集训课-18：递归算法

课程目标

01 递归的思想

02 递归函数的概念

03 递归函数的两大要素

04 递归函数的经典应用

05 知识回顾

Part 1

递归的思想

递归的思想

- 递归思想：自己用自己
- 递归的能力在于用有限的语句来定义对象的无限集合
- 例如，在数学上，所有偶数的集合可递归地定义为：

  ① 0 是一个偶数

  ② 一个偶数与 2 的和是一个偶数
- 仅需两句话就能定义一个由无穷多个元素组成的集合

递归的思想

从前有座山，山里有座庙，庙里有个老和尚和小和尚在讲故事，讲的是：“从前有座山，山里山里有座庙，庙里有个老和尚和小和尚在讲故事……”

Part 2

递归函数的概念

递归函数的概念

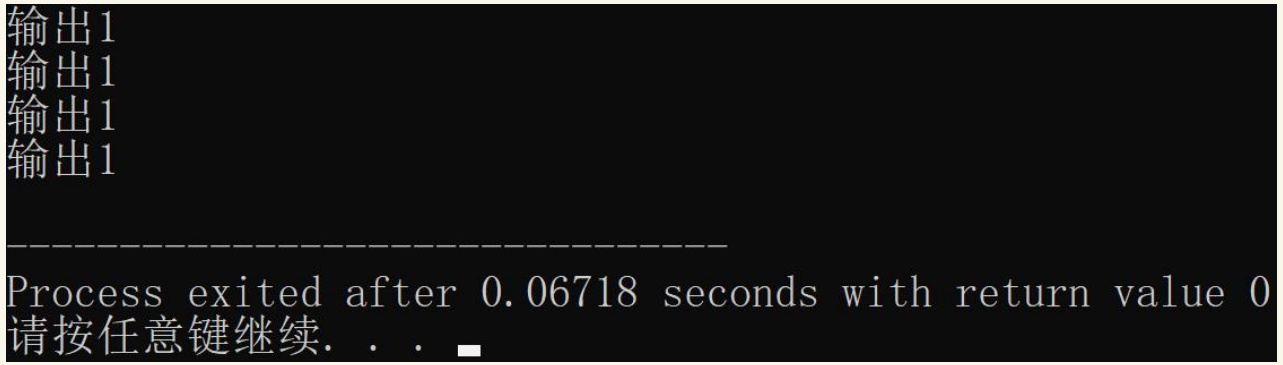
- 自己调用自己
- 函数的定义中，在内部直接或间接地出现对自身的调用
- 在程序中，递归是通过函数的调用来实现的。函数直接调用其自身，称为直接递归；函数间接调用其自身，称为间接递归。

```
A() { // 函数 A 的实现
   A() // 调用函数 A
}
```

```
A() { // 函数 A 的实现
   B() // 调用函数 B
}

B() { // 函数 B 的实现
   A() // 调用函数 A
}
```

递归函数的概念

```
#include<iostream>
using namespace std;
void f(int t){
    if (t == 5){
        return;
    }
    cout << "输出" << 1 << endl ;
    f(t + 1);
}
int main(){
    f(1);
    return 0;
}
```



Part 3

递归函数的两大要素

递归函数的两大要素

- 递归函数的两大要素
  - 递归式或递归调用(如何调用自己)
  - 递归边界(什么时候结束)

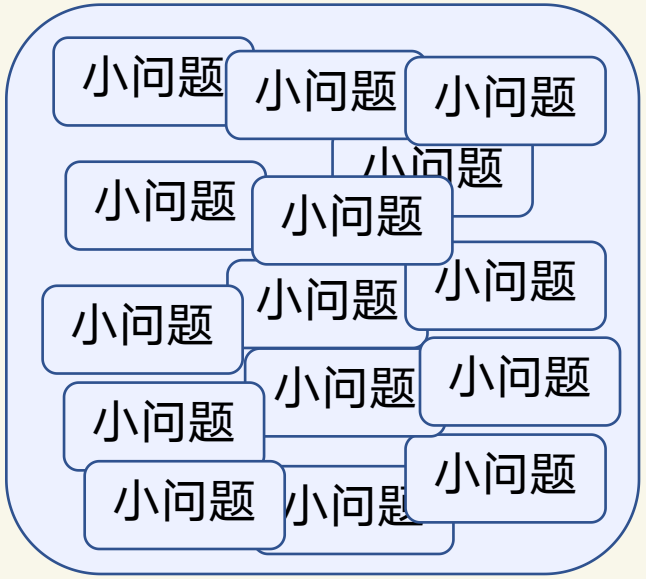
```
//下列代码为一个无返回值的递归函数
void f(){
    if (边界条件){
        return;
    }
    f();
}
```

递归边界

递归式（或称递归调用）

递归函数的两大要素

大问题

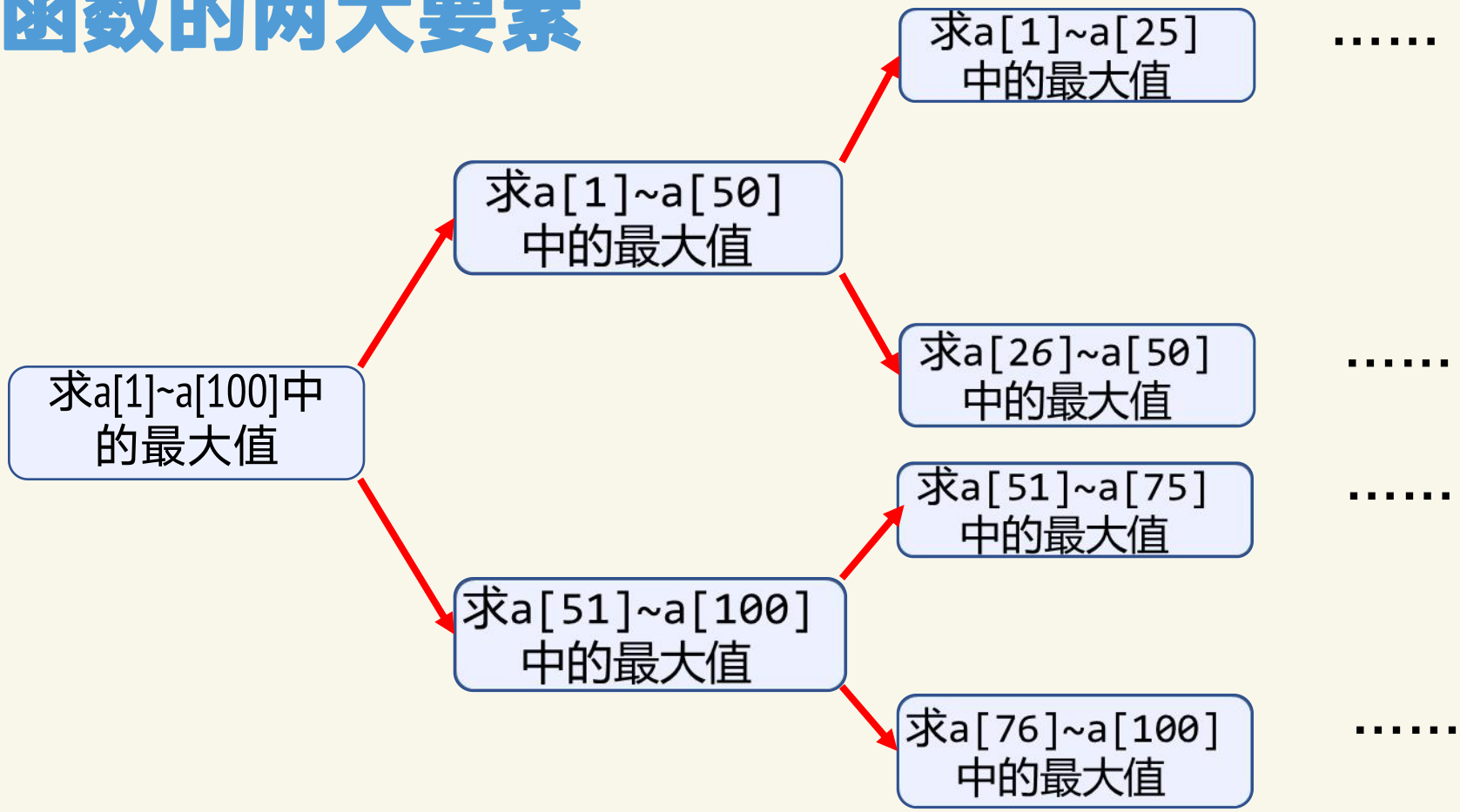


(大型、复杂)
问题

层层转化

(与原问题结构相同/相似、规模较小)
问题

递归函数的两大要素



把a[1]~a[100]这个大区间转化成很多个小区间，小区间的最大值解决了，那么大区间的最大值也就解决了。

Part 4

递归函数的经典应用

递归函数的经典应用

输入 n，求 $1\sim n$ 的整数和。

注意：请用递归完成

(大型、复杂)问题 → **层层转化** → (与原问题结构相同/相似、规模较小)问题

$(1\sim n$ 的和$) = n + (1\sim(n-1)$ 的和$)$

$(1\sim(n-1)$ 的和$) = (n-1) + (1\sim(n-2)$ 的和$)$

……

$(1\sim 2$ 的和$) = 2 + (1\sim(2-1)$ 的和$)$

$(1\sim 1$ 的和$) = 1$

代码如何执行？

$sum(n)$表示 $1\sim n$ 的和

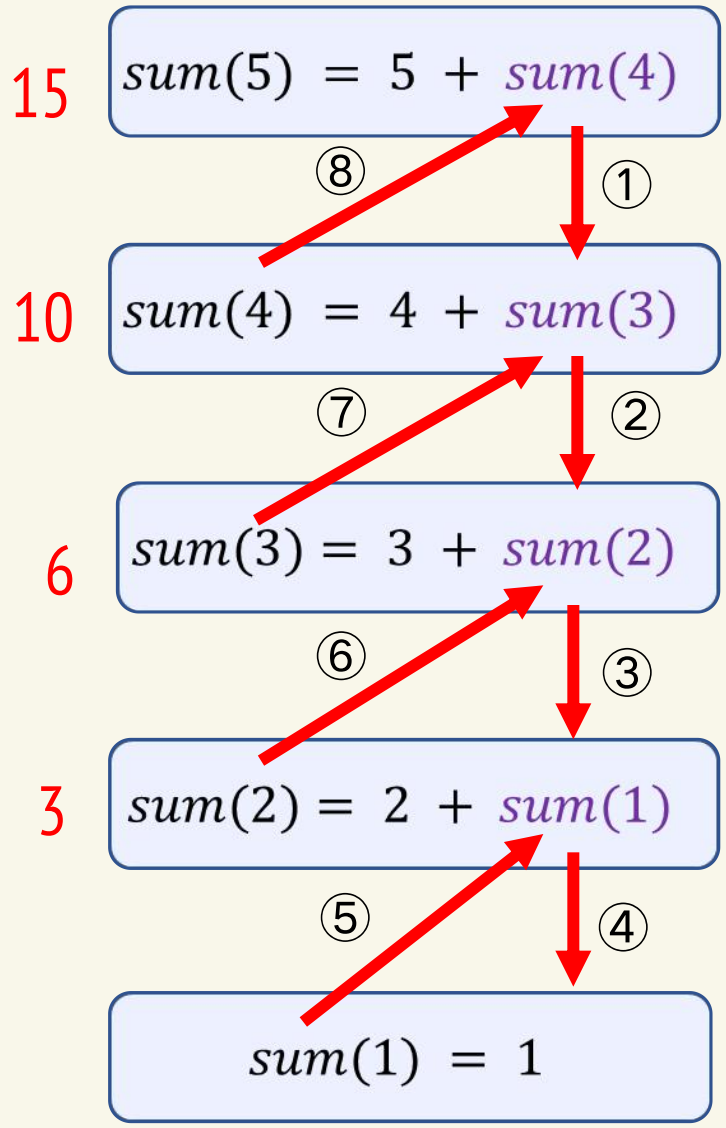
递归调用：$sum(n) = n + sum(n-1)$

递归边界：$n = 1$ 时，$sum(n) = 1$

```cpp
#include <iostream>
using namespace std;

int sum(int n) {
    if(n == 1) return 1; // 递归边界
    return n + sum(n-1); // 递归调用
}

int main() {

    int n;
    cin >> n;
    int ans = sum(n); // 调用sum函数
    cout << ans;

    return 0;
}
```

递归函数的经典应用

递归过程示意图 $(n = 5)$



(大型、复杂)问题 **层层转化** (与原问题结构相同/相似、规模较小)问题

```cpp
#include <iostream>
using namespace std;

int sum(int n) {
    if(n == 1) return 1; // 递归边界
    return n + sum(n-1); // 递归调用
}

int main() {

    int n;
    cin >> n;
    int ans = sum(n); // 调用sum函数
    cout << ans;

    return 0;
}
```

递归函数的经典应用

斐波那契数列（$Fibonacci\ sequence$），又称黄金分割数列，因数学家莱昂纳多·斐波那契（$Leonardo\ Fibonacci$）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、13、21、34、……

输入 n，求斐波那契数列的第 n 项。注意：请用递归完成

第 n 项 = 第 $n-1$ 项 + 第 $n-2$ 项

第 $n-1$ 项 = 第 $n-2$ 项 + 第 $n-3$ 项

……

第 3 项 = 第 $(3-1)$ 项 + 第 $(3-2)$ 项

第 2 项 = 1

第 1 项 = 1

(大型、复杂)问题 **层层转化** (与原问题结构相同/相似、规模较小)问题

- $fib(n)$表示斐波那契数列的第 n 项

递归调用：$fib(n) = fib(n-1) + fib(n-2)$

递归边界：$n = 1$ 时，$fib(n) = 1$

$n = 2$ 时，$fib(n) = 1$

递归函数的经典应用

第 n 项 = 第 $n-1$ 项 + 第 $n-2$ 项

第 $n-1$ 项 = 第 $n-2$ 项 + 第 $n-3$ 项

……

第 3 项 = 第 $(3-1)$ 项 + 第 $(3-2)$ 项

第 2 项 = 1

第 1 项 = 1

递归调用：$fib(n) = fib(n-1) + fib(n-2)$

递归边界：$n = 1$ 时，$fib(n) = 1$

$n = 2$ 时，$fib(n) = 1$

代码如何执行？

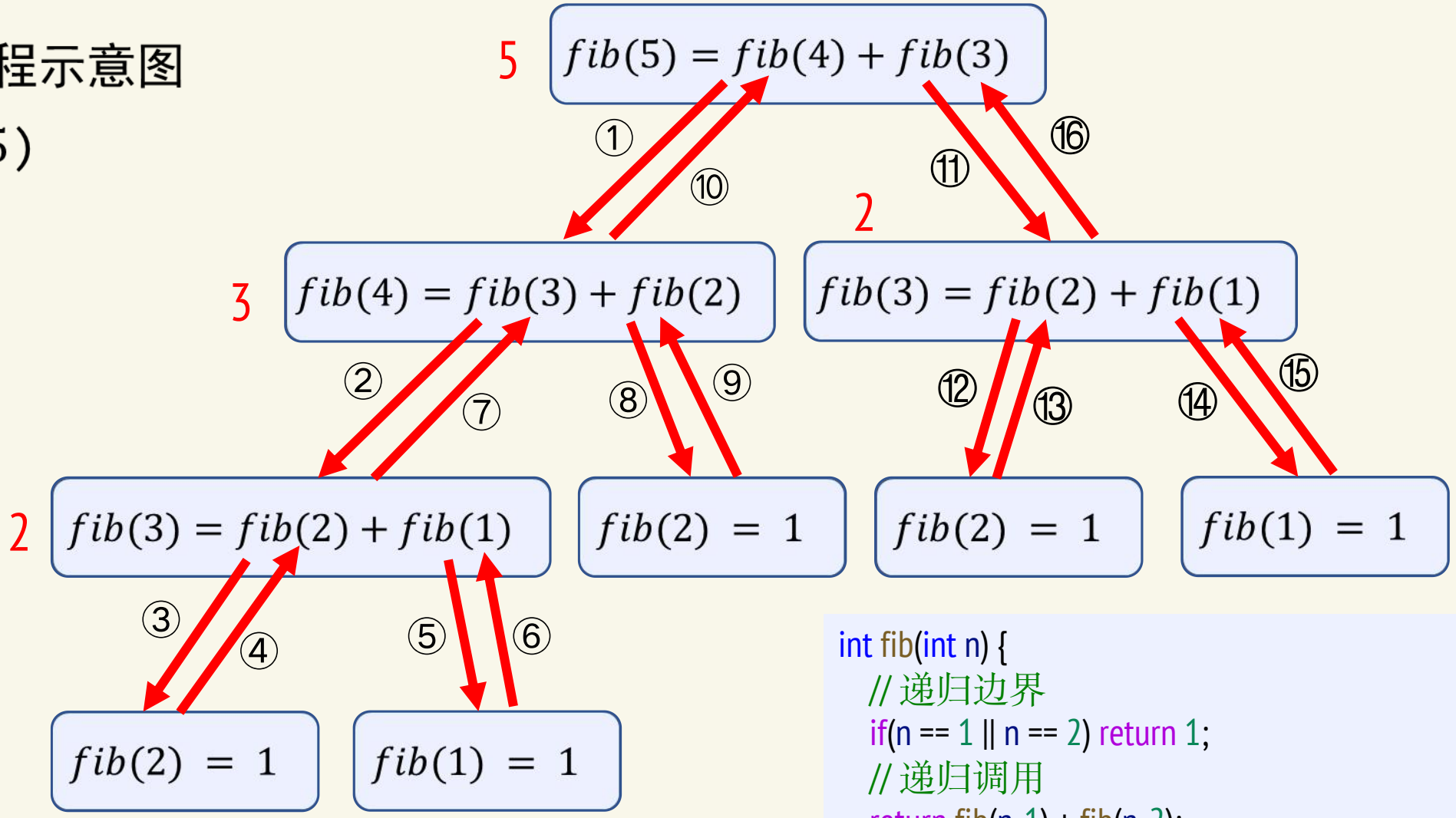
(大型、复杂)问题 **层层转化** (与原问题结构相同/相似、规模较小)问题

```
#include<iostream>
using namespace std;

int fib(int n) {
    // 递归边界
    if(n == 1 || n == 2) return 1;
    // 递归调用
    return fib(n-1) + fib(n-2);
}

int main() {
    int n;
    cin >> n;
    int ans = fib(n); // 调用fib函数
    cout << ans;
    return 0;
}
```

递归函数的经典应用

递归过程示意图

$(n = 5)$




```
int fib(int n) {
    // 递归边界
    if(n == 1 || n == 2) return 1;
    // 递归调用
    return fib(n-1) + fib(n-2);
}
```

递归函数的经典应用

给出一个正整数 n，你需要将 n 进行倒序输出。$n(1 \leq n \leq 10^{18})$

注意：请使用递归完成。

(大型、复杂)问题 **层层转化** (与原问题结构相同/相似、规模较小)问题

倒序输出 n 的每一位 = 输出 n 的个位 和 倒序输出 $n/10$ 的每一位

倒序输出 $n/10$ 的每一位 = 输出 $n/10$ 的个位 和 输出 $n/10/10$ 的每一位

……

- $dfs(n)$ 表示倒序输出 n 的每一位

递归调用：$dfs(n/10)$

递归边界：$n = 0$ 时，返回

递归函数的经典应用

倒序输出 n 的每一位 = 输出 n 的个位 和 倒序输出 $n/10$ 的每一位

倒序输出 $n/10$ 的每一位 = 输出 $n/10$ 的个位 和 输出 $n/10/10$ 的每一位

……

- $dfs(n)$ 表示倒序输出 n 的每一位

递归调用：$dfs(n/10)$

递归边界：$n = 0$ 时，返回

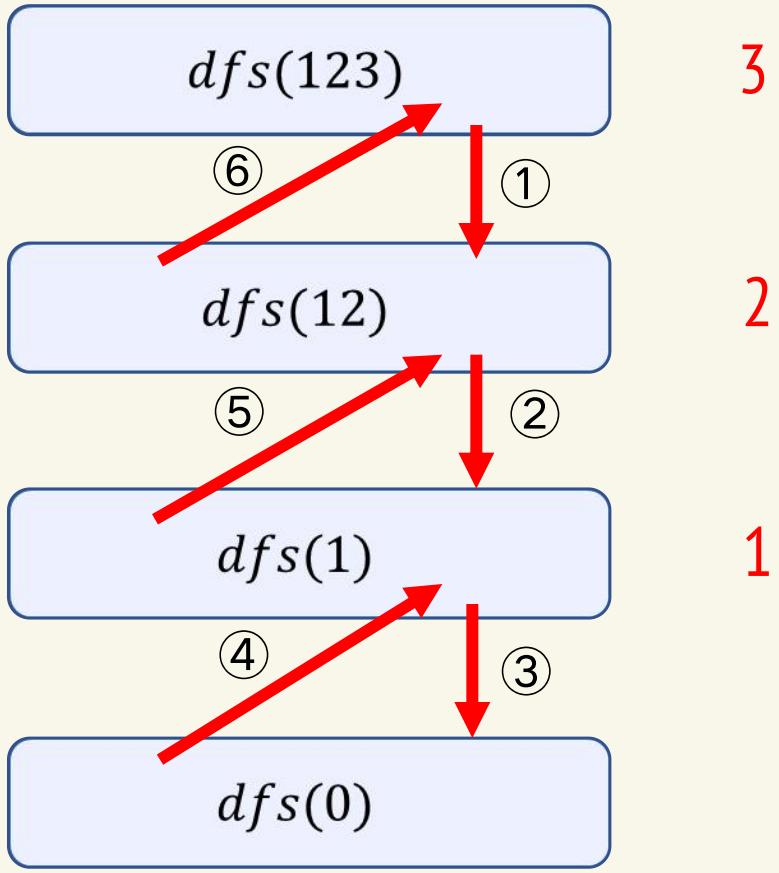
代码如何执行？

```
#include <iostream>
using namespace std;

void dfs(long long x) {
    if (x == 0) {
        return;
    }
    cout << x % 10;
    dfs(x / 10);
}
int main() {
    long long n;
    cin >> n;
    dfs(n);
    return 0;
}
```

递归函数的经典应用

递归过程示意图

$(n = 123)$



输出：

321

```
#include <iostream>
using namespace std;

void dfs(long long x) {
    if (x == 0) {
        return;
    }
    cout << x % 10;
    dfs(x / 10);
}
int main() {
    long long n;
    cin >> n;
    dfs(n);
    return 0;
}
```

递归函数的经典应用

输入一个十进制整数 n，转成对应的二进制数。

注意：请用递归完成

+表示拼接

n 的二进制 = $n / 2$ 的二进制 + $n \% 2$

$(n / 2)$的二进制 = $(n / 2) / 2$的二进制+$(n / 2) \% 2$

……

+表示拼接

(大型、复杂)问题 **层层转化** (与原问题结构相同/相似、规模较小)问题

- $solve(n)$ 表示将 n 转成二进制

递归调用：$solve(n/2)$

递归边界：$n = 0$ 时，返回

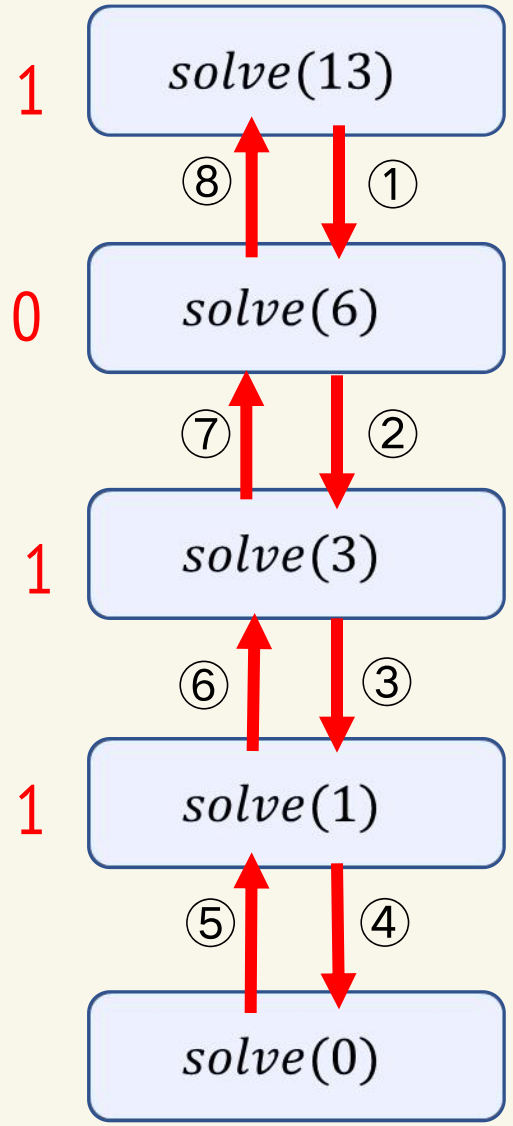
代码如何执行？

```
#include <iostream>
using namespace std;
void solve(int n) {
    // 递归边界
    if(n == 0) return; //商为0时停止
    solve(n / 2);  // 递归调用
    cout << n % 2; // 输出余数
}
int main(){
    int n;
    cin >> n;
    solve(n);  // 调用solve函数
    return 0;
}
```

递归函数的经典应用

递归过程示意图

$(n = 13)$



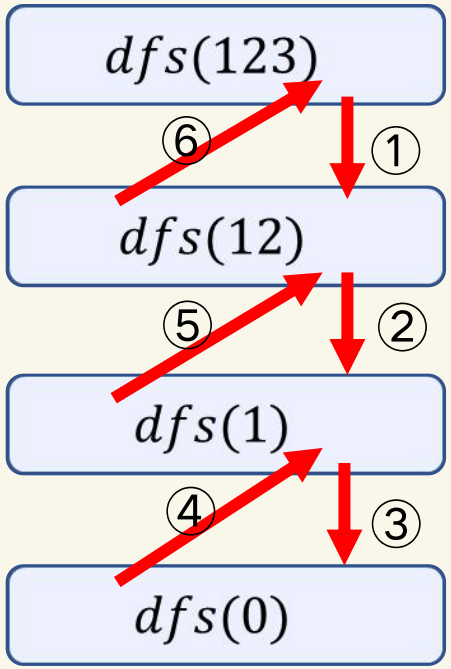
输出：

1101

(大型、复杂)问题 **层层转化** (与原问题结构相同/相似、规模较小)问题

```
#include <iostream>
using namespace std;
void solve(int n) {
    // 递归边界
    if(n == 0) return;  //商为0时停止
    solve(n / 2);  // 递归调用
    cout << n % 2; // 输出余数
}
int main(){
    int n;
    cin >> n;
    solve(n);  // 调用solve函数
    return 0;
}
```

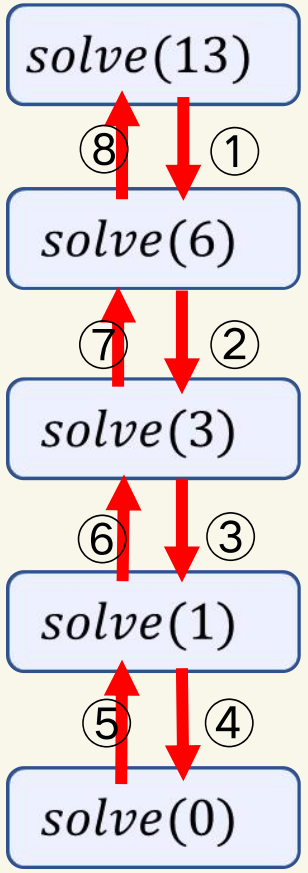
递归函数的经典应用




```
void dfs(long long x) {
    if (x == 0) {
        return;
    }
    cout << x % 10;
    dfs(x / 10);
}
```

输出：

321




```
void solve(int n) {
    // 递归边界
    if(n == 0) return;
    solve(n / 2);
    cout << n % 2;
}
```

输出：

1101

输出写在递归调用之前，结果按照调用顺序正序输出

输出写在递归调用之后，结果按照调用顺序逆序输出

Part 5

总结回顾

总结回顾

- 递归函数：自己调用自己
- 函数的定义中，其内部操作直接或间接地出现对自身的调用
- 递归两大要素：
  - 递归式或递归调用(如何调用自己)
  - 递归边界(什么时候结束)

(大型、复杂)问题 **层层转化** → (与原问题结构相同/相似、规模较小)问题

Class is over