Recurrencees

Shirley B. Chu

De La Salle University

10 Apr 2023

Recurrence

- **Recurrence** is used to describe the running time of a recursive algorithm.

Recurrence

- **Recurrence** is used to describe the running time of a recursive algorithm.

Iterative, not recursive

```
int factorial (int n)
{
    int result = 1;
    while (n > 0)
    {
        result *= n;
        n--;
    }
    return result;
}
```

Recurrence

- **Recurrence** is used to describe the running time of a recursive algorithm.

Iterative, not recursive

```
int factorial (int n)
{
    int result = 1;
    while (n > 0)
    {
        result *= n;
        n--;
    }
    return result;
}
```

Recursive

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

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

Recurrence

- **Recurrence** is used to describe the running time of a recursive algorithm.

Iterative, not recursive

```
int factorial (int n)
{
    int result = 1;
    while (n > 0)
    {
        result *= n;
        n--;
    }
    return result;
}
```

Recursive

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

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

- A general form of recurrence is an equality or inequality that describes a function over integers or reals using the function itself.

Recurrence

- **Recurrence** is used to describe the running time of a recursive algorithm.

Iterative, not recursive

```
int factorial (int n)
{
    int result = 1;
    while (n > 0)
    {
        result *= n;
        n--;
    }
    return result;
}
```

Recursive

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

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

- A general form of recurrence is an equality or inequality that describes a function over integers or reals using the function itself.
- It contains **more than one** cases, depending on the argument.

Recurrence

- **Recurrence** is used to describe the running time of a recursive algorithm.

Iterative, not recursive

```
int factorial (int n)
{
    int result = 1;
    while (n > 0)
    {
        result *= n;
        n--;
    }
    return result;
}
```

Recursive

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

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

- A general form of recurrence is an equality or inequality that describes a function over integers or reals using the function itself.
- It contains **more than one** cases, depending on the argument.
    - **recursive case**

Recurrence

- **Recurrence** is used to describe the running time of a recursive algorithm.

Iterative, not recursive

```
int factorial (int n)
{
    int result = 1;
    while (n > 0)
    {
        result *= n;
        n--;
    }
    return result;
}
```

Recursive

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

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

- A general form of recurrence is an equality or inequality that describes a function over integers or reals using the function itself.
- It contains **more than one** cases, depending on the argument.
  - **recursive case**
  - **base case**

Example 1: factorial()

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

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

Example 1: `factorial()`

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

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

$T(n) =$

Example 1: factorial()

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

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

$$T(n) = \left\{ \right.$$

Example 1: factorial()

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

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

$$T(n) = \begin{cases} & n == 0 \\ & \end{cases}$$

Example 1: factorial()

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

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

$$T(n) = \begin{cases} a & n == 0 \\ & \end{cases}$$

Example 1: `factorial()`

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

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

$$T(n) = \begin{cases} a & n == 0 \\ & \end{cases}$$

Example 1: factorial()

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

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

$$T(n) = \begin{cases} a & n == 0 \\ & \end{cases}$$

Example 1: factorial()

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

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

$$T(n) = \begin{cases} a & n == 0 \\ & n > 0 \end{cases}$$

Example 1: `factorial()`

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

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

$$T(n) = \begin{cases} a & n == 0 \\ T(n-1) & n > 0 \end{cases}$$

Example 1: factorial()

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

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

$$T(n) = \begin{cases} a & n == 0 \\ T(n-1)+b & n > 0 \end{cases}$$

Example 1: factorial()

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

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

$$T(n) = \begin{cases} a & n == 0 \\ T(n-1)+b & n > 0 \end{cases}$$

Recurrence is algorithmic

A recurrence $T(n)$ is *algorithmic* if, for every sufficiently large threshold constant $n_0 > 0$, the following properties hold:

1. $\forall\, n < n_0, T(n) = \Theta(1)$
2. $\forall\, n \geq n_0$, every path of recursion terminates in a defined base case within a finite number of recursive invocations.

Convention for recurrences: *Whenever a recurrence is stated without an explicit base case, we assume that the recurrence is algorithmic.*

Example 2: displayLine()

```
void displayLine (int n)
{
   if (n > 0)
   {
      print ('=');
      displayLine (n - 1);
   }
}
```

$$\begin{cases} \\ \end{cases}$$

Example 2: `displayLine()`

```
void displayLine (int n)
{
   if (n > 0)
   {
      print ('=');
      displayLine (n - 1);
   }
}
```

$$T(n) = \begin{cases} \end{cases}$$

Example 2: displayLine()

```
void displayLine (int n)
{
   if (n > 0)
   {
      print ('=');
      displayLine (n - 1);
   }
}
```

$T(n) = \begin{cases} \end{cases}$

Example 2: `displayLine()`

```
void displayLine (int n)
{
  if (n > 0)
  {
    print ('=');
    displayLine (n - 1);
  }
}
```

$$T(n) = \begin{cases} & n > 0 \end{cases}$$

Example 2: displayLine()

```
void displayLine (int n)
{
    if (n > 0)
    {
        print ('=');
        displayLine (n - 1);
    }
}
```

$$T(n) = \begin{cases} \\ T(n-1) & n > 0 \end{cases}$$

Example 2: displayLine()

```
void displayLine (int n)
{
    if (n > 0)
    {
        print ('=');
        displayLine (n - 1);
    }
}
```

$$T(n) = \begin{cases} \\ T(n-1)+b & n > 0 \end{cases}$$

Example 2: displayLine()

```
void displayLine (int n)
{
   if (n > 0)
   {
      print ('=');
      displayLine (n - 1);
   }
}
```

$$T(n) = \begin{cases} & n == 0 \\ T(n-1)+b & n > 0 \end{cases}$$

Example 2: displayLine()

```
void displayLine (int n)
{
   if (n > 0)
   {
      print ('=');
      displayLine (n - 1);
   }
}
```

$$T(n) = \begin{cases} a & n == 0 \\ T(n-1) + b & n > 0 \end{cases}$$

Example 2: displayLine()

```
void displayLine (int n)
{
   if (n > 0)
   {
      print ('=');
      displayLine (n - 1);
   }
}
```

$$T(n) = \begin{cases} a & n == 0 \\ T(n-1)+b & n > 0 \end{cases}$$

Exercise 1

Write the recursive function and derive its recurrence relation.

1. `sum (n)` : computes for $\sum_{j=1}^{n} j$
2. `power (x, y)` : computes for x^y
3. `fibonacci (n)` : returns the n^{th} fibonacci number.
   Fibonacci numbers are: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$T(5)$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$T(5) = T(5) \tag{1}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$T(5) = T(4) + 3 \tag{1}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(4) + 3 && (2) \end{aligned}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \end{aligned}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(3) + 3 + 3 && (3) \end{aligned}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \end{aligned}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{align} T(5) &= T(4) + 3 \tag{1} \\ &= T(3) + 3 + 3 \tag{2} \\ &= T(2) + 3 + 3 + 3 \tag{3} \\ &= T(2) + 3 + 3 + 3 \tag{4} \end{align}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{align} T(5) &= T(4) + 3 \tag{1} \\ &= T(3) + 3 + 3 \tag{2} \\ &= T(2) + 3 + 3 + 3 \tag{3} \\ &= T(1) + 3 + 3 + 3 + 3 \tag{4} \end{align}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= T(1) + 3 + 3 + 3 + 3 && (5) \end{aligned}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \end{aligned}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

If $n = 5$, __ $= 4$.

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

If $n = 5$, __ $= 4$.

$$T(5) = 2 + 3(4) \qquad (7)$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

$$T(n) = 2 + 3(n-1) \qquad (7)$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

$$T(n) = 2 + 3(n-1) \qquad (7)$$

Simplify!

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

$$\begin{aligned} T(n) &= 2 + 3(n-1) && (7) \\ &= 2 + 3n - 3 && (8) \end{aligned}$$

Simplify!

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

$$\begin{aligned} T(n) &= 2 + 3(n-1) && (7) \\ &= 2 + 3n - 3 && (8) \\ &= 3n - 1 && (9) \end{aligned}$$

Simplify!

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned}
T(5) &= T(4) + 3 && (1)\\
&= T(3) + 3 + 3 && (2)\\
&= T(2) + 3 + 3 + 3 && (3)\\
&= T(1) + 3 + 3 + 3 + 3 && (4)\\
&= 2 + 3 + 3 + 3 + 3 && (5)\\
T(5) &= 2 + 3(4) && (6)
\end{aligned}$$

$$\begin{aligned}
T(n) &= 2 + 3(n-1) && (7)\\
&= 2 + 3n - 3 && (8)\\
\end{aligned}$$

$$\boxed{T(n) = 3n - 1} \qquad (9)$$

Solving recurrences: Iteration Method

Determine the closed form or explicit formula of $T(n)$.

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + 3 & n > 1 \end{cases}$$

set $n = 5$,

$$\begin{aligned} T(5) &= T(4) + 3 && (1) \\ &= T(3) + 3 + 3 && (2) \\ &= T(2) + 3 + 3 + 3 && (3) \\ &= T(1) + 3 + 3 + 3 + 3 && (4) \\ &= 2 + 3 + 3 + 3 + 3 && (5) \\ T(5) &= 2 + 3(4) && (6) \end{aligned}$$

$$\begin{aligned} T(n) &= 2 + 3(n-1) && (7) \\ &= 2 + 3n - 3 && (8) \end{aligned}$$

$$\boxed{T(n) = 3n - 1 \qquad O(n)} \qquad (9)$$

Exercise 2

Use the iteration method to determine the explicit formula and the *Big-Oh* of the following recurrences defined.

1. $$T(n) = \begin{cases} a & n = 1 \\ T(n-1) + b & n > 1 \end{cases}$$

2. $$T(n) = \begin{cases} 5 & n = 0 \\ T(n-1) + 2n - 1 & n > 0 \end{cases}$$

3. $$T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{2}\right) + 1 & n > 1 \end{cases}$$

4. $$T(n) = \begin{cases} 8 & n = 1 \\ 2T(n-1) + 5 & n > 1 \end{cases}$$

5. $$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + 1 & n > 1 \end{cases}$$

Formulas to **recall** : Summations

$$\sum_{j=a}^{b} 1 = b - a + 1$$

$$\sum_{j=0}^{n} j = \frac{n(n+1)}{2}$$

$$\sum_{j=0}^{n} j^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{j=0}^{n} j^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{j=0}^{n} x^j = \frac{x^{n+1}-1}{x-1}$$

Logarithms

Notations:

$\lg n = \log_2 n$	binary logarithm
$\ln n = \log_e n$	natural logarithm
$\lg^k n = (\lg n)^k$	exponentiation
$\lg \lg n = \lg(\lg n)$	composition
$\log n + 1 = (\log n) + 1$	$\neq \log(n+1)$

For any constant $b > 1$,

$$\log_b n = \begin{cases} \text{undefined} & n \leq 0 \\ \text{strictly increasing} & n > 0 \\ \text{negative} & 0 < n < 1 \\ \text{positive} & n > 1 \\ 0 & n = 1 \end{cases}$$

Formulas to **remember** : Logarithms

For all real $a > 0, b > 0, c > 0, n$, and logarithm bases $\neq 1$

$$a = b^{\log_b a}$$

$$\log_c ab = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b \frac{1}{a} = -\log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

Reference

Antioquia, A. (2020). Recurrences. CCDSALG Lecture.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2022). Introduction to Algorithms, fourth edition. MIT Press.