数据结构与算法分析

华中科技大学软件学院

2014年秋

大纲

关于本课程

- 课程内容与目标

关于本课程

- 课程内容与目标
- 教学方式

关于本课程

- 课程内容与目标
- 教学方式
- 考试与作业

关于本课程

- 课程内容与目标
- 教学方式
- 考试与作业
- 其他事项

课程内容

- 数据结构： 组织和访问大量数据的系统方法
- 算法： 在有限时间内执行某个任务的分步过程
- 程序=数据+算法
- 教材： Data Structures and Algorithms Analysis in C, by Mark Allen Weiss, China Machine Press
- 教师： 沈刚， 邮箱： hust_shen@126.com， 办公室： Room 211， 软件学院

课程目标

- 课程结束时，希望能够掌握基本的数据结构并理解与算法的关系
- 能够判别哪些数据结构和算法更好地解决给定的问题
- 熟练地阅读代码
- 成为熟练的C语言程序员

教学方式

- 老师：导游，学生：游客。导游可以引导，但是能否欣赏到美景，在于游客是否有一双善于发现的眼睛和一颗开放乐观的心
- 老师：教练，学生：队员 。教练可以部署战术，但是能否取得胜利在于平素的训练质量和临场高昂的斗志

考试与作业

- 考试：期末闭卷考试，150分钟，10道题
- 作业：5次上机实验，按时提交实验报告
- 成绩＝考试成绩×60%+作业成绩×30%+考勤×10%，可能会根据实际情况调整权重

其它事项

- 安全问题
- 意见与反馈

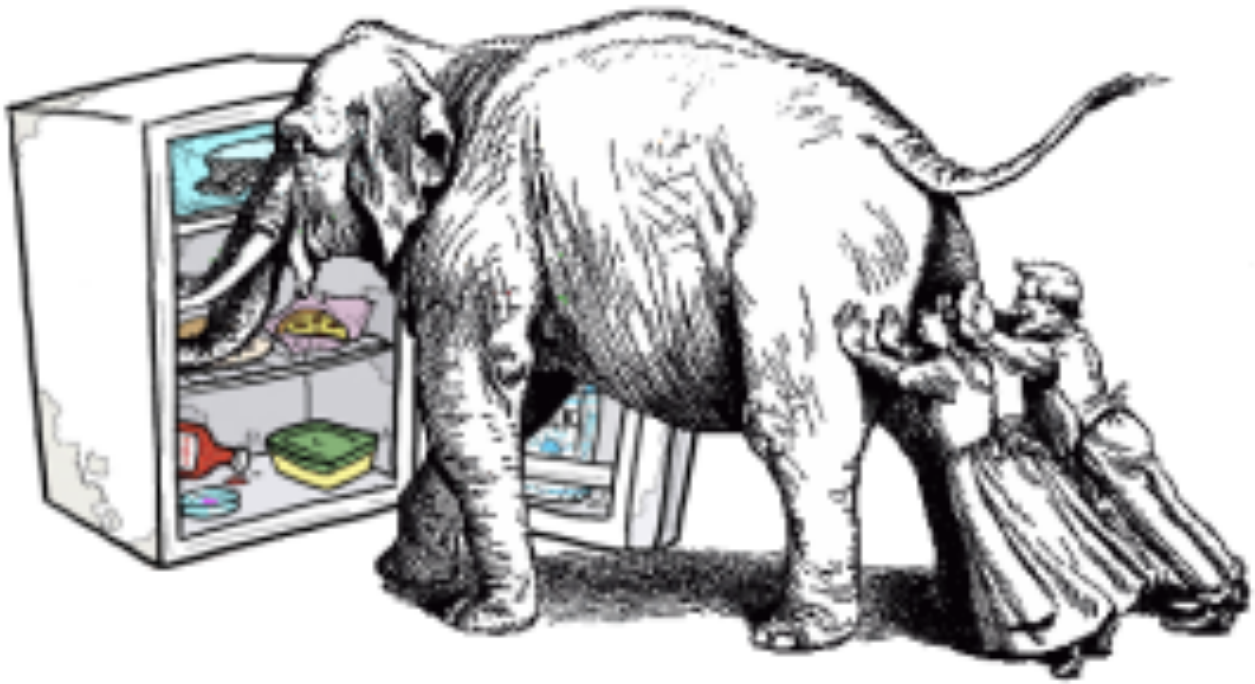
Roadmap

第一个C语言程序

```c
#include <stdio.h>

int main (void)
{
    printf ("Hello, World!\n");
    return(0);
}
```

如何把大象关进冰箱



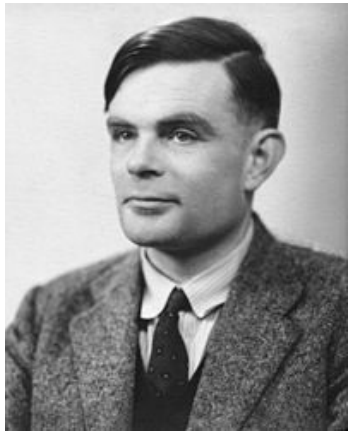
如何把大象关进冰箱

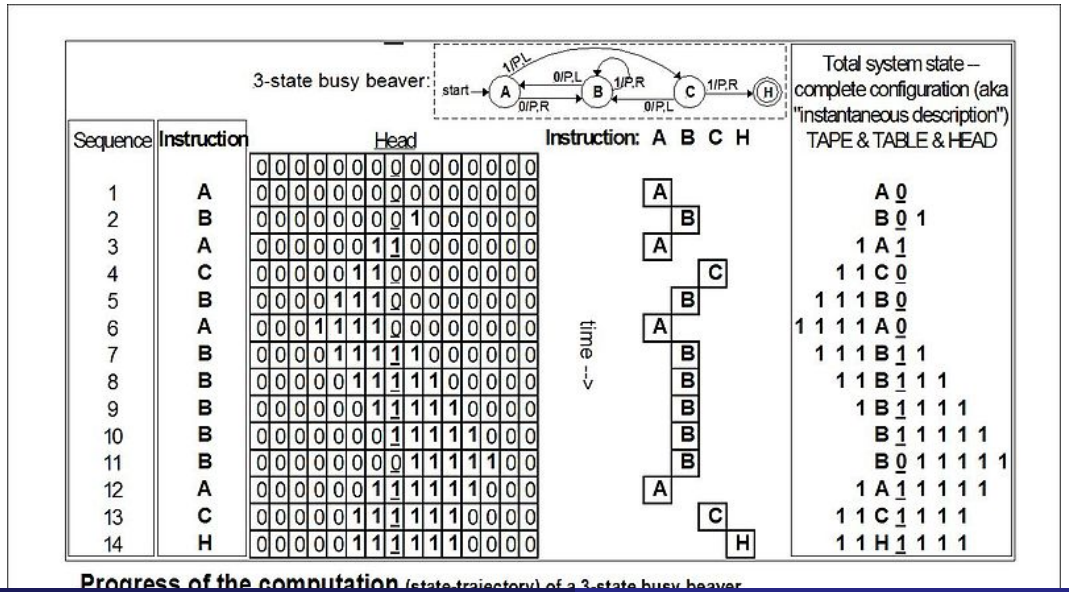
- 如何把大象关进冰箱
  - 打开冰箱门
  - 插入大象
  - 关上冰箱门
- 如何把长颈鹿关入冰箱
  - 打开冰箱门
  - 移走大象
  - 插入长颈鹿
  - 关上冰箱门

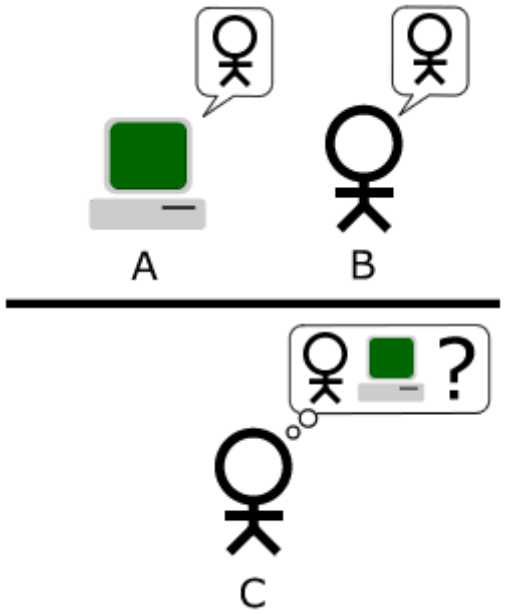
算法

- 花剌子密（al-Khwārizmī）：用阿拉伯数字进行算数运算的规则，求解线性和二次方程的方法
- 在计算机领域，指的是一个分布完成运算的过程
- 算法可以有效地表示为一个指令列表，通过有限步的运算把输入转化为输出
- 相关工具包括
  - Gódel-Herbrand-Kleene递归函数
  - Alonzo Churchλ−演算
  - Alan Turing图灵机

图灵和图灵机



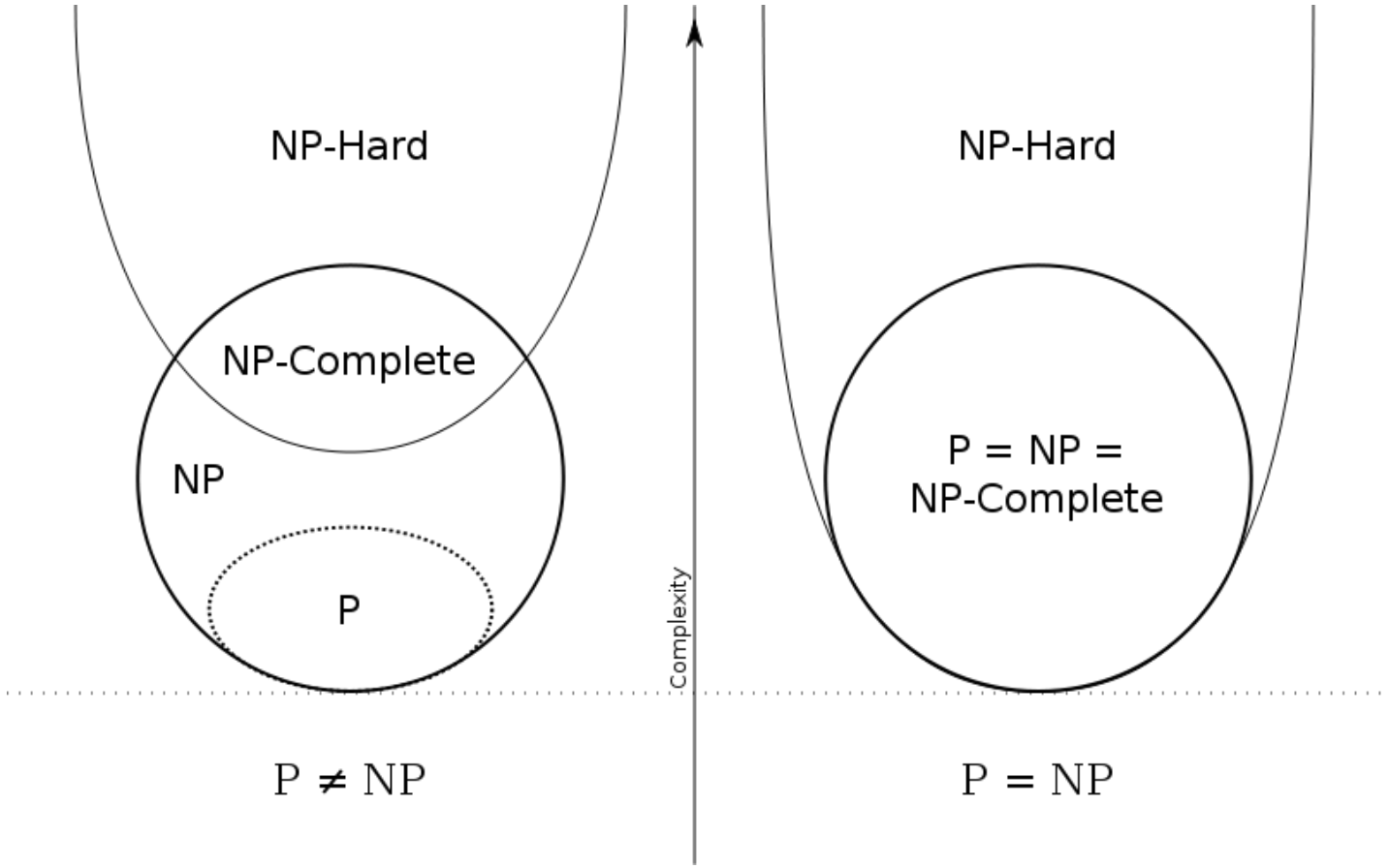

3-state busy beaver:
start
A
B
C
H
1/P,L
0/P,L
1/P,R
0/P,R
Total system state – complete configuration (aka "instantaneous description") TAPE & TABLE & HEAD
Sequence
Instruction
Head
Instruction: A B C H
time -->
Progress of the computation (state-trajectory) of a 3-state busy beaver


A
B
C
?

可计算问题

- Church-Turing论题：所有有效计算的问题可以归结为图灵机可计算问题
- 定义了什么是可机械计算问题
- 递归函数、λ—演算和图灵机等价

计算问题的复杂性


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

福尔摩斯基本演绎法S2E2

P和NP

- 计算机科学中尚未解决问题之一：一个问题的解如果可以被计算机快速验证，那么这个问题是否可以被计算机快速求解？例如加密和解密
- 1971年Stephen Cook引入这一问题，是克雷数学研究所选出的世纪大奖问题之一，奖金一百万美元
- P：一类问题可以有算法在多项式时间求解。NP：没有已知算法在多项式时间求解，但是可以用多项式时间验证一个答案是否其解
- 如果$P \neq NP$，则存在NP问题（NPC），它们的求解比验证要困难。例如，旅行销售员问题就是NPC问题

数据结构与算法

- 算法：程序背后的想法，与具体编程语言无关，由一系列解决问题的步骤构成
  - 有限
  - 意义明确
  - 每一步都可处理
- 算法是一种从输入到输出的映射，输出可以是
  - 数值
  - 决策结果，如质数还是合数
  - 或者一个数据结构，如排序的数组

处理复杂性的方法

- 模块化：分而治之，把问题分解为易于处理的部分
- 封装：把相似的东西放在一起，把可预期的变化局部化
- 抽象
  - 功能上，一般化，参数化
  - 实施上，使用简单的接口，隐藏细节

设计算法的目标

- 如何评价算法？要看是否满足对算法的需求
- 理想情况下，算法应该
  - 正确
  - 有效
  - 最优化
- 有时必须在几个因素间妥协
  - 显而易见的算法可能运行缓慢
  - 快速执行的算法可能难以理解

思考下面几个问题

- 整数求和：从1到n之间的整数相加，和是多少？用C语言实现函数，输入n，返回和

思考下面几个问题

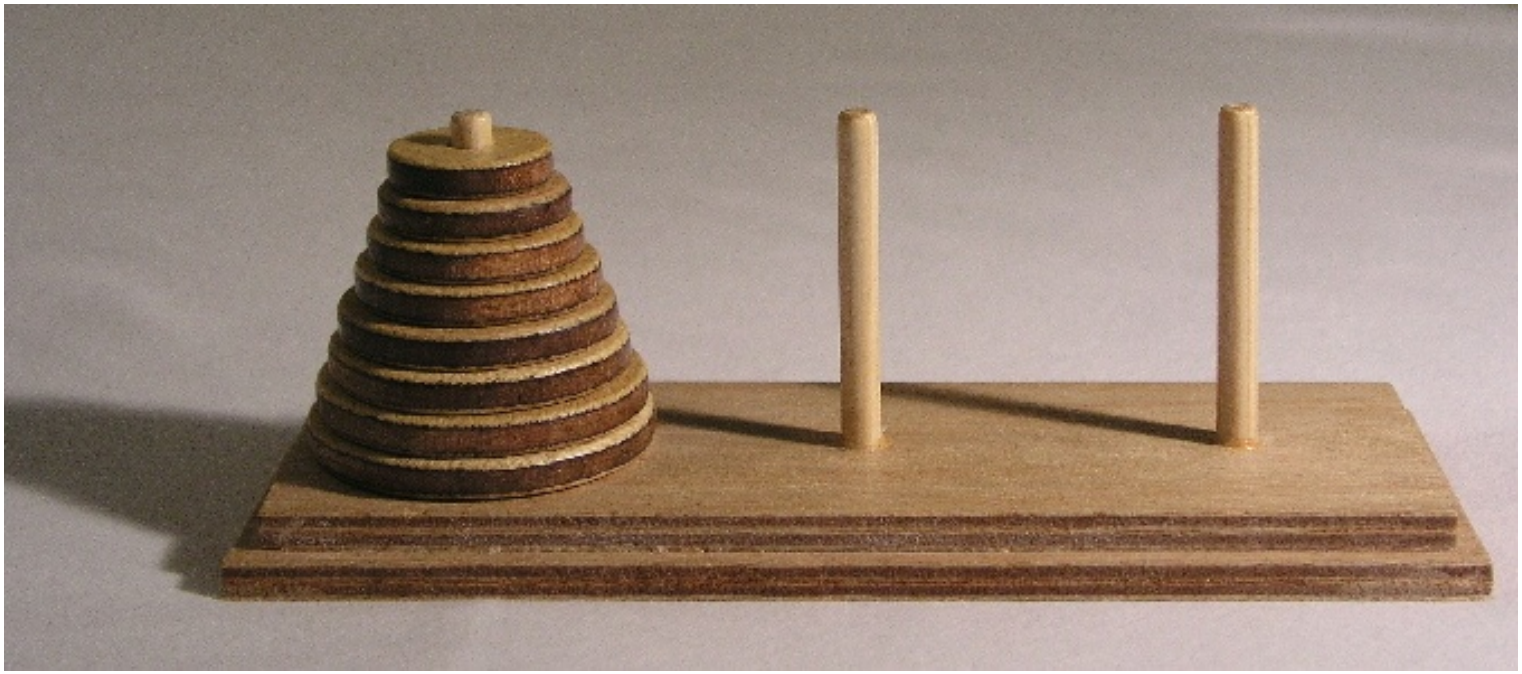
- 整数求和：从1到n之间的整数相加，和是多少？用C语言实现函数，输入n，返回和
- 切披萨：把一张披萨饼切n刀，最多能得到多少块饼？用C语言实现函数，输入参数为n，返回结果

思考下面几个问题

- 整数求和：从1到n之间的整数相加，和是多少？用C语言实现函数，输入n，返回和
- 切披萨：把一张披萨饼切n刀，最多能得到多少块饼？用C语言实现函数，输入参数为n，返回结果
- 河内塔问题：移动n层的塔，总的移动次数是多少？

河内塔

1883年，法国数学家Edouard Lucas提出：一次只能移动一层，大的不能放在小的上面。可以使用临时场所暂存中间结果



河内国旗塔

算法、数学归纳法与递归

- 递归函数：调用自身的函数
- 分而治之：如果复杂问题可以表示为一系列较为简单的问题，可以用相同的算法先求解这些小问题，继而得到原来问题的求解
- 数学归纳法：基础，假设，演绎
- 如何避免无限递归？基础，平凡解：无需递归即可得知的情况
  - 求和：$\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n$，基础：$S(1) = 1$
  - 切披萨：$P(n) = P(n-1) + n$，基础：$P(1) = 2$
  - 河内塔：$M(n) = 2 \times M(n-1) + 1$，基础：$M(1) = 1$

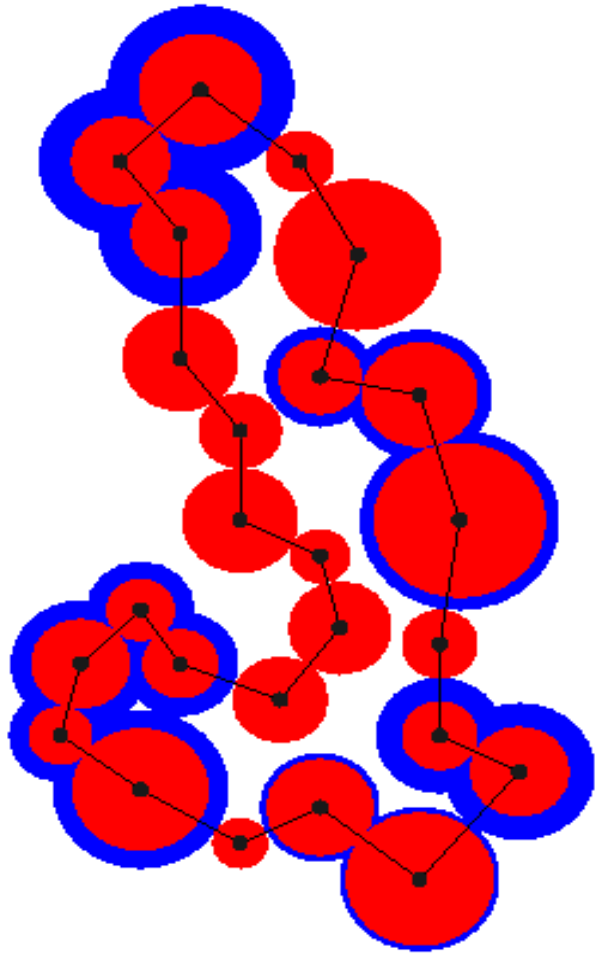
河内塔的代码

```
void HanoiTower (int n, char source, char temp,
                 char target)
{
    if (n < 1)
        return;
    if (1 == n)
    {
        printf ("%c->%c, ", source, target);
        return;
    }
    HanoiTower (n - 1, source, target, temp);
    printf ("%c->%c, ", source, target);
    HanoiTower (n - 1, temp, source, target);

    return;
}
```

旅行推销员问题

- 一个推销员需要访问N个城市，每个城市访问一次，然后回到出发地
- 需要节省时间或费用
- 如何规划路线才能达到最优？



Roadmap

常用的数学基础

- 指数与对数
- 序列求和
- 证明方法

指数运算

- $x^a * x^b = x^{a+b}$
- $\frac{x^a}{x^b} = x^{a-b}$
- $(x^a)^b = x^{ab}$
- $x^n + x^n = 2x^n$
- $2^n + 2^n = 2^{n+1}$

对数运算

- $\log \frac{a}{b} = \log a - \log b$
- $\log a^b = b \log a$
- $\log x < x, \forall x > 0$
- $\log 1 = 0, \log 2 = 1, \log 1024 = 10, \log 1000000 \approx 20$

对数换底

命题：$\forall b > 0, c > 0, \log_b X = O(\log_c X)$

证明：

假设 $Y = \log_b X$

则根据定义 $b^Y = b^{\log_b X} = X$

那么 $\log_c b^Y = \log_c X = Y \log_c b = \log_c X$

因此 $Y = \frac{\log_c X}{\log_c b}$

由假设 $Y = \log_b X$

有 $\log_b X = \frac{\log_c X}{\log_c b}$

也就是$\log_b X \approx \log_c X$，差别是一个常数$\log_c b$

常用的数量

- $2^{10} = 1024 = 1\mathrm{kilo}$
- $2^{8} = 256$
- $2^{16} = 64\mathrm{k}$
- $2^{20} = 1\mathrm{mega}$
- $2^{30} = 1\mathrm{giga}$
- $2^{\mathrm{n}} = (2^{3})^{\frac{\mathrm{n}}{3}} <= 10^{\frac{\mathrm{n}}{3}}$

常用序列和

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} 2^i = 2^{n+1} - 1$
- $\sum_{i=1}^{n} A^i = \frac{A^{n+1}-1}{A-1}$
- $H_n = \sum_{i=1}^{n} \frac{1}{i} \approx \ln n$，调和序列

数学证明方法

- 直接证明，演绎法
- 假定否命题成立，推出矛盾，反证法
- 数学归纳法

直接证明法

应用逻辑规则和假设推出结论
例如，n^2 是偶数当且仅当 n 是偶数

- 如果n是偶数，$n = 2m$，则$n^2 = (2m)^2 = 4m^2$也是偶数
- 反之，如果$n^2 = 2m$，则n也是偶数

反证法

假定结论的反命题成立，推出矛盾
例如，证明 $\sqrt{2}$是无理数

- 假设$\sqrt{2}$是有理数，则$\exists p, q \in I$，使$\sqrt{2} = \frac{p}{q}$
- 取一对互质的p, q，$p^2 = 2q^2$，p是偶数
- 由于$q^2 = \frac{p^2}{2}$，q也是偶数，继而推出矛盾

数学归纳法

- 演绎与归纳
  - 明天太阳否依旧升起？
  - 数学归纳法是演绎，而非归纳
- 结构化方法：基础，假设，演绎
- 证明$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

Roadmap

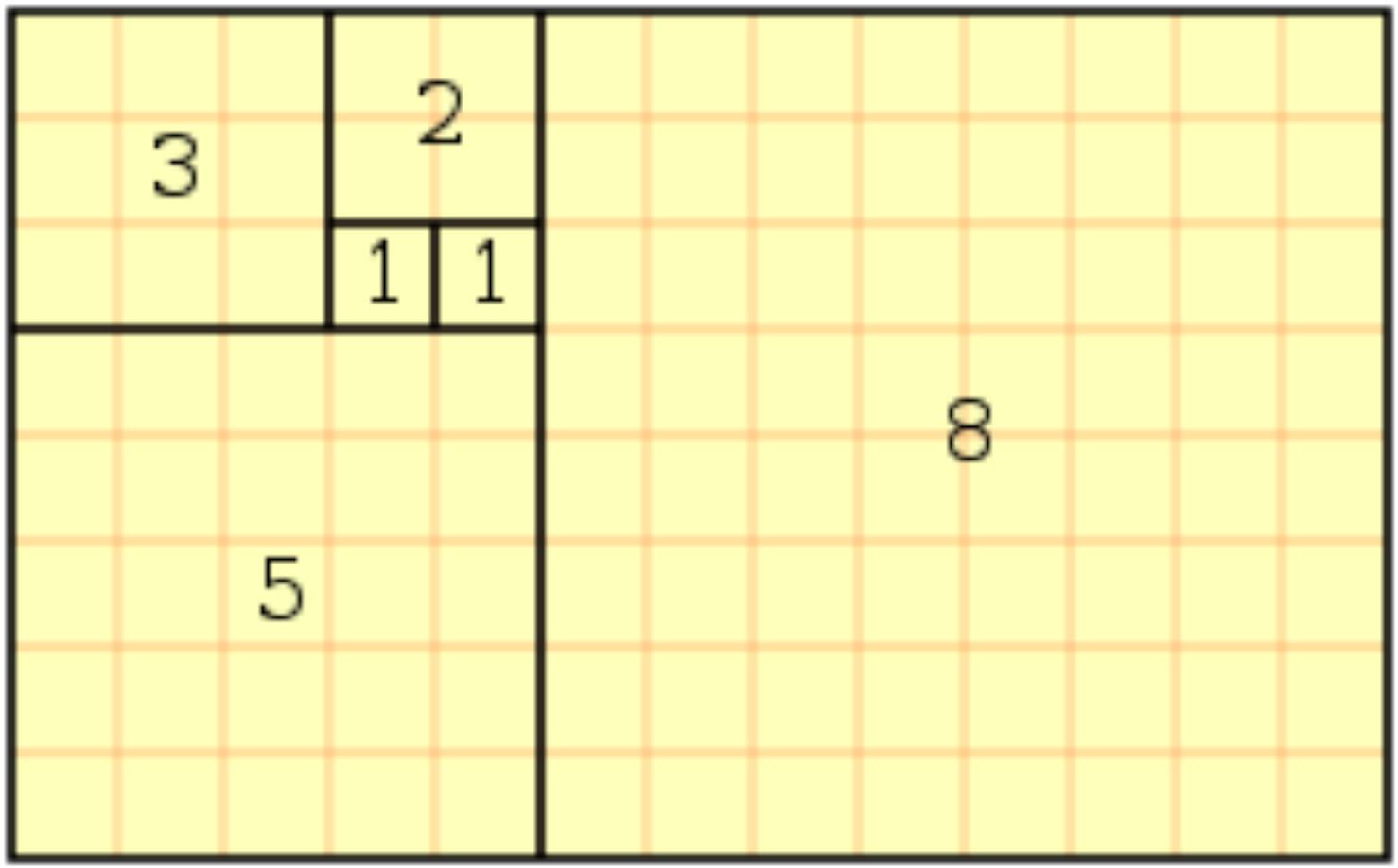
繁殖过程

假设

- 第一个月有一对刚诞生的兔子
- 第二个月之后它们可以生育
- 每月每对可生育的兔子会诞生下一对新兔子
- 兔子永不死去

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$$

Fibonacci数列的图形表示

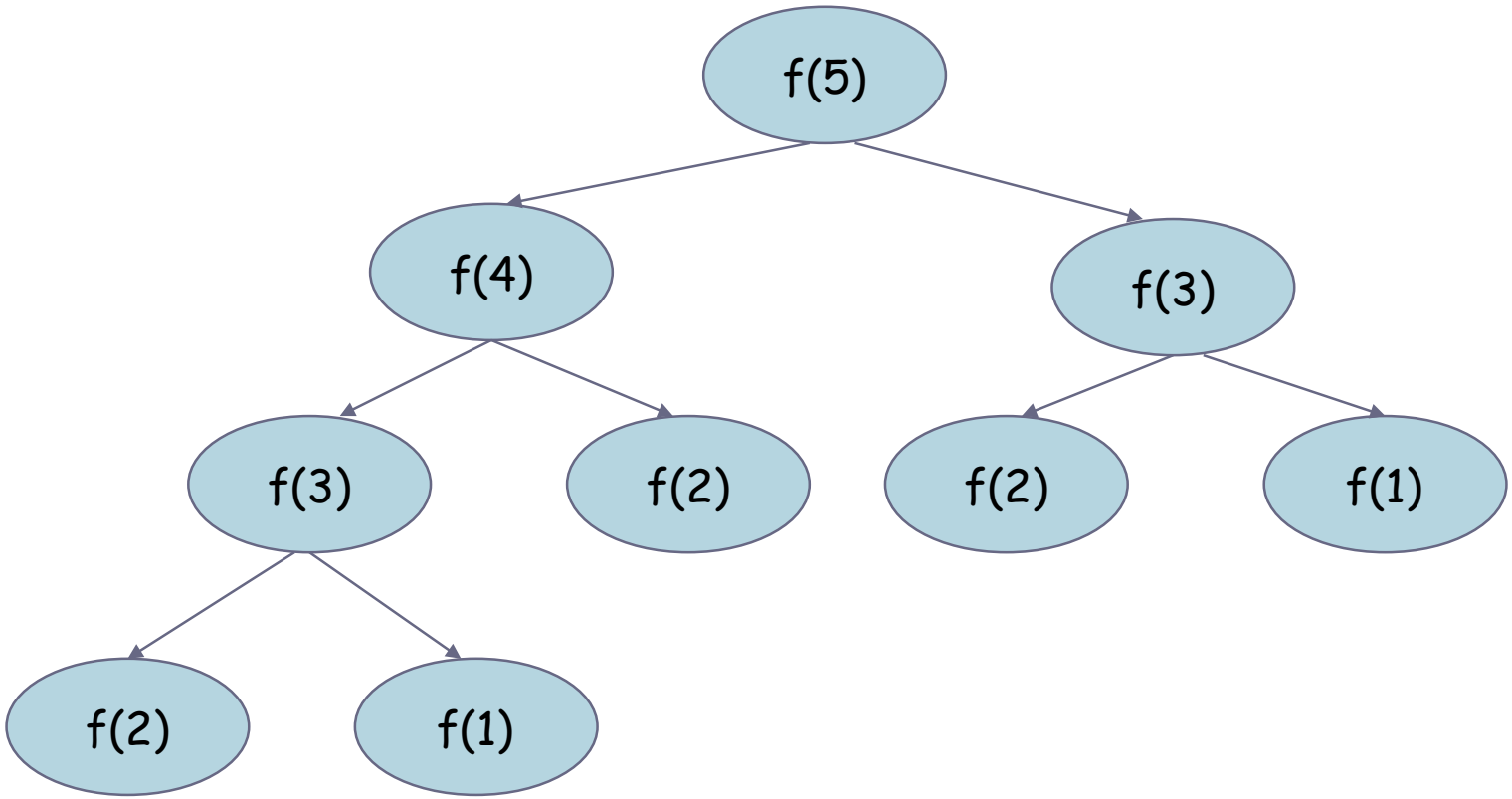


递归实现

```
int fib (int n)
{
    if (n == 1 || n == 2)
        return (1);

    return (fib (n-1) + fib (n-2));
}
```

递归树



递归中的问题

- 结果是否正确？正确，可以用数学归纳法证明
- 程序是否好？不好
  - 效率低下，递归树表明存在大量重复计算
  - 缺乏参数验证保护，易造成死循环，例如fib(-2)

使用迭代

```
int fib (int n)
{
    int tmp1 = 1, tmp2 = 1, res;
     /*fib(n-1), fib(n-2), fib(n)*/

    if (n < 3)
        return (1);

    while (--n >= 2)
    {
        res = tmp1 + tmp2;
        tmp2 = tmp1;
        tmp1 = res;
    }
    return (res);
}
```

其它方法

- 构造新序列，令$f_n + af_{n-1} = b(f_{n-1} + af_{n-2})$，得到 $a = \frac{\sqrt{5}-1}{2}, b = \frac{\sqrt{5}+1}{2}$。因此

$$f_n = \frac{b^n - a^n}{\sqrt{5}}$$

- 使用矩阵运算

$$\begin{bmatrix} f_{n-1} \\ f_n \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_{n-2} \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

递归中的陷阱

```
iint bad (int n)
{
    if (n == 0)
        return (0);

    return (bad (n/3 + 1) + n-1);
}
```

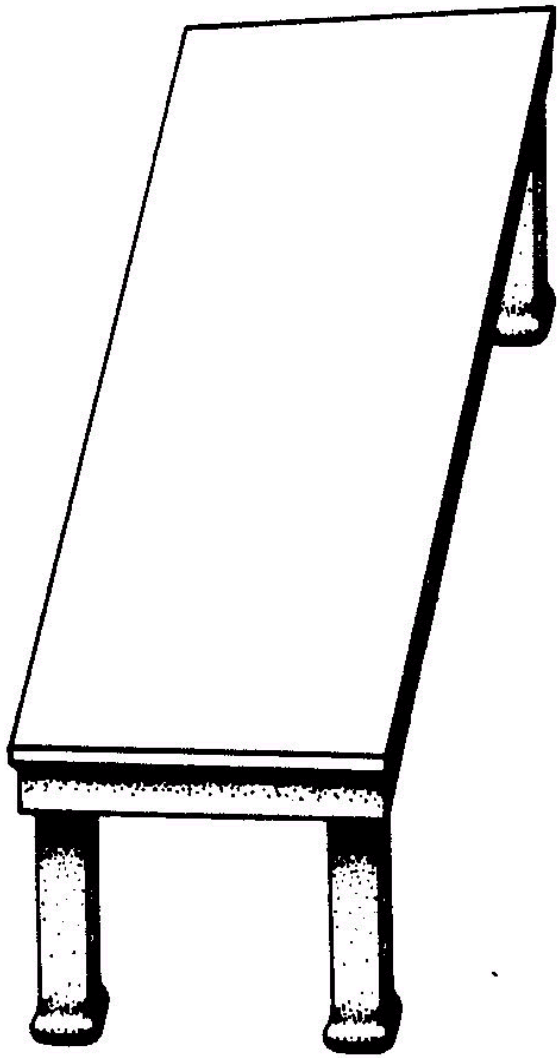
调用bad(4)会发生什么情况?

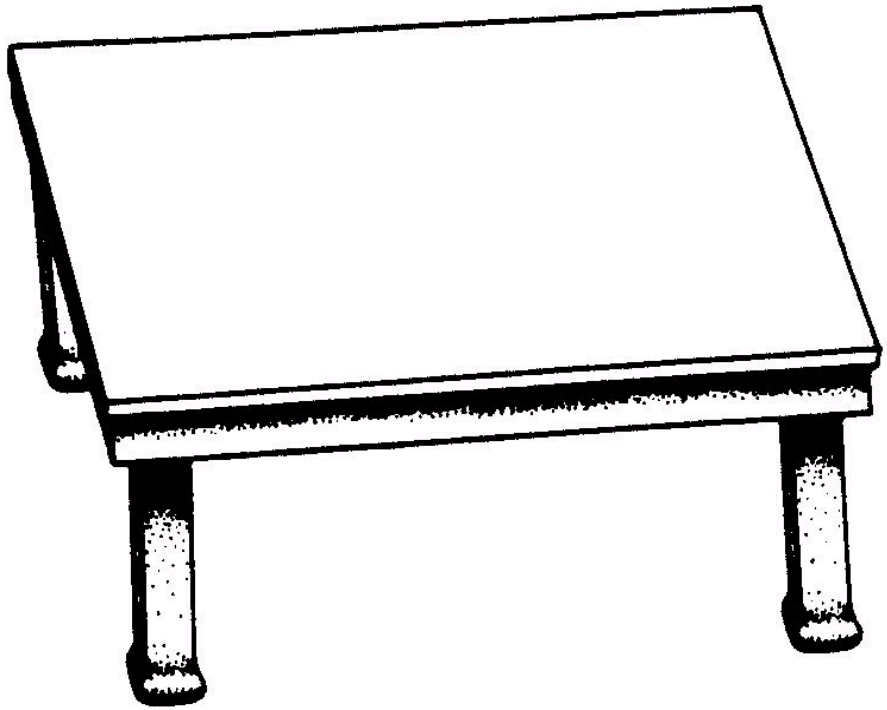
编写递归的一些规则

- 一定要能够到达某一个基础
- 每一步都需要朝着基础变化
- 写代码时可以假定前面的步骤自然成立
- 不要重复已有工作
- 如无必要，不使用递归，而用迭代实现

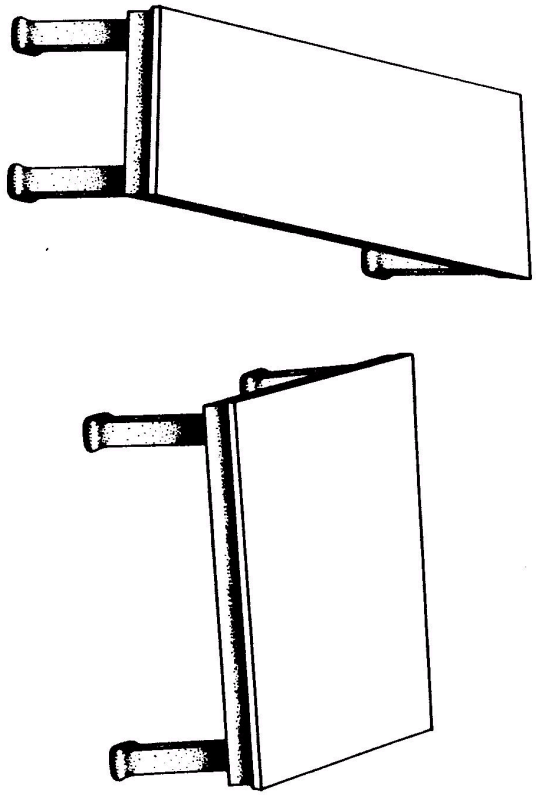
Roadmap

为什么需要度量





为什么需要度量



clock ()函数

- Return the number of clock ticks used by the program
- Synopsis:

```
#include <time.h>
clock_t clock( void );
```

- Description: The clock() function returns the number of clock ticks of processor time used by the program since it started executing. You can convert the number of ticks into seconds by dividing by the value CLOCKS_PER_SEC

clock ()函数举例

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

void compute( void )
{
    int i, j;
    double x; x = 0.0;

    for ( i = 1; i <= 100; i++ )
    {
        for ( j = 1; j <= 100; j++ )
        {
            x += sqrt ((double) i * j );
        }
    }
    printf ( "%16.7f\n", x );
}

int main (void)
{
    clock_t start_time, end_time;

    start_time = clock ();
    compute ();
    end_time = clock ();
    printf ("Execution time was %lu seconds\n", (long)((end_time - start_time)/CLOCKS_PER_SEC) );
    return (0);
}
```

复杂度的设置

Can measure complexity in different settings: best, average, worst

- Best case: sometimes easy to figure, but not very interesting
- Average case: representative, the "expected" time, averaged over all inputs, often hard to figure, sometimes doesn't make sense
- Worst-case: what we almost always want, strong guarantee, can promise your boss: will never be worse than this

RAM模型

- The abstract model for machine independent algorithm design
  - Not Turing machine
  - Random Access Machine
- Consider adding two numbers
  - On a TM, would take log time; just to write one down
  - On a real computer, would happen in essentially one instruction, very fast
- Multiplying takes a little longer, but still very fast

RAM模型

- On RAM model, each "simple" operation takes one unit of time
- Real life is slightly more complicated
  - Multiplication takes longer, but not much!
  - adding billions takes same time as adding 2+2, if using integers
  - data types have fixed sizes! e.g. 32 bits
- Same with memory access, calling a function, etc.
- But loops do matter: time for loop = time for body * number of repetitions

举例

```
int ftn (int n)
{
    int res = 1;

    for (int i = 0; i < n; i++)
        res += i*i*i;

    return (res);
}
```

What is its time complexity in terms of n?

大O表示

- 思路1:度量实际执行时间
  - 问题：通用算法用于解决一般性的问题
  - 可以用于不同输入值的各种情况
  - 无论什么算法，对10000个数排序总比对10个数排序要慢
- 思路2：把执行时间作为输入值的函数来度量
  - 对于数字输入而言有意义
  - 对于输入是复杂对象的算法难以实现

大O表示

- 思路3：把执行时间作为输入规模的函数来度量
  - 问题：执行算法的计算机存在性能差异，有快慢之分
- 思路4：在抽象的RAM上度量作为输入规模函数的操作次数

从复杂度到大O表示

并不需要精确的数值，只需要表示随输入规模变化的趋势。规则：

- Each simple operation: 1
- Sequence of statements: add
- Loops: #reps * body complexity
- Test: 1 + max of children
- Function call: 1 + function complexity

In general, only variable-repetition statements matter

小结

- 数据结构研究大量数据的组织
- 算法按照步骤解决特定问题
- 不同算法存在难易程度的差异
- 解决同一问题的算法有好坏之分
- 时间复杂度可以按照在RAM上简单操作的计数来度量

实验1

- 用C语言编程实现求整数和、切pizza和Hanoi塔等问题的求解
- 在程序中加入clock ()来计算求解时间
- 使用不同的输入值得到对应的时间值
- 分析算法的时间复杂度并与测量结果进行比较
- 如果存在差异，解释原因
- 写出实验报告