算法分析与设计

陈长建

计算机科学系

教学组

- 陈长建
  - 计算机科学系
  - 办公室：超算1号楼310
  - 邮箱：changjianchen@hnu.edu.cn
  - 研究领域：人机交互、机器学习
  - 主页： https://changjianchen.github.io/

教学组 – 助教

- 吕斐
  - 办公室：超算1号楼420
  - 邮箱：feilv@hnu.edu.cn
  - 电话：18272084258
- 关亚龙
- 王鹏程

课程群

https://mooc1.chaoxing.com/course/245394486.html

算法分析与设计课（计...

群号：857829147

什么是算法?

- 算法 = 计算机?

- 算法 = 人工智能?

算法是解决问题的过程和方法

例子-数人数

- 如何数一个房间里的人数?

算法1-单个枚举法

- 一个一个数

算法1-单个枚举法

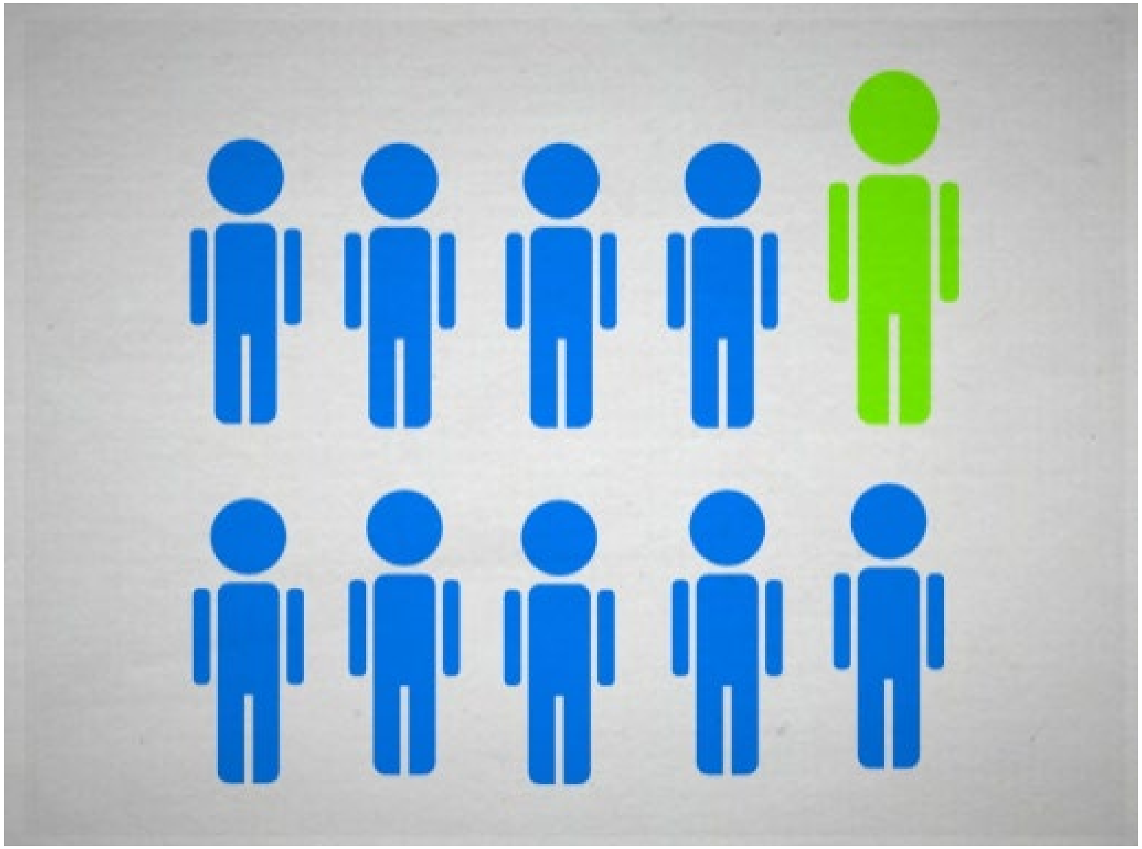
- 人数多的时候，效率变慢

算法2-多个枚举法

- 一次数两个，效率提高一倍

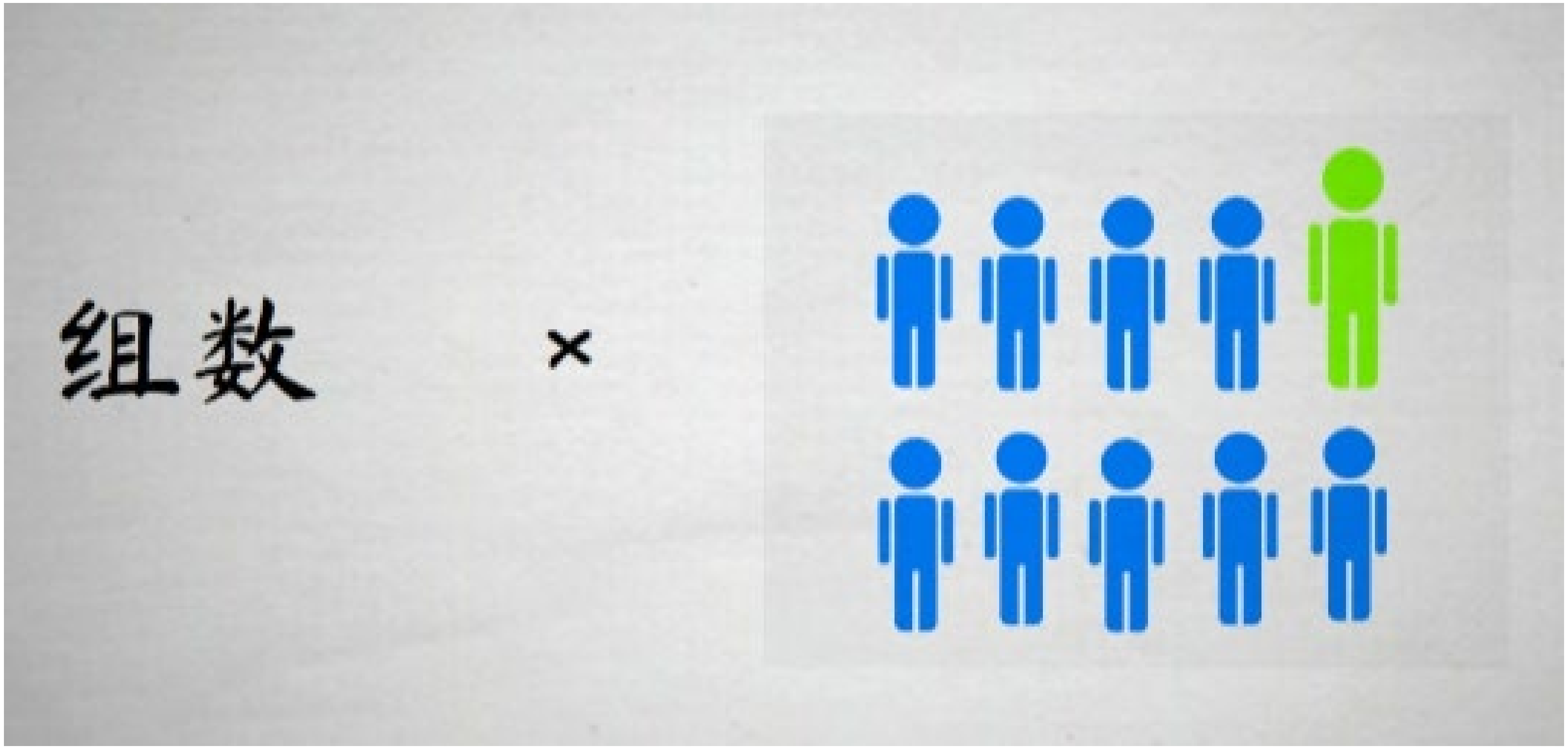
算法2-多个枚举法

- 一次数十个？ 效率并没有提升！

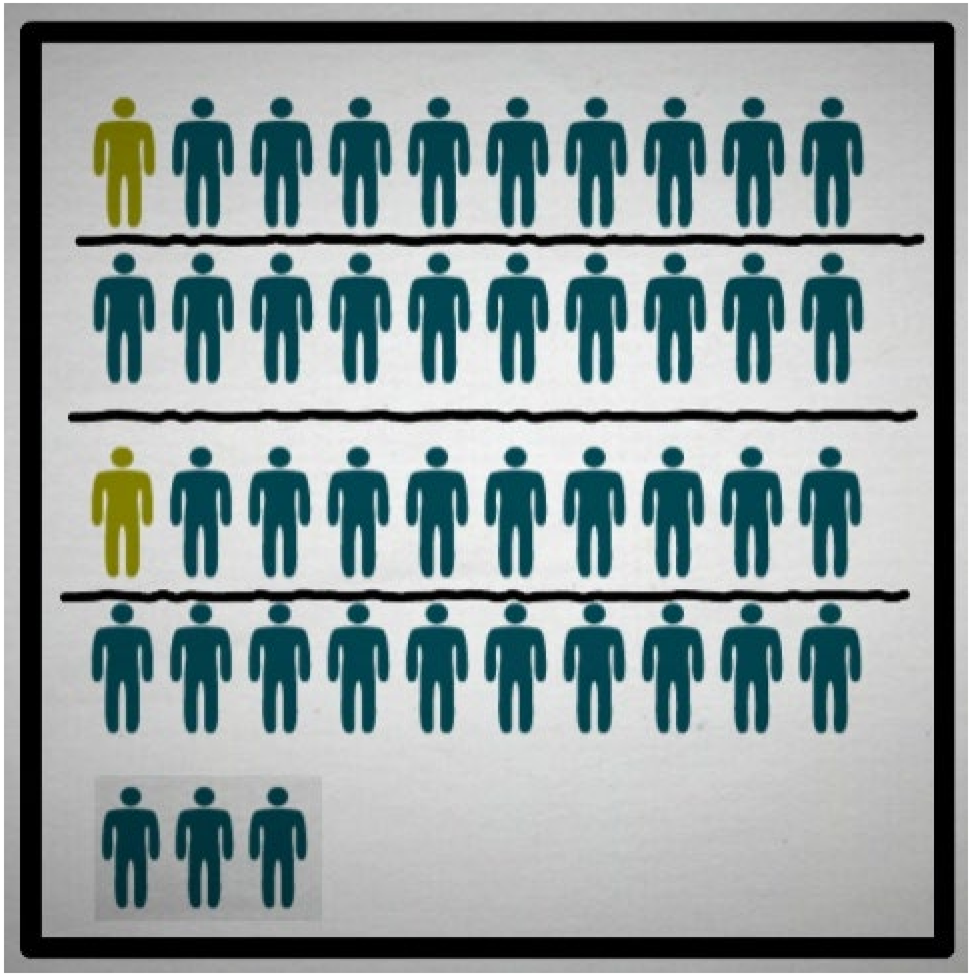


算法2-多个枚举法

- 如何避免每个组内的数数?



算法3-分组法

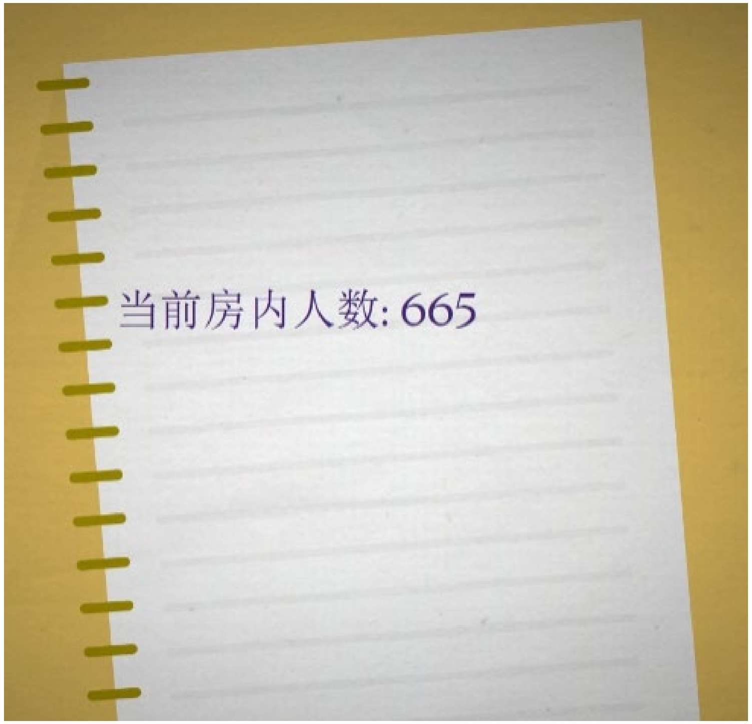


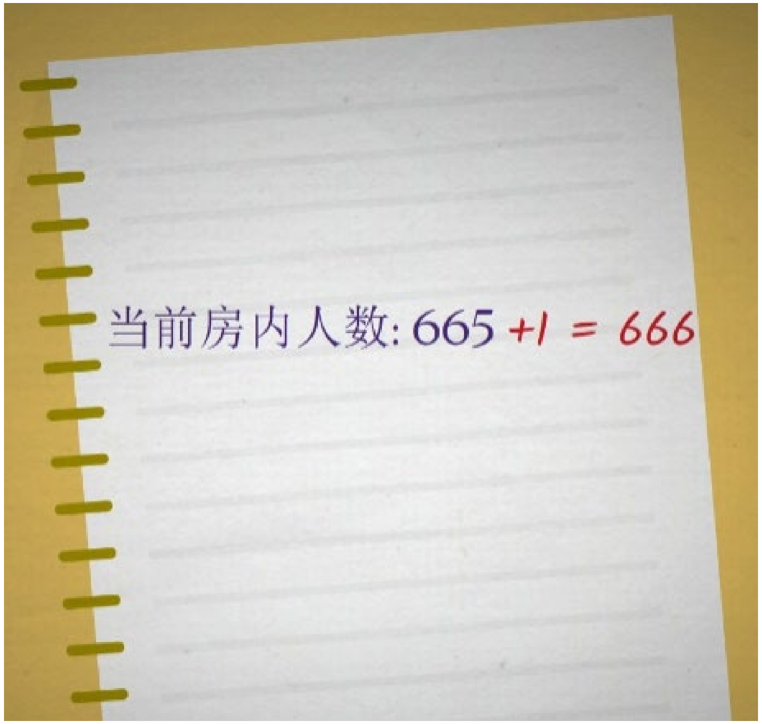
算法4-?

- 是否还有更加高效的方法?

算法4-计数法

- 维护一个记事本





算法与程序

算法

程序

可计算否

可计算性理论（哪些问题能被解决？哪些问题不能被解决）

能行可计算否

计算复杂性理论（一个问题怎样才能被有效的解决）

算法设计与分析

可计算问题的算法的设计与分析

用计算机语言实现算法

软件系统

解决一个计算问题的过程

Algorithms are the heart of computer science, **and the subject has countless practical applications as well as intellectual depth**-斯坦福大学计科专业

算法的重要性

- 生活中到处是算法

搜索算法

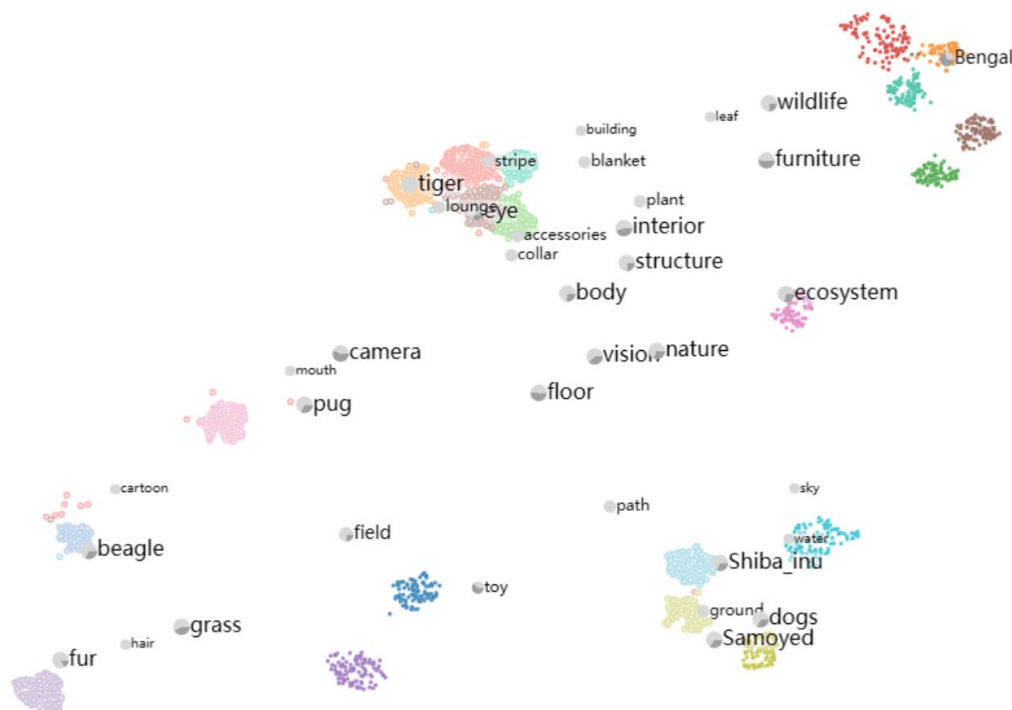
推荐算法

路径规划算法

…

算法的重要性

- 算法很实用
  - 两类数据：图像、文本
  - 如何映射到二维平面，并保持他们的相似性？
    - 例如：哈士奇和阿拉斯加



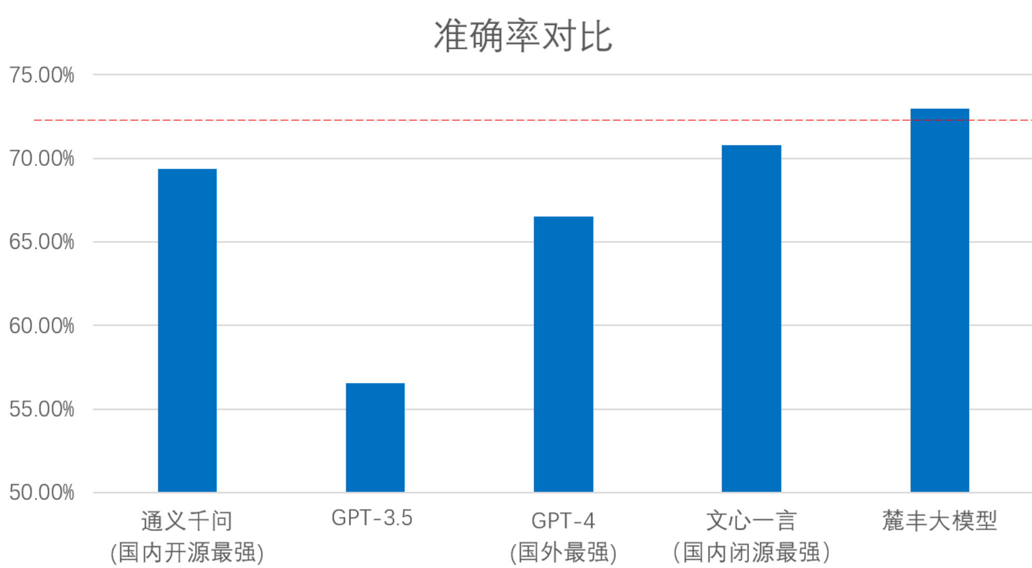
CCF-A类会议投稿，余胜杰（21级，国奖），周之翔（21级，国奖）

算法的重要性

- 种业大模型

选取了国内外具有代表性的大模型进行对比

- **GPT3.5, GPT4 (国外最强）**
- **通义千问（国内开源最强）**

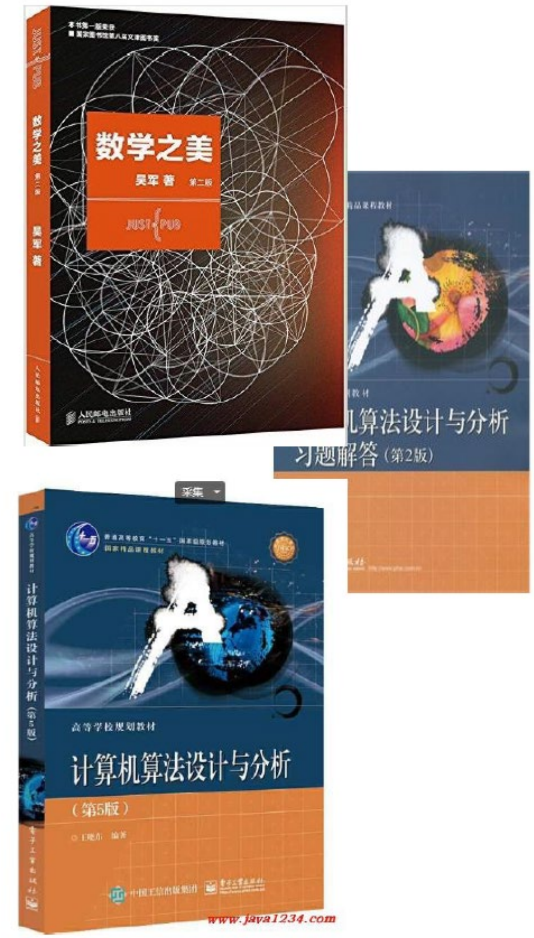


在近千道选择题测试下，**麓丰大模型**要好过国内外最优的大模型

梅炳寅（计科21，第一名）

课程教材

- 教材、资料
  - 《计算机算法设计与分析（第5版）》王晓东 编著, 出版社：电子工业出版社, 2018.
  - 《计算机算法设计与分析习题解答（第3版》
  - 《数学之美(第2版)》 吴军
  - Wikipedia
  - Google



MOOC资源推荐

- 北方交通大学 李清勇
  - http://www.icourse163.org/learn/NJTU-1003359012?tid=1206796214#/learn/content?type=detail&id=1211656960&cid=1214478516
- 北京航空航天大学 童咏昕
  - https://www.icourse163.org/course/BUAA-1449777166?utm_source=weixin&utm_medium=iphoneShare&utm_campaign=share
- 青岛大学 李劲华等
  - https://mooc1.chaoxing.com/course/236563000.html

关于成绩

- 平时成绩（含考勤） - 20%
  - 课堂参与、小班讨论、平时作业
- 实验成绩 （含考勤）- 30%
  - 实验准备、实验执行、实验总结报告
- 期中考试 - 15%
- 期末考试 - 35%

小班讨论课安排

- 讨论主题分为3类：
  - 算法分析练习
  - 算法实现练习
  - 《数学之美》等课外资料阅读
- 分组选题（每班分6组）
  - 每组选择1位组长，实行组长负责制
  - 每次讨论课，每组选1类题，即每次课参与讨论分析题、实现题与《 数学之美》三类题的各有2组
  - 每组在2次讨论课中，必须轮流选择3类题目

关于小班讨论

以组为单位,每组一题,每组人人参与,合理分工,ppt中标记分工,尽量都有代码演示,第8周上台报告

- 主题三分
  - 算法分析题 2-10、2-15(要求：有ppt (选:代码演示))
  - 算法实现题 2-4、2-5(要求：有ppt和代码演示讲解)
  - 数学之美分主题 2个(要求：有ppt)

    （1） P89 第9章 图论和网络爬虫

    （2）P249 第29章 各个击破算法和Google云计算的基础

每组讲解时间：10-12分钟+3-5分钟提问讨论

额外奖励（Bonus）

- 小班讨论
  - 一等奖（1组）：键盘 IKBC W200
  - 二等奖（2组）：鼠标（罗技M330）
  - 三等奖（3组）： 32G U盘

- 课堂回答问题奖励（5位）： 小礼物
- 平时作业优秀奖（5位）： 小礼物

关于实验

- 主题三分
  - 离线测试题 (要求：有三种规模测试数据和实验报告)
  - 在线测试题（系统自动判断）

- 总计4次实验
- 验收时间：在第7周和第13周（暂定）
  - 需要与各位班长协调时间

第1章 算法引论

- 算法与程序
  - 什么是算法?
  - 算法与程序的关系
  - 什么时候需要算法?
- 算法复杂性分析
  - 什么是一个好算法?
  - 计算复杂度概念
  - 渐进复杂性的数学表述

第1章 算法引论

- 算法与程序
  - 什么是算法?
  - 算法与程序的关系
  - 什么时候需要算法?
- 算法复杂性分析
  - 什么是一个好算法?
  - 计算复杂度概念
  - 渐进复杂性的数学表述

算法的定义

- 算法是满足下述性质的指令序列
  - 输　入：有零个或多个外部量作为算法的输入。
  - 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清晰、无歧义。
  - 有限性：算法中每条指令的执行次数有限，执行每条指令的时间也有限。

程序（Program）

- 程序是算法用某种程序设计语言的具体实现。
- 程序可以不满足算法的性质(4)。
  - 例如操作系统，是一个在无限循环中执行的程序，因而不是一个算法。操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

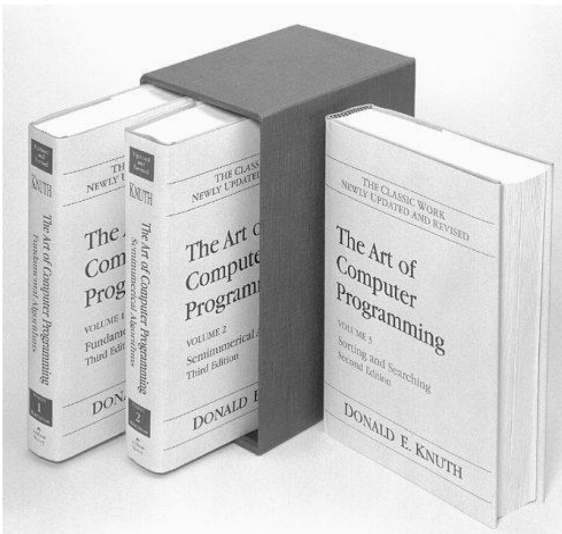
算法和程序

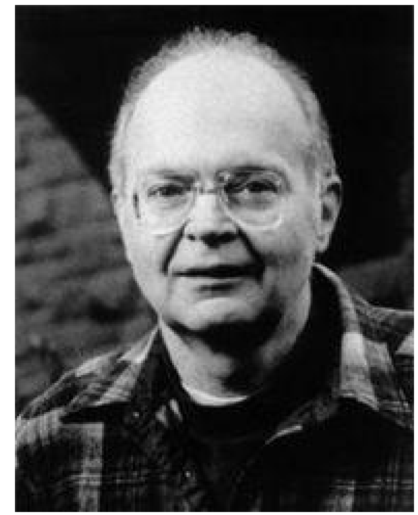
- 程序=算法+数据结构

主要贡献：
提出了结构化程序设计的思想
（逐步细化，或称为逐步求精，
stepwise refinement)

1984年图灵奖得主：
瑞士的Niklaus Wirth

算法和程序





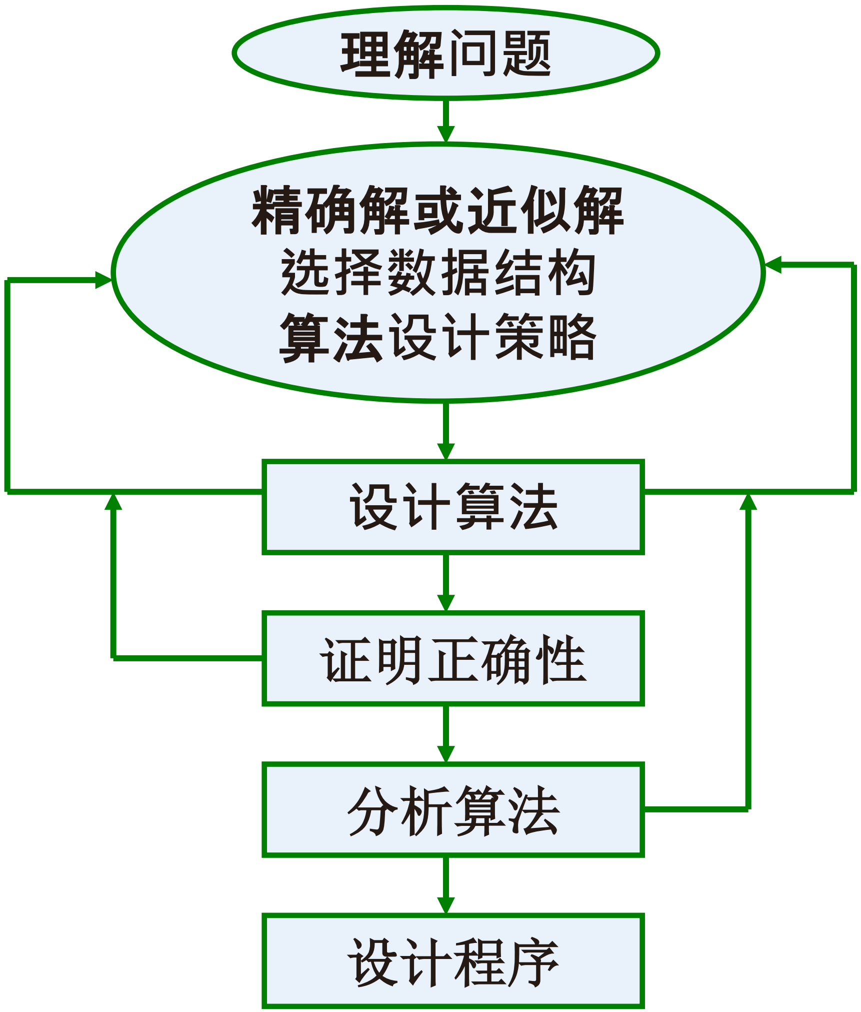
在算法分析和程序语言设计领域的杰出贡献，特别是其著名的“The Art of Computer Programming"系列丛书。

1974年图灵奖得主：美国的Donald E. Knuth（高德纳）

When do we need algorithm

Everything, Everytime, Everywhere

问题求解（Problem Solving）



第1章 算法引论

- 算法与程序
  - 什么是算法?
  - 算法与程序的关系
  - 什么时候需要算法?
- 算法复杂性分析
  - 什么是一个好算法?
  - 计算复杂度概念
  - 渐进复杂性的数学表述

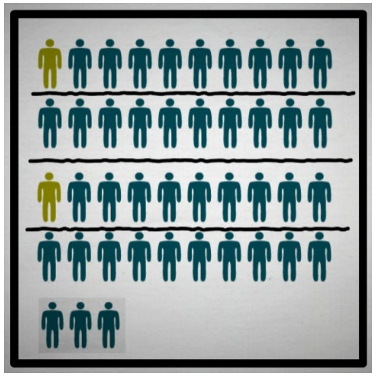
什么是一个好的算法?

- 以数人数为例

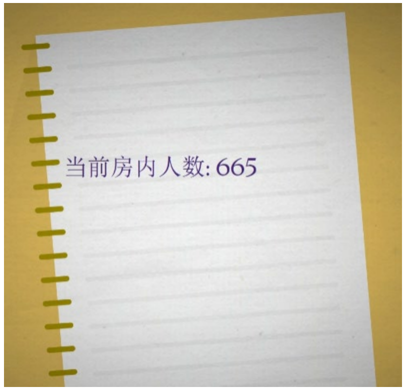
单个枚举法

多个枚举法

分组法



计数法



什么是一个好的算法?

- 什么是好算法?
  - Edmonds1975年提出了一个被沿用至今的标准。
  - Edmonds算法标准指出具有多项式时间的算法为好算法。
  - 多项式时间算法：如果Q是任意一个问题，对Q存在着一个算法，它的时间复杂性为$O(n^k)$,其中n为输入规模，k为非负整数，就认为存在着一个解问题Q的多项式时间算法。多项式时间算法的可实现性远大于指数时间算法。
- 好的算法
  - 提高求解问题的效率
  - 节省存储空间

提高算法质量

- 质量：正确性、可靠性、健壮性、可读性
- 请看两组操作：
- (1) a:=a+b； b:=a-b；a:=a-b；
- (2) t:=a；   a:=b；   b:=t；

功能： "交换变量a、b中的数据"

第一组操作节省了一个存储空间，但失去了可读性。

算法的正确性

- 定义：(算法正确性)
  - 一个算法是正确的，如果它对于每一个输入都最终停止，而且产生正确的输出。
  - 不正确算法：
    - ①不停止(在某个输入上)
    - ②对所有输入都停止，但对某输入产生不正确结果
  - 近似算法
    - ①对所有输入都停止
    - ②产生近似正确的解或产生不多的不正确解

如何描述算法的好坏?

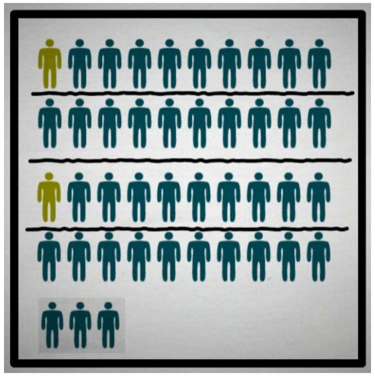
- 算法复杂度分析
  - 算法复杂性是算法运行所需要的计算机资源的量
  - 需要时间资源的量称为**时间复杂性 T(n)**，需要的空间资源的量称为**空间复杂性 S(n)**。n是问题规模。
  - 这个量应该只依赖于算法要解的问题的规模、算法的输入和算法本身的函数。如果分别用N、I和A表示算法要解问题的规模、算法的输入和算法本身，而且用C表示复杂性，那么，应该有C=F(N,I,A)。一般把时间复杂性和空间复杂性分开，并分别用T和S来 表示，则有：T=T(N,I,A)和S=S(N,I,A) 。

空间复杂度

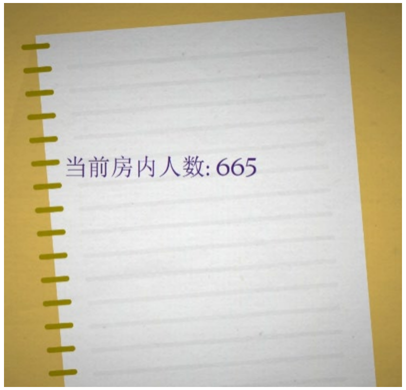
单个枚举法

多个枚举法

分组法



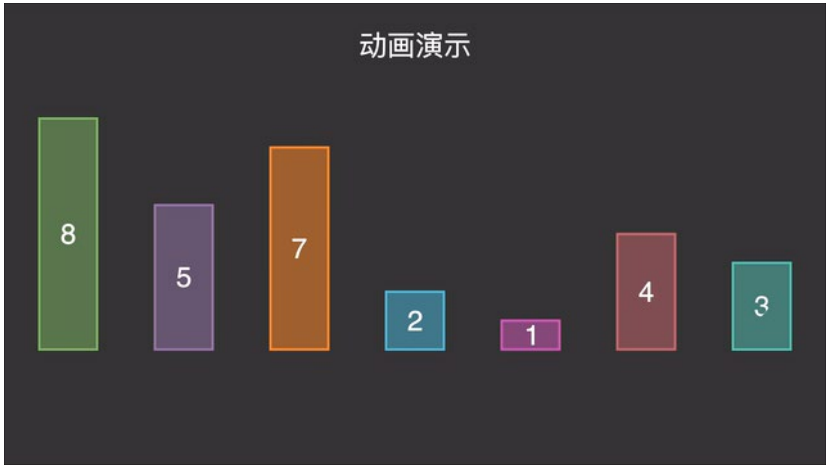
计数法



空间复杂度

- 冒泡排序

```
void BubbleSort(int* a, int n)
{
    assert(a);
    for (size_t end = n; end > 0; --end)
    {
        int exchange = 0;
        for (size_t i = 1; i < end; ++i)
        {
         if (a[i-1] > a[i])
         {
                Swap(&a[i-1], &a[i]);
                exchange = 1;
         }
       }
    if (exchange == 0)
      break;
    }
}
```



$S(n)=1$

单选题 1分

设置

上述冒泡排序算法空间复杂度是多少?

A 1

B 2

C n

D n^2

提交

空间复杂度

- 例：Fibonacci数列
- 无穷数列1，1，2，3，5，8，13，21，34，55，……，称为Fibonacci数列。它可以递归地定义为：

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

空间复杂度

```
// 计算Fibonacci的空间复杂度?
// 返回斐波那契数列的前n项
long long* Fibonacci(size_t n)
{
	if(n==0)
	return NULL;

	long long * fibArray = (long long *)malloc((n+1) * sizeof(long long));
	fibArray[0] = 0;
	fibArray[1] = 1;
	for (int i = 2; i <= n ; ++i)
	{
	 fibArray[i] = fibArray[i - 1] + fibArray [i - 2];
	}
	return fibArray;
}
```

$$S(n)=n$$

时间复杂度

- （1）**最坏情况**下的时间复杂性
  - $T_{max}(n)$ = max{ T(I) | size(I)=n }
- （2）**最好情况**下的时间复杂性
  - $T_{min}(n)$ = min{ T(I) | size(I)=n }
- （3）**平均情况**下的时间复杂性
  - $T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$
  - 其中I是问题的规模为n的实例，p(I)是实例I出现的概率。

时间复杂度

- 算法渐近复杂性

$T(n)\rightarrow\infty$， as $n\rightarrow\infty$；

$(T(n) - t(n))/T(n)\rightarrow 0$， as $n\rightarrow\infty$;

$t(n)$是$T(n)$的渐近性态，为算法的渐近复杂性。

- 在数学上， $t(n)$是$T(n)$的**渐近表达式**，是$T(n)$略去低阶项留下的主项。它比$T(n)$ 简单。
- 例子： $T(n)=n^2+n$, $t(n)=n^2$

渐近分析的记号

在下面的讨论中，对所有n，$f(n) \geq 0$，$g(n) \geq 0$。

- （1）**渐近上界记号O**

阶越低、评估越准确、价值越大

  – $O(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0，使得对所有$n \geq n_0$有：

  $0 \leq f(n) \leq cg(n) \}$

- （2）**非紧上界记号o**

  – $o(g(n)) = \{ f(n) \mid$ 对于任何正常数c和n_0，使得对所有$n \geq n_0$有：$0 \leq f(n) < cg(n) \}$

  – 等价于 $f(n) / g(n) \to 0$，as $n \to \infty$。

渐近分析的记号

- （3）**渐近下界记号Ω**
  - $\Omega(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$有：

  $0 \leq cg(n) \leq f(n) \}$
- （4）**非紧下界记号ω**
  - $\omega(g(n)) = \{ f(n) \mid$ 对于任何正常数$c>0$，存在正数和$n_0>0$使得对所有$n \geq n_0$有：$0 \leq cg(n) < f(n) \}$
  - 等价于 $f(n) / g(n) \to \infty$，as $n \to \infty$。
  - $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

渐近分析的记号

- （5）**紧渐近界记号Θ**
  - $\Theta(g(n)) = \{ f(n) \mid$ 存在正常数c_1, c_2和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：** $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

例子

- $g(n)=n^2+2n$
  - $O(g(n)) = n^2, n^3, \ldots$
  - $\Omega\ (g(n)) = 1, n, \ldots$
  - $\Theta\ (g(n)) = n^2$

渐近分析中的函数比较

- $f(n) = O(g(n)) \approx a \le b;$
- $f(n) = o(g(n)) \approx a < b;$
- $f(n) = \Theta(g(n)) \approx a = b;$
- $f(n) = \Omega(g(n)) \approx a \ge b;$
- $f(n) = \omega(g(n)) \approx a > b.$

算法分析中常见的复杂性函数

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	$N \log N$
N^2	Quadratic
N^3	Cubic
2^N	Exponential

算法复杂度分析

- 多项式函数与指数函数

时间复杂度函数	问题规模					
	10	20	30	40	50	60
n	10^{-5}	$2*10^{-5}$	$3*10^{-5}$	$4*10^{-5}$	$5*10^{-5}$	$6*10^{-5}$
n^2	10^{-4}	$4*10^{-4}$	$9*10^{-4}$	$16*10^{-4}$	$25*10^{-4}$	$36*10^{-4}$
n^3	10^{-3}	$8*10^{-3}$	$27*10^{-3}$	$64*10^{-3}$	$125*10^{-3}$	$216*10^{-3}$
n^5	10^{-1}	3.2	24.3	1.7 分	5.2 分	13.0 分
2^n	.001 秒	1.0 秒	17.9 分	12.7 天	35.7 年	366 世纪
3^n	.059 秒	58 分	6.5 年	3855 世纪	$2*10^8$ 世纪	$1.3*10^{13}$ 世纪

算法分析的基本法则

非递归算法:

- 顺序语句
  - 各语句计算时间相加;
- if-else语句
  - if语句计算时间和else语句计算时间的较大者;
- for / while 循环
  - 循环体内计算时间*循环次数;
- 嵌套循环
  - 循环体内计算时间*所有循环次数。

算法分析方法-顺序搜索举例

- 顺序搜索算法

```
template<class Type>
int seqSearch(Type *a, int n, Type k)
{
    for(int i=0;i<n;i++)
        if (a[i]==k) return i;
    return -1;
}
```

算法分析方法-顺序搜索举例

（1） $T_{max}(n)$ = max{ T(I) | size(I)= n }= $O(n)$

（2） $T_{min}(n)$ = min{ T(I) | size(I)= n }= $O(1)$

（3）在平均情况下，假设：

(a) 搜索成功的概率为 p ($0 \le p \le 1$);

(b) 在数组的每个位置 i ($0 \le i < n$)搜索成功的概率相同，均为 p / n。

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

$$= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}\right) + n \cdot (1 - p)$$

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

算法分析方法-插入排序举例

```
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 og (ti-1)
         }
      a[j+1]=key;                     //   c7        n-1
      }
}
```

5 3 4 7 2

Insertion Sort

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

算法分析方法-插入排序举例

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

- 在最好情况下， $t_i=1$, for $1 \le i < n$;

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

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

- 在最坏情况下， $t_i \le i+1$, for $1 \le i < n$;

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

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

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

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

$$= O(n^2)$$

最优算法

- 问题的计算时间下界为Ω(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Ω(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。

问题复杂度 VS 算法复杂度

- 举例：Fibonacci 递归算法

```
int fibonacci(int n)
{
if (n <= 1) return 1;
return fibonacci(n-1)+fibonacci(n-2);
}
```

- Lower bound: $\Omega(2^{(n/2)})$
- Upper bound: $O(2^n)$

问题复杂度 VS 算法复杂度

- 举例：Fibonacci 非递归算法

```
// 计算Fibonacci的空间复杂度?
// 返回斐波那契数列的前n项
long long* Fibonacci(size_t n)
{
     if(n==0)
     return NULL;

     long long * fibArray = (long long *)malloc((n+1) * sizeof(long long));
     fibArray[0] = 0;
     fibArray[1] = 1;
     for (int i = 2; i <= n ; ++i)
     {
      fibArray[i] = fibArray[i - 1] + fibArray [i - 2];
     }
     return fibArray;
}
```

$T(n)=n$　　　　$S(n)=n$

NP完全性理论

- 是否每个问题都有多项式时间算法?
- 实际上可计算的问题
  - 多项式时间可解的问题

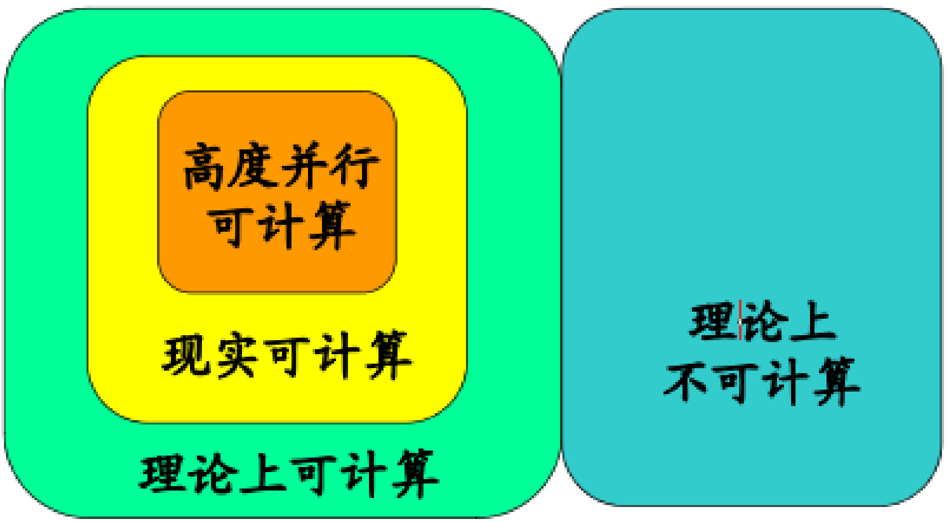
- NP=P?

问题求解-理解问题

- 理论上的可计算：可计算理论
- 研究目标：
  - 确定什么问题是可计算的，即存在求解算法
- 合理的计算模型
  - 已有的：递归函数、 **Turing**机、 λ演算、 **Post**系统、正则算法等
  - 条件： 计算一个函数只要有限条指令；每条指令可以由模型中的有限个计算步骤完成；指令执行的过程是确定的
- 可计算性是不依赖于计算模型的客观性质

问题求解-理解问题

- **理论上与现实上可计算性**



  - 算法至少具有指数时间： 理论上可计算——难解
  - 多项式时间的算法：现实上可计算——多项式时间可解
  - 对数多项式时间的算法： 高度并行可解

算法复杂性分析

问题的复杂度分类

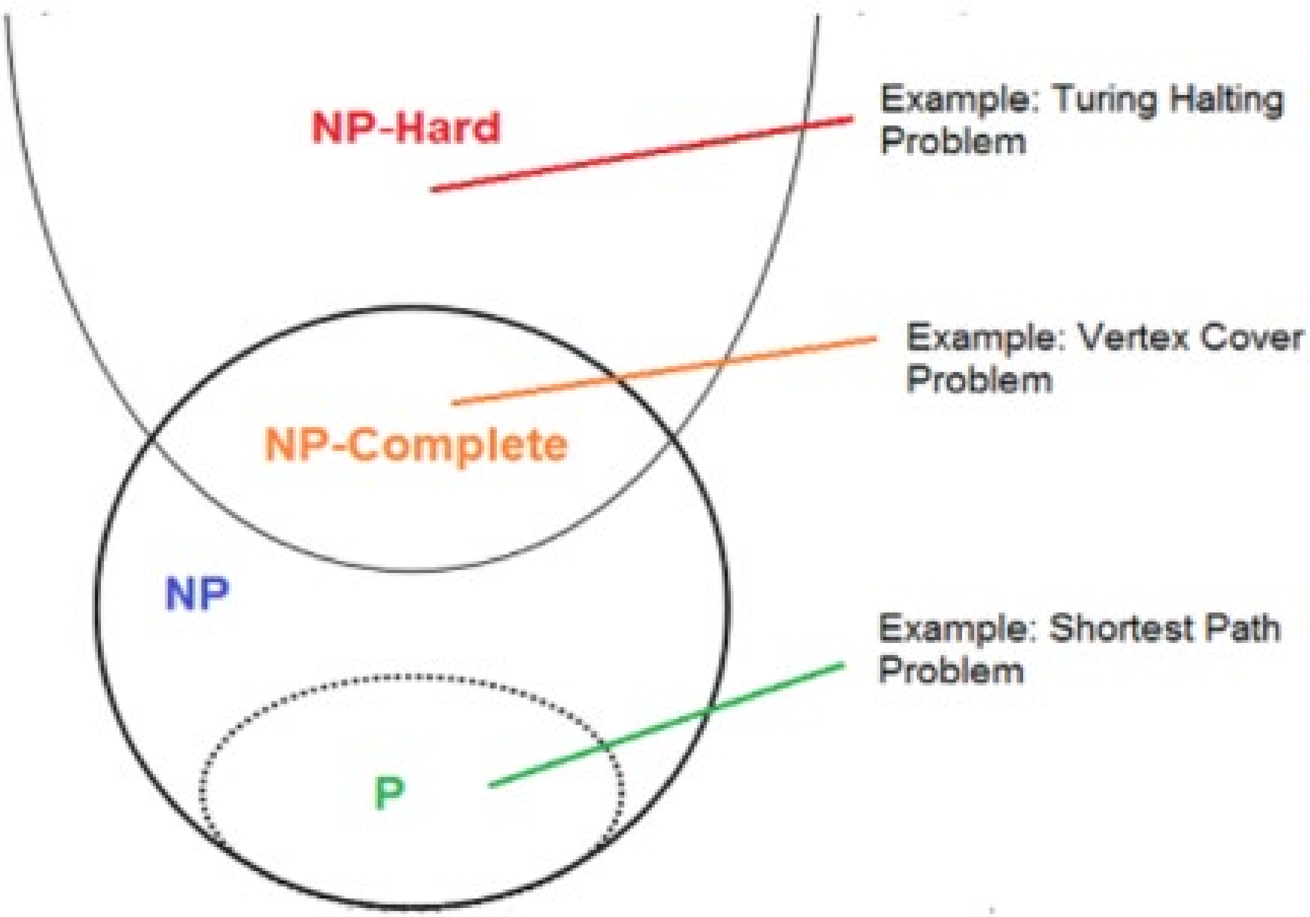
- *P*：多项式时间可解的问题
  - 存在着解*P*的多项式时间的算法
  - P = {L|L是一个能够在多项式时间内被一台确定性图灵机所接受的语言}
- NP：难解的问题*P*，Nondeterministic Polynomial
  - 不存在解*P*的多项式时间的算法
  - NP = {L|L是一个能够在多项式时间内被一台非确定性图灵机所接受的语言}

算法复杂性分析

- NP难问题：
  - NP难问题至少和NP问题一样难
- NP完全问题：
  - 这一类问题满足两个性质

1. 在多项式时间内可以验证一个候选答案是不是真正的解
2. 可把任何一个NP问题在多项式的时间内将其输入转化，使之成为一个NP完全问题（即**归约**）。

P、 NP、 NP完全的包容关系



This diagram assumes that P != NP

NP类问题

- 例：旅行商（TSP）：一个商品推销员要去若干个城市推销商品，该推销员从一个城市出发，需要经过所有城市后，回到出发地。应如何选择行进路线，以使总的行程最短。

NP类问题 – 集合覆盖

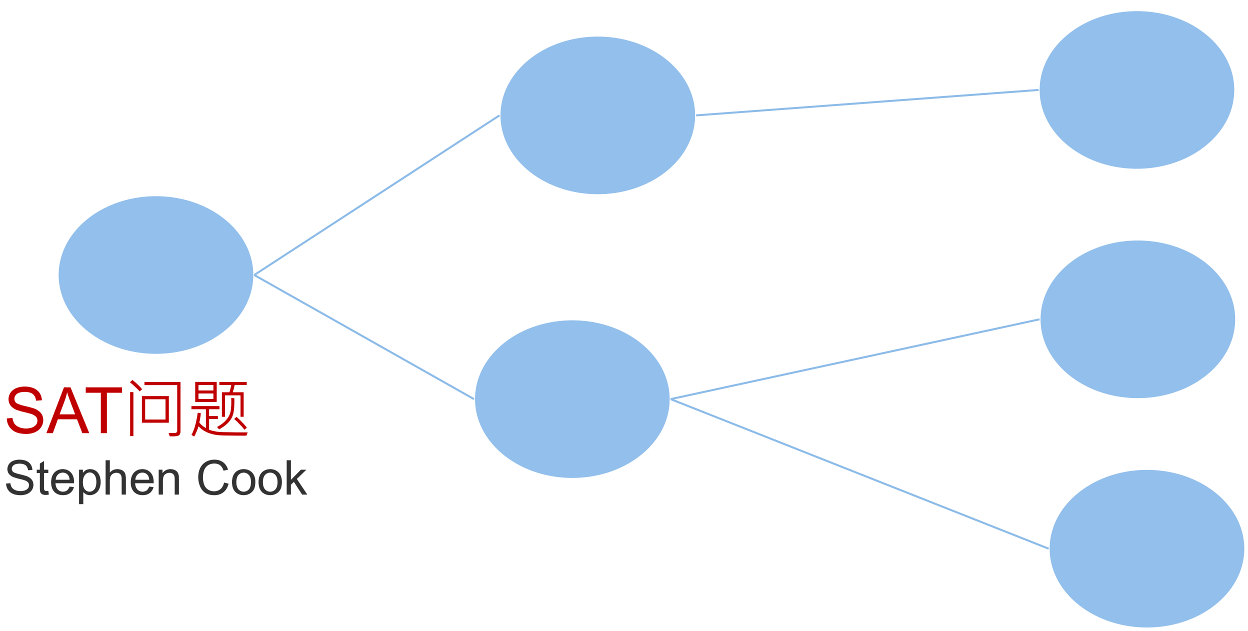
问题描述：假设我们有个全集U (Universal Set), 以及m个子集合 $S_1, S_2, \ldots, S_m$ ，目标是要寻找最少的集合，使得集合的union等于U．例子: U={1,2,3,4,5}，S : { S_1 ={1,2,3}, S_2 ={2,4}, S_3 ={1,3}, S_4 ={4}， S_5 ={3,4}, S_6 ={4,5} }，最少的集合为 : {1,2,3}, {4,5}，集合个数为2.

如何证明NPC

- 问题A归约问题B （问题A不难于问题B）：
  - 将问题A的输入变换为问题B的适当输入
  - 求解问题B
  - 把问题B的输出变换为问题A的正确解
- 例子：
  - 问题A: ax+c=0
  - 问题B: $\begin{cases} ax+by+c=0 \\ dx+ey+f=0 \end{cases}$ $x=\dfrac{c\times e-b\times f}{b\times d-a\times e}$ $y=\dfrac{a\times f-c\times d}{b\times d-a\times e}$

NPC问题树

- 证明一个问题Q是NPC问题
  - 证明Q是NP问题
  - 找到一个已知NPC问题，证明可以归约到Q



NP完全问题的研究现状

- 已发现3000多个NP完全问题
- 2000年5月24日，美国克雷数学研究所在巴黎法兰西学院宣布了七个“千年数学难题”，解决其中一个可获百万美元奖励。

谢谢