CSC 384
Introduction to Artificial Intelligence

CSP 1

Alice Gao and Randy Hickey

Winter 2023

Learning Goals

By the end of this lecture, you should be able to

Formulating a CSP

- Formulate a CSP as a Search Problem.
- Define a constraint using a table or using an expression.
- Explain the relative advantages and disadvantages of multiple CSP formulations.

Backtracking Search

- Trace the execution of the Backtracking Search algorithm.

Heuristics

- Choose a variable by using the Minimum-Remaining-Value heuristic or the Degree heuristic.
- Choose a value for a variable using the Least-Constraining-Value heuristic.

Outline

1. CSP Examples
2. Formulating a CSP
3. Backtracking Search
4. Heuristics

CSP EXAMPLES

Example: Scheduling

Want to schedule a time and a space for each final exam so that

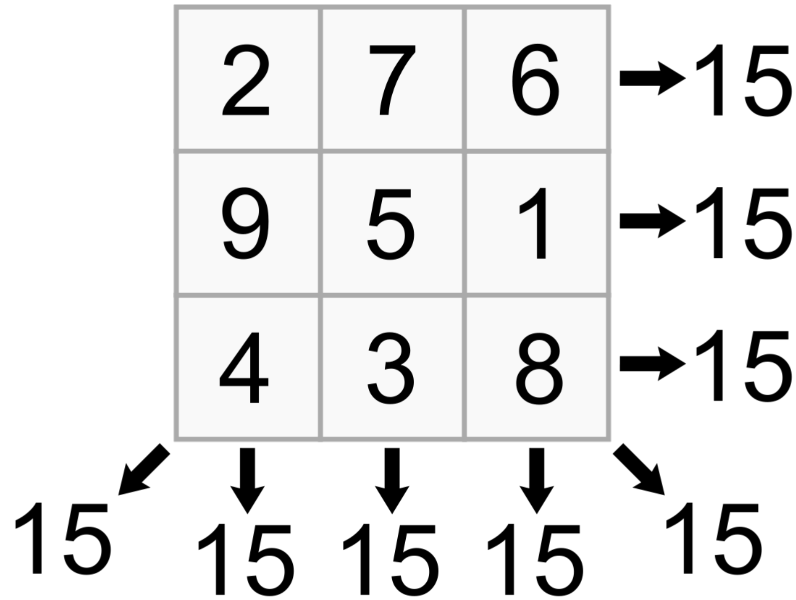
- No student is scheduled to take more than one final exam at the same time.
- The space allocated must be available at the time set.
- The space must be large enough to accommodate all the students taking the exam.

Example: Sudoku

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

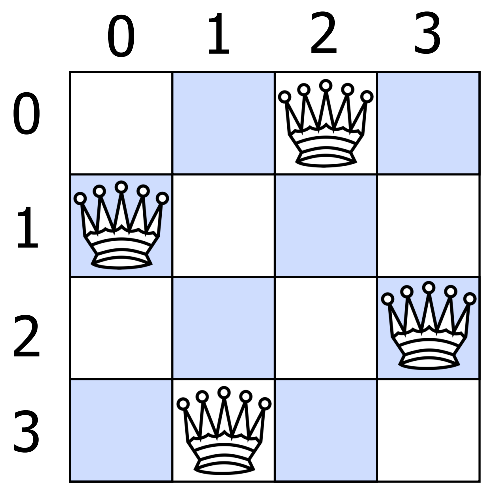
1	2	6	4	3	7	9	5	8
8	9	5	6	2	1	4	7	3
3	7	4	9	8	5	1	2	6
4	5	7	1	9	3	8	6	2
9	8	3	2	4	6	5	1	7
6	1	2	5	7	8	3	9	4
2	6	9	3	1	4	7	8	5
5	4	8	7	6	9	2	3	1
7	3	1	8	5	2	6	4	9

Example: Magic Square



FORMULATING A CSP

4-Queens Problem



4-Queens as a Search Problem

- State:
- Initial State:
- Successor Function:
- Goal States:
- (Optionally) Cost Function:
- (Optionally) Heuristic Function:

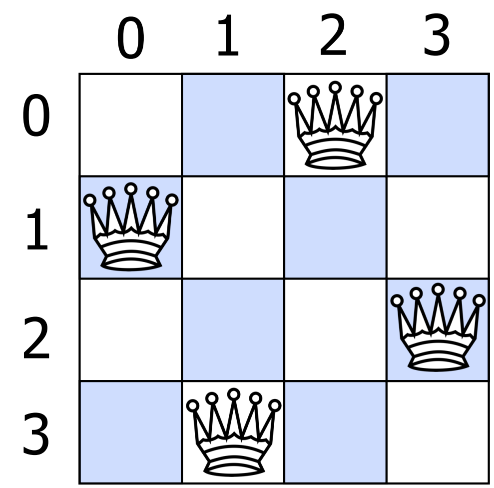
Defining a State of a CSP

Each state contains

- A set of variables.
- A domain of possible values for each variable.
- A set of constraints specifying the allowable value combinations.

A State for 4-Queens Problem

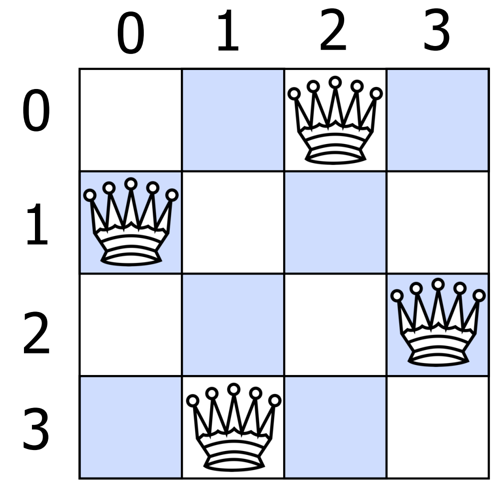
- Variables:



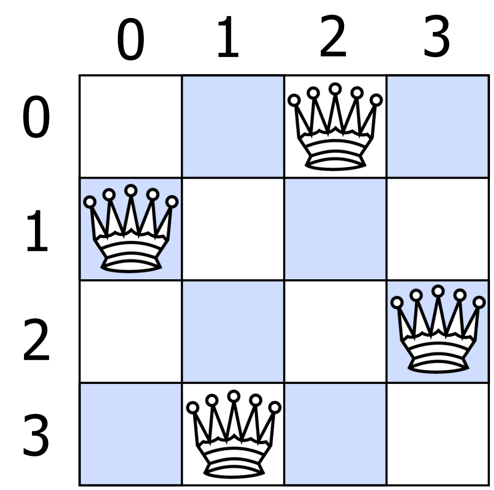
- Domains:
- Constraints:

A State for 4-Queens Problem (A)

- Variables: (x_i, y_i) is the position of the ith queen.
- Domains: $x_i \in \{0, 1, 2, 3\}$, $y_i \in \{0, 1, 2, 3\}$
- Constraints: No two queens are in the same row, column, or diagonal.



A State for 4-Queens Problem (B)



- Variables: $x_0\ x_1\ x_2\ x_3$
  - At most one queen in each column.
  - x_i is the row position of the queen in column i, where $i \in \{0, 1, 2, 3\}$.
  - $x_i = N$ means no queen is in column i.

- Domains: $D_{\{x_i\}} = \{N, 0, 1, 2, 3\}$ for all x_i.

- Constraints: No two queens are in the same row or diagonal.

Which 4-Queens Formulation Do You Prefer?

Formulation A

- Variables: (x_i, y_i) is the position of the ith queen.
- Domains: $x_i \in \{0, 1, 2, 3\}$, $y_i \in \{0, 1, 2, 3\}$
- Constraints: No two queens are in the same row, column, or diagonal.

Formulation B

- Variables: $x_0\ x_1\ x_2\ x_3$
  - At most one queen in each column.
  - x_i is the row position of the queen in column i, where $i \in \{N, 0, 1, 2, 3\}$.
  - $x_i = N$ means no queen is in column i.
- Domains: $D_{\{x_i\}} = \{N, 0, 1, 2, 3\}$ for all x_i.
- Constraints: No two queens are in the same row or diagonal.

4-Queens as a Search Problem

- State: see the previous slide.
- Initial state:
- Goal states:
- Successor function:

4-Queens as a Search Problem

- State:
  - Variables: $x_0\ x_1\ x_2\ x_3$,
    - At most one queen in each column. x_i is the row position of the queen in column i, $i \in \{0, 1, 2, 3\}$. $x_i = N$ means no queen is in column i.
  - Domains: $D_{\{x_i\}} = \{N, 0, 1, 2, 3\}$ for all x_i.
  - Constraints: No two queens are in the same row or diagonal.

- Initial state: the empty board.

- Goal state:
  - 4 queens on the board.
  - No two queens are in the same row or diagonal.

- Successor function:
  - Add a queen to the leftmost empty column.

Question: Which ones are the successors?

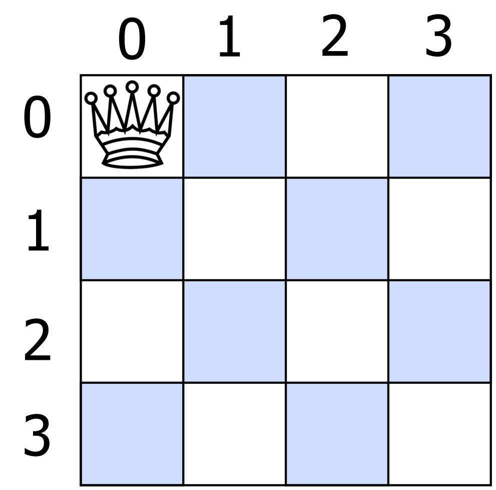
Which of the following is/are successor(s) of $0NNN$?

A. 00NN

B. 01NN

C. 02NN

D. 03NN



Answer: Which ones are the successors?

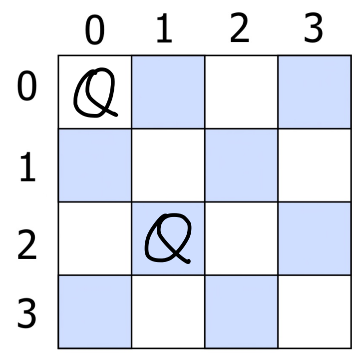
Which of the following is/are successor(s) of $0NNN$?

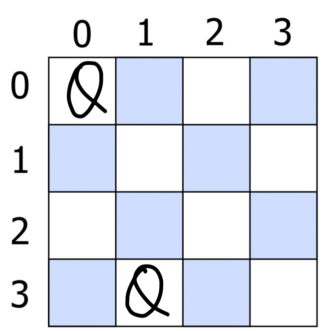
A. 00NN

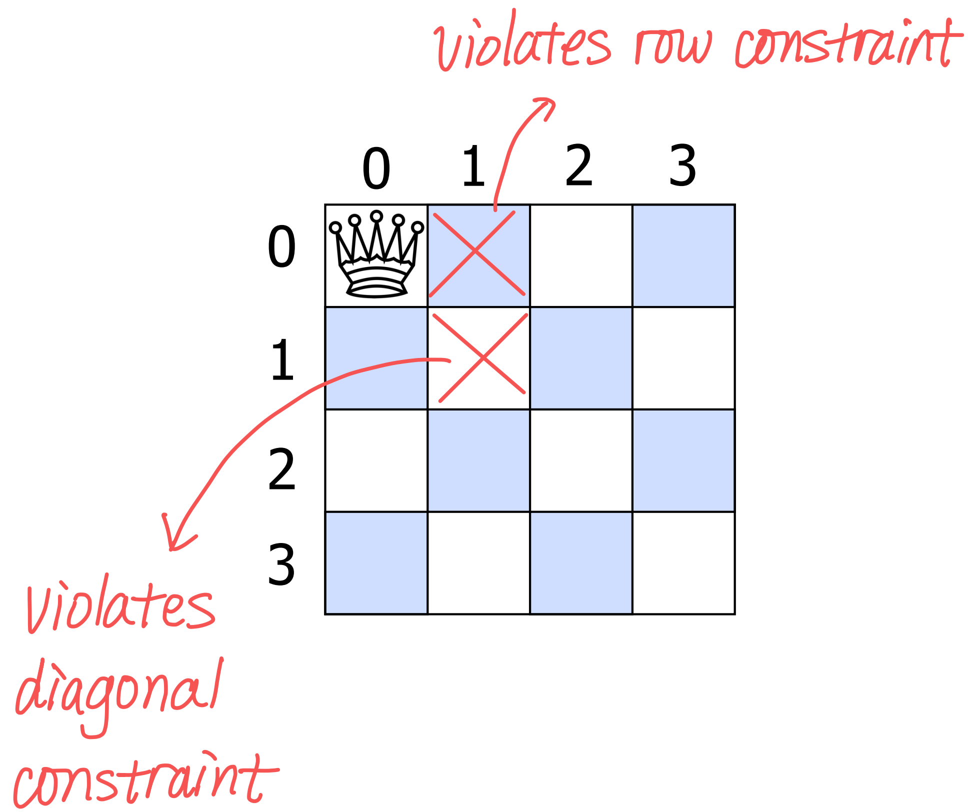
B. 01NN

C. 02NN

D. 03NN







Defining a Constraint in Two Ways

- List all allowable value combinations with a table.

x	y
0	3
1	2
2	1
…	…

- Write a logical expression (a compact version).

$$x + y = 3$$

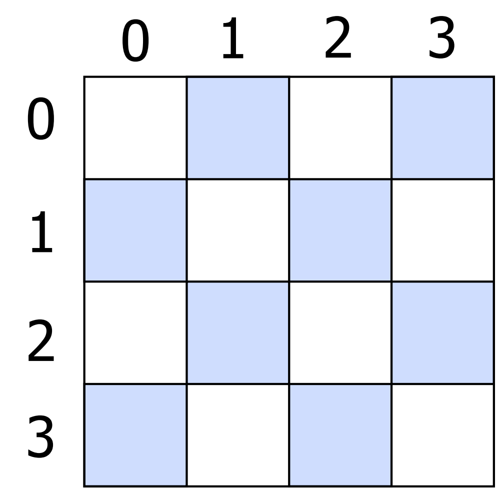
Defining a Constraint with a Table

How can we define the constraint below?

“queens x_1 and x_3 are not in the same row or diagonal.”

(1) Define this constraint in a table.

x_1	x_3



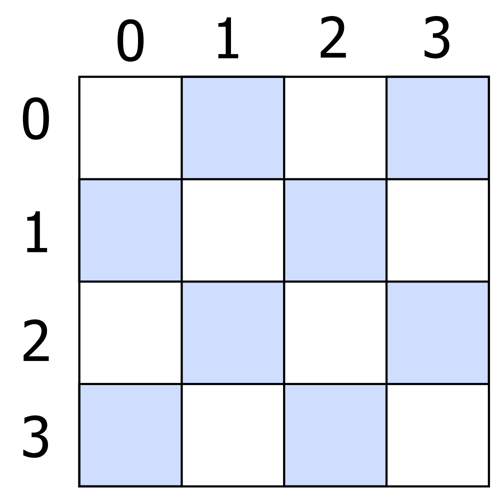
Defining a Constraint with a Table

How can we define the constraint below?

"queens x_1 and x_3 are not in the same row or diagonal."

(1) Define this constraint in a table.

x_1	x_3
0	1
0	3
1	0
1	2
2	1
2	3
3	0
3	2

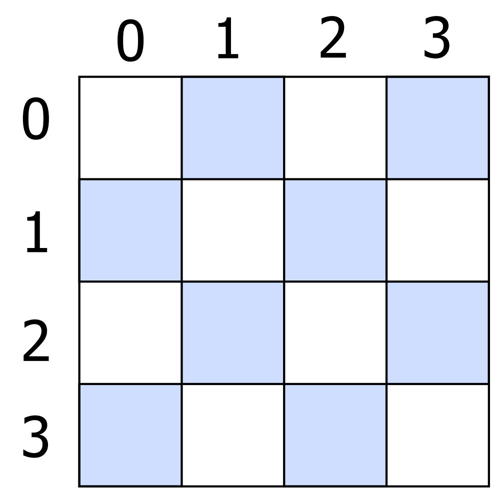


Defining a Constraint with an Expression

How can we define the constraint below?

"queens x_1 and x_3 are not in the same row or diagonal."

(2) Define this constraint using a logical expression.



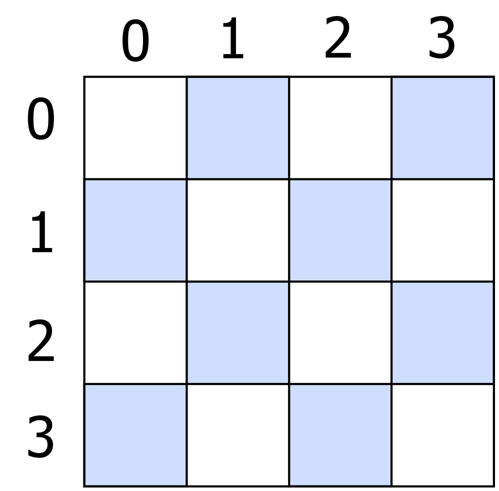
Defining a Constraint with an Expression

How can we define the constraint below?

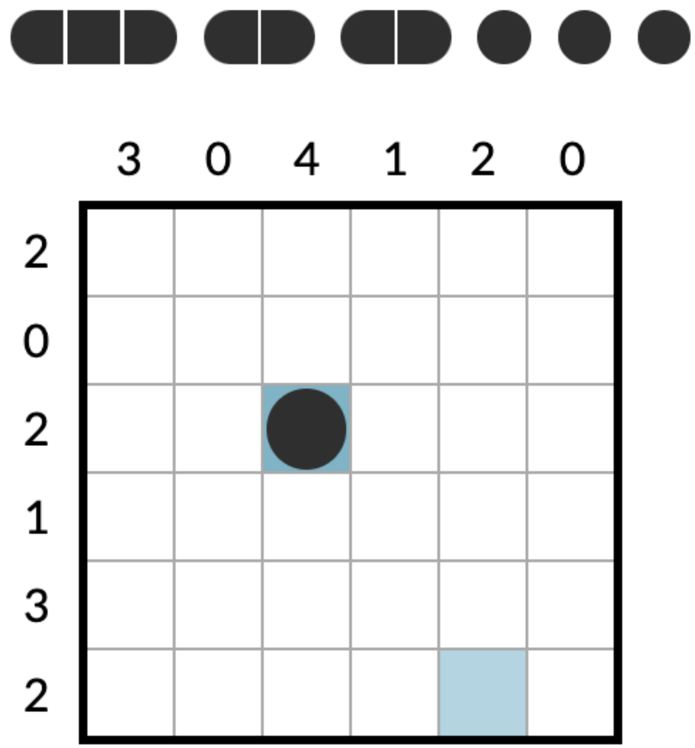
"queens x_1 and x_3 are not in the same row or diagonal."

(2) Define this constraint using a logical expression.

$$x_1 \neq x_3 \land |x_1 - x_3| \neq 2$$



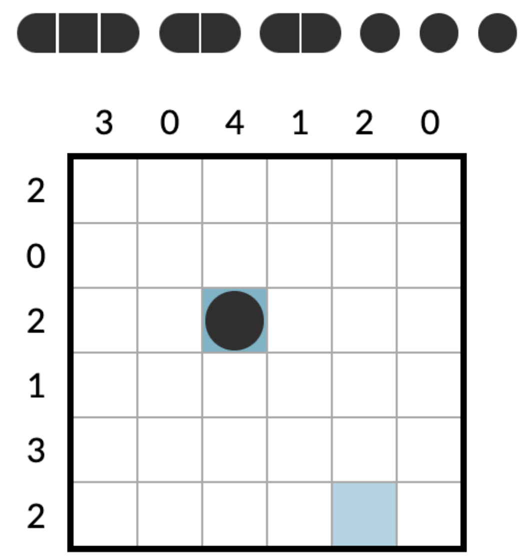
Battleship Solitaire



Play here: https://lukerissacher.com/battleships

A State for Battleship Solitaire (Cell Based)

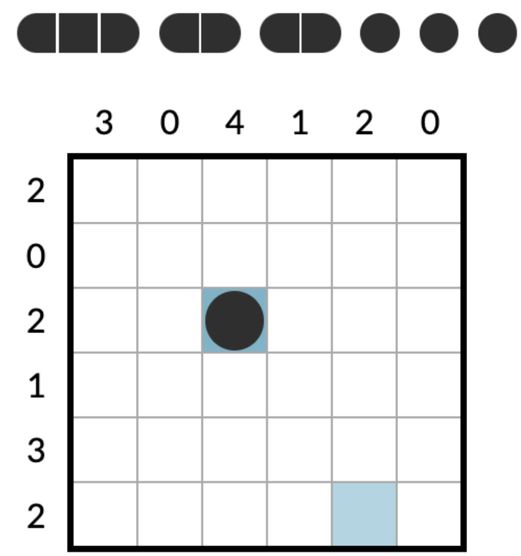
- Variables: each cell (x, y). $0 \leq x, y \leq dim$.
- Domains:
- Constraints:



A State for Battleship Solitaire (Cell Based)

- Variables: each cell (x, y). $0 \leq x, y \leq dim$.

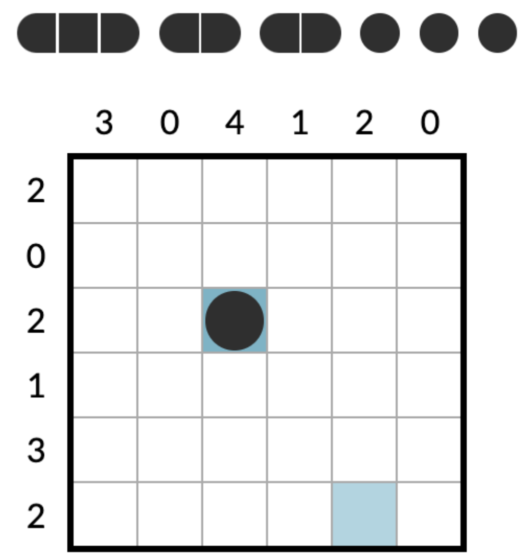
- Domains:
  - undetermined
  - water
  - an end of a ship (<, >, ^, v, or single)
  - a middle segment of a ship
  - an undetermined ship part



- Constraints:
  - **row** constraints: total ship parts in each row matches the number for the row.
  - **column** constraints: total ship parts in each column matches the number for the column.
  - **ship** constraints:
    - surround each ship completely with water.
    - ship parts form valid ships.
    - ships on the board match the set of ships.

A State for Battleship Solitaire (Ship Based)

- Variables: each ship's top-left corner location (x, y) and orientation (h or v). $0 \leq x, y \leq dim$.



- Domains:

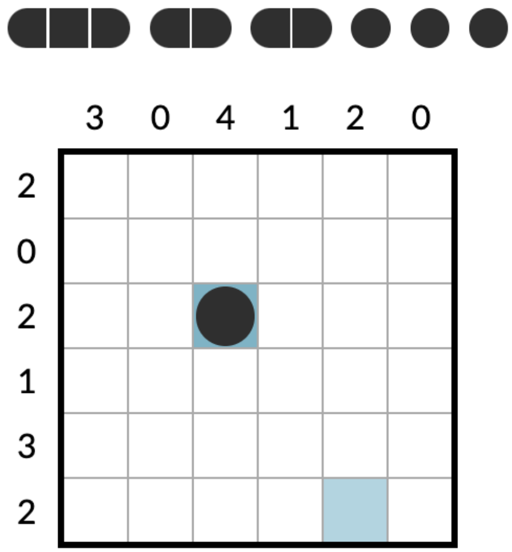
- Constraints:

A State for Battleship Solitaire (Ship Based)

- Variables: each ship's top-left corner location (x, y) and orientation (h or v). $0 \leq x, y \leq dim$.

- Domains:
  - undetermined.
  - all possible locations of a ship.

- Constraints:
  - **row** constraints: total ship parts in each row matches the number for the row.
  - **column** constraints: total ship parts in each column matches the number for the column.
  - surround each ship completely with water.
  - **ship** constraints:
    - surround each ship completely with water.
    - ships on the board match the set of ships.
  - ship locations match the **hints**.

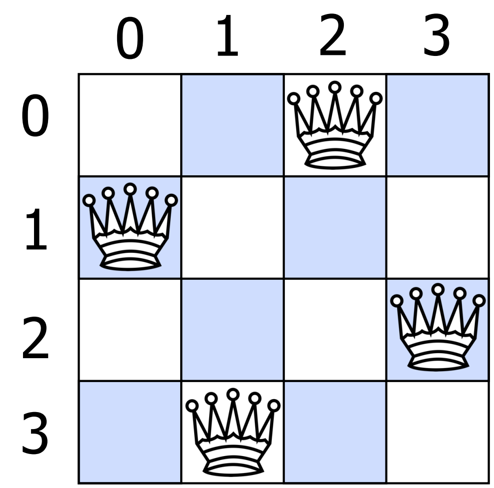


BACKTRACKING SEARCH

How are solving the two puzzles different?

Sliding Puzzle

4-Queens



How are solving the two puzzles different?

Sliding Puzzle

- **Know** what the goal state looks like (partially).
- Goal is to find a sequence of moves to a goal state.

4-Queens

- **Do not know** what the goal state looks like in advance.
- Goal is to find a goal state.
- Do not care about the sequence of moves to the goal state.

Solving a CSP as a Search Problem

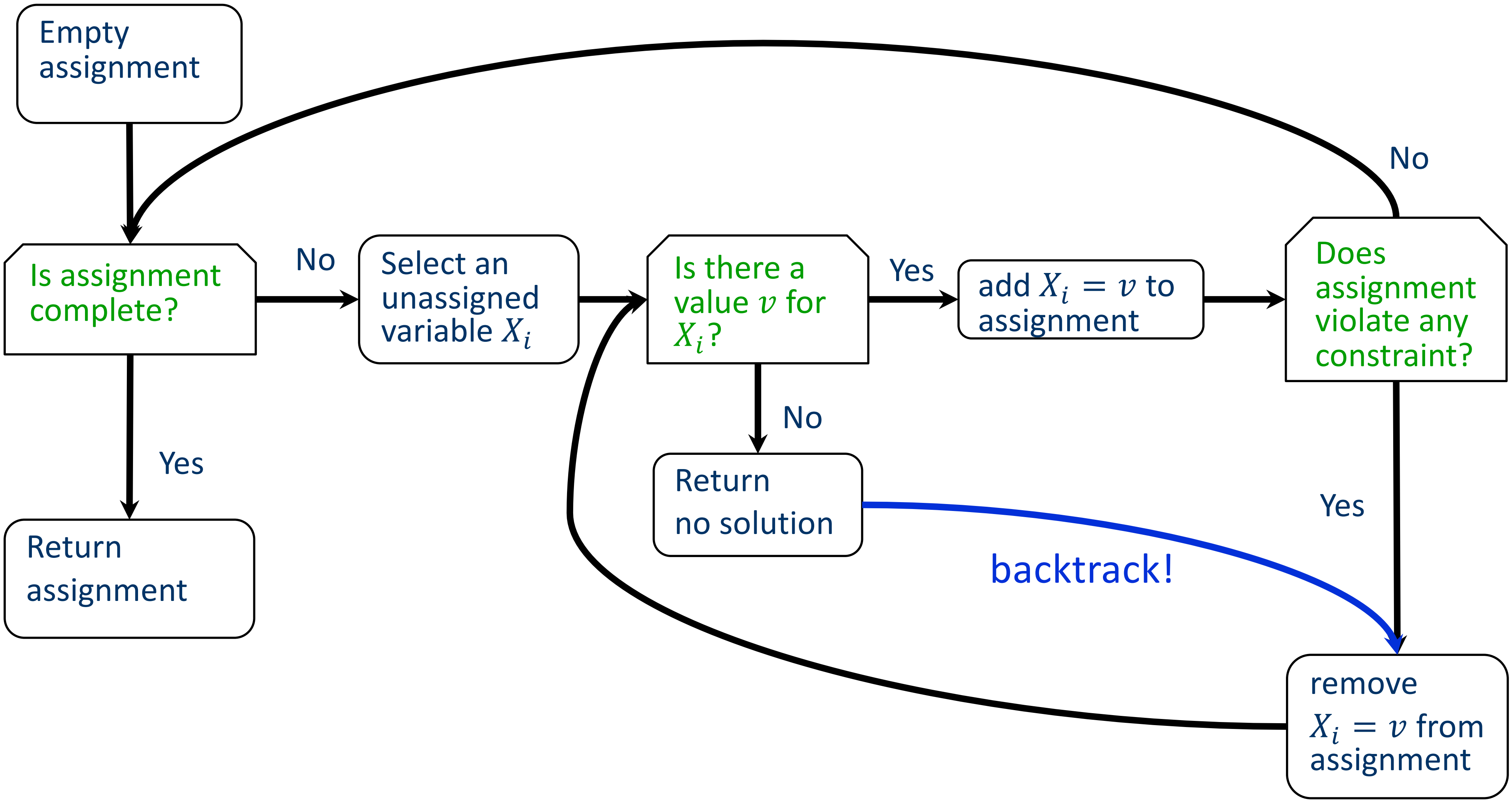
Do not care about finding a path to the goal.

Can use a specialized version of **depth-first search**.

Key ideas of **backtracking search**:

1. Start with the empty assignment.
2. Search through the partial assignments.
3. At each step, assign a value to an unassigned variable.
4. If a partial assignment violates a constraint, backtrack.

Backtracking Search (Flowchart)



Backtracking Search (Pseudocode)

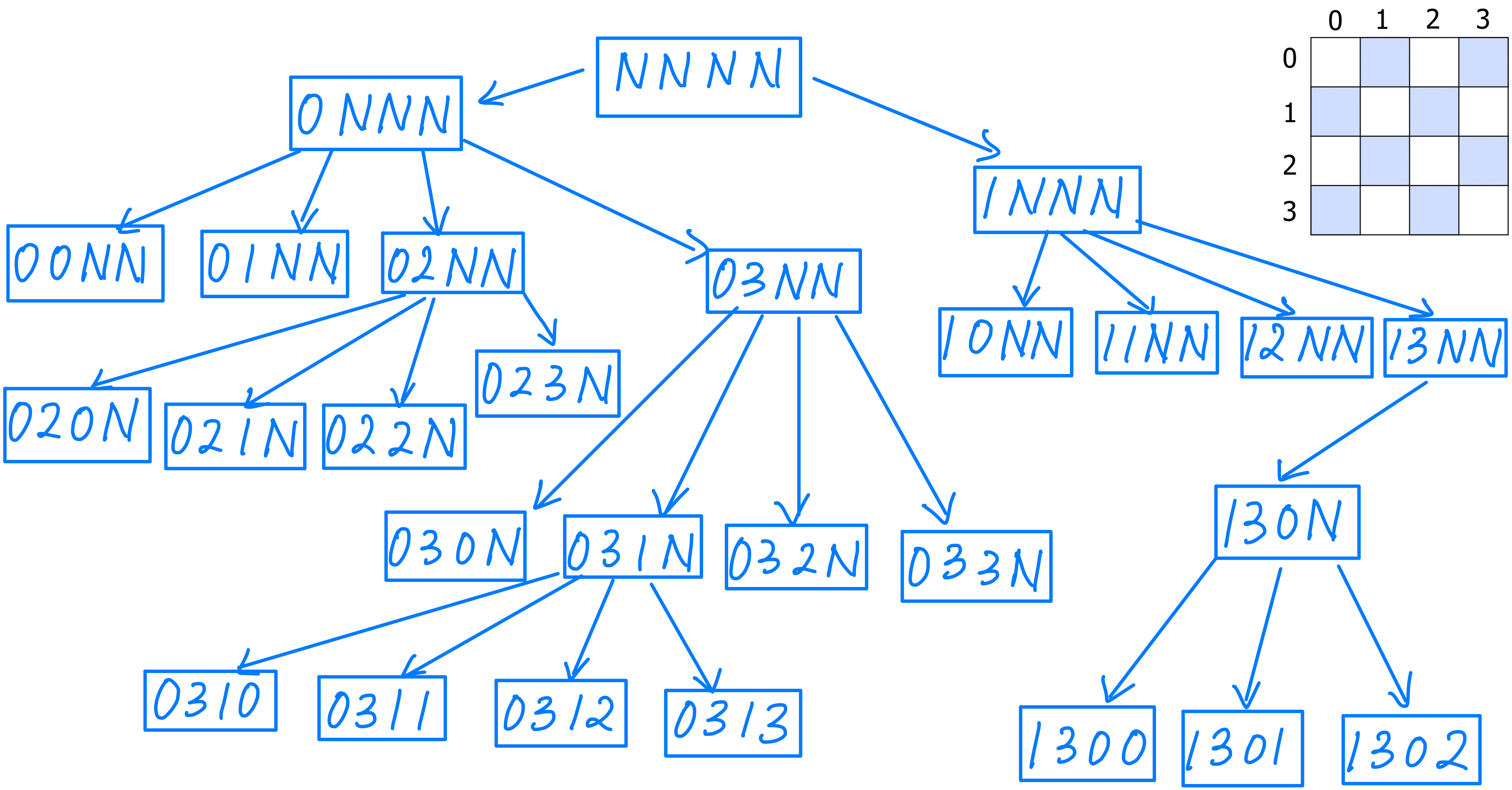
```
function BACKTRACKING-SEARCH(csp)
    return BACKTRACK({}, CSP)

function BACKTRACK(assignment, csp)
    if assignment is complete then return assignment
    var <- SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        add {var = value} to assignment
        if assignment does not violate any constraint then
            result <- BACKTRACK(assignment, csp)
            if result ≠ no solution then
                return result
        remove {var = value} from assignment
    return no solution
```

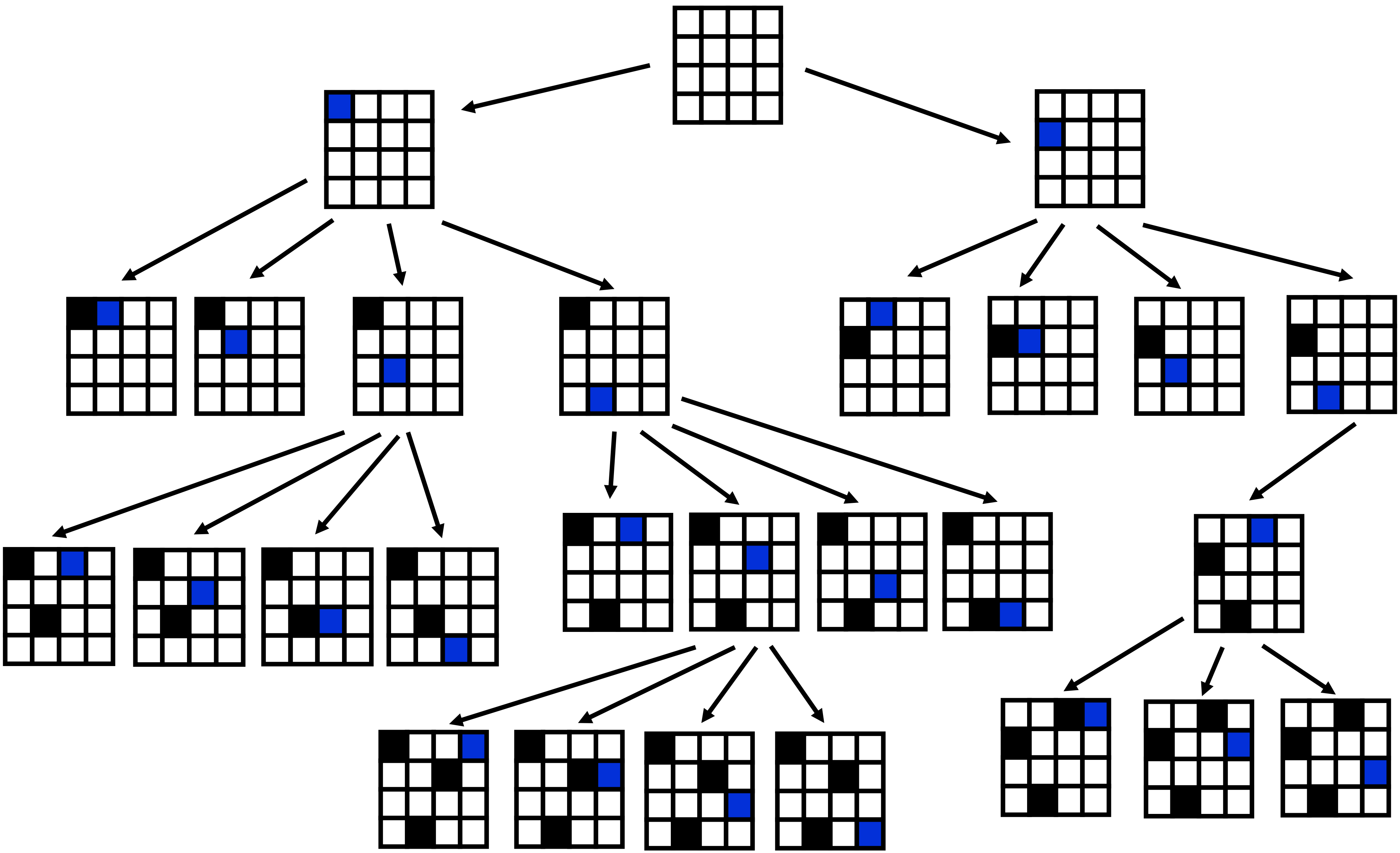
backtracking happens in 2 cases:

1. assignment violates a constraint.
2. recursive call returns "no solution"

Solving 4-Queens using Backtracking Search



Solving 4-Queens using Backtracking Search



HEURISTICS

Heuristics for Variable and Value Ordering

- How do we choose the next **variable** to consider?
  - Minimum-remaining-values heuristic
    - Most powerful
    - "fail-first" heuristic.
  - Degree heuristic
    - Helpful for choosing the first variable.
    - A useful tie-breaker.
- How do we choose the next **value** to consider?
  - Least-constraining-value heuristic
    - "fail-last" heuristic.

Heuristics to Choose the Next Variable

Minimum-Remaining-Values (MRV) Heuristic

- Choose the variable with the fewest “legal” values
- a.k.a. the “most constrained variable” heuristic
- a.k.a. the “fail-first” heuristic.
  - picks a variable that is most likely to cause a failure soon.
  - prune large parts of the tree earlier.

Degree Heuristic

- Select the variable that is involved in the largest number of constraints on other unassigned variables.

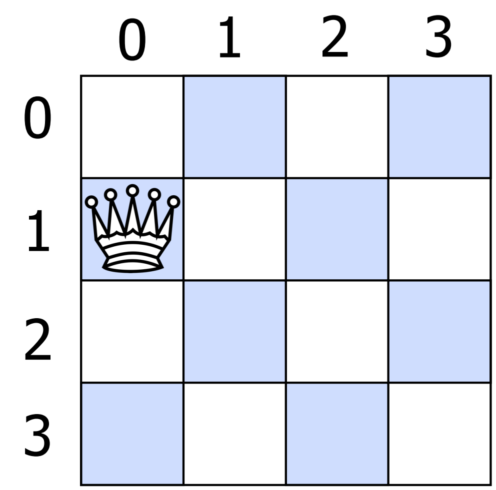
Heuristics to choose the next variable

- **MRV** heuristic is more powerful than degree heuristic.

- **Degree** heuristic is helpful for choosing the first variable.
- **Degree** heuristic is a useful tie-breaker.

Q1: Applying the MRV Heuristic

Consider the 4-queens state.



Based on the MRV heuristic,

which **variable** should we choose next?

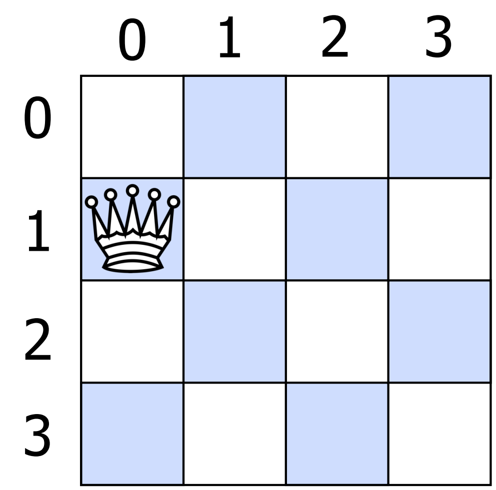
A. x_1

B. x_2

C. x_3

Answer 1: Applying the MRV Heuristic

Consider the 4-queens state.



Based on the MRV heuristic,

which **variable** should we choose next?

A. x_1

B. x_2

C. x_3

Q2: Applying the Degree Heuristic

Based on the degree heuristic,

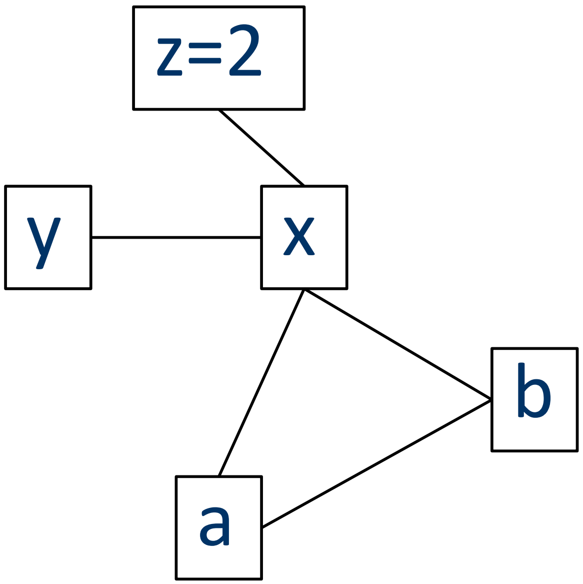
which **variable** should we choose next?

A. a

B. b

C. x

D. y



Answer 2: Applying the Degree Heuristic

Based on the degree heuristic,

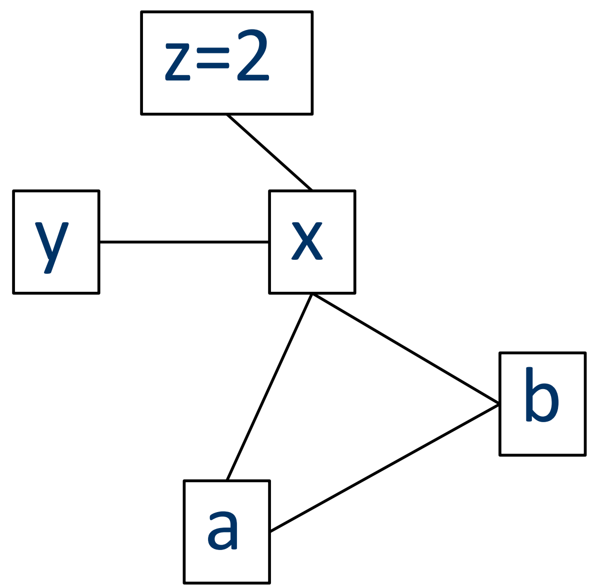
which **variable** should we choose next?

A. a

B. b

C. x

D. y



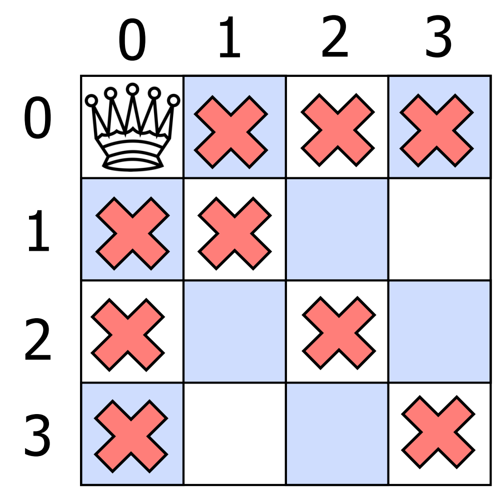
Least-Constraining-Value Heuristic

- How do we select which **value** to examine first?
- Choose the value that rules out the fewest choices for the neighboring variables.
- Leave the maximum flexibility for subsequent variable assignments.
- a.k.a. “fail-last” heuristic.
  - Only need one solution. Look for the most likely values first.

Q3: Applying the LCV Heuristic

Consider the 4-queens state.

Consider the variable x_1 next.



Based on the LCV heuristic,

which **value** should we choose next?

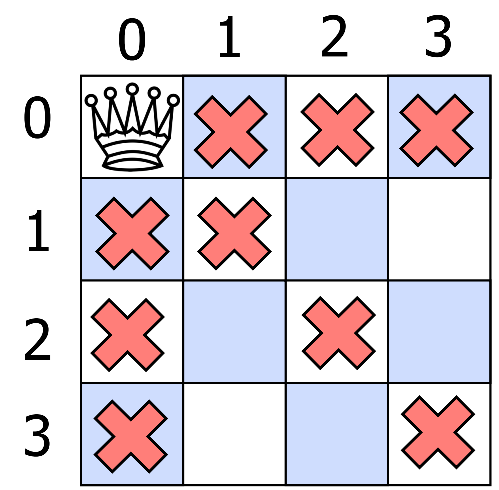
$A.\ \ x_1 = 2$

$B.\ \ x_1 = 3$

Answer 3: Applying the LCV Heuristic

Consider the 4-queens state.

Consider the variable x_1 next.



Based on the LCV heuristic,

which **value** should we choose next?

A. $x_1 = 2$

B. $x_1 = 3$