计算机导论

第八章 算法

本章内容

- 递归函数
- 分治算法
- 动态规划

递归函数

- 如果一个函数在内部调用自身本身，这个函数就是递归函数。
- 例子：计算阶乘，用 fact(n) 表示求 n!

```
fact(n) = n! = 1 x 2 x 3 x ... x (n-1) x n
             = (n-1)! x n
             = fact(n-1) x n
```

```
def fact(n):
    if n == 1:
        return 1
    return n * fact(n - 1)
```

递归调用结束的过程

递归调用继续的过程

递归函数调用过程

- 计算 fact(5)，可以根据函数定义看到计算过程如下：

```
===> fact(5)
===> 5 * fact(4)
===> 5 * (4 * fact(3))
===> 5 * (4 * (3 * fact(2)))
===> 5 * (4 * (3 * (2 * fact(1))))
===> 5 * (4 * (3 * (2 * 1)))
===> 5 * (4 * (3 * 2))
===> 5 * (4 * 6)
===> 5 * 24
===> 120
```

递归函数原理

- 递归原理

　　问题的求解可通过降低问题规模实现，而小规模的问题求解方式与原问题的一样，小规模问题的解决导致问题的最终解决。

- 递归调用应该能够在有限次数内终止递归
  - 递归调用如果不加以限制，将无数次的循环调用
  - 必须在函数内部加控制语句，只有当满足一定条件时，递归终止

递归练习

- 斐波那契数列

　　指这样一个数列：1、1、2、3、5、8、13、...，编写函数 fib(n) 求第n项的斐波那契数，如 fib(4) = 3。

- 角谷猜想

　　输入一个自然数，若为偶数，则把它除以2，若为奇数，则把它乘以3加1。经过有限次运算后，总可以得到自然数值1。

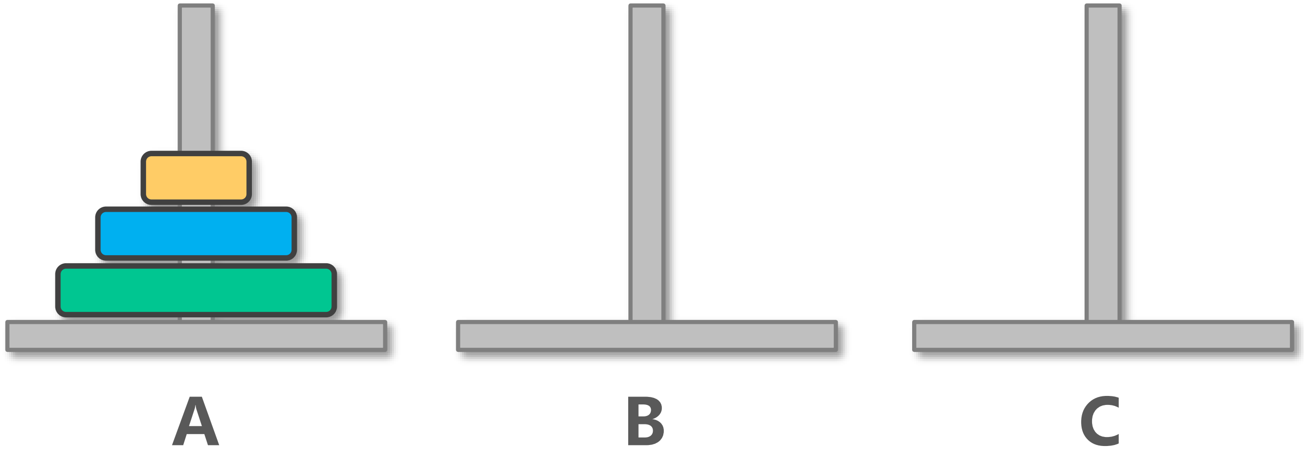
　　编写函数 jiaogu(n) 求n经过转换的次数，如：输入10，需经过 10->5->16->8->4->2->1，共6次转换。

递归练习

- 汉诺塔

编写函数 move(n, a, b, c)，求解汉诺塔移动步骤，其中a、b、c表示三根柱子的名称，n表示a柱子上圆盘数量。

如：move(3, "A", "B", "C")，应输出：




```
A ---> C
A ---> B
C ---> B
A ---> C
B ---> A
B ---> C
A ---> C
```

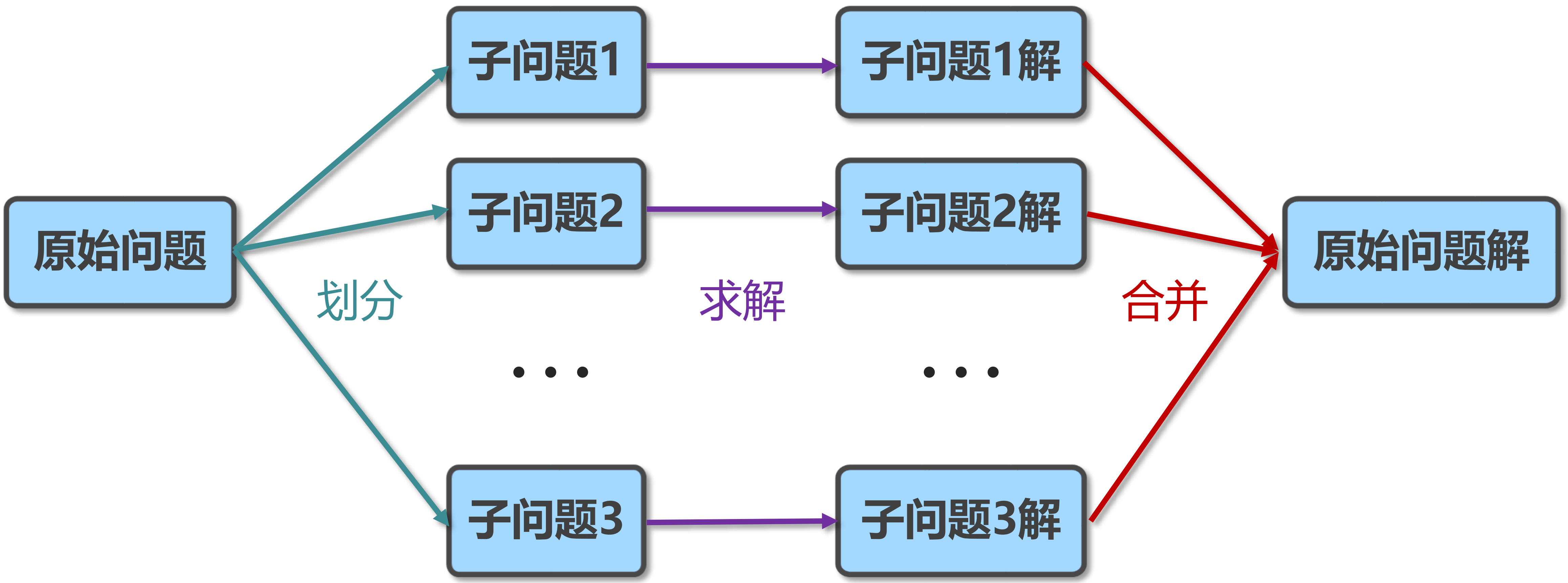
本章内容

- 递归函数
- 分治算法
- 动态规划

分治算法

- 在计算机科学中，分治算法是一种很重要的算法。字面上的解释是 "分而治之"，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题，直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。

分治算法



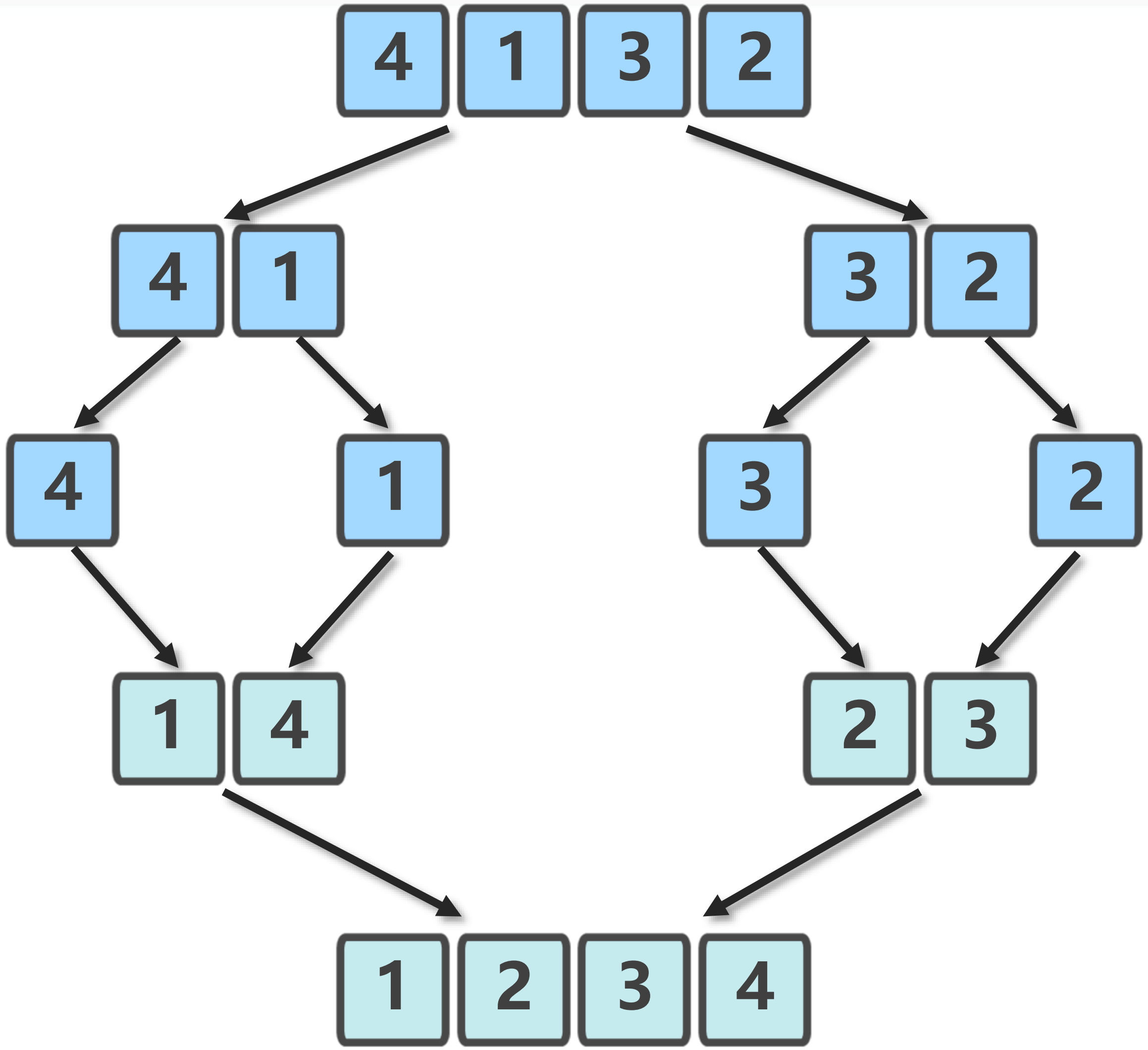
分治算法所解决问题的特征

- 分治算法所能解决的问题一般具有以下几个特征：
  1. 该问题的规模缩小到一定的程度就可以容易地解决；
  2. 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；
  3. 利用该问题分解出的子问题的解可以合并为该问题的解；
  4. 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

归并排序

- 归并排序（MERGE-SORT）是建立在归并操作上的一种有效的排序算法，该算法是采用分治算法的一个非常典型的应用。
- 将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使合并后的序列有序。若将两个有序表合并成一个有序表，称为二路归并。

归并排序示意图



合并有序序列

A: 2 4 7 8

C: 1 2 3 4 5 6 7 8

B: 1 3 5 6

归并排序实现

```python
def merge(a, b):
    c = []
    i, j = 0, 0
    while i < len(a) and j < len(b):
        if a[i] <= b[j]:
            c.append(a[i])
            i += 1
        else:
            c.append(b[j])
            j += 1
    c.extend(a[i:])
    c.extend(b[j:])
    return c
```

- 归并排序实现

```python
def merge_sort(lists):
    if len(lists) == 1:
        return lists
    middle = len(lists) // 2
    left = lists[:middle]
    right = lists[middle:]
    return merge(left, right)
```

- 全排列
  - 排列是从n个元素中任取m个元素，并按照一定的顺序进行排列，称为排列；而当n==m时，称为全排列。
  - 编写函数 perm( ) 打印数列的全排列，并统计全排列数量。
  - 比如：集合 {1,2,3} 的全排列为：

```
{ 1 2 3 }
{ 1 3 2 }
{ 2 1 3 }
{ 2 3 1 }
{ 3 2 1 }
{ 3 1 2 }
```

- 递归函数
- 分治算法
- 动态规划

数字三角形

在数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。路径上的每一步都只能往左下或右下走。只需要求出这个最大和即可，不必给出具体路径。

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

数字三角形

- 解题思路：

  用矩阵 nums 存放数字三角形。

  nums[i, j]：表示矩阵中相应数字

  max_sum(i, j)：从 nums[i, j] 到底边的各条路径中，最佳路径的数字之和。

  问题：求 max_sum(0, 0)

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

问题：求 max_sum(0, 0) —— 典型的递归问题。

nums[i, j] 出发，下一步只能走 nums[i+1, j] 或者 nums[i+1, j+1]。故对于 n 行的三角形：

```
if(i == n-1)
    max_sum(i,j) = nums[i,j]
else
    max_sum(i,j) = max(max_sum(i+1,j),
                       max_sum(i+1,j+1))
                   + nums[i,j]
```

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

递归存在问题

- 存在问题

  如果采用递规的方法，深度遍历每条路径，存在**大量重复计算**。则时间复杂度为 O(2^n)。

71

31 81

81 12 01

21 73 43 41

41 54 26 64 51

- 改进

　　如果每算出一个 max_sum(i, j) 就保存起来，下次用到其值的时候直接取用，则可免去重复计算。那么可以用$O(n^2)$时间完成计算。因为三角形的数字总数是 $n(n+1)/2$。

动态规划与分治相似，都是通过组合子问题的解来求解原问题，与之区别在于动态规划算法应用于子问题重叠的情况，即不同的子问题拥有共同的子子问题。这种情况分治算法会重复的求解子问题，而动态规划算法对每个子问题只求解一次，将其解保存在一个表格中，从而无需重复求解。

动态规划通常用来求解最优化问题，这类问题可以有很多可行解，而希望寻找的只有最优解（最大值或最小值）。

递归到动规的一般转化方法

递归函数有n个索引参数，就定义一个n维的数组，数组的下标是递归函数参数的取值范围，数组元素的值是递归函数的返回值，这样就可以从边界值开始，逐步填充数组。

数据数组

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

sum_arr数组

	0	1	2	3	4
0	30				
1	23	21			
2	20	13	10		
3	7	12	10	10	
4	4	5	2	6	5

```python
def max_sum(i, j):
    global nums
    global sum_arr
    if not sum_arr[i][j] is None:
        return sum_arr[i][j]
    if i == len(nums) - 1:
        sum_arr[i][j] = nums[i][j]
    else:
        sum_arr[i][j] = max(max_sum(i + 1, j), max_sum(i + 1, j + 1)) + nums[i][j]
    return sum_arr[i][j]

nums = [[7], [3, 8], [8, 1, 0], [2, 7, 4, 4], [4, 5, 2, 6, 5]]
sum_arr = [None] * len(nums)
for i in range(len(nums)):
    sum_arr[i] = [None] * len(nums)
print(max_sum(0, 0))
```

递归函数有n个参数，就定值开始，逐步填充数组。

				30				
			23		21			
		20		13		10		
	7		12		10		10	
4		5		2		6		5

- 01背包问题
  - 有n 个物品，它们有各自的重量和价值，现有给定容量的背包，如何让背包里装入的物品具有最大的价值总和?
  - 如下表，W表示物品重量列表，V表示物品价值列表，在背包容量为10时，背包最大价值为15。

	0	1	2	3	4
W	2	2	6	4	5
V	6	3	5	4	6

Questions?