算法设计

刘权辉

2024 春

我的经历

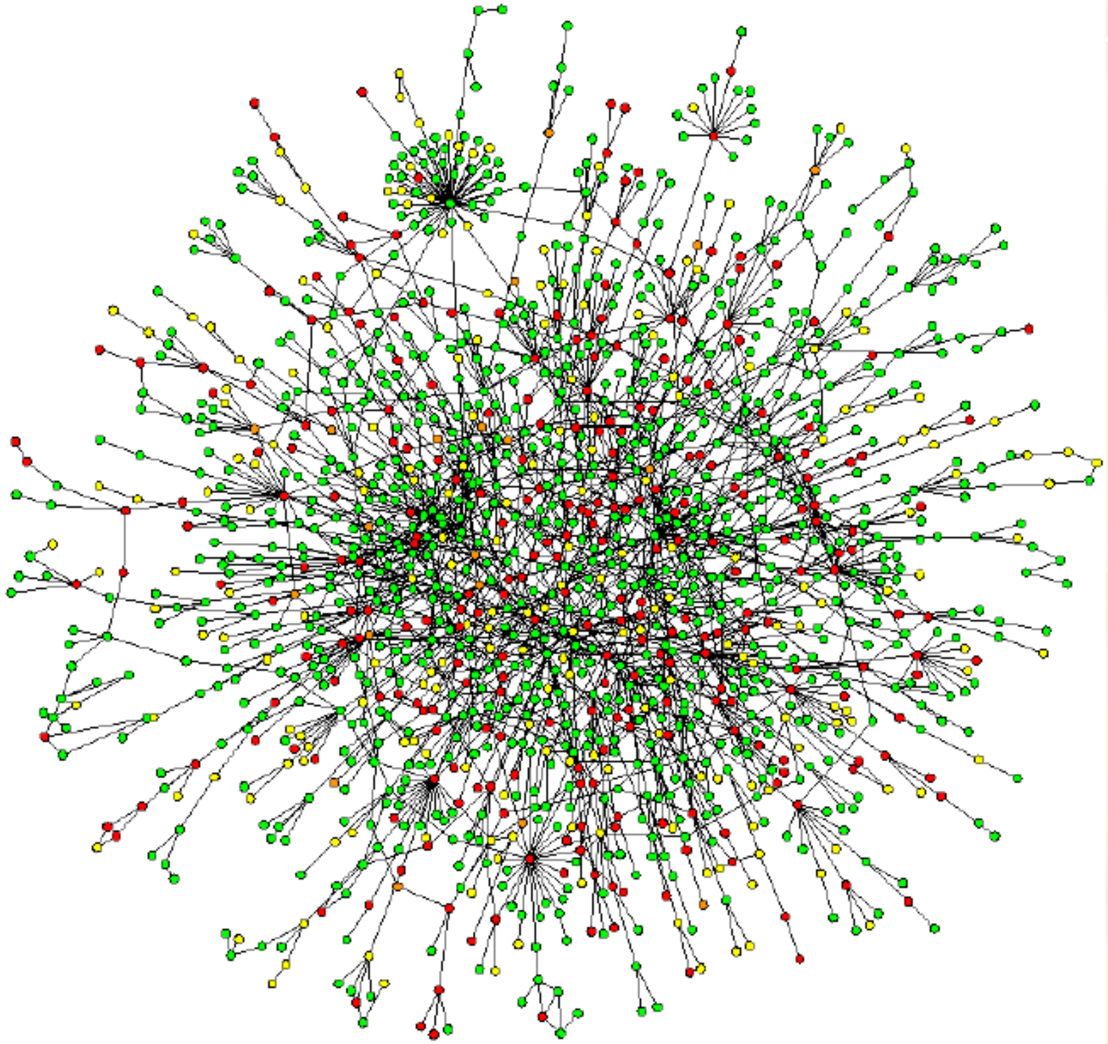
□教育经历

- ➢2009/09 – 2013/06，电子科技大学，计算机学院，本科
- ➢2013/09 – 2019/06，电子科技大学，计算机学院，博士
- ➢2016/09 – 2018/09，美国东北大学， 信息科学系，公派联合培养博士

□工作经历

- ➢2019/06 – 至今，四川大学计算机学院，特聘副研究员，副教授，博导，四川大学校百人B计划入选者

科研方向—网络科学与理论



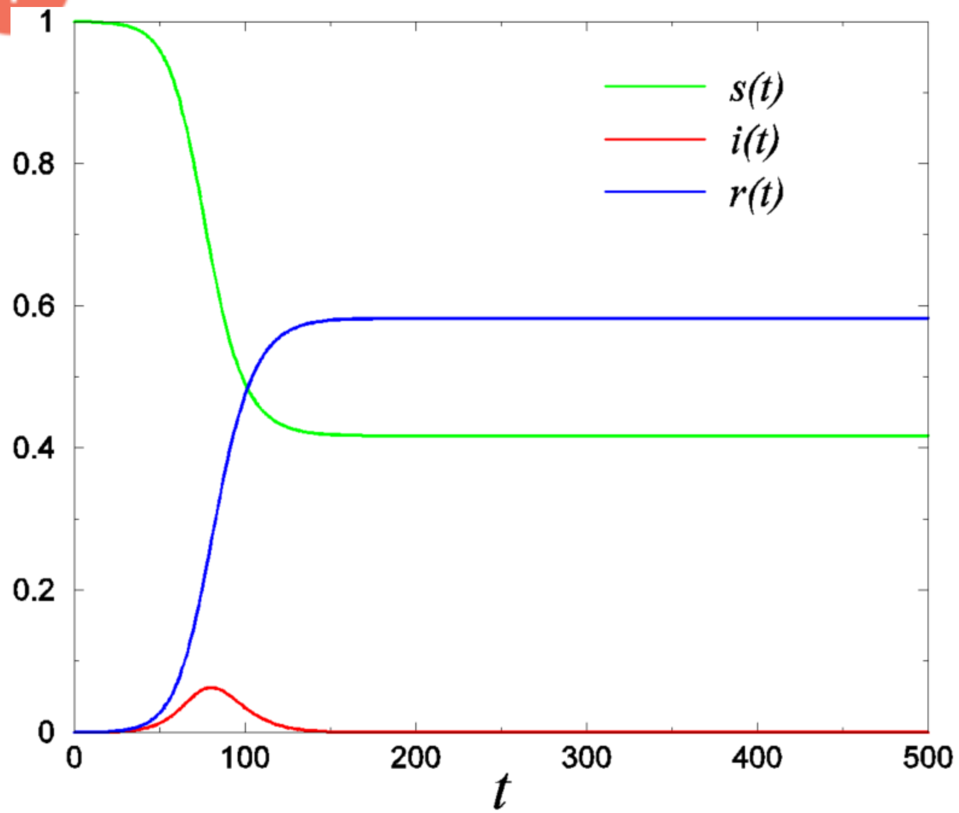
传播动力学机制模型

□模型：SIR， SIS， SEIR， etc.

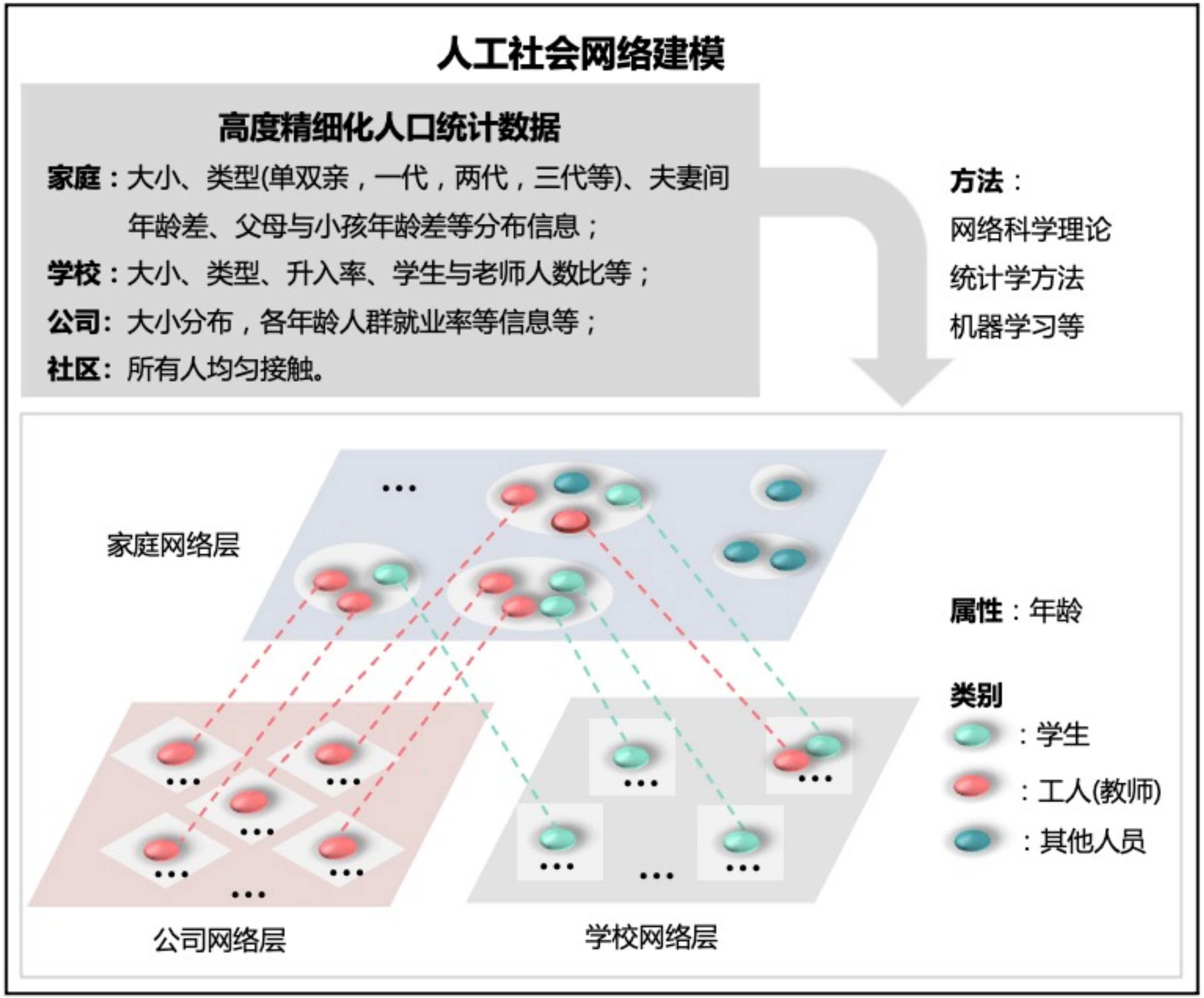
- **SIR (susceptible-infected-recovered): 易感染态-感染态-恢复态**
- **SIR 传播过程：** $S + I \xrightarrow{\beta} I + I$
- **SIR 恢复过程：** $I \xrightarrow{\gamma} R$
- **SIR 动力学过程刻画：** $S'(t) = -\beta \frac{I(t)}{N} S(t)$

$$I'(t) = \beta \frac{I(t)}{N} S(t) - \Upsilon I(t)$$

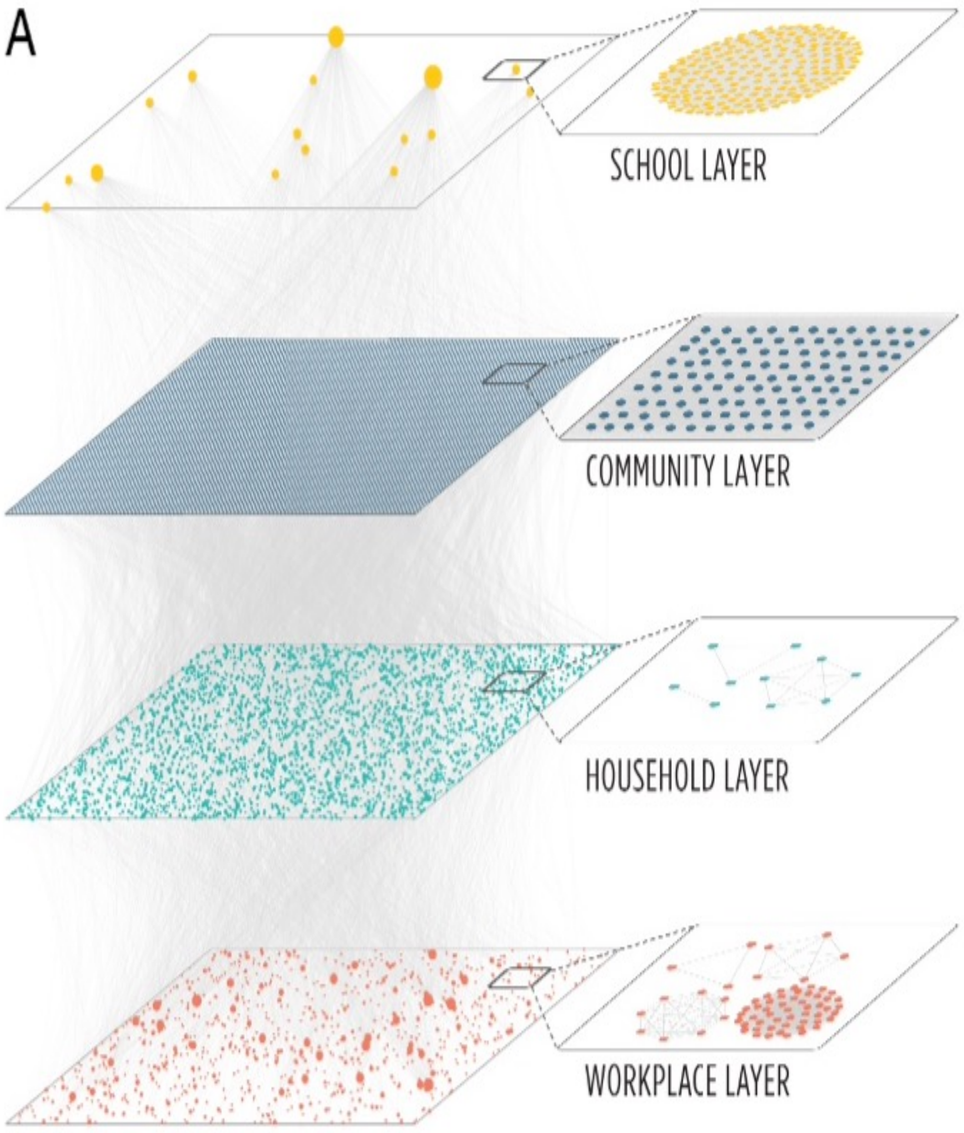
$$R'(t) = \Upsilon I(t)$$



数据驱动的网络建模

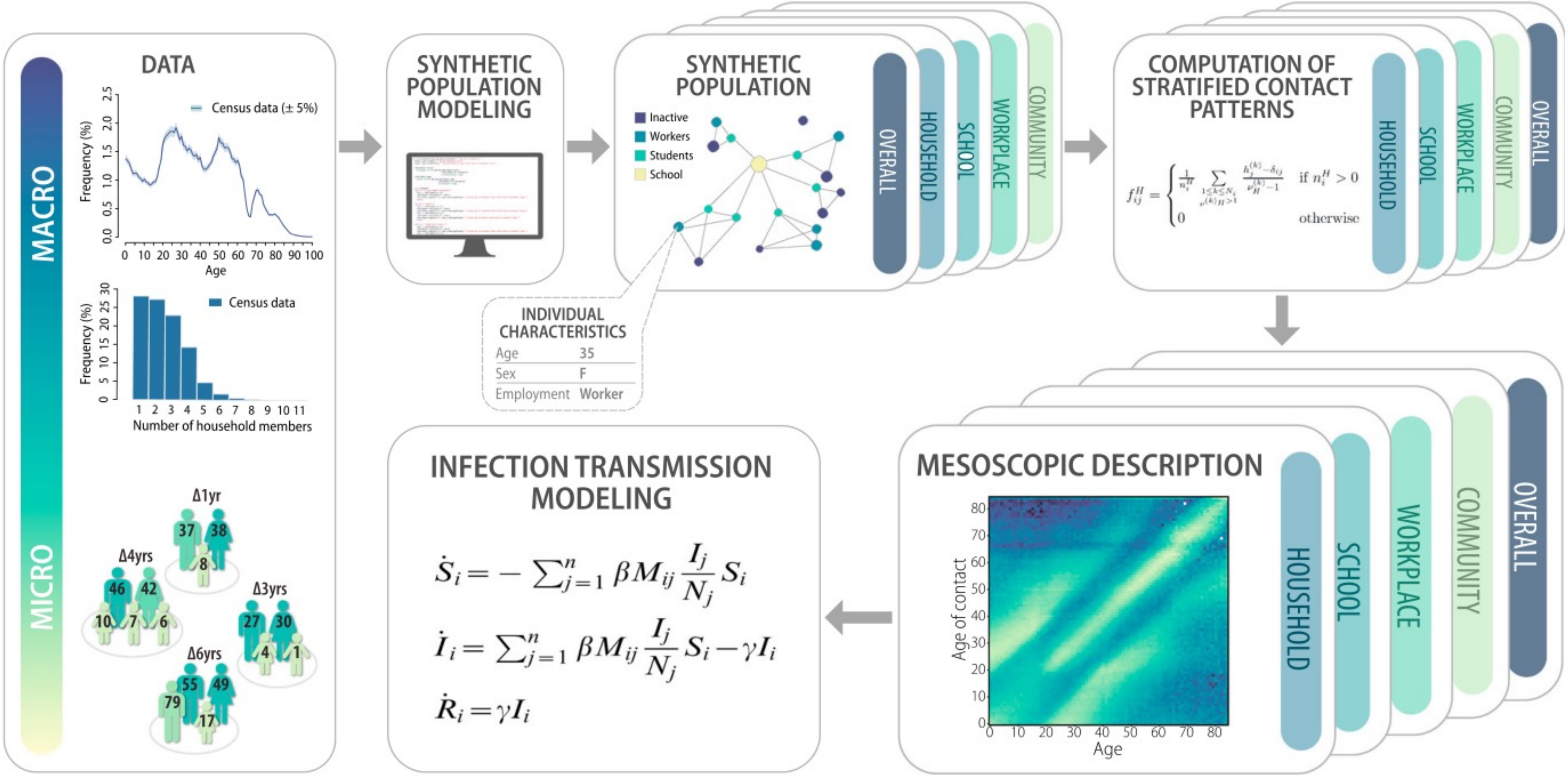


人工社会网络建模
高度精细化人口统计数据
家庭：大小、类型(单双亲，一代，两代，三代等)、夫妻间年龄差、父母与小孩年龄差等分布信息；
学校：大小、类型、升入率、学生与老师人数比等；
公司：大小分布，各年龄人群就业率等信息等；
社区：所有人均匀接触。
方法：
网络科学理论
统计学方法
机器学习等
家庭网络层
属性：年龄
类别
：学生
：工人(教师)
：其他人员
公司网络层
学校网络层

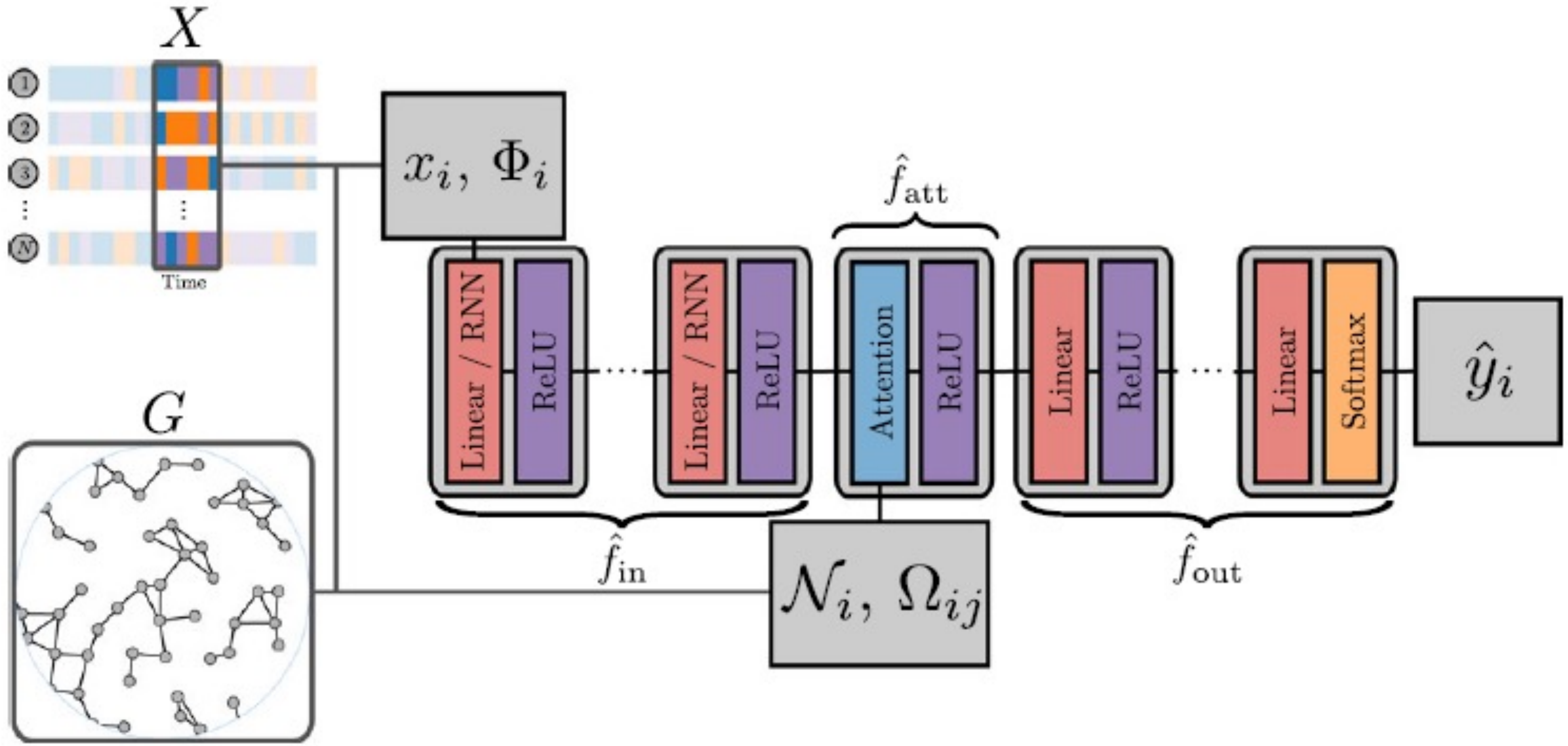


A
SCHOOL LAYER
COMMUNITY LAYER
HOUSEHOLD LAYER
WORKPLACE LAYER

数据驱动网络上的传播动力学建模



基于神经网络的传播动力学建模



- 图神经网络
- 自然语言处理
- 时间序列
- **AI4Science**
- 谣言识别
- 传染病溯源
- 传染病预测
- 等

参考教材

- 余详宣等《计算机算法基础》1998年，华中科技大学出版社。
- 潘彦 译《算法设计与分析基础》2007年，清华大学出版社。
- 霍红卫 译《算法分析与设计》2006年，人民邮电出版社。
- 刘任任 《算法设计与分析》2003年，武汉理工大学出版社。
- 《计算机算法引导》2000年，机械出版社。
- 王晓东 《计算机算法设计与分析》第5版。

联系方式

- **本课程：2学分（选修课）；**
  - 平时成绩(签到+作业) (50%) +期末课程报告(50%)；
- **办公地点**
  - 望江校区基础教学楼B座318A
  - 邮件：quanhuiliu@scu.edu.cn
- **助教：王唯一**
  - 邮件： 1175230433@qq.com
  - 课程QQ群： 867537658
  - 作业提交方式
    - 邮件标题和附件命名规则：学号+姓名+第X次算法设计作业

主要内容

第一章：算法概述

第二章：递归与分治策略

第三章：动态规划

第四章：贪心算法

第五章：回溯法

第六章：分支限界法

算法概述

要点

- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述。
- 掌握用C++语言描述算法的方法。

1.1.1算法的基本概念

◘算法（Algorithm）

➢指解决问题的一种方法或一个过程。严格的讲，由若干条指令组成的有穷序列。

◘算法的性质：

➢输入：有零个或多个由外部提供的量作为算法的输入。

➢输出：算法产生至少一个量作为输出。

➢确定性：组成算法的每条指令是清晰、无歧义的。

➢有限性：算法中每条指令的执行次数是有限的、执行每条指令的时间也是有限的。

1.1.1算法的基本概念

- 程序（Program）
  - 程序是算法用某种程序设计语言的具体实现。
- 程序不同于算法，可以不满足算法的有限性
  - 如操作系统，是一个在无限循环中执行的程序，所以不是算法。
- 算法的描述
  - 可以采用多种形式来描述。本课程中采用C++ 或 Java语言进行描述。

1.1.2 算法的描述

- 欧几里得算法

  求两个正整数最大公约数的算法 (辗转相除法)

- 例如：求1997 和 615两个正整数的最大公约数步骤

  m（被除数）=1997, n（除数）=615, r 为余数

1. 1997 / 615 = 3 (余 152)
2. 615 / 152 = 4(余7)
3. 152 / 7 = 21(余5)
4. 7 / 5 = 1 (余2)
5. 5 / 2 = 2 (余1)
6. 2 / 1 = 2 (余0)

以除数和余数反复做除法运算，当余数为0时，取当前算式除数为最大公约数，所以1997和615的最大公约数为1。

1.1.2算法的描述

◘自然语言 （m是被除数，n是除数，r是余数）

例： 欧几里德算法

① 输入m和n；

② 求m除以n的余数r；

③ 若r等于0，则n为最大公约数，算法结束；否则执行第④步；

④ 将n的值放在m中，将r的值放在n中；

⑤ 重新执行第②步。

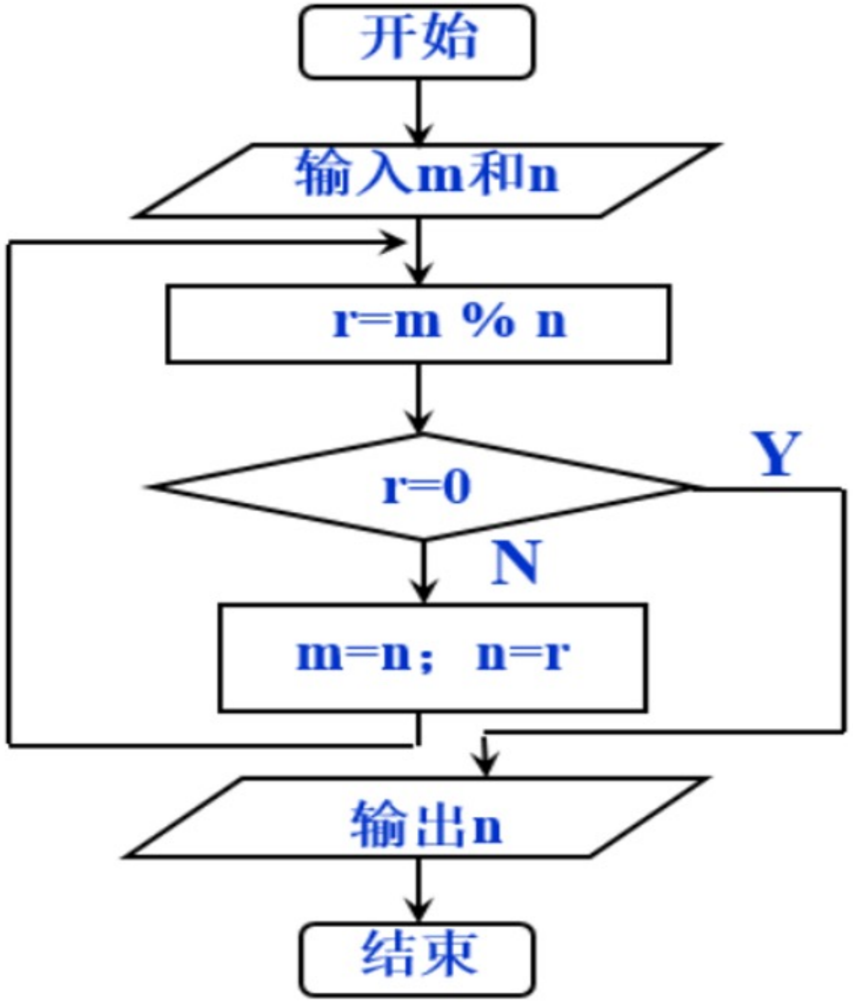
m=1997, n=615, r 为余数

1. 1997 / 615 = 3 (余 152)
2. 615 / 152 = 4(余7)
3. 152 / 7 = 21(余5)
4. 7 / 5 = 1 (余2)
5. 5 / 2 = 2 (余1)
6. 2 / 1 = 2 (余0)

- 优点：容易理解
- 缺点：冗长、二义性

1.1.2算法的描述

□ 流程图



m=1997, n=615, r 为余数

1. 1997 / 615 = 3 (余 152)
2. 615 / 152 = 4(余7)
3. 152 / 7 = 21(余5)
4. 7 / 5 = 1 (余2)
5. 5 / 2 = 2 (余1)
6. 2 / 1 = 2 (余0)

- 优点：流程直观（主要用于描述简单算法）
- 缺点：缺少严密性、灵活性

1.1.2算法的描述

- 程序设计语言 （**m**是被除数，**n**是除数，**r**是余数）

```
#include <iostream.h>
int CommonFactor(int m, int n)
{
    int r=m%n;
    while(r!=0)
    {     m=n;  n=r;  r=m%n;    }
    return n;
}
void main()
{
    cout<<CommonFactor(1997,615)<<endl;
}
```

m=1997, n=615, r 为余数

1. 1997 / 615 = 3 (余 152)
2. 615 / 152 = 4(余7)
3. 152 / 7 = 21(余5)
4. 7 / 5 = 1 (余2)
5. 5 / 2 = 2 (余1)
6. 2 / 1 = 2 (余0)

- 优点：计算机可直接执行
- 缺点：抽象性差，对语言要求高
- 使用方法：算法需要验证（将算法写成函数）

1.1.2算法的描述

□ 伪代码 （Pseudocode，算法设计语言）

- 介于自然语言和程序设计语言之间的方法，它采用某种设计语言的基本语法，操作指令可以结合语言来设计
- 欧几里得算法伪代码

```
1. r=m%n;
    2. 循环直到r等于0
        2.1 m=n;
        2.2 n=r;
        2.3 r=m%n;
    3. 输出n;
```

m=1997, n=615, r 为余数

1. 1997 / 615 = 3 (余 152)
2. 615 / 152 = 4(余7)
3. 152 / 7 = 21(余5)
4. 7 / 5 = 1 (余2)
5. 5 / 2 = 2 (余1)
6. 2 / 1 = 2 (余0)

- 优点：表达能力强，抽象性强，容易理解

1.1.2算法的描述

自然语言

容易理解、冗长、二义性

例： 欧几里德算法

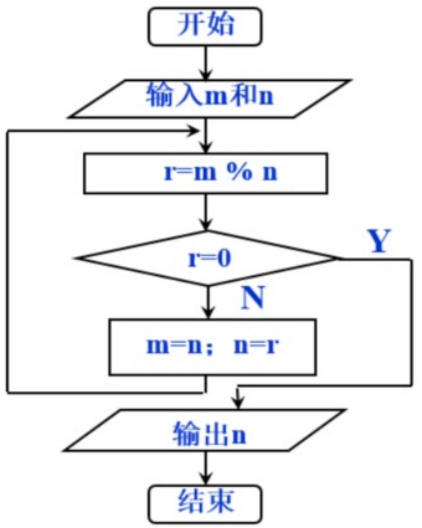
① 输入*m* 和*n*；

② 求*m*除以*n*的余数*r*；

③ 若*r*等于0，则*n*为最大公约数，算法结束；
否则执行第④步；

④ 将*n*的值放在*m*中，将*r*的值放在*n*中；

⑤ 重新执行第②步。



流程图

直观、
不严密、
不灵活

程序设计语言

可执行、对语言要求高

```
#include <iostream.h>
int CommonFactor(int m, int n)
{
    int r=m%n;
    while(r!=0)
    {
        m=n;  n=r;  r=m%n;
    }
    return n;
}

void main()
{
   cout<<CommonFactor(1997,615)<<endl;
}
```

伪代码

自然语言和程序设计语言之间

1. r=m%n;
2. 循环直到r等于0
    2.1 m=n;
    2.2 n=r;
    2.3 r=m%n;
3. 输出n；

1.1.3算法设计一般流程

1. 理解问题
2. 预测所有可能的输入
3. 在精确解和近似解间做选择
4. 确定适当的数据结构
5. 算法设计技术
6. 描述算法
7. 跟踪算法
8. 分析算法的效率
9. 根据算法编写代码

1.1.4算法设计的重要类型问题

1. 查找问题
2. 排序问题
3. 图论
4. 组合问题
5. 几何问题
6. 动态规划
7. 树
8. 贪心
9. 数论

…

1.2.1算法复杂性分析

- 一个算法复杂性的高低体现在运行该算法所需要的计算机资源的多少上。
- 计算机资源
  - 时间资源：算法的时间复杂性T(n)；（主要讨论）
  - 空间资源：空间复杂性 S(n);
  - n是问题的规模（输入大小）
- 算法设计的目标：
  - 设计的算法的复杂性要尽可能低
- 算法复杂性依赖于：
  - 求解问题的规模、算法的输入、算法本身

1.2.1算法复杂性分析

- **例：分别用N、I 和 A 表示算法要解决的问题的规模、算法的具体输入和算法本身。**
  - 若用C表示复杂性，则 **C=F(N,I,A)**
  - 时间复杂性： **T= (N,I,A)**，简记： **T=(N,I)**
  - 空间复杂性：**S=(N,I,A)**， 简记： **S=(N,I)**
- **时间复杂性 T(N,I)的计算：算法在一台抽象计算机上运行所需的时间。**
  - 元运算操作：$O_1, O_2, \ldots, O_k$
  - 元运算时间：$t_1, t_2, \ldots, t_k$
  - 算法中，元运算O_i的执行次数为e_i, i=1,2,…,k,

  则：

$$\mathbf{T(N,I)=\sum_i t_i e_i(N,I)}$$

无法对规模N的每种合法输入I都去统计$e_i(N, I)$

1.2.2 算法的时间复杂性

- 三种情况下的时间复杂性：$T(N,I)=\sum_i t_i e_i(N,I)$
  - 最坏情况下的时间复杂性

$$T_{\max}(n) = \max\{T(I) \mid \text{size}(I) = n\}$$

  - 最好情况下的时间复杂性

$$T_{\min}(n) = \min\{T(I) \mid \text{size}(I) = n\}$$

  - 平均情况下的时间复杂性

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

实践表明，可操作性最好且最有价值的是最坏情况下的时间复杂性。因此，本课程的重点是讨论最坏情况下的时间复杂性分析。

- 其中I是问题n的规模为的实例，$p(I)$是实例I出现的概率。

1.2.3算法的渐进复杂性

- 问题越来越复杂、问题规模也越来越大，问题性质可能发生巨大变化 （量变到质变）
- 算法渐进复杂性

若 $T(n) \to \infty,\ \text{as}\ n \to \infty$

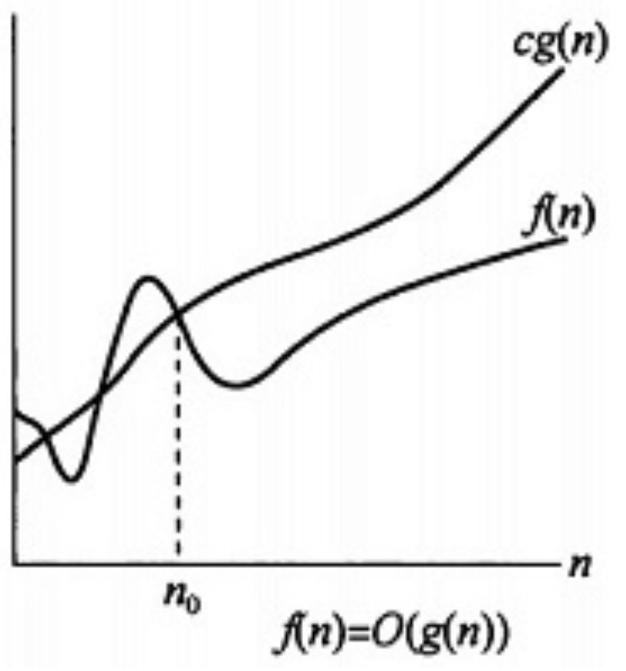
又 $\dfrac{T(n)-t(n)}{T(n)} \to 0, asn \to \infty$

- $t(n)$是$T(n)$的渐进性态，为算法的渐进复杂性。
- 数学上$t(n)$是$T(n)$的渐近表达式，是$T(n)$略去低阶项留下的主项，比$T(n)$ 简单。例： $\mathbf{T(n) = n^3 + n + 1, so\ t(n) = n^3}$

1.2.4渐进分析的记号

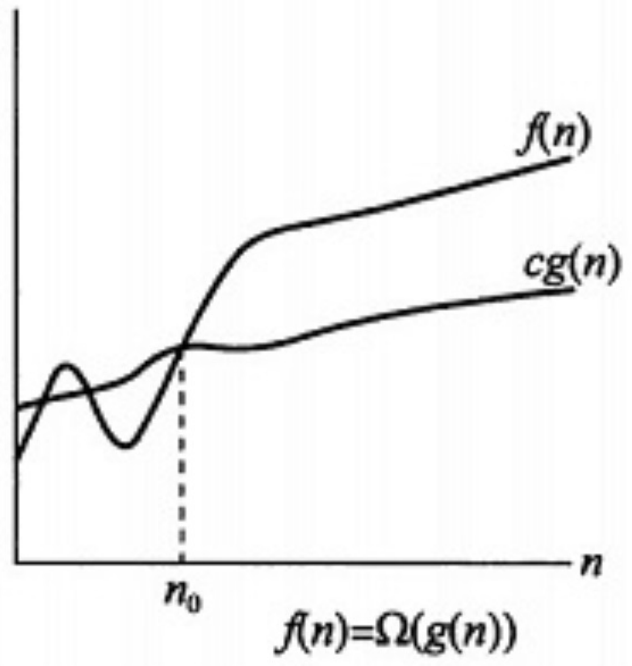
- 对所有 n，f(n)≥0, g(n) ≥0，定义：
  - 渐进上界记号O (小于等于)：

如果存在正的常数c和自然数n_0，使得对所有$n \ge n_0$有：$0 \le f(n) \le cg(n)$，则$f(n)$当n充分大时，有上界，且$g(n)$是它的一个上界，记为：$f(n) = O(g(n))$



  - 渐进下界记号Ω (omega:大于等于)：

如果存在正的常数c和自然数n_0，使得对所有$n \ge n_0$有：$f(n) \ge cg(n)$，则$f(n)$当n充分大时，有下界，且$g(n)$是它的一个下界，记为：$f(n) = \Omega(g(n))$



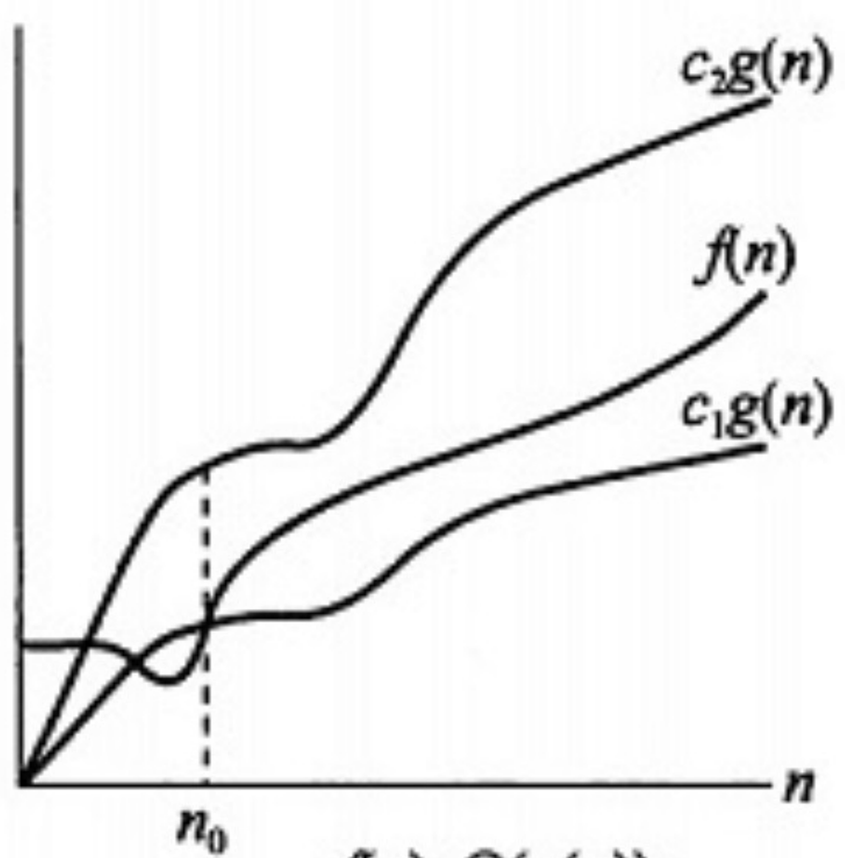
1.2.4渐进分析的记号

- 对所有n，f(n)≥0, g(n) ≥0，定义：
  - **渐进紧界记号Θ (theta)：**

    如果存在正的常数c_1, c_2和自然数n_0，使得对所有$n \geq n_0$有：$c_1 g(n) \leq f(n) \leq c_2 g(n)$

  - **定理1：**

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$



f(n)=Θ(g(n))

1.2.5渐进分析的数学基础

- $f(n) = \Theta(g(n))$ 的确切意义是：$f(n) \in \Theta(g(n))$

- 一般情况下，等式和不等式中的渐近记号$\boldsymbol{\Theta(g(n))}$表示$\boldsymbol{\Theta(g(n))}$中的某个函数

  - 例如：$\mathbf{2}\boldsymbol{n}^{\mathbf{2}} + \mathbf{3}\boldsymbol{n} + \mathbf{1} = \mathbf{2}\boldsymbol{n}^{\mathbf{2}} + \boldsymbol{\Theta}(\mathbf{n})$表示：

$$\mathbf{2}\boldsymbol{n}^{\mathbf{2}} + \mathbf{3}\boldsymbol{n} + \mathbf{1} = \mathbf{2}\boldsymbol{n}^{\mathbf{2}} + \boldsymbol{f}(\boldsymbol{n}),$$

- 其中$\boldsymbol{f}(\boldsymbol{n})$ 是$\boldsymbol{\Theta}(\mathbf{n})$中某个函数，其他记号$\boldsymbol{O}, \boldsymbol{\Omega}$类似

1.2.5渐进分析的数学基础

□ 渐进分析记号的若干性质

（1）传递性：

$$f(n)=\Theta(g(n)),\quad g(n)=\Theta(h(n))\Rightarrow f(n)=\Theta(h(n));$$
$$f(n)=O(g(n)),\quad g(n)=O(h(n))\Rightarrow f(n)=O(h(n));$$
$$f(n)=\Omega(g(n)),\quad g(n)=\Omega(h(n))\Rightarrow f(n)=\Omega(h(n));$$

O 对所有$n \ge n_0$有：$0 \le f(n) \le cg(n)$
Ω 对所有$n \ge n_0$有：$f(n) \ge cg(n)$
Θ对所有$n \ge n_0$有：$c1g(n) \le f(n) \le c2g(n)$

1.2.5渐进分析的数学基础

O 对所有$n \geq n_0$有： $0 \leq f(n) \leq cg(n)$

Ω 对所有$n \geq n_0$有： $f(n) \geq cg(n)$

Θ对所有$n \geq n_0$有： $c1g(n) \leq f(n) \leq c2g(n)$

（2）反身性：

$$f(n) = \Theta(f(n));$$
$$f(n) = O(f(n));$$
$$f(n) = \Omega(f(n)).$$

（3）对称性：

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

（4）互对称性 （o w相对于O,Omega,无等号）：

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n));$$
$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n));$$

1.2.5渐进分析的数学基础

□ 渐进性分析中的常用运算：

$$
\begin{aligned}
O(f(n)+g(n)) &= O(\max\{f(n),g(n)\}) \\
O(f(n))+O(g(n)) &= O(f(n)+g(n)) \\
O(f(n))*O(g(n)) &= O(f(n)*g(n)) \\
O(cf(n)) &= O(f(n))
\end{aligned}
$$

O 对所有$n \geq n_0$有：$0 \leq f(n) \leq cg(n)$

Ω 对所有$n \geq n_0$有：$f(n) \geq cg(n)$

Θ对所有$n \geq n_0$有：$c1g(n) \leq f(n) \leq c2g(n)$

1.2.6渐进分析的常用函数

单调函数

- 单调递增：$m \le n \Rightarrow f(m) \le f(n)$;
- 单调递减：$m \le n \Rightarrow f(m) \ge f(n)$;
- 严格单调递增：$m < n \Rightarrow f(m) < f(n)$;
- 严格单调递减：$m < n \Rightarrow f(m) > f(n)$.

取整函数

- $\lfloor x \rfloor$：不大于x的最大整数；
- $\lceil x \rceil$：不小于x的最小整数。

多项式函数

- $p(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$； $a_d>0$;
- $p(n) = \Theta(n^d)$;
- $f(n) = O(n^k) \Leftrightarrow f(n)$多项式有界；
- $f(n) = O(1) \Leftrightarrow f(n) \le c$;
- $k \ge d \Rightarrow p(n) = O(n^k)$;
- $k \le d \Rightarrow p(n) = \Omega(n^k)$;

1.2.6渐进分析的常用函数

□指数函数

- 对于正整数m,n和实数$a>0$:
- $a^0=1$;
- $a^1=a$;
- $a^{-1}=1/a$;
- $(a^m)^n = a^{mn}$;

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

- $(a^m)^n = (a^n)^m$;
- $a^m a^n = a^{m+n}$;
- $a>1 \Rightarrow a^n$为单调递增函数;
- $a>1 \Rightarrow \lim_{n\to\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = O(a^n)$
- $e^x \geq 1+x$;
- $|x| \leq 1 \Rightarrow 1 + x \leq e^x \leq 1 + x + x^2$
- $e^x = 1 + x + \Theta(x^2)$, as $x \to 0$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x$$

1.2.6渐进分析的常用函数

□ 对数函数

- $\log n = \log_2 n$;
- $\lg n = \log_{10} n$;
- $\ln n = \log_e n$;
- $\log^k n = (\log n)^k$;
- $\log\log n = \log(\log n)$;
- for a>0,b>0,c>0

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

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

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

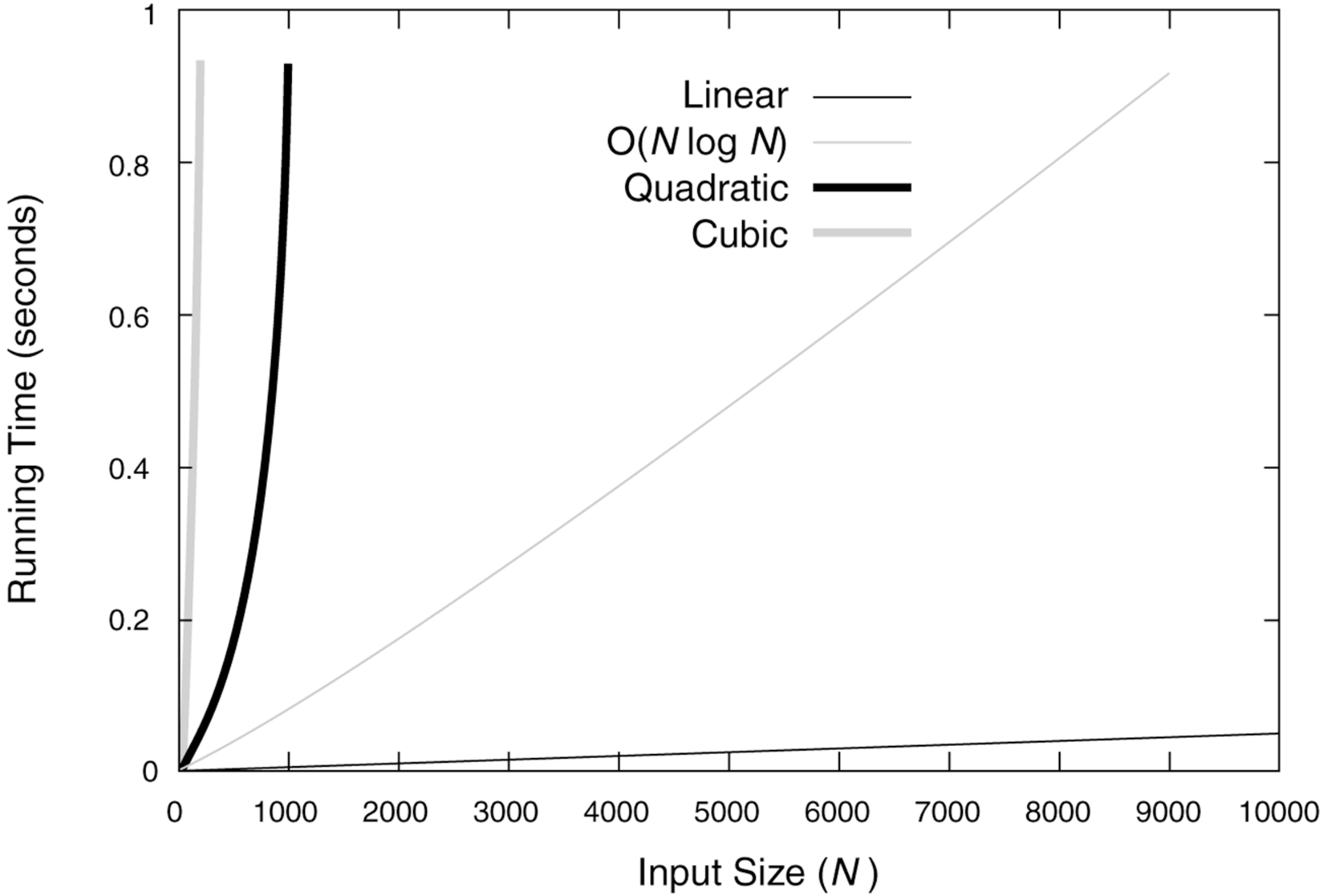
$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

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

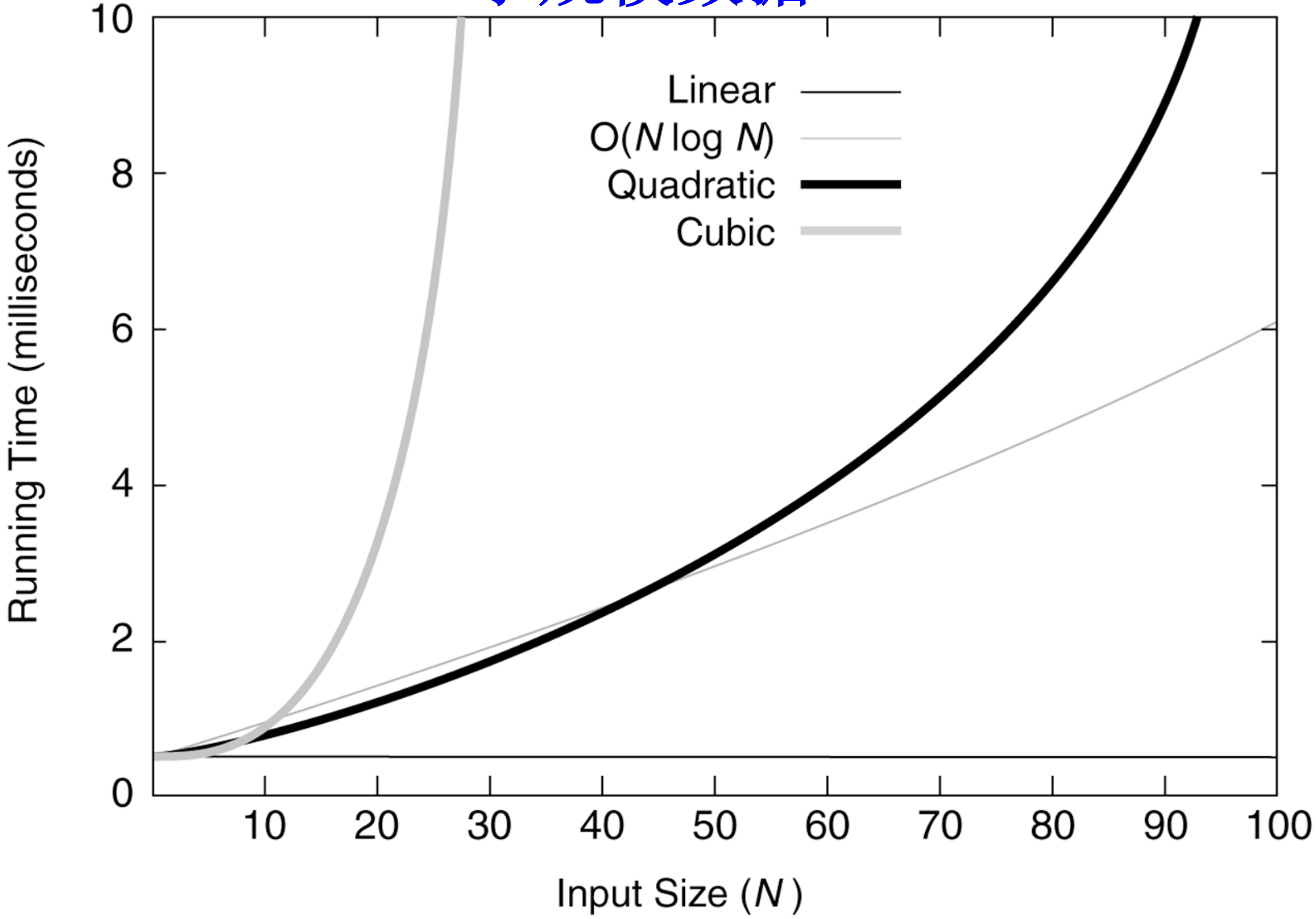
$$a^{\log_b c} = c^{\log_b a}$$

中等规模数据



Linear
O(N log N)
Quadratic
Cubic
Running Time (seconds)
Input Size (N)
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

小规模数据



10
8
6
4
2
0
Running Time (milliseconds)
Linear
O(N log N)
Quadratic
Cubic
10
20
30
40
50
60
70
80
90
100
Input Size (N)

例：顺序搜索的时间复杂性分析

- **最优**情况下： $T_{\min}(n)=\min\{T(I)\mid \mathrm{size}(I)=n\}=O(1)$

- **最坏**情况下： $T_{\max}(n)=\max\{T(I)\mid \mathrm{size}(I)=n\}=O(n)$

- **在平均情况下**，假设：
  - 搜索成功的概率为$\boldsymbol{p}(\mathbf{0}\leq \boldsymbol{p}\leq \mathbf{1})$；
  - 在数组的每个位置$\boldsymbol{i}(\mathbf{0}\leq \boldsymbol{i}<\boldsymbol{n})$搜索成功的概率相同，均为 $\frac{\boldsymbol{p}}{\boldsymbol{n}}$。

$$
\begin{aligned}
T_{\text{avg}}(n) &= \sum_{size(I)=n} p(I)T(I) \\
&= \left(1\cdot\frac{p}{n}+2\cdot\frac{p}{n}+3\cdot\frac{p}{n}+\cdots+n\cdot\frac{p}{n}\right)+n\cdot(1-p) \\
&= \frac{p}{n}\sum_{i=1}^{n} i+n(1-p)=\frac{p(n+1)}{2}+n(1-p)
\end{aligned}
$$

例：插入排序的时间复杂性分析

```
template<class Type>
void insertion_sort(Type *a, int n)
{
    Type key;                                  //  cost   times
    for (int i = 1; i < n; i++){               //  c1     n
        key=a[i];                              //  c2     n-1
        int j=i-1;                             //  c3     n-1
        while( j>=0 && a[j]>key ){             //  c4     sum of ti
         a[j+1]=a[j];                          //  c5     sum of (ti-1)
         j--;                                  //  c6     sum of (ti-1)
         }
       a[j+1]=key;                             //  c7     n-1
     }
}
```

最好输入

1	2	3	4	5

最坏输入

5	4	3	2	1

例：插入排序的时间复杂性分析

□ 时间

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=1}^{n-1} t_i + c_5 \sum_{i=1}^{n-1} (t_i - 1) + c_6 \sum_{i=1}^{n-1} (t_i - 1) + c_7(n-1)$$

□ 最好情况（递增），$t_i = 1$，对于$1 < i < n$:

1	2	3	4	5

$$T_{\min}(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = O(n)$$

□ 最坏情况（递减），$t_i = i + 1$，对于$1 \le i < n$;

5	4	3	2	1

$$\sum_{i=1}^{n-1} (i+1) = \frac{n(n+1)}{2} - 1 \qquad \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

$$T_{\max}(n) \le c_1 n + c_2(n-1) + c_3(n-1) +$$

$$c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$= \frac{c_4+c_5+c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4-c_5-c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

$$= O(n^2)$$

例：插入排序的时间复杂性分析

□对于输入数据a[i]=n-i,i=0,1,…,n-1，算法insertion_sort达到其最坏情形。因此，

$$T_{\max}(n) \geq \frac{c_4+c_5+c_6}{2}n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4-c_5-c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$
$$= \Omega\left(n^2\right)$$

➢由此可见

$$T_{\max}(n) = \Theta\left(n^2\right)$$

递归算法的分析

- 关键：根据递归过程建立递归关系式，然后求解这个递归关系式。
- 常用方法：
  - 猜测技术：对递归关系式估计一个上限，然后用数学归纳法证明它正确。
  - 递归归纳技术：

$$T(n)=\begin{cases} 7 & n=1 \\ 2T\left(\frac{n}{2}\right)+5n^2 & n>1 \end{cases}$$

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right)+5n^2 \\
&= 2\left(2T\left(\frac{n}{4}\right)+5\left(\frac{n}{2}\right)^2\right)+5n^2 \\
&= 2\left(2\left(2T\left(\frac{n}{8}\right)+5\left(\frac{n}{4}\right)^2\right)+5\left(\frac{n}{2}\right)^2\right)+5n^2 \\
&= 2^kT(1)+2^{k-1}5\left(\frac{n}{2^{k-1}}\right)^2+\cdots+2*5\left(\frac{n}{2}\right)^2)+5n^2
\end{aligned}$$

$$T(n)=7n+5\sum_{i=0}^{k-1}\left(\frac{n}{2^i}\right)^2=7n+5n^2\left(2-\frac{1}{2^{k-1}}\right)=10n^2-3n\le 10n^2=O(n^2)$$

递归算法的分析

■常用方法：

➢**通用分治递推式：**大小为**n的原问题**分解成若干个**大小为n/b的子问题**，其中**a**个子问题需要求解，而cn^k是合并各个子问题的解需要的工作量。

$$T(n)=\begin{cases} c & n=1 \\ aT\left(\frac{n}{b}\right)+cn^k & n>1 \end{cases} \qquad T(n)=\begin{cases} O\left(n^{\log_b a}\right) & a>b^k \\ O\left(n^k \log_b a\right) & a=b^k \\ O\left(n^k\right) & a<b^k \end{cases}$$

算法复杂性分析

□**最优算法：**

- **问题的计算时间下界为 $\Omega(f(n))$，则计算时间复杂性为 O(f(n))的算法是最优算法；**
- **例如，排序问题的计算时间下界为 $\Omega(nlogn)$，计算时间复杂性为O(nlogn)的排序算法是最优算法。**
- **堆排序算法是最优算法。**

End

补充材料-递推方程求解

- **递推方程定义:**
  给定数列f(0),f(1),…,f(n), 一个把f(n)和某些f(i)，$0 \le i < n$，联系起来的等式称为递推方程
- 给定关于f(n)的递推方程和初值，求f(n)称为**解递推方程**

- **求解方法**

  公式法　　　　换元法

  迭代归纳法　　　差消法

  Master定理

补充材料-递推方程求解

■ 1. 常系数线性齐次递推方程的求解（公式法）

标准形式：k阶 $H(n)-a_1H(n-1)-a_2H(n-2)-\cdots-a_kH(n-k)=0,$

$n\geq k, a_1,a_2,\ldots,a_k$是常数， $a_k\neq 0$

求解步骤：

（1）求出特征方程 $x^k-a_1x^{k-1}-\ldots-a_k=0$ 的k个根

（2）如果没有重根，则该递推方程的通解为

$$H(n)=C_1{q_1}^n+C_2{q_2}^n+\ldots+C_k{q_k}^n$$

$C_1,C_2,\ldots,C_k$待定常数

如果有重根，如果q是e重特征根，通解对应于根q的部分为

$$(C_1+C_2n+\ldots+C_en^{e-1})q^n$$

整个通解为各个不等的特征根的对应部分之和

（3）代入初值确定待定常数。

补充材料-递推方程求解

例 Fibonacci数列

$$f_n = f_{n-1} + f_{n-2}$$
$$f_0 = 1, f_1 = 1$$

解：x²-x-1 =0 的根为

$$\frac{1+\sqrt{5}}{2}, \frac{1-\sqrt{5}}{2}$$

$$f_n = C_1\left(\frac{1+\sqrt{5}}{2}\right)^n + C_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

带入初值 得

$$\begin{cases} C_1 + C_2 = 1 \\ C_1\left(\frac{1+\sqrt{5}}{2}\right) + C_2\left(\frac{1-\sqrt{5}}{2}\right) = 1 \end{cases}$$

解得

$$C_1 = \frac{1}{\sqrt{5}}\frac{1+\sqrt{5}}{2}, \quad C_2 = -\frac{1}{\sqrt{5}}\frac{1-\sqrt{5}}{2}$$

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}$$

补充材料-递推方程求解

例7 H(n)+H(n-1)-3H(n-2)-5H(n-3)-2H(n-4) = 0

H(0) = 1, H(1) = 0, H(2) = 1, H(3) = 2

特征方程 $x^4+x^3-3x^2-5x-2=0$，**特征根**-1，-1，-1，2

通解为

$$H(n)=(C_1+C_2n+C_3n^2)(-1)^n+C_4 2^n$$

$$\begin{cases} C_1+C_4=1 \\ -C_1-C_2-C_3+2C_4=0 \\ C_1+2C_2+4C_3+4C_4=1 \\ -C_1-3C_2-9C_3+8C_4=2 \end{cases}$$

解得

$$C_1=\frac{7}{9}, C_2=-\frac{1}{3}, C_3=0, C_4=\frac{2}{9}$$

解为

$$H(n)=\frac{7}{9}(-1)^n-\frac{1}{3}n(-1)^n+\frac{2}{9}2^n$$

补充材料-递推方程求解

- **常系数线性非齐次递推方程求解（公式法）**

标准形:

$$H(n)-a_1H(n-1)-a_2H(n-2)-\ldots-a_kH(n-k)=f(n)$$
$$H(0)=d_0,H(1)=d_1,H(2)=d_2,\ldots,H(k-1)=d_{k-1}$$

通解为对应的齐次通解加上特解:

$$H(n)=\overline{H}(n)+H^*(n)$$

特解的函数形式依赖于f(n)

求解的关键是用待定系数法确定一个特解$H^*(n)$

补充材料-递推方程求解

- **f(n)为n的t次多项式，一般$H^*(n)$也为n的t次多项式**

例 求$a_n+5a_{n-1}+6a_{n-2}=3n^2$的通解

设 $a_n^*=P_1n^2+P_2n+P_3$，

代入得

$$P_1n^2+P_2n+P_3+5[P_1(n-1)^2+P_2(n-1)+P_3]+6[P_1(n-2)^2+P_2(n-2)+P_3]=3n^2$$

从而得到方程组

$$12P_1=3$$

$$-34P_1+12P_2=0$$

$$29P_1-17P_2+12P_3=0$$

$$P_1=\frac{1}{4},\quad P_2=\frac{17}{24},\quad P_3=\frac{115}{288}$$

$$a_n^*=\frac{1}{4}n^2+\frac{17}{24}n+\frac{115}{288}$$

通解为: $a_n=C_1(-2)^n+C_2(-3)^n+\frac{1}{4}n^2+\frac{17}{24}n+\frac{115}{288}$

补充材料-递推方程求解

f(n)为指数函数 β^n，特解也为指数形式

若β不是特征根，则特解为$H^*(n) = P\beta^n$

若β是e重特征根，则特解为$Pn^e\beta^n$

例 $H(n) + 5H(n-1) + 6H(n-2) = 42\cdot 4^n$

令 $H^*(n) = P\,4^n$,

代入得

$$P\,4^n + 5P\,4^{n-1} + 6P\,4^{n-2} = 42\cdot 4^n$$

$$42P = 42\cdot 16,\quad P = 16$$

通解为 $H(n) = C_1(-2)^n + C_2(-3)^n + 4^{n+2}$

补充材料-递推方程求解

■2. 转化成常系数线性递推方程求解---换元法

例 归并排序

$T(n) = 2\,T(n/2) + n-1$, $n = 2^k$

$T(2) = 1$

$H(k) = 2H(k-1) + 2^k - 1$

$H(1) = 1$

令 $H^*(k) = P_1 k 2^k + P_2$ **,解得**

$P_1 = P_2 = 1$, $H^*(k) = k2^k + 1$

通解 $H(k) = C\,2^k + k2^k + 1$,

代入初值,得

$C = -1$

$H(k) = -2^k + k2^k + 1$

$T(n) = n \log n - n + 1$

补充材料-递推方程求解

■ **3．叠代归纳法 (*较常用*)**

例13　H(n) = (4n-6) H(n-1)

H(1) =1

$$
\begin{aligned}
H(n) &= (4n-6)H(n-1) \\
&= (4n-6)(4n-10)H(n-2) \\
&= \ldots \\
&= (4n-6)(4n-10)\ldots 6\cdot 2\cdot H(1) \\
&= 2^{n-1}[(2n-3)(2n-5)\ldots 3\cdot 1] \\
&= 2^{n-1}\frac{(2n-2)!}{(2n-1)(2n-4)\ldots 4\cdot 2} \\
&= \frac{(2n-2)!}{(n-1)!}
\end{aligned}
$$

用归纳法验证。

注：此法需要较强的观察、归纳能力

补充材料-递推方程求解

■ 4．差消法----化简递推方程（了解）

例14 求解递推方程

$$T(n)=\frac{2}{n}\sum_{i=1}^{n-1}T(i)+n-1,\quad n\geq 2$$

$$T(1)=0$$

乘以n

$$nT(n)=2\sum_{i=1}^{n-1}T(i)+n^2-n$$

$$(n-1)T(n-1)=2\sum_{i=1}^{n-2}T(i)+(n-1)^2-(n-1)$$

相减并化简得

$$nT(n)=(n+1)T(n-1)+2n-2$$

除以(n+1)

$$\frac{T(n)}{n+1}=\frac{T(n-1)}{n}+\frac{2}{n+1}-\frac{2}{(n+1)n}$$

由叠代得

$$\frac{T(n)}{n+1}=\frac{2}{n+1}+\frac{2}{n}+\frac{2}{n-1}+\ldots+\frac{T(1)}{2}-O(n$$

$$=2(\frac{1}{n+1}+\frac{1}{n}+\ldots+\frac{1}{3})-O(n)$$

T(n)=O(nlog n)

补充材料-递推方程求解

■5.Master定理

设$a \ge 1, b > 1$为常数，$f(n)$为函数

$$T(n) = aT(n/b) + f(n),$$

$T(n)$为非负整数

1. $f(n) = O(n^{\log_b a-\varepsilon}), \varepsilon > 0,$

   那么$T(n) = \Theta(n^{\log_b a})$

2. $f(n) = \Theta(n^{\log_b a}),$

   那么$T(n) = \Theta(n^{\log_b a}\log n)$

3. $f(n) = \Omega(n^{\log_b a+\varepsilon}), \varepsilon > 0,$

   且对于某个常数$c < 1$和所有的充分大的 n有

   $$af(n/b) \le cf(n),$$

   那么$T(n) = \Theta(f(n))$

注； 此类型的问题通常也可递推归纳法

补充材料-递推方程求解

■5.Master定理

例1 $T(n)=9T(n/3)+n$

$a=9, b=3, f(n)=n, \quad n^{\log_3 9}=n^2,$

$f(n)=O(n^{\log_3 9-1}), \quad T(n)=\Theta(n^2)$

例2 $T(n)=T(2n/3)+1$

$a=1, b=3/2, f(n)=1, n^{\log_{3/2} 1}=n^0=1,$

$f(n)=\Theta(n^{\log_{3/2} 1}), T(n)=\Theta(\log n)$

例3 $T(n)=3T(n/4)+n\log n$

$a=3, b=4, f(n)=n\log n, n^{\log_4 3}=O(n^{0.793}),$

$f(n)=\Omega(n^{\log_4 3+\varepsilon}), \varepsilon\approx 0.2,$

$af(n/b)=3(n/4)\log(n/4)\leq(3/4)n\log n=cf(n), c=3/4, n$充分大

$T(n)=\Theta(n\log n)$