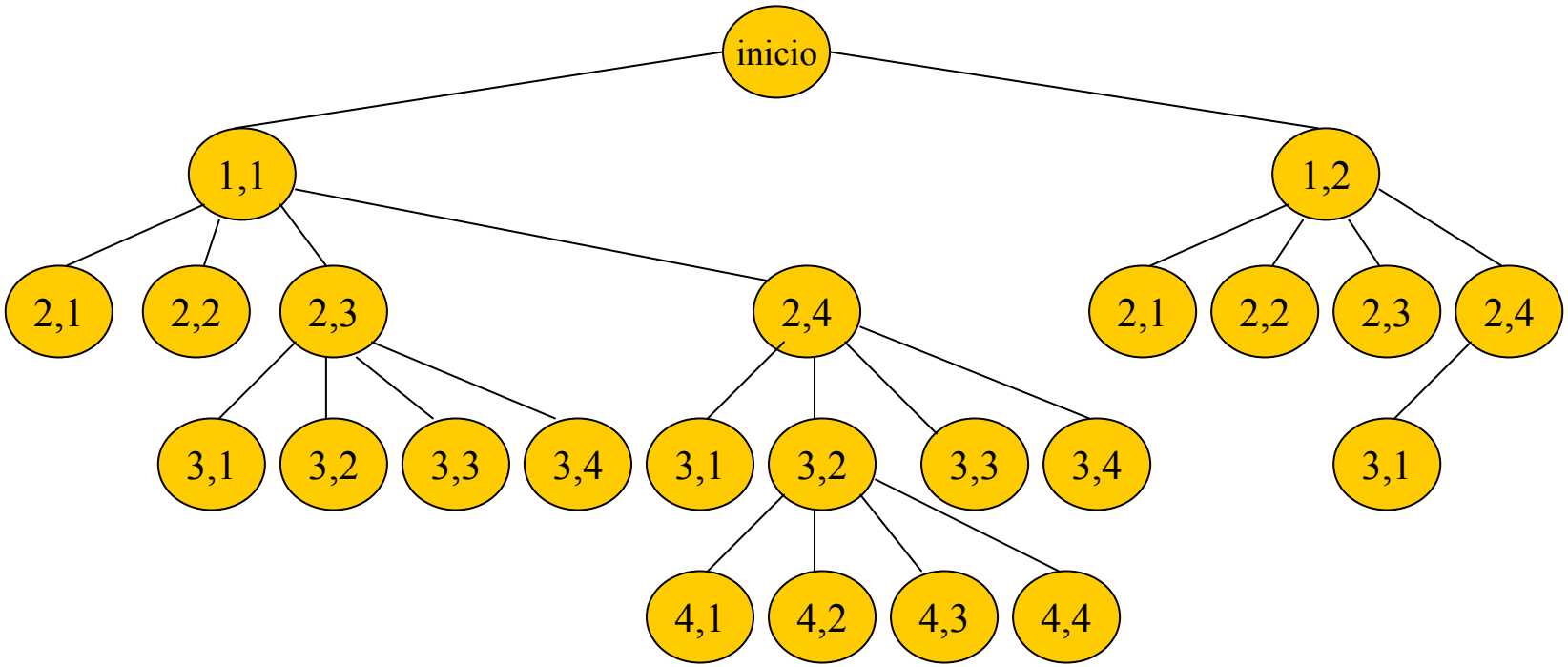
NO cumple criterio
(misma diagonal)

Ejemplo… para *n* = 4



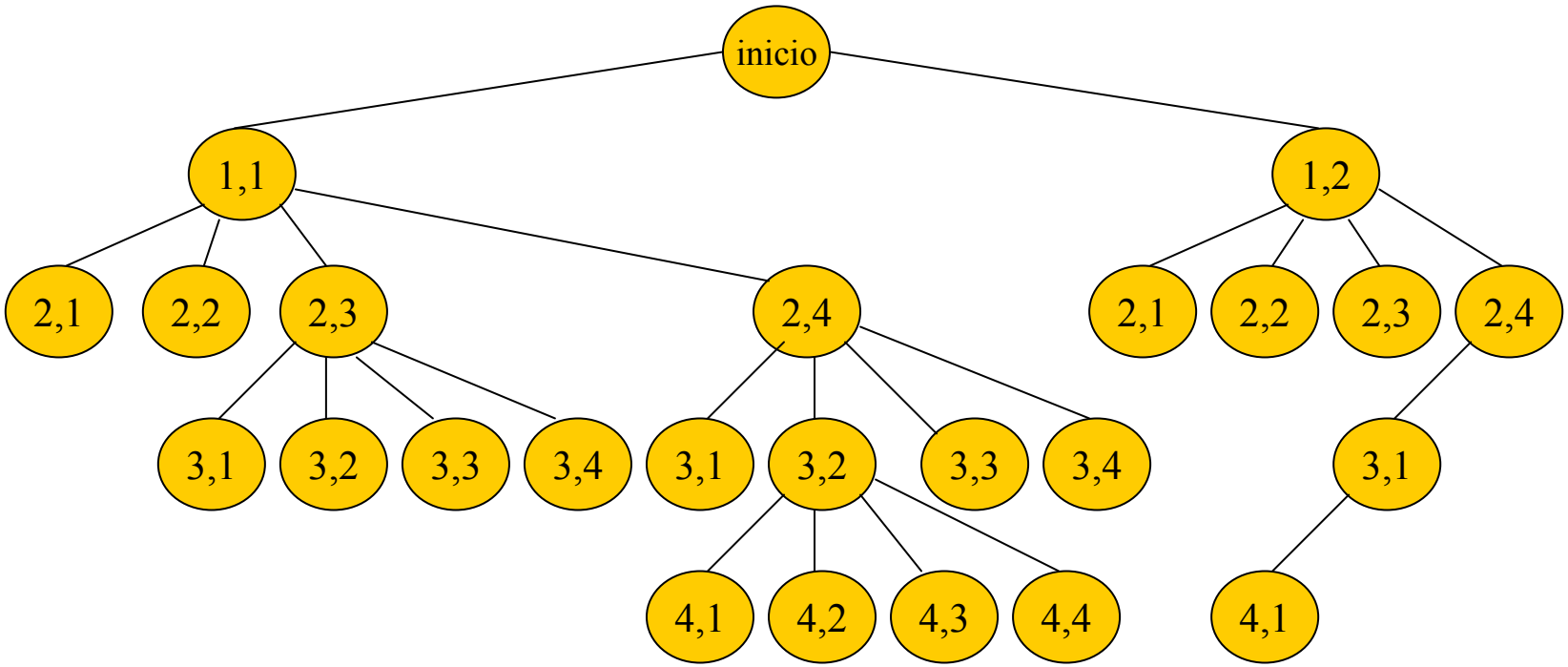
OK… adelante con la búsqueda!

Ejemplo… para *n* = 4



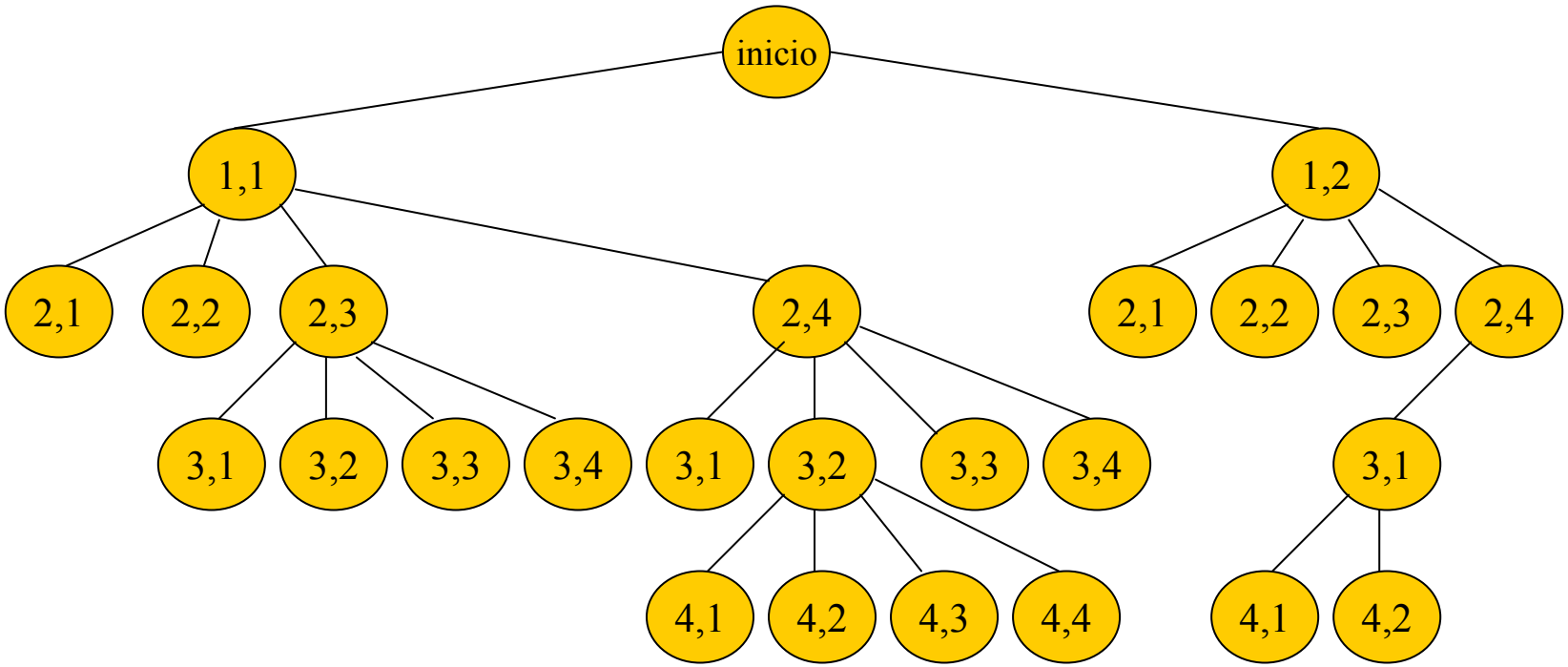
OK… adelante con la búsqueda!

Ejemplo… para *n* = 4



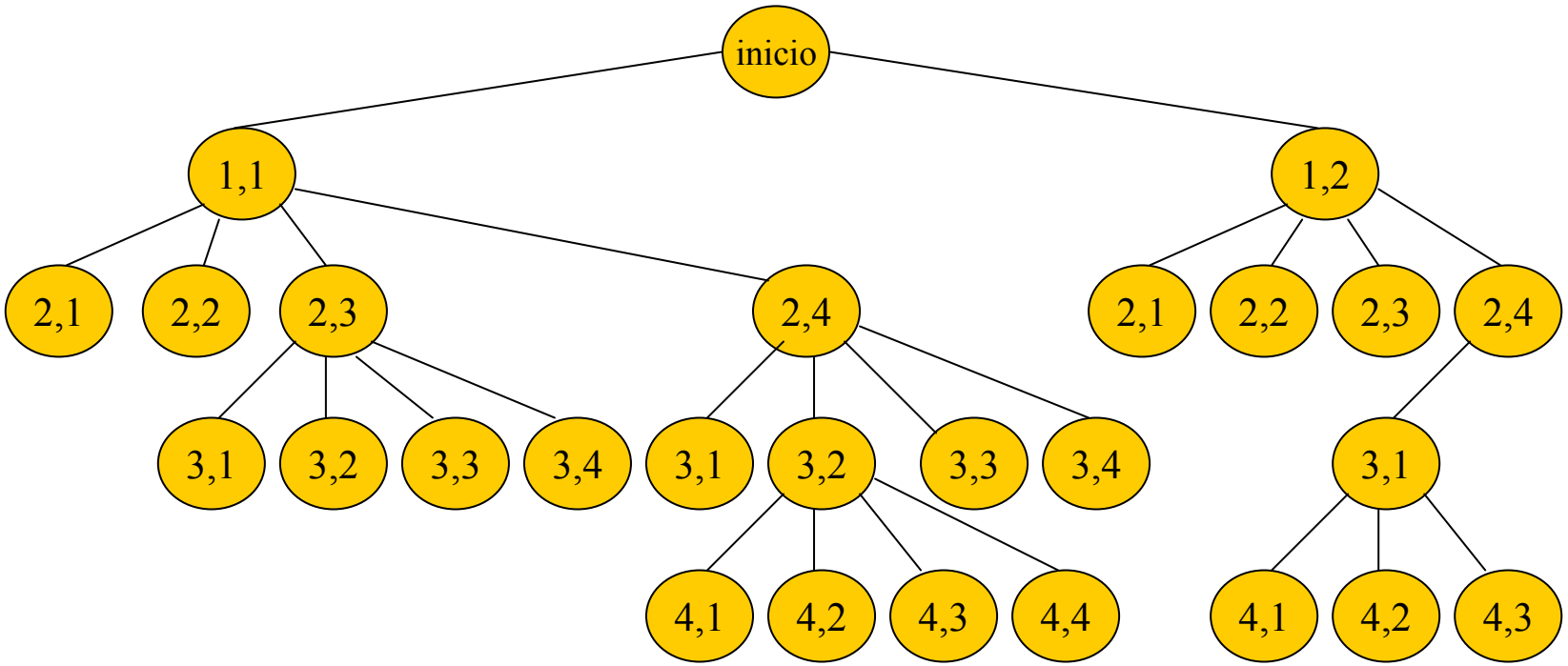
NO cumple el criterio (misma columna)

Ejemplo… para *n* = 4



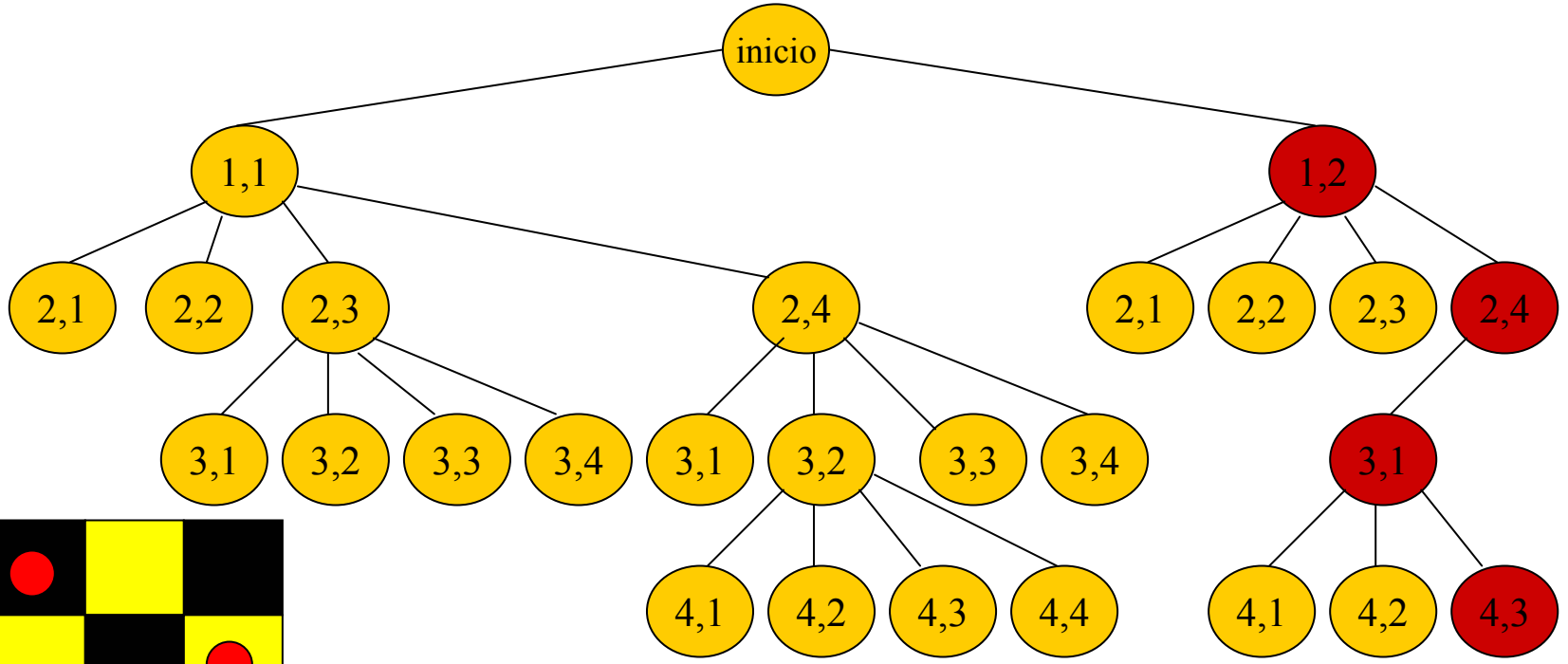
NO cumple el criterio (misma columna)

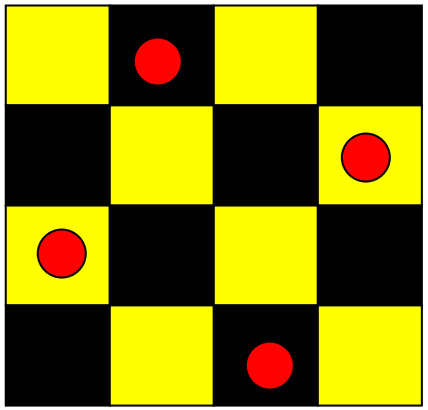
Ejemplo… para *n* = 4



¡¡OK… se encontró solución !!

Ejemplo… para *n* = 4





Se comprobarón 27 nodos… Sin backtracking se hubieran comprobado 155 nodos

Algoritmo Backtracking

- Podemos presentar una formulación general, aunque precisa, del proceso de backtracking.
- Supondremos que hay que encontrar todos los nodos respuesta, y no solo uno.
- Sea $(x_1, x_2,..., x_i)$ un camino desde la raíz hasta un nodo en el árbol de estados.
- Sea $T(x_1, x_2,..., x_i)$ el conjunto de todos los posibles valores de x_{i+1} tales $(x_1, x_2,..., x_{i+1})$ es también un camino hacia un estado del problema.
- Suponemos la existencia de funciones de acotación B_{i+1} (expresadas como predicados) tales que $B_{i+1}(x_1,x_2,...,x_{i+1})$ es falsa para un camino $(x_1, x_2,..., x_{i+1})$ desde el nodo raíz hasta un estado del problema si el camino no puede extenderse para alcanzar un nodo respuesta.
- Así los candidatos para la posición i+1 del vector solución X(1..n) son aquellos valores que son generados por T y satisfacen B_{i+1}.

Algoritmo Backtracking

- El Procedimiento Backtrack, representa el esquema general backtracking haciendo uso de T y B_{i+1}.

Procedimiento BACKTRACK(n)

{Todas las soluciones se generan en X(1..n) y se imprimen tan pronto como se encuentran. T(X(1),...,X(k-1)) da todos los posibles valores de X(k) dado que habíamos escogido X(1),..., X(k-1). El predicado B_k(X(1),..., X(k)) determina los elementos X(k) que satisfacen las restricciones implícitas}

```
Begin
  k = 1
    While k > 0 do
      If queda algún X(k) no probado tal que
            X(k) ∈ T(X(1),..., X(k-1)) and B_k (X(1),..., X(k)) = true
      Then if (X(1),..., X(k)) es un camino hacia un nodo respuesta
            Then print (X(1),..., X(k))
             k = k+1
      else k = k-1
end
```

Algoritmo Backtracking recursivo

- El siguiente algoritmo, Rbacktrack, presenta una formulación recursiva del método, ya que como backtracking básicamente es un recorrido en postorden, es natural describirlo así,

Procedimiento RBACTRACK(K)

{Se supone que los primeros k-1 valores X(1),…,X(k-1) del vector solución X(1..n) han sido asignados}

```
Begin
  for cada X(k) tal que
     X(k)∈ T(X(1),…,X(k-1)) and B (X(1),…, X(k)) = true do
       If (X(1),…,X(k)) es un camino hacia un nodo respuesta
       Then print (X(1),…, X(k))
        RBACKTRACK(K+1)
End
```

Eficiencia de backtracking

- La eficiencia de los algoritmos backtracking depende básicamente de cuatro factores:
  - el tiempo necesario para generar el siguiente X(k),
  - el numero de X(k) que satisfagan las restricciones explícitas,
  - el tiempo para las funciones de acotación Bi, y
  - el numero de X(k) que satisfagan las B_i para todo i.
- Las funciones de acotación se entienden buenas si reducen considerablemente el numero de nodos que generan.
- Las buenas funciones de acotación consumen mucho tiempo en evaluaciones, por lo que hay que buscar un equilibrio entre el tiempo global de computación, y la reducción del numero de nodos generados.

Eficiencia de backtracking

- De los 4 factores que determinan el tiempo requerido por un algoritmo backtracking, solo la cuarta, el numero de nodos generados, varia de un caso a otro.
- Un algoritmo backtracking en un caso podría generar solo O(n) nodos, mientras que en otro (relativamente parecido) podría generar casi todos los nodos del árbol de espacio de estados.
- Si el numero de nodos en el espacio solución es 2^n o $n!$, el tiempo del peor caso para el algoritmo backtracking seria generalmente $O(p(n)2^n)$ u $O(q(n)n!)$ respectivamente, con p y q polinomios en n.
- La importancia del backtracking reside en su capacidad para resolver casos con grandes valores de n en muy poco tiempo.
- La dificultad esta en predecir la conducta del algoritmo backtrack en el caso que deseemos resolver.

Eficiencia de backtracking

- Podemos estimar el numero de nodos que se generaran con un algoritmo backtracking usando el método de Monte Carlo.
- Se trata de generar un camino aleatorio en el árbol de estados.
- Sea X un nodo en ese camino aleatorio. Supongamos que X esta en el nivel i del árbol del espacio de estados.
- Las funciones de acotación se usan en el nodo X para determinar el numero m_i de sus hijos que no hay que acotar. El siguiente nodo en el camino se obtiene seleccionando aleatoriamente uno de estos m_i hijos que no se han acotado.
- La generación del camino termina en un nodo que sea una hoja o cuyos hijos vayan a acotarse. Usando estos m_i's podemos estimar el numero total, m, de nodos en el árbol del espacio de estados que no se acotaran.