Introduction to Recursive Sequences and Functions

Understanding Recursion in Mathematics and Programming

November 2024

Table of Contents

CS12Recursive Functions/Screenshot 2024-11-11 150024.png

What is Recursion?

- A process where the definition refers to the thing being defined
- Example: The Sleeping Story
  - A child couldn't sleep, so mother told a story about...
  - a frog who couldn't sleep, so its mother told a story about...
  - a bear who couldn't sleep, so its mother told a story about...
  - a weasel
  - ...who fell asleep.
  - ...and the little bear fell asleep;
  - ...and the little frog fell asleep;
  - ...and the child fell asleep.

Recursive Functions

CS12Recursive Functions/Screenshot 2024-11-11 152930.png

Table of Contents

Recursive Sequences: Basic Form

- A recursive sequence is defined by:
  - Initial term(s)
  - A rule for finding subsequent terms
- General Form for Geometric Sequences:

$$t_n = t_1 r^{n-1}$$

where:

- r is the common ratio
- $r = \frac{t_i}{t_{i-1}}$ for $i > 1$

Types of Recursive Rules

Simple Recursive Rule

$$t_n = \begin{cases} t_1 = 1 \\ t_n = t_{n-1} + 1 \end{cases}$$

Multiple Term Dependencies

$$t_n = \begin{cases} t_1 = t_2 = 1 \\ t_n = t_{n-1} + t_{n-2} \text{ (Fibonacci)} \end{cases}$$

Common Examples

1. Arithmetic: $a_n = a_{n-1} + d$
2. Geometric: $a_n = a_{n-1} \cdot r$
3. Fibonacci: $a_n = a_{n-1} + a_{n-2}$
4. Complex: $a_n = n + a_{n-1} + 6$

Table of Contents

Common Programming Exercises

- Fibonacci Sequence Implementation
- Counting Digits Recursively
- Sum of Digits Using Recursion
- Binary Conversion
- Greatest Common Factor (GCD)
- Lowest Common Multiple (LCM)

Common Mistake 1: Forgetting Base Cases

Incorrect

```
int factorial(int n) {
    return n * factorial(n
        - 1);
}
```

Correct

```
int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial(n
        - 1);
}
```

Common Mistake 2: Infinite Recursion

Incorrect

```
int countDown(int n) {
    cout << n << " ";
    return countDown(n -
        1);
}
```

Correct

```
int countDown(int n) {
    if (n < 0)
        return 0;
    cout << n << " ";
    return countDown(n -
        1);
}
```

Common Mistake 3: Stack Overflow

Incorrect

```
int fibonacci(int n) {
    return fibonacci(n-1)
        + fibonacci(n-2);
}
```

Correct

```
int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n-1)
        + fibonacci(n-2);
}
```

Stack Overflow Explained

Each recursive call adds a new layer to the program's memory stack, and without proper base cases, these layers pile up until the computer runs out of memory space.

Common Mistake 4: Incorrect Recursive Step

Incorrect

```c
int sum(int n) {
    if (n == 0)
        return 0;
    return n + n-1; //
       Wrong!
}
```

Correct

```c
int sum(int n) {
    if (n == 0)
        return 0;
    return n + sum(n-1);
}
```

Common Mistake 5: Edge Cases in Recursion

Key Point

Always check the input's validity before processing!

Missing Edge Case

```
int countDown(int n) {
    cout << n << " ";
    return countDown(n-1);
}
```

With Edge Case

```
int countDown(int n) {
    if (n < 0) return 0;
    cout << n << " ";
    return countDown(n-1);
}
```

Table of Contents

Exercise Types

Write the first 5 terms for sequences with rules like:

1. $a_1 = 4$ and $a_n = n + a_{n-1} + 6$
2. $a_1 = 0$ and $a_n = a_{n-1} - n^2$
3. $a_1 = 2$ and $a_n = (a_{n-1})^2 + 2$

Real-World Application

Example (Swimming Pool Problem)

You add chlorine to a pool:

- First week: 750mL
- Every week after: 350mL
- 40% evaporates each week

Write a recursive rule for the amount of chlorine each week.

Table of Contents

Piece-wise Recursive Rules

Example:

$$a_n = \begin{cases} 7 & \text{if } n = 1 \\ \frac{a_{n-1}}{2} & \text{if } a_{n-1} \text{ is even} \\ 3a_{n-1} + 1 & \text{if } a_{n-1} \text{ is odd} \end{cases}$$

Explicit vs. Recursive Rules

Explicit Rule:

$$t_n = n$$

Recursive Rule:

$$t_n = \begin{cases} t_1 = 1 \\ t_n = t_{n-1} + 1 \end{cases}$$

Conclusion

- Recursion is a powerful mathematical and programming tool
- Key concepts:
  - Base cases
  - Recursive steps
  - Multiple approaches (explicit vs. recursive)
- Practice with both mathematical and programming problems