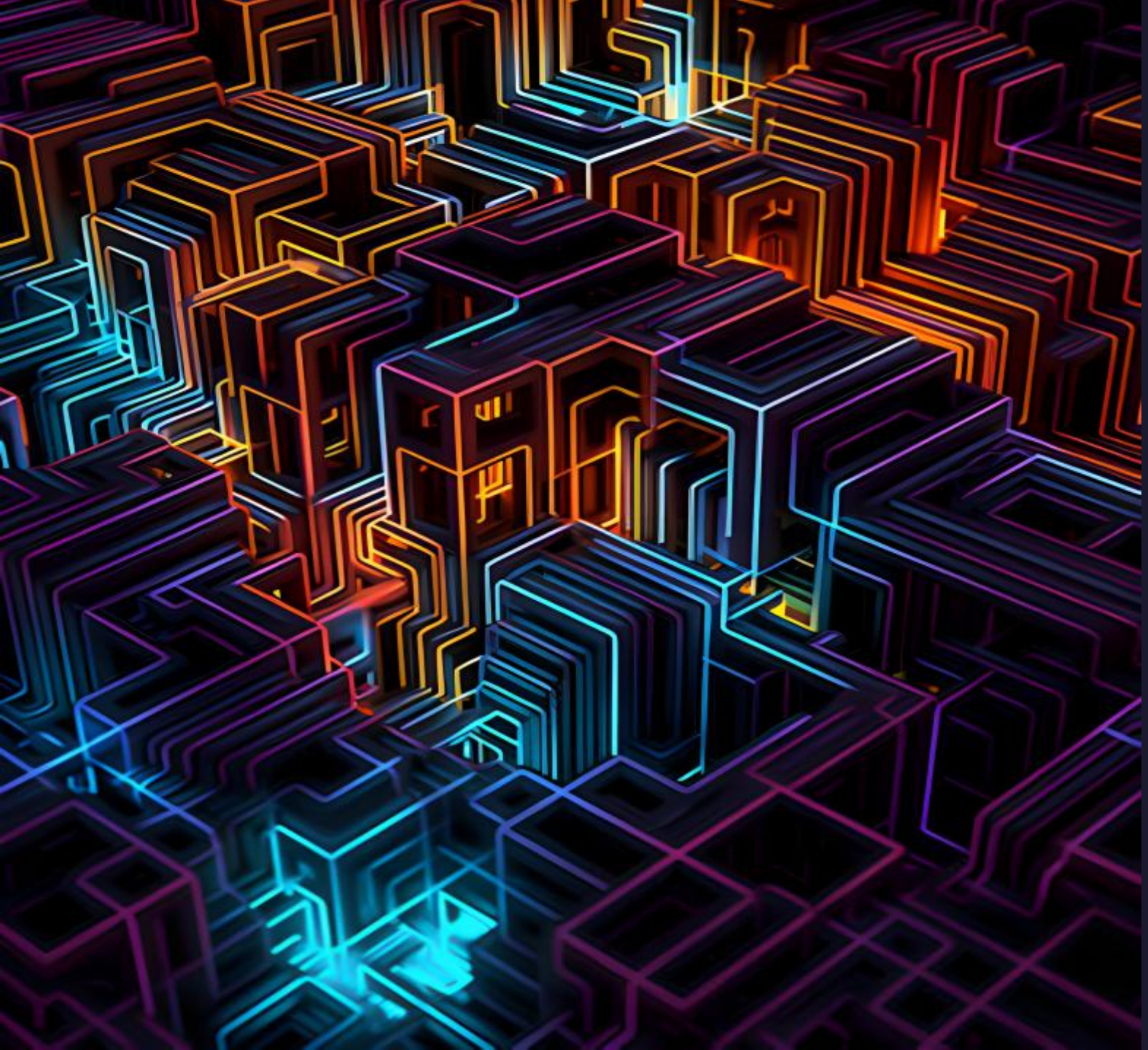


Técnicas de diseño de algoritmos: backtracking

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

Tecnológico de Monterrey

Problemas

¿Cuáles son los problemas intratables?

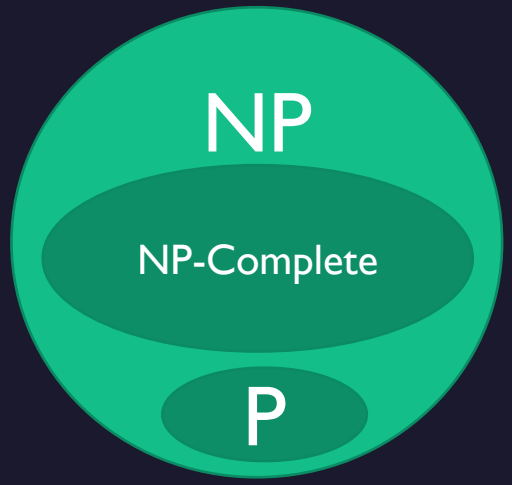
De decisión	Sobre verificar si una solución es la "correcta", o si una solución que cumpla ciertas características existe	
De satisfactibilidad	Sobre encontrar una asignación para variables, que satisfaga ciertas condiciones	SAT
De optimización • Combinatoria💚 • Numérica	Sobre encontrar una asignación para variables (discretas o continuas), que resulte en valores mínimos o máximos	TSP Coloreado de grafos Knapsack

¿Porque son tan difíciles?

En general, porque tienen muchísimas soluciones posibles, y no podemos saber que una es la mejor sin tener que revisar todas

Y hacerlo tomaría millones de años…

Clases de problemas



Son problemas de decisión

P: toman poly-time en DTM
NP: toman poly-time en NDTM

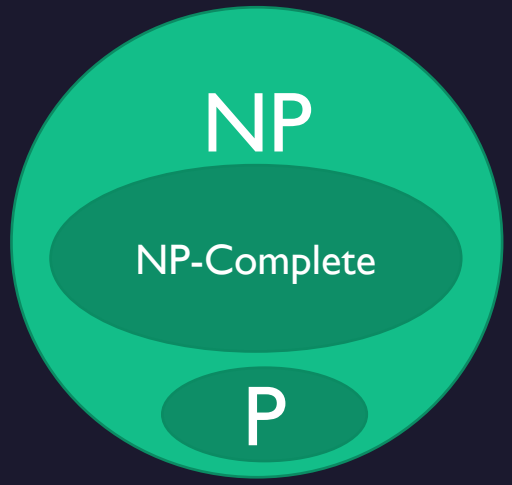
NP-Complete: pueden transformarse a otros NP en poly-time

Son problemas de optimización

PO: toman poly-time en DTM
NPO: toman poly-time en NDTM

NP-Hard: pueden transformarse a otros NP en poly-time

Clases de problemas



Problemas de decisión

Ej. En TSP ¿hay un tour de costo $\leq k$?

Crearlo puede ser difícil de crear, pero verificarlo cuando ya existe es fácil

Problemas de optimización

Ej. En TSP ¿cuál es el tour óptimo?

Es difícil verificar si un tour dado cumple con ser óptimo

Problemas con combinaciones

Una solución x, formada por varias variables

?	?	?	?	?

x_1 x_n

Cada variable x_i tiene valores válidos que puede tomar (dominio y rango)

¿Cómo crear combinaciones que cumplan con ciertas características buscadas?

Por ejemplo:

Que al evaluar una cierta función $f(x)$, el resultado sea

- lo menor posible, o…
- lo máximo posible, o…
- dentro de un umbral, o…
- que se cumplan con restricciones matemáticas prestablecidas (constraints)

La optimización (💚) es una área de la computación dedicada a estudiar este tipo de problemas

La perspectiva (mas o menos) completa

- Algoritmos
  - **Exactos**
    Óptimos asegurados
    - Búsqueda exhaustiva
    - Backtraking B&B
    - Solvers declarativos (como CPLEX)
  - **Aproximados**
    Óptimos no asegurados
    - **Heurísticas**
      Intuiciones educadas sobre el problema, para construir soluciones
      - Algs. Greedy
      - Programación dinámica
      - Randomized
    - **Metaheurísticas**
      Heurísticas sobre heurísticas: Mejorar soluciones existentes, vía modificación y/o combinación
      - Monosolución
      - Poblacionales
    - **Hiperheurísticas**
      Optimización de metaheurísticas y sus componentes

Cuando tratamos con problemas de optimización, hay dos grandes ramas de algoritmos: los exactos y los aproximados

En estas ramas, la investigación se centra en: algoritmos para problemas nuevos, algoritmos mejores que los existentes, estudios empíricos de dificultad… etc., etc., etc.

…

Técnicas de diseño: backtracking

Primero, veamos un problema:

Set Sum Problem

Dado un conjunto S, buscamos un subconjunto de S, cuyo total sume d

Si $S = \{1, 2, 5, 6, 8\}$ y $d = 9$
las soluciones son $\{1, 8\}$ y $\{1, 2, 6\}$

¿Cómo seria la solución de fuerza bruta con búsqueda exhaustiva?

Backtracking es una técnica planteada como mejora de la búsqueda exhaustiva

Es parte de los métodos exactos

Técnicas de diseño: backtracking

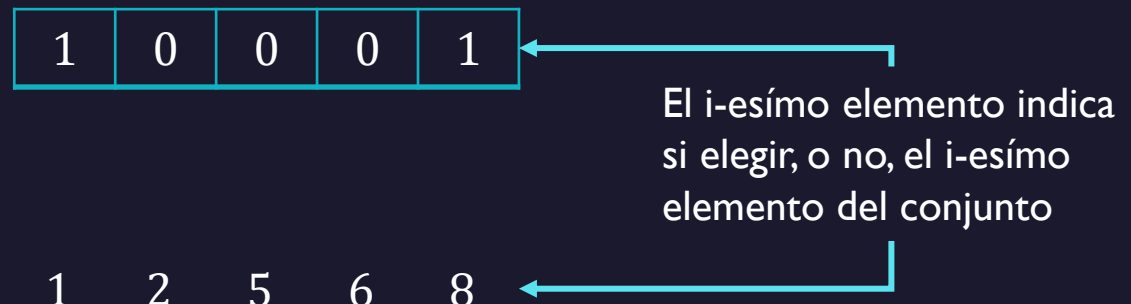
Primero, veamos un problema:

Set Sum Problem

Dado un conjunto S, buscamos un subconjunto de S, cuyo total sume d

Si $S = \{1, 2, 5, 6, 8\}$ y $d = 9$
las soluciones son $\{1, 8\}$ y $\{1, 2, 6\}$

Representación de la solución:
un arreglo de binarios



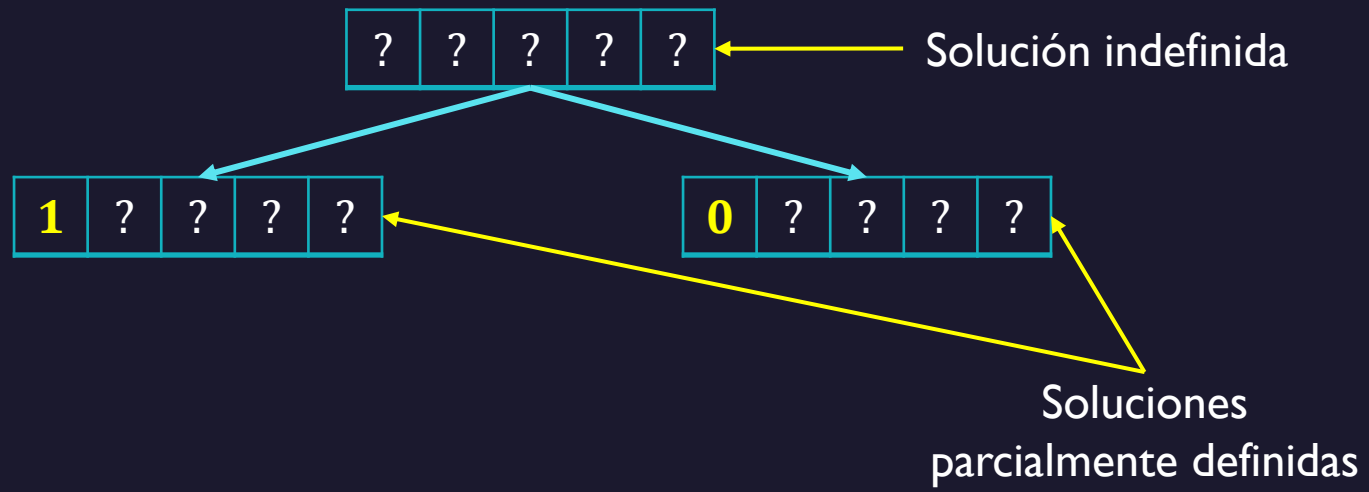
Entonces hay $n = |S|$ variables, y cada una tiene como dominio $\{0,1\}$

Es decir, habrá 2^n soluciones candidatas

¿Cómo podríamos organizarlas?

Técnicas de diseño: backtracking

Backtracking se basa en la idea de iniciar con una solución indefinida, y extenderla poco a poco, en diferentes ramas



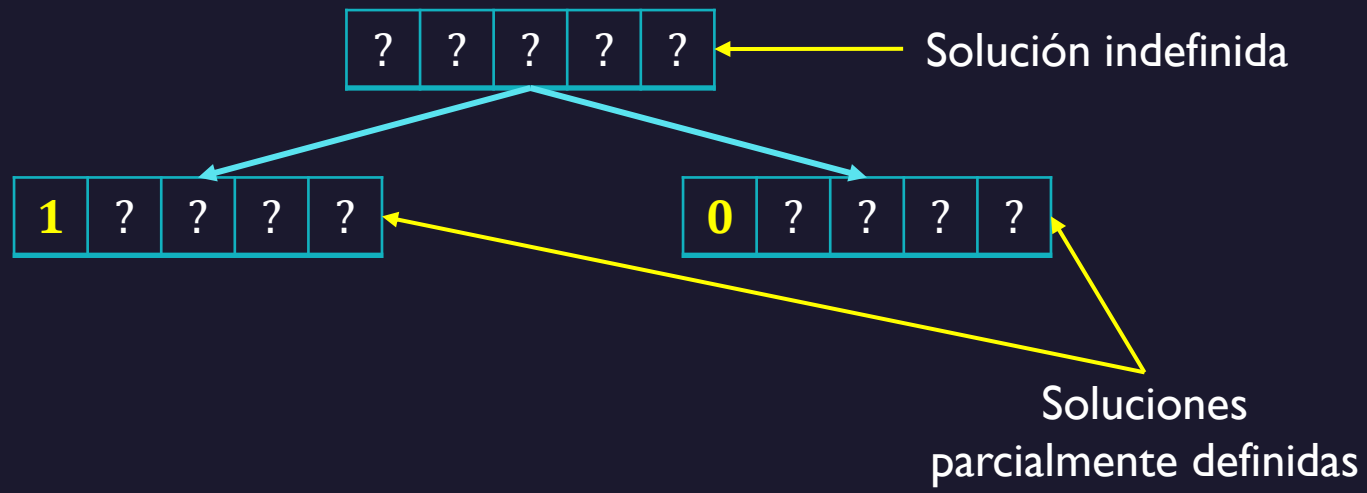
Podemos imaginarlo como un árbol:

- La raíz en la solución indefinida inicial
- Sus nodos hijos son soluciones parcialmente definidas

¿Por qué la raíz tiene 2 nodos hijos?

Técnicas de diseño: backtracking

Backtracking se basa en la idea de iniciar con una solución indefinida, y extenderla poco a poco, en diferentes ramas



Podemos imaginarlo como un árbol:

- La raíz en la solución indefinida inicial
- Sus nodos hijos son soluciones parcialmente definidas

¿Por qué la raíz tiene 2 nodos hijos?

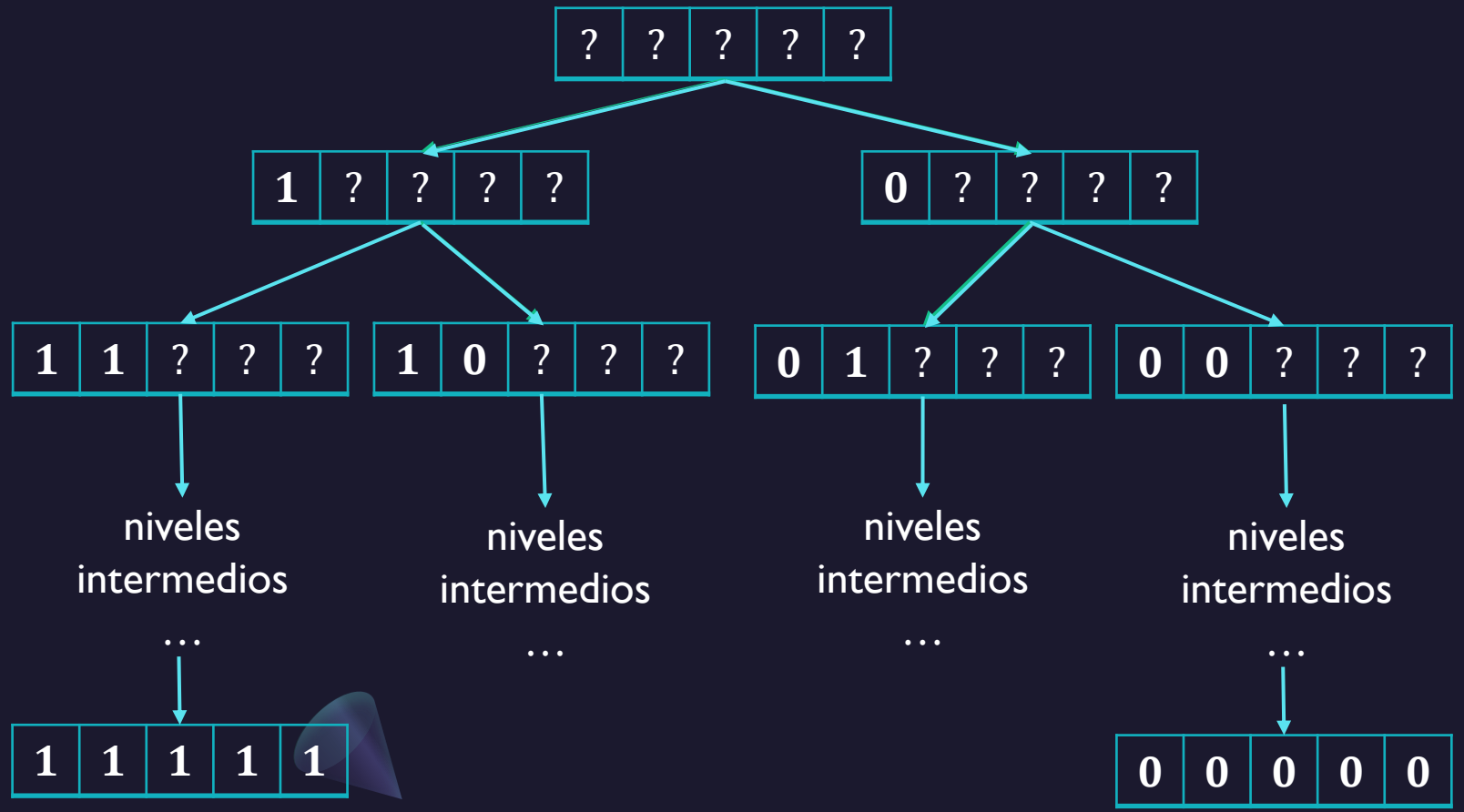
Porque la primer variable indefinida puede tener 2 valores: 0 o 1

Técnicas de diseño: backtracking

Podemos imaginarlo como un árbol:

- La raíz en la solución indefinida inicial
- Sus nodos hijos son soluciones parcialmente definidas
- Las hojas del árbol son soluciones completamente definidas

- Se llama árbol de estados (estados = soluciones)



Técnicas de diseño: backtracking

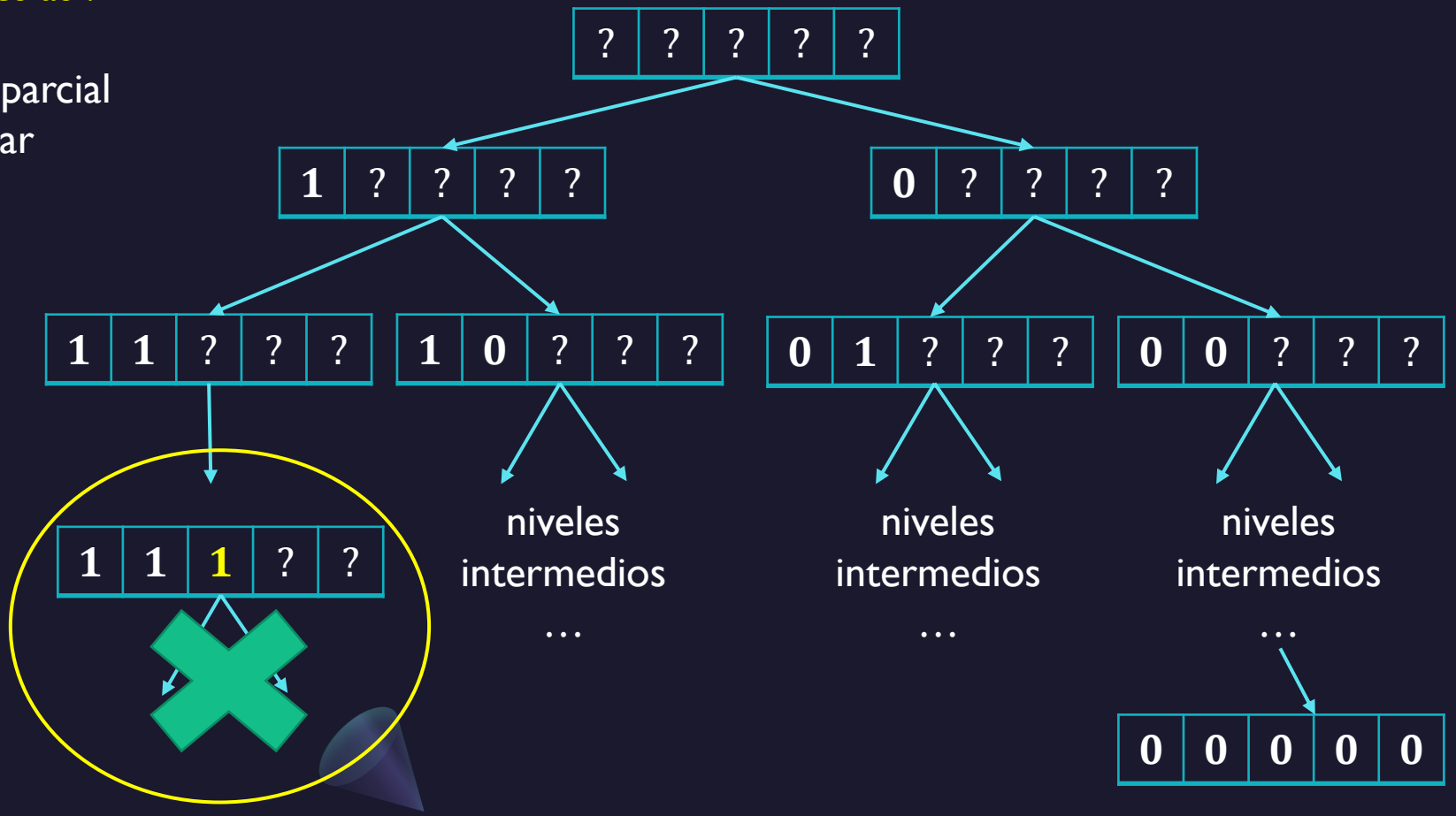
¿De que sirve imaginar las soluciones así?

Para que al encontrar una solución parcial que resulte invalida, sea posible evitar ramificar mas a partir de ella

En el caso de sum-set:
Si elegir el 3er elemento resulta en una suma que sobrepasa d, la solución es invalida
Y ya no será ramificada

Así, se evita computar todos los estados

¿Qué pasará en el peor caso?



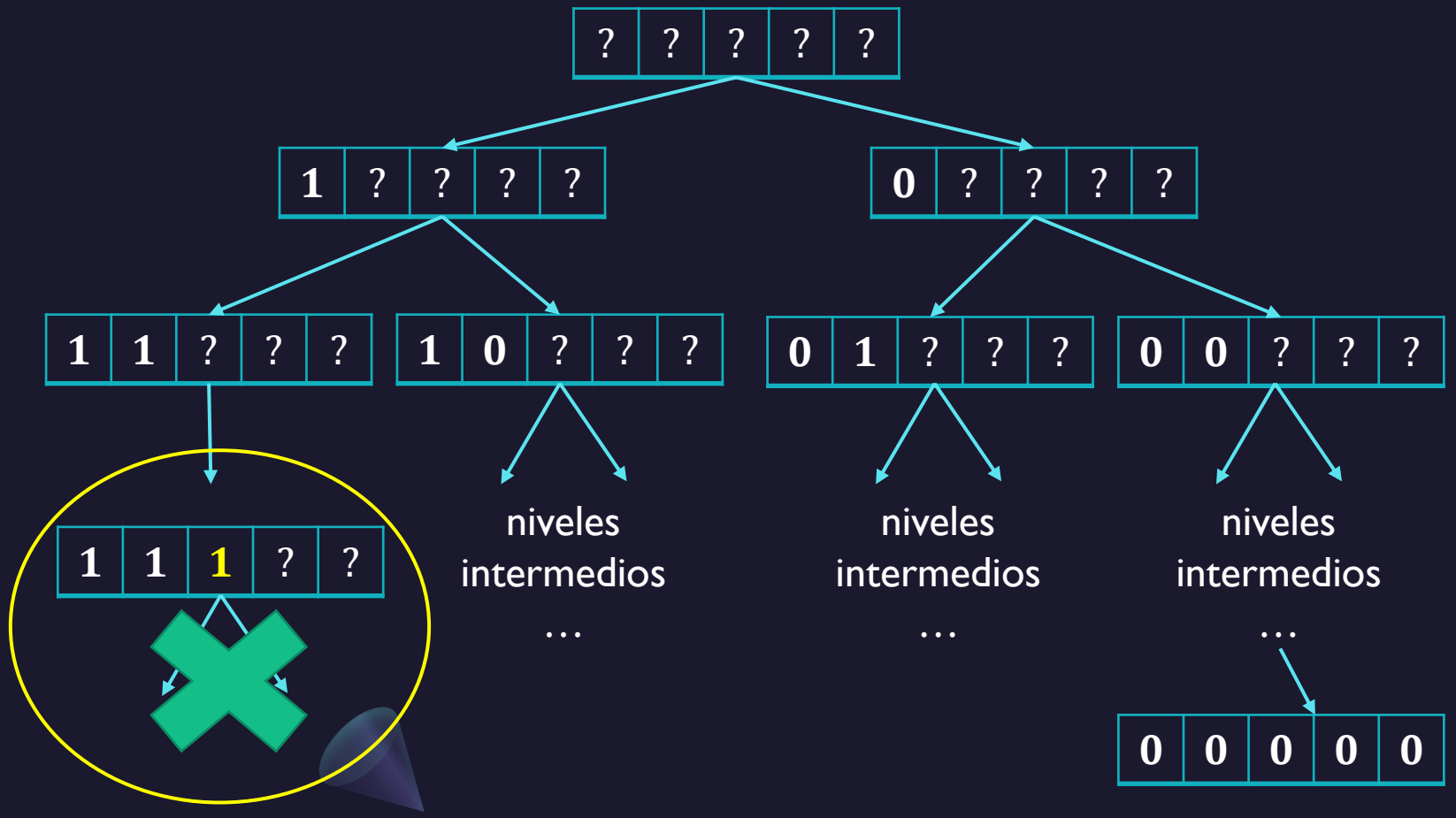
Técnicas de diseño: backtracking

¿Qué pasará en el peor caso?

En el peor caso, puede que se necesite computar todos los estados, si no fue posible descartar ramas

Así que la complejidad es exponencial, o factorial, según el problema

Para sum-set es $O(2^n)$



Técnicas de diseño: backtracking

¿Cómo se implementa?

Buena noticia: No necesitas mantener en memoria todo el árbol
¡Es suficiente con una fila (queue)!

El algoritmo general de Backtraking:

1. La solución raíz entra en la fila
2. Mientras la fila no este vacía, o no hayamos encontrado una solución completa:
    3. $partial\_sol$ = solución que sale de la fila
    4. i = primer variable indefinida de $partial\_sol$
    5. Por cada valor j en el dominio de i:
        6. Crea una solución hija, dándole a la variable i el valor j
        7. Evalúa si la solución hija es valida
        8. Si lo es, agregarla a la fila

¿Cómo evaluamos si la solución es válida para sum-set?

Técnicas de diseño: backtraking

Backtracking trabaja con problemas de satisfacción de restricciones

Por ejemplo, en sum-set buscamos un subconjunto que satisfaga la condición de que su suma sea d

Satisfacción de restricciones vs optimización:
la diferencia es ¿qué buscamos?

Solución factible: asignación de variables que cumple las restricciones

Solución optima: asignación de variables que minimiza (o maximiza) una función de costo

B&B: es un algoritmo muy parecido a backtracking, pero para problemas de optimización (próxima clase)

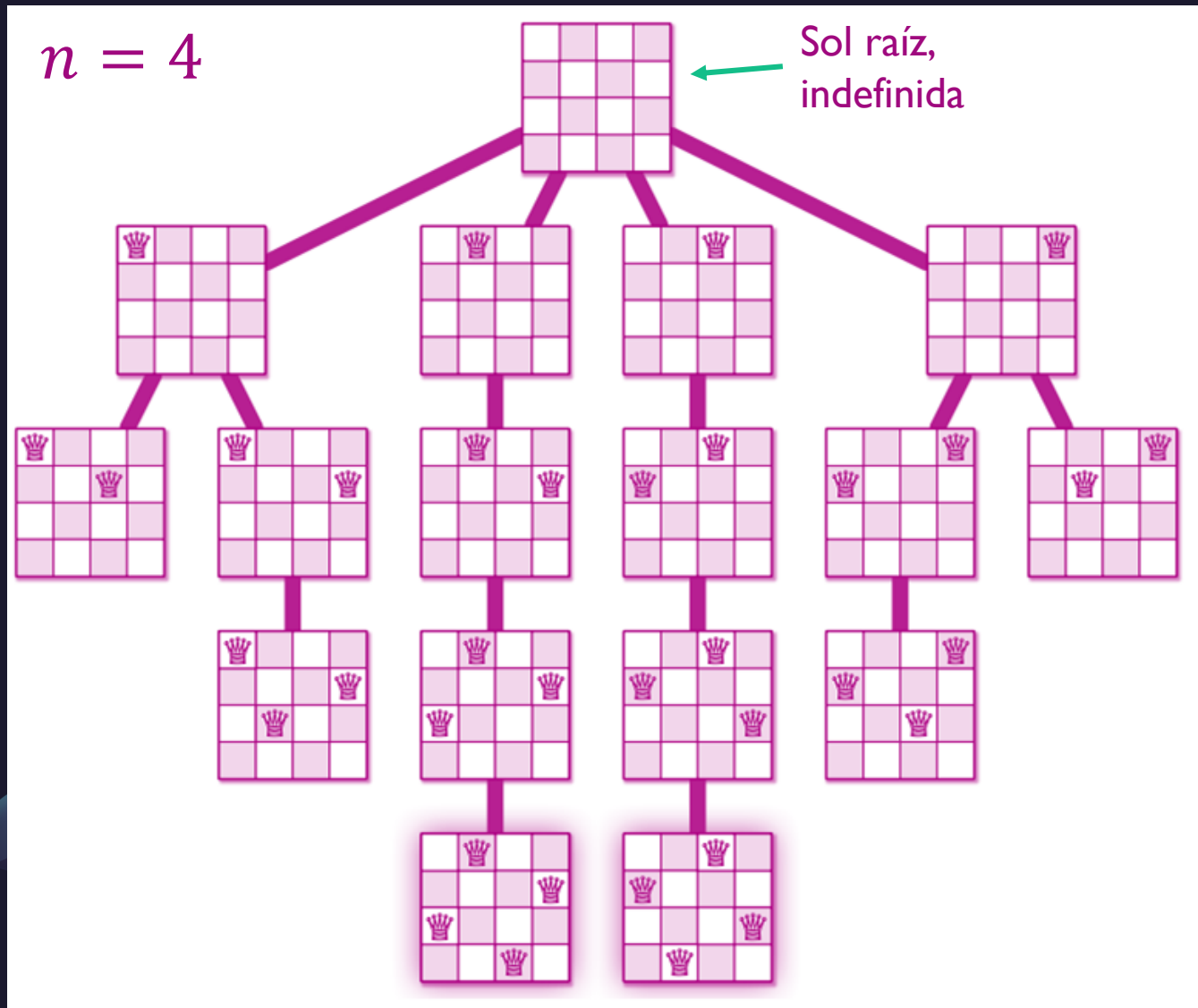
Técnicas de diseño: backtraking

Otro ejemplo: n-queens

Es el problema de colocar n reinas, en un tablero de ajedrez de $n \times n$

¿Cómo representas soluciones parciales?
Recuerda que:

- No necesitas todo el tablero
- No necesitas todo el árbol, la fila basta
- Y puedes asumir que habrá una reina por fila



Técnicas de diseño: backtraking

Basta con n variables: cada una es la tupla de coordenadas de una reina

?	?	?	?

← Solución indefinida, raíz

La primer variable indefinida: indica donde va la reina de la primer fila

Podría ser: (1,1), (1,2), (1,3) o (1,4)

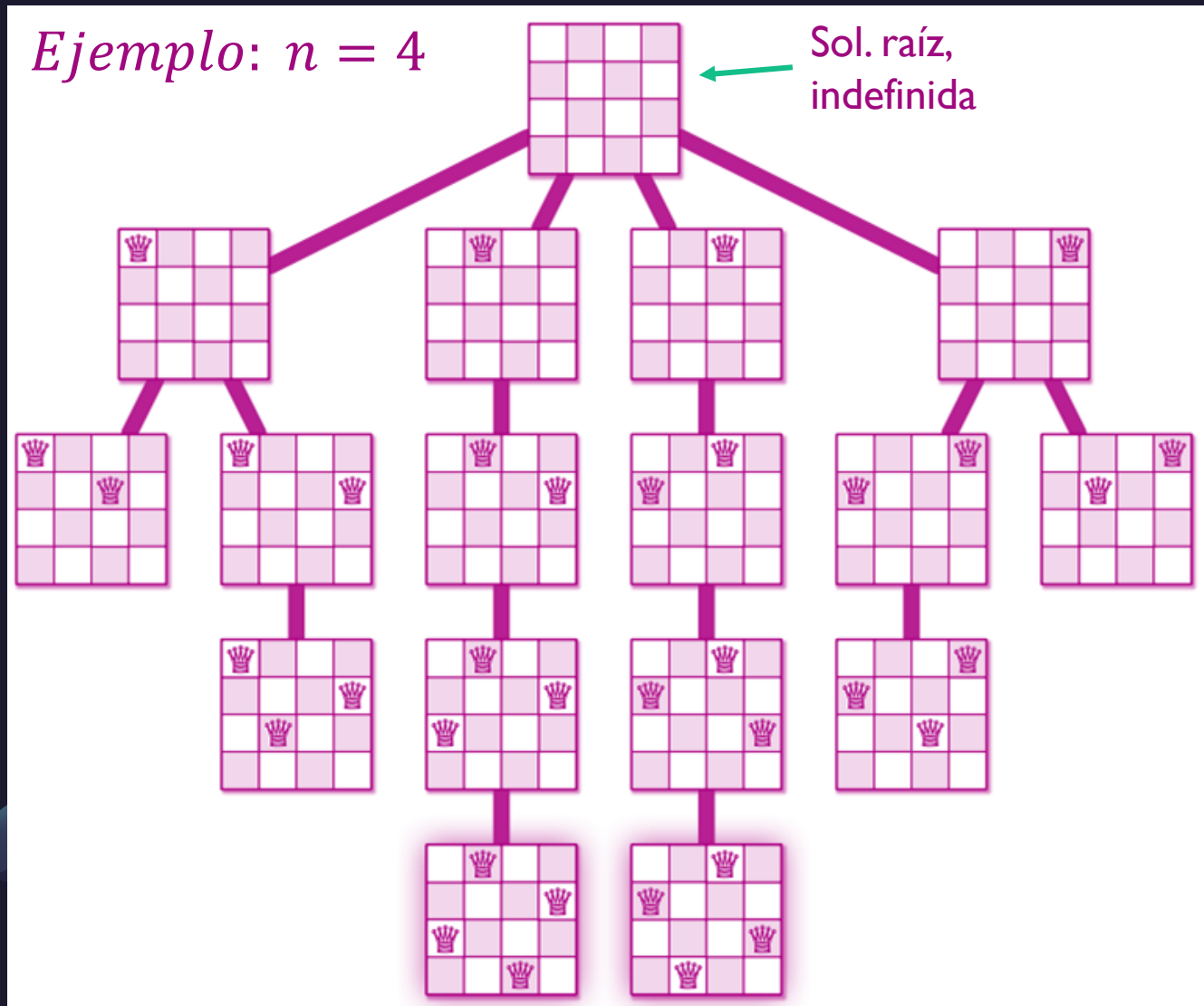
1,1	?	?	?

1,2	?	?	?

1,3	?	?	?

1,4	?	?	?

Las 4 posibles soluciones hijas de la raíz, obtenidas al definir la primer variable



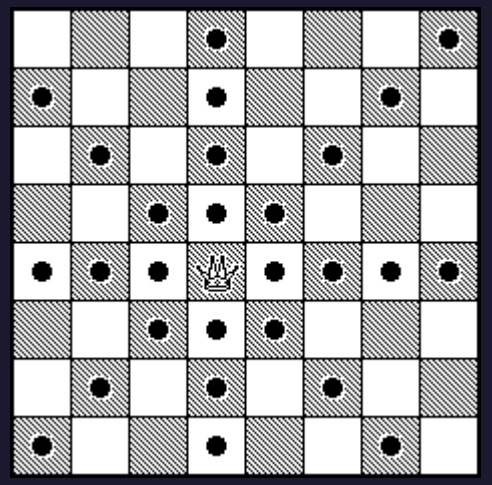
Técnicas de diseño: backtraking

¿Como verificas que las siguiente reina no rompa las condiciones del problema?

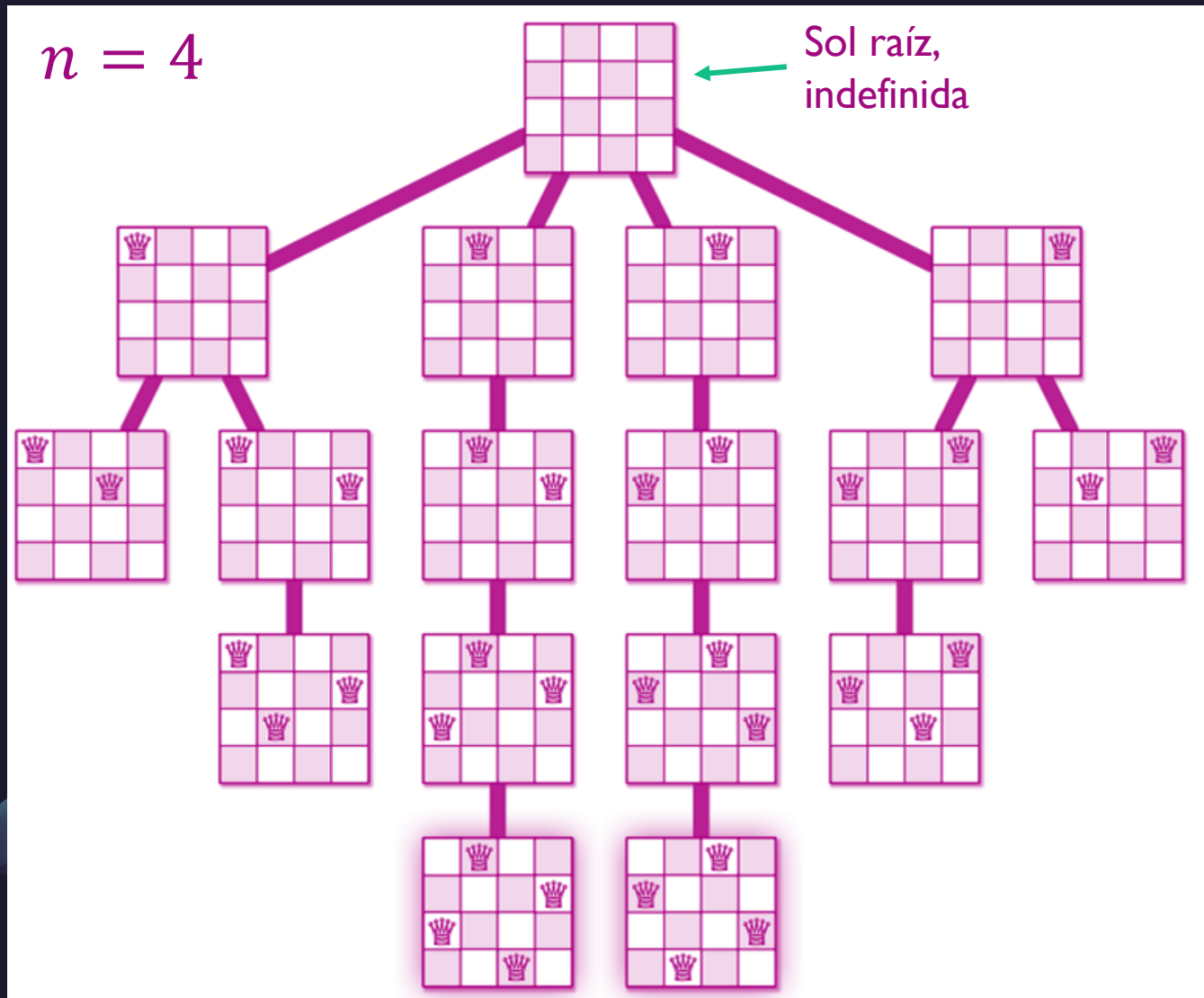
1,1	2,1	?	?

1,1	2,4	?	?

¿Serán válidas?



Una reina ataca toda su fila, columna y diagonales

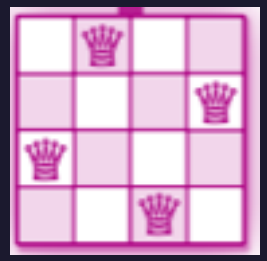


Actividad 1.4: n-Queens con backtraking

- Implementa algoritmo de backtracking para n-Queens en Python o C/C++ descrito en la diapositiva 10.
  Usa como base el notebook de ejemplo para sum set

- Debe funcionar con valores variables de n
- Debe usar una fila y ser iterativo
- Debe mostrar las soluciones como matrices que representan el tablero

Formato de salida para $n = 4$



=>

```
- R - -
- - - R
R - - -
- - R -
```

- Pregunta: ¿como se podría mejorar para llegar a una solución completamente definida mas rápido?
  - Pista: ¿de que podría servir otro tipo de estructura de datos para organizar a las soluciones hijas?