Recursion

Recursion is a programming technique in which a method calls itself to fulfill its purpose.

A recursive method must provide two parts of definition:
- Base case: that solves the a problem without recursive call
- Recursive case: that solves the problem with one or more recursive calls

Fail to define the base case can cause an infinite recursion!!

Example

N factorial:

\[ n! = n \times (n-1) \times \cdots \times 2 \times 1 = \prod_{i=1}^{n} i \]

- Base case: 1! = 1
- Recursive case: \( k! = k \times (k-1)! \)
Sum of Natural Numbers

Example

Iterative solution:

\[
\text{sum}(n) = n + (n-1) + (n-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i
\]

Recursive solution:

- Base case: \(\text{sum}(1) = 1\)
- Recursive case: \(\text{sum}(n) = n + \text{sum}(n-1)\)

Recursion vs Iteration

- Recursion is natural for many problems, but it incurs high overhead: nested method calls, storage for local variables, difficult to debug.
- Iteration is direct and efficient, but it can be much more complex than recursion.
- Many recursive methods can be converted into iterative methods.
- There exist problems that can only be reasonably solved by recursion.

Tower of Hanoi

- Rules
  - Move one block at a time.
  - No larger block on top of a smaller block.
  - Each block must be on a peg, unless it is in transit between pegs.

\[
\begin{array}{c}
\text{Initial State} \\
\text{Goal State}
\end{array}
\]
Solving Problems Using Recursion

Towers of Hanoi Problem

Recursive Solution

- **Base case:** If there is only one disk to move, move directly.
- **Recursive case:** To move a stack of \( N \) blocks from source peg to destination peg,
  - Move the topmost \( N-1 \) blocks from the source peg to the extra peg.
  - Move the largest block from the source peg to the destination peg.
  - Move the \( N-1 \) blocks from the extra peg to the destination peg.

Traversing a Maze

- A maze can be represented using a rectangular-shaped two-dimensional integer array:
  - 1 represents road and 0 represents no road.
  - Starts at (0, 0) and ends at \((n, m)\), the bottom-right corner.

```
1 1 0 0 0
1 0 0 0 0
1 1 0 0 0
0 1 1 1 1
```

- **Base case:** If already at \((n, m)\), maze is solved.
- **Recursive case:** If at \((i, j)\), for each of the four positions adjacent to \((i, j)\), try to traverse from that position.
  - The four adjacent positions are:
    - down: \((i+1, j)\)
    - right: \((i, j+1)\)
    - up: \((i-1, j)\)
    - left: \((i, j-1)\)
Eight Queens Problem

- Find all possible ways to place 8 queens on an 8 by 8 board so that no queen attacks others.
- A queen attacks any queen that is in the same row, column or diagonal lines.

Solutions to 8 Queens Problem

- Each queen is placed on a row, so for each row, we need to find a column for that queen.
- Recursive thinking for finding one solution:
  - Base case: If this is the 8th queen, place it at the next valid column.
  - Recursive case: If this is for queen i, place it at the next valid column, and solve the problem for the next queen.
- To find all solutions, we need to backtrack from previous solution, and consider subsequent columns in each row.

Check for Position Validity

- Use arrays to flag rows, columns, forward diagonals and backward diagonals that are attacked by existing queens.
- Forward diagonals are numbered by row# - column# + 8 - 1, backward diagonals are numbered by row# + column#.