BACKTRACKING ALGORITHMS

Problems

- N-queen
- N-puzzle
- Sudoku
- And many more……..

N-Queen Problem

Problem Definition

- Place N chess queens on an N×N chessboard so that no two queens threaten each other.
- No two queens share the same row, column, or diagonal.
- The goal is to find all possible configurations (or one configuration, depending on the variant) that satisfy the constraint.

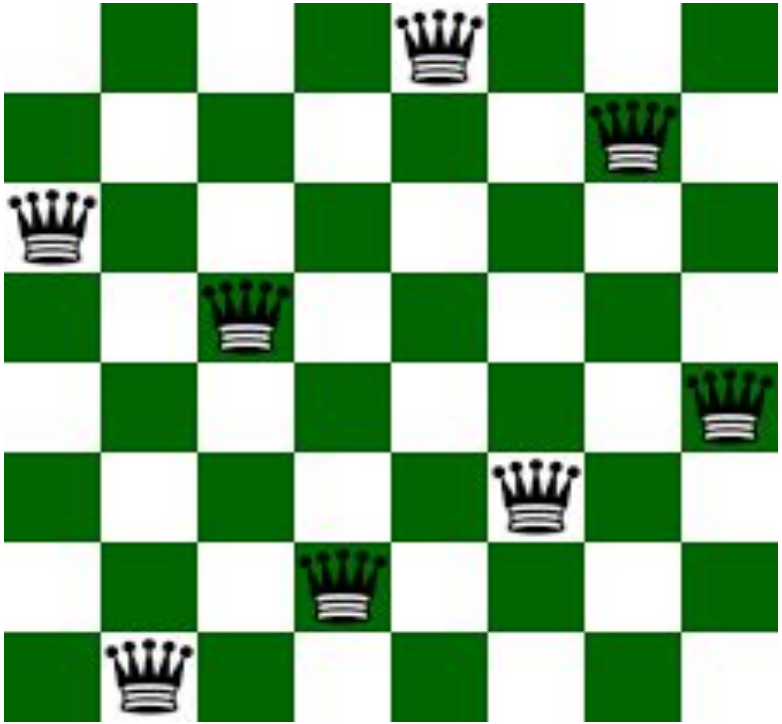
N-Queen Problem

Input

- A single integer n — the size of the chessboard and the number of queens to place.

N-Queen Problem

Sample Output



N-Queen Problem

Observations

- Only one queen per row and one queen per column — simplifies the problem.
- We can iterate row by row, placing one queen per row.

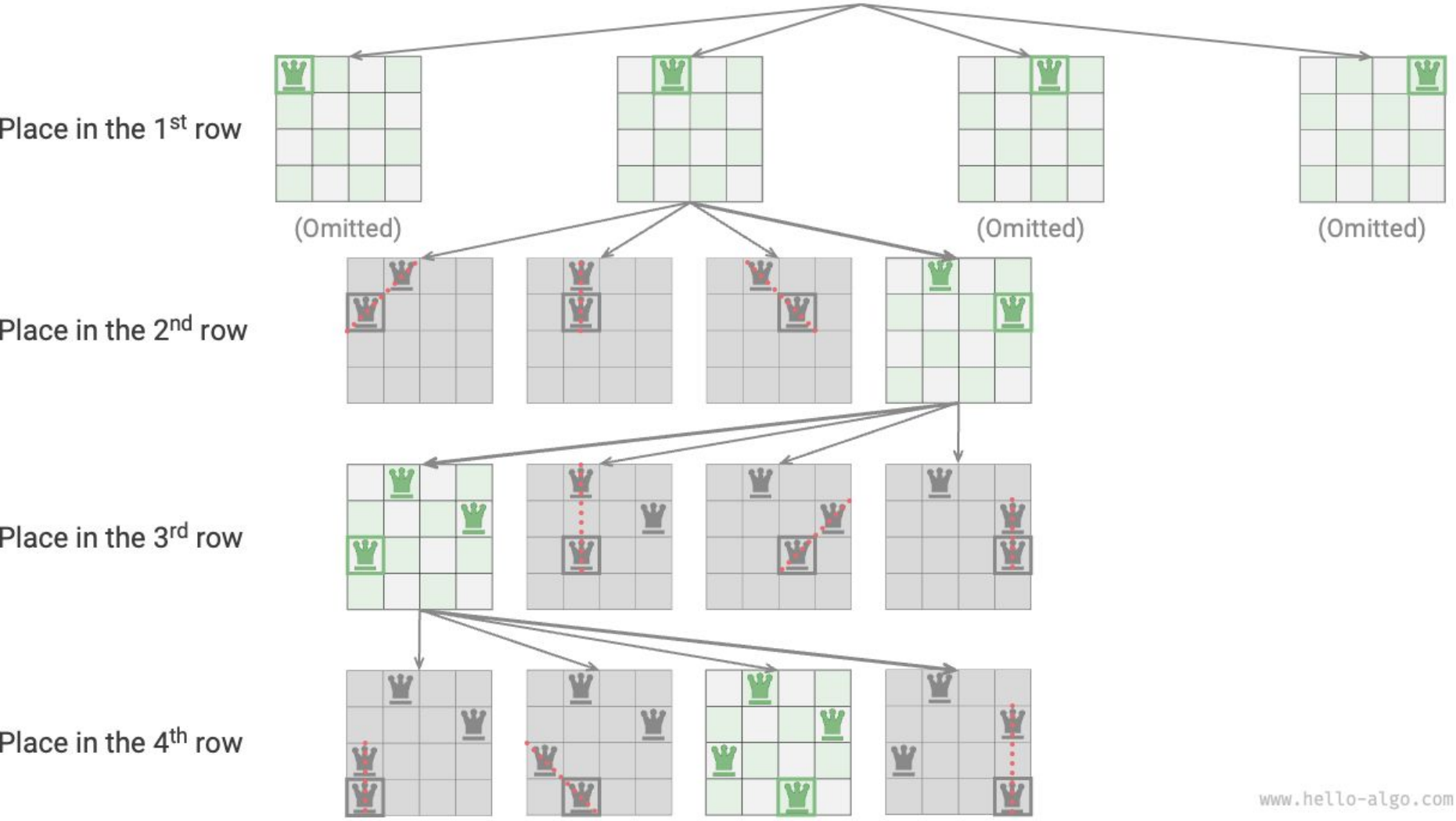
Valid queen placement must ensure:

- No other queen is in the same column.
- No other queen is on the same diagonal.

N-Queen Problem

Steps to Solve (Backtracking)

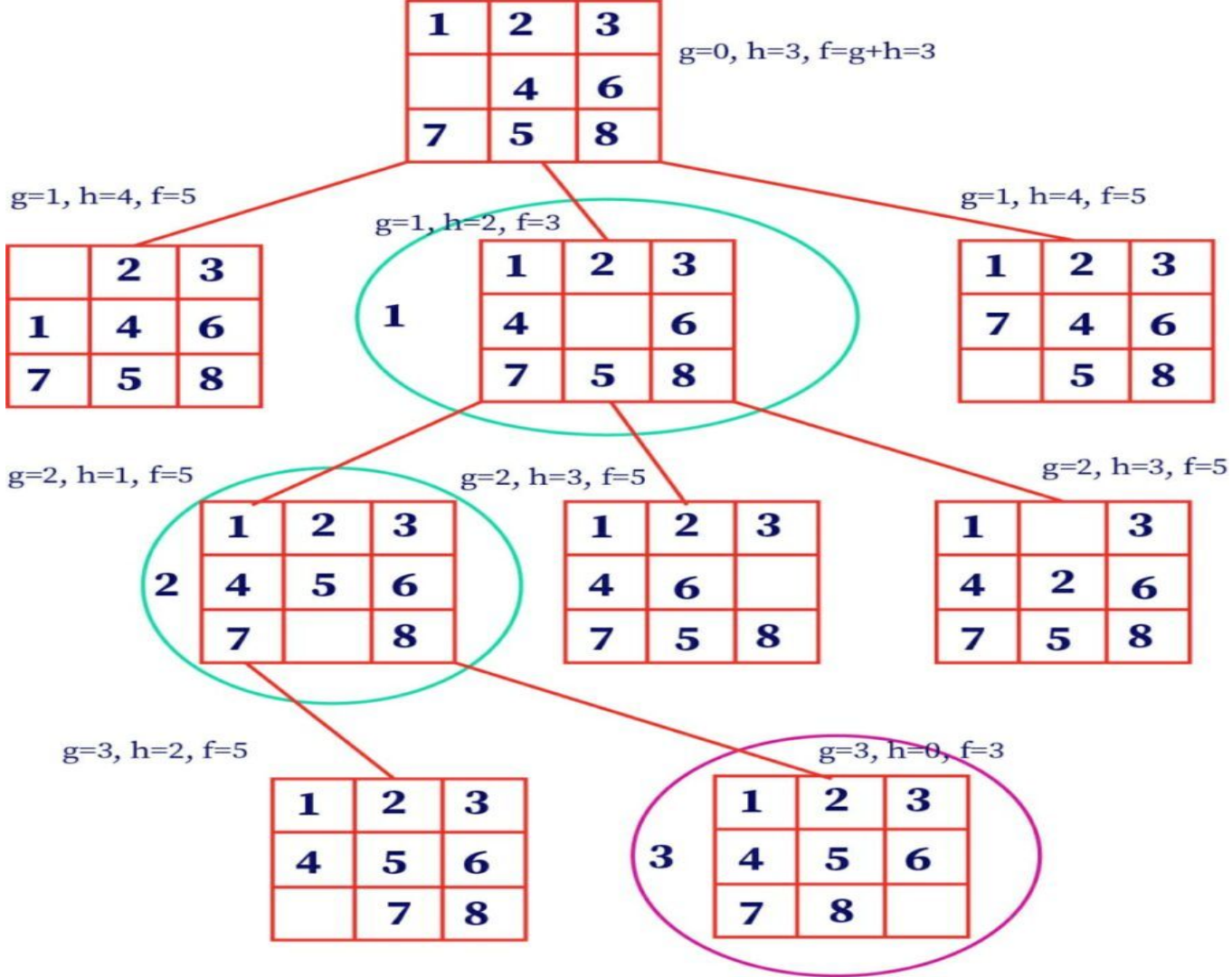
1. Start from row 1.
2. For each column col in row:
    - Check if placing a queen at (row, col) is valid.
3. If valid:
    - Place queen, mark column and diagonals.
    - Recur for row + 1.
    - Backtrack: remove queen and unmark.
4. If row == (n+1), a valid configuration is found. Save it.


Place in the 1st row
(Omitted)
(Omitted)
(Omitted)
Place in the 2nd row
Place in the 3rd row
Place in the 4th row
www.hello-algo.com

In general, how Backtracking algo works?

- Define ***'State'*** and ***'Goal State'*** criteria.
- Start at ***initial state***.
- Figure out what the ***possible moves/options*** are.
- Make decision.
- Check for ***Validity.***
- ***Backtrack*** if necessary.
- Continue ***Exploring***.
- Find the ***Solution*** or ***Exhaust All Options.***

N-Puzzle


1 2 3
4 6
7 5 8
g=0, h=3, f=g+h=3
g=1, h=4, f=5
2 3
1 4 6
7 5 8
g=1, h=2, f=3
1
1 2 3
4 6
7 5 8
g=1, h=4, f=5
1 2 3
7 4 6
5 8
g=2, h=1, f=5
2
1 2 3
4 5 6
7 8
g=2, h=3, f=5
1 2 3
4 6
7 5 8
g=2, h=3, f=5
1 3
4 2 6
7 5 8
g=3, h=2, f=5
1 2 3
4 5 6
7 8
g=3, h=0, f=3
3
1 2 3
4 5 6
7 8

Define ***'State'*** and ***'Goal State'*** criteria.

State

A state represents the current configuration of the puzzle board. For an n-puzzle:

Representation: 2D array or 1D array representing tile positions

Example (3x3): [[1,2,3], [4,0,6], [7,5,8]] where 0 represents the empty space

Goal State

The target configuration we want to reach:

Standard goal: [[1,2,3], [4,5,6], [7,8,0]] for 8-puzzle

General form: Numbers 1 to n-1 in order, with empty space (0) at bottom-right

Start at ***initial state***

Starting configuration of the puzzle:

- Given scrambled arrangement of tiles
- Must be a valid, solvable configuration

Figure out what the ***possible moves/options*** are

Possible Moves/Options

Available actions from ***current state:***

Move empty space: Up, Down, Left, Right

Equivalent: Slide adjacent tile into empty space

Constraints: Cannot move outside board boundaries

Make decision

Choose which move to explore:

- Select one of the valid moves from current state
- Apply move to generate new state
- Add new state to exploration path

Check for ***Validity***

Verify if the chosen move/state is acceptable:

- **Boundary check**: Move doesn't go outside grid
- **Cycle detection**: New state hasn't been visited before in current path
- **Goal check**: Determine if goal state is reached

Backtrack if necessary

When current path leads to dead end:

Trigger: No more valid moves available or maximum depth reached

Action: Return to previous state, undo last move

Try alternative: Explore different move from previous state

Continue ***Exploring***

Systematic exploration process:

- If goal not found and valid moves exist, continue exploring
- Maintain path history to avoid cycles
- Use depth-first search approach

Solution or Exhaust All Options

Terminal conditions:

Solution found: Return the sequence of moves leading to goal

No solution: All possible states explored without finding goal

Output: Either move sequence or "no solution exists"

General Steps for Writing Backtracking Pseudocode

1. Define the recursive function with state parameters
2. Check base cases (goal reached or invalid state)
3. Generate all possible moves from current state
4. For each valid move:
    - Apply the move (make decision)
    - Recursively call function with new state
    - If solution found, return it
    - If not, undo the move (backtrack)
5. Return failure if no moves lead to solution

N-Puzzle Pseudocode

https://docs.google.com/document/d/1ezGm7yWCuZMw2LZbiOF9e-frDTQMFRYaDmoD11xyDvM/edit?usp=sharing

More Problems on Backtracking

- https://www.geeksforgeeks.org/dsa/rat-in-a-maze/
- https://www.geeksforgeeks.org/dsa/sudoku-backtracking-7/
- https://www.geeksforgeeks.org/dsa/tug-of-war/