算法与计算理论

课程内容

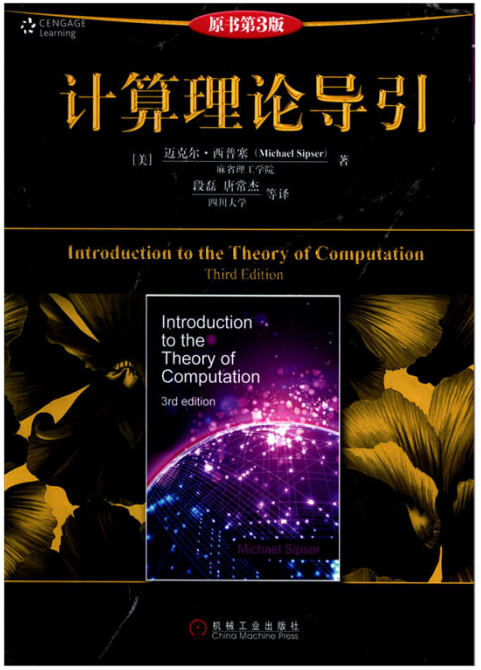
数据结构

概述	数组与广义表	图	外部排序
线性表	串	查找	
栈与队列	树	内部排序	

算法与计算理论

概述	贪心	……	计算模型
分治	回溯	……	可计算理论
动态规划	……	……	计算复杂性

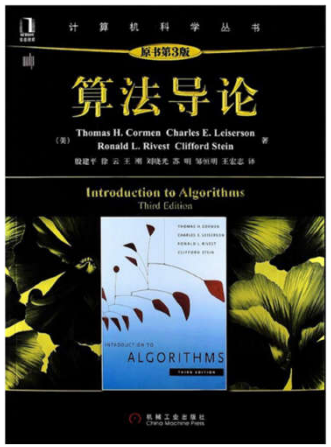
教材



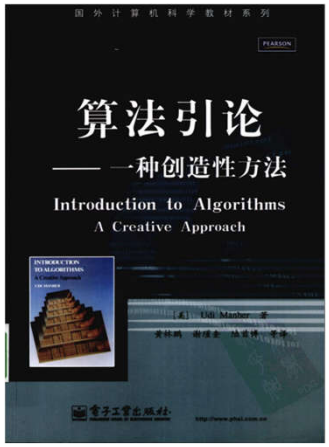
王晓东,计算机算法设计与分析(第5版),电子工业出版社.

段磊,唐常杰等译, Sipser著, 计算理论导引, 机械工业.出版社

参考资料

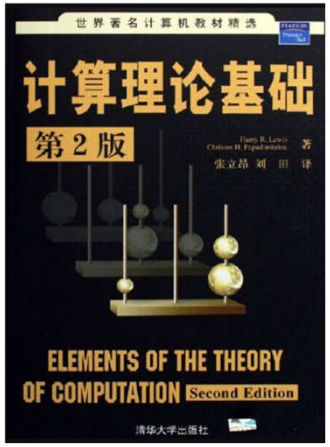


潘金贵等译, Cormen等著, 算法导论, 机械工业出版社



黄林鹏等译, Manber著, 算法引论-一种创造性方法, 电子工业出版社

刘汝佳等, 算法艺术与信息学竞赛, 清华大学出版社



Lewis等著, 计算理论基础, 清华大学出版社

Contents

本章内容

什么是算法

算法复杂性分析

P问题与NP完全问题

算法分析中的数学基础

什么是算法

算法（Knuth）：是一组有穷规则的集合，这些规则给出了求解特点类型问题的运算序列，具有重要的5个特征:

(1) 输入(Input): 零个或多个外部提供的量

(2) 输出(Output): 产生至少一个量作为输出

(3) 确定性(Definiteness): 每条指令是清晰确定的, 无歧义

(4) 有穷性(Finiteness): 每条指令的执行次数, 时间有限

(5) 有效性(Effectiveness,能行性)： 算法中的所有运算或操作必须充分基本，原则上可以由人用笔或纸在有限的时间里准确地完成。

一个满足算法其他特征但不满足有穷性的过程，可以称之为计算方法。

什么是算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

算法的有效性（能行性）

从计算机的角度上看，无法实现的算法步骤，例如：除0操作。
从数学角度上看，可以实现，但计算机无法实现的；如算珠排序。

什么是算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

算法的正确性

从计算机角度上看，给定每一组输入，都能得到正确的结果，可看作算法正确。
但计算机跑出来的正确结果，就是正确的吗？

从数学角度上看，经过数学证明的算法，才是正确的。

不正确的算法包括：
算法不停止；给出错误的结果；可能是有用的；

什么是算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

算法导论[C]:

解决可计算问题的一个工具

将输入转换为输出的一个可计算步骤序列

韦氏大学词典:

求解问题的一个过程, 步骤有限, 通常有重复操作;
广义地说, 是按部就班解决一个问题的过程.

什么是算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

这些都是算法的直观解释, 包含了严格定义的所有要素

找最大数问题

输入: 一个实数序列a1,…,an.

输出: max { a1,…,an }.

输入样例: 1.1, 2.3, 4, 10, 8, 7, 3, 8
对应输出: 10

算法:

1. 令max=a1.
2. 对 i = 2 到 n,
3. 　若 max > ai , 则 max = ai .
4. 输出max.

输入,输出

确定性

有限性

可计算步骤

一个过程

有重复操作

无二义性

算法的正确性证明

正确的算法：

对任意一个输入，算法都能得到一个正确的输出

循环不变量：

与程序变量有关的一组性质，它在循环刚开始前为真，在循环的每个迭代执行后为真，特别是在循环结束后，仍然为真。

算法的正确性证明

利用循环不变量证明算法的正确性

找到循环不变量，即某个特性L_j，然后证明循环不变量为真（j=1.....n)，然后利用类似数学归纳法的证明方法：

1.初始步：在循环的迭代开始前，L_1为真

2.归纳步：如果在循环的第$_i$个迭代之前，L_{i-1}为真，则在循环第j_{i+1}个迭代之前，L_i为真；

3.终止步：当循环终止时，L_n为真

算法的正确性证明

证明插入排序算法的正确性

```
for (i=1;i<n;i++){
        key = a[i]
        j = i-1
        while(j>=0 && a[j] > key) {
                a[j+1] = a[j]；  j--;
        }
        a[j+1] = key
}
```

算法的其他特征

正确性：针对任何合法的输入，都应该得到正确的结果。

可读性：算法容易被人理解和实现。晦涩难懂的算法可能存在一些不易发现的错误。

健壮性：对非法的输入应当能识别并作出处理。

通用性：能够处理泛化问题，并便于扩展，便于移植，避免过度依赖硬件要素。

高效性：应该具有较短的执行时间和占用较少的辅助空间。

效率是算法的灵魂

Contents

本章内容

什么是算法

算法复杂性分析

P问题与NP完全问题

算法分析中的数学基础

关于算法复杂度的度量

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

如何计算算法的复杂度？ 主要考虑三方面的要素: 输入, 时间, 空间

时间和空间复杂度, F(N,I,A) 规模, 输入, 算法

T(N,I,A) 简化为 T(N,I) ， 进一步将形式简化为T(N)

最好时间复杂度， 最坏时间复杂度, 平均时间复杂度

输入有很多不同形式:数列, 图, 大数等?

需要约定输入规模的度量

系统不同、缓存大小不同等 诸多影响因素?

需要约定计数哪些步

如何计算问题的复杂度?

关于算法复杂度的度量

构造简单的RAM计算模型

（假定一种通用的单任务处理器，Random Access Machine）

只包含真实计算机中最基本的指令：

算术指令（加、减、乘、除、取余，向上取整，向下取整）：

逻辑运算指令（与或非）：

数据移动指令（装入、存储、复制）：

程序控制指令（条件转移、非条件转移，函数调用，函数返回）），这些指令所需要的时间都为常数单位。

限定整数、浮点数的字长为常数单位；

忽略硬件特性；

插入排序算法分析过程

	cost	times
for (i=1;i<n;i++){	c1	n-1
key = a[i]	c2	n-1
j = i-1	c3	n-1
while(j>=0 && a[j] > key) {	c4	$\sum_{k=0}^{i-1} t_j$
a[j+1] = a[j];	c5	$\sum_{k=0}^{i-1} t_j - 1$
j--;	c6	$\sum_{k=0}^{i-1} t_j - 1$
}		
a[j+1] = key	c7	n-1
}		

插入排序算法分析过程

	cost	times
最好情况下：输入数据已经有序。	c1	n-1
T(n) = c1(n-1)+c2(n-1)+c3(n-1)	c2	n-1
+c4(n-1)+c7(n-1)	c3	n-1
= (c1+c2+c3+c4+c7)n-	c4	$\sum_{k=0}^{i-1} t_j$
(c1+c2+c3+c4+c7)	c5	$\sum_{k=0}^{i-1} t_j - 1$
	c6	$\sum_{k=0}^{i-1} t_j - 1$
T(n) = an+b (a,b是由c_i确定的常数)		
T(n)是n的线性函数；	c7	n-1

插入排序算法分析过程

	cost	times
最坏情况下：输入数据倒序排列。	c1	n-1
T(n) = c1(n-1)+c2(n-1)+c3(n-1)	c2	n-1
+c4(n(n+1)/2)-1)+c5(n(n-1)/2)	c3	n-1
+c6(n(n-1)/2)+c7(n-1)	c4	$\sum_{j=0}^{i-1} t_j$
= (c4/2+c5/2+c6/2)n^2	c5	$\sum_{j=0}^{i-1} t_j - 1$
+(c1+c2+c3+c4/2-c5/2-c6/2)n	c6	$\sum_{j=0}^{i-1} t_j - 1$
-(c1+c2+c3)		
T(n) = an^2+bn+c (a,b,c是由c_i确定的常数)	c7	n-1

T(n)是n的一元二次函数;

找最大数问题

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: max $\{ a_1,\ldots,a_n \}$.

序列	3	2	5	7	2	9	1	8
最大	3	3	5	7	7	9	9	9
修改次数	1	0	1	1	0	1	0	0
比较次数	0	1	1	1	1	1	1	1

输入样例 I = {3,2,5,7,2,9,1,8}

约定规模为序列中数的个数: N = 8

以比较次数计算时间复杂度比较合理

时间复杂度: T(N) = N-1

找最大次大数问题([M]): 方法一

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3						
次大		2						
比较次数		1						

输入规模 N = 8, I = {3,2,5,7,2,9,1,8}

约定比较次数为时间复杂度

找最大次大数问题([M]): 方法一

- 输入: 一个实数序列$a_1,\ldots,a_n$.
- 输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3	5					
次大		2	3					
比较次数		1	1					

- 输入规模 N = 8, I = {3,2,5,7,2,9,1,8}
- 约定比较次数为时间复杂度

找最大次大数问题([M]): 方法一

- 输入: 一个实数序列$a_1,\ldots,a_n$.
- 输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3	5	7				
次大		2	3	5				
比较次数		1	1	1				

- 输入规模 N = 8, I = {3,2,5,7,2,9,1,8}
- 约定比较次数为时间复杂度

找最大次大数问题([M]): 方法一

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3	5	7	7			
次大		2	3	5	5			
比较次数		1	1	1	2			

输入规模 N = 8, I = {3,2,5,7,2,9,1,8}

约定比较次数为时间复杂度

找最大次大数问题([M]): 方法一

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: 最大和次大的数.

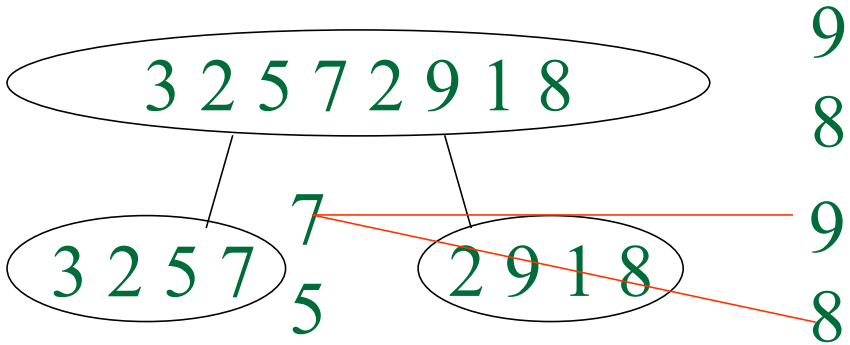
序列	3	2	5	7	2	9	1	8
最大		3	5	7	7	9	9	9
次大		2	3	5	5	7	7	8
比较次数		1	1	1	2	1	2	2

输入规模 N = 8, I = {3,2,5,7,2,9,1,8}

对所有规模N输入, N-1 ≤ T(N) ≤ 2(N-2)+1=2N-3 ?

能否进一步改进?

最大次大([M]): 方法二-分治



- 分治法: 设规模n比较最多T(n)次
  T(n) = 2T(n/2) + 2,
  T(2)=1, T(3)=3
- 若$n=2^k$, T(n)=1.5n-2; ([M])
- 若$n=3\cdot 2^k$, T(n)=5n/3-2
- 对一般n, 可归纳证明 $1.5n-2 \le T(n) \le 5n/3-2$

  证明见本ppt附录. 能否进一步改进?

$$
\begin{aligned}
T(2^k) &= 2T(2^{k-1})+2 \\
&= 2(2T(2^{k-2})+2)+2 \\
&= 2^2T(2^{k-2})+2^2+2 \\
&= 2^3T(2^{k-3})+2^3+2^2+2 \\
&= \dots \\
&= 2^{k-1}\,T(2)+2^{k-1}+\dots+2^2+2 \\
&= 2^{k-1}+2^{k-1}+\dots+2^2+2 \\
&= 3\times 2^{k-1}-2 \\
&= 3n/2-2
\end{aligned}
$$

找最大次大数问题([M]): 方法三

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3						
次大		2						
比较次数		1						

找最大次大数问题([M]): 方法三

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3	7	7				
次大		2	5	5				
比较次数		1	1	2				

找最大次大数问题([M]): 方法三

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3	7	7	9	9		
次大		2	5	5	2	7		
比较次数		1	1	2	1	2		

找最大次大数问题([M]): 方法三

输入: 一个实数序列$a_1,\ldots,a_n$.

输出: 最大和次大的数.

序列	3	2	5	7	2	9	1	8
最大		3	7	7	9	9	8	9
次大		2	5	5	2	7	1	8
比较次数		1	1	2	1	2	1	2

对所有规模n输入, 比较次数最多

$3\lceil(n-2)/2\rceil+1 = 3\lceil n/2\rceil - 2 \sim 1.5n-2$

注: [C]中对最大最小数问题使用了这个算法.

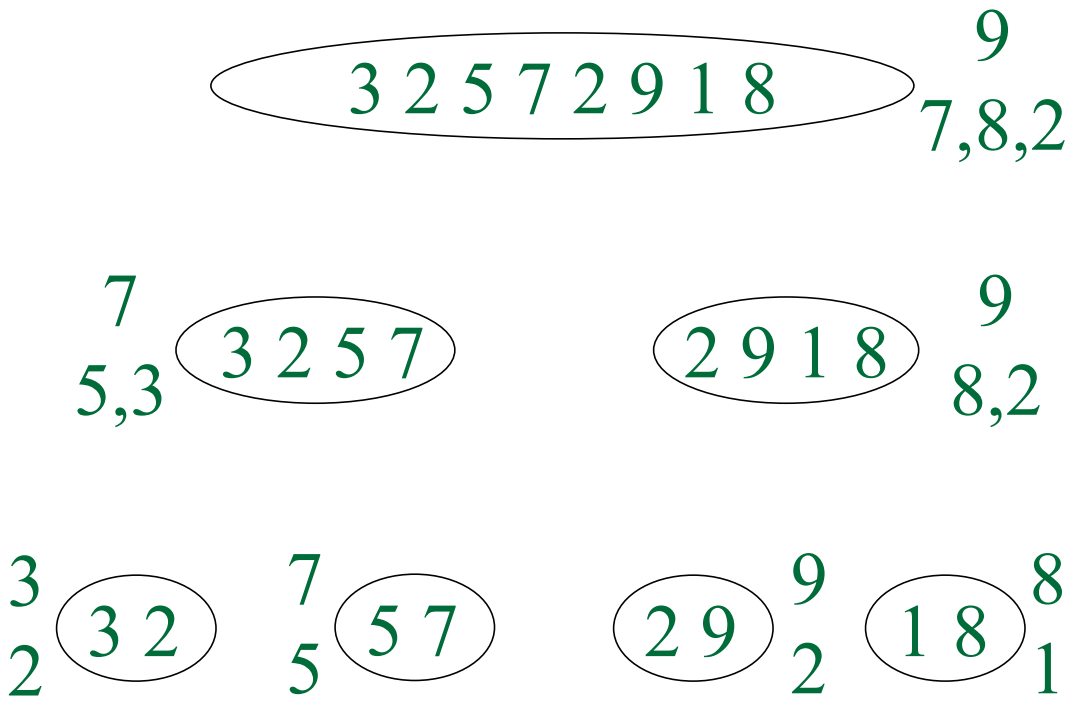
找最大次大数问题([M]): 方法四

修改的分治:

维护可能次大集

比较次数:

$n-1+\lfloor \log_2 n \rfloor$



斐波那契数列

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & if \quad n > 1 \\ 1 & if \quad n = 1 \\ 0 & if \quad n = 0 \end{cases}$$

斐波那契数列-算法1

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

算法1：

```
fib1(int n) {
        if(1==n) return 0;
        if(1==n) return 1;
        return fib(n-1)+fib(n-2);
}
```

问题

1：时间花费？

2：能不能改进？

斐波那契数列

- T(n):计算F(n)所需要的计算次数

$$T(n) \le 2 \quad n \le 1$$

$$T(n) = T(n-1) + T(n-2) + 3 \quad n > 1$$

$$T(200) \ge F(200) \ge 2^{138}$$

$$F(n) = 2^{0.694n} \approx 1.6^n$$

地球模拟器：太阳变成红巨星以后

IBM Roadrunner（2008年7月）快30倍，可以多算7个数

- 摩尔定律：每年计算速度增加1.6倍
  - 如果F(n)可以在一年内计算出来，下一年可以计算F(n+1)

为什么？

太多重复计算

斐波那契数列-算法2

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

算法2：

```
int fib2(int n) {
   int val,*array= (int*)malloc ((n+1)*sizeof(int));
   array[0]=0;
   array[1]=1;
   for(i=2;i<=n;i++)array[i]=array[i-1]+array[i-2];
   val= array[n];
   free(array);
   return val;
}
```

斐波那契数列-算法2

所花时间n,代价空间n

空间换时间或者时间换空间　　非常常见的策略

可以轻易地计算F_{200},F_{200000}和$F_{200,000,000}$

有没有更好的算法?

斐波那契数列-算法3

$a_1 = a_2 = 1$， $a_{n+2} = a_{n+1} + a_n \ (n \in N_+)$

整理得： $a_{n+2} - a_{n+1} - a_n = 0$

其特征方程为： $x^2 - x - 1$

解得： $\begin{cases} x_1 = \frac{1+\sqrt{5}}{2} \\ x_2 = \frac{1-\sqrt{5}}{2} \end{cases}$

$$f(n) = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

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

斐波那契数列-算法4

$$F_1 = F_0 \qquad F_2 = F_1 + F_0$$

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$F^2 = F * F$

$F^4 = F^2 * F^2$

$F^8 = F^4 * F^4$

将n表示为二进制形式，例如n = 13，其二进制形式为1101。

从右到左，对于每一位i，

如果该位为1，则将矩阵F乘以自身的2^i次方。

最终的结果矩阵为F^n，其中F(n)即为所求的斐波那契数列的第n项。

斐波那契数列

算法1：2^n

算法2：n

算法3：时间复杂度$2^{0.694n}$

算法4：时间复杂度logn

算法的效率

什么是高效的算法？运行速度和占用空间可以用来度量效率

输入规模n 耗费时间		1小时/天/年可解的问题实例的最大规模		
		计算机	快100倍计算机	快1000倍计算机
多项式时间	n	N_1	100 N_1	1000 N_1
	n^2	N_2	10 N_2	31.6 N_2
	n^3	N_3	4.64 N_3	10 N_3
	n^5	N_4	2.5 N_4	3.98 N_4
指数时间	2^n	N_5	N_5 + 6.64	N_5 + 9.97
	3^n	N_6	N_6 + 4.19	N_6 + 6.29

算法的效率分析

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

算法效率的分析是指：算法求解一个问题所需要的时间和空间

时间资源和空间资源

算法时间资源的估算

对算法执行基本运算（或步数）的数量分析

算法的效率分析

对算法时间资源的估算：
对算法执行基本运算（或步数）的数量分析

度量算法运行时间的三种方式：
最好情形下的时间复杂度
最坏情形下的时间复杂度
平均情形下的时间复杂度

算法的效率分析

对算法时间资源的估算：
对算法执行基本运算（或步数）的数量分析

度量算法运行时间的三种方式：
最好情形下的时间复杂度
最坏情形下的时间复杂度
平均情形下的时间复杂度

算法分析中的概率分析方法

给定一个数组A，寻找数组中是否存在值为x的元素，若找到，则返回数组下标，若不存在，则返回-1；

```
k = 0;
while (k<=n-1 && a[k]!=x)
    k++;
if (k==n) return -1
else return k;
```

算法分析中的概率分析方法

给定一个数组A，寻找数组中是否存在值为x的元素，若找到，则返回数组下标，若不存在，则返回-1；

```
k = 0;
while (k<=n-1 && a[k]!=x)
    k++;
if (k==n) return -1
else return k;
```

最好情形：1次比较

最坏情形：n次比较

找不到的情形：n次比较

算法分析中的概率分析方法

给定一个数组A，寻找数组中是否存在值为x的元素，若找到，则返回数组下标，若不存在，则返回-1；

```
k = 0;
while (k<=n-1 && a[k]!=x)
    k++;
if (k==n) return -1
else return k;
```

假定x存在于A中，出现在A的位置概率是均等的，那么查找成功的时间复杂度为：

$$\mathrm{T}(n) = \frac{1}{n}\sum_{k=0}^{n-1} k = \frac{1}{n}\cdot\frac{n(n+1)}{2} = \frac{n+1}{2}$$

算法分析中的概率分析方法

给定一个数组A，寻找数组中是否存在值为x的元素，若找到，则返回数组下标，若不存在，则返回-1；

```
k = 0;
while (k<=n-1 && a[k]!=x)
    k++;
if (k==n) return -1
else return k;
```

假定x存在于A中，出现在A的位置概率是均等的，那么查找成功的时间复杂度为：

$$\mathrm{T}(n) = \frac{1}{n}\sum_{k=0}^{n-1} k = \frac{1}{n}\cdot\frac{n(n+1)}{2} = \frac{n+1}{2}$$

算法分析中的概率分析方法

给定一个数组A，寻找数组中是否存在值为x的元素，若找到，则返回数组下标，若不存在，则返回-1；

```
k = 0;
while (k<=n-1 && a[k]!=x)
    k++;
if (k==n) return -1
else return k;
```

假定x出现在A中的概率为p，出现在A的位置概率是均等的，那么总的查找时间复杂度为：

$$\mathrm{T}(n) = \mathrm{p}(\mathrm{T}(n)_{\text{成功}}) + (1-\mathrm{p})(\mathrm{T}(n)_{\text{不成功}}) = \mathrm{p}\left(\frac{n+1}{2}\right) + (1-\mathrm{p})\mathrm{n}$$

插入排序算法分析过程

平均情况下：输入数据随机排列。

$$\sum_{j=0}^{i-1} t_j = \frac{i-1-1}{i-1} + \sum_{k=0}^{i-1} \frac{1}{i-1}(i-1-k+1) = \frac{i-1}{2} - \frac{1}{i-1} + \frac{1}{2}$$

$$\sum_{i=1}^{n-1}\left(\frac{i-1}{2} - \frac{1}{i-1} + \frac{1}{2}\right) = \frac{n^2}{4} + \frac{3n}{4} - \sum_{j=1}^{n}\frac{1}{j}$$

$$T(n) = \boldsymbol{\Theta}(n^2)$$

cost	times
c1	n-1
c2	n-1
c3	n-1
c4	$\sum_{j=0}^{i-1} t_j$
c5	$\sum_{j=0}^{i-1} t_j - 1$
c6	$\sum_{j=0}^{i-1} t_j - 1$
c7	n-1

均摊分析

直接估计、递推关系

均摊分析

- 累计法
- 记账法
- 势能法

均摊分析-累计法

考虑栈的操作：

进栈和出栈是基本操作

考虑栈的一系列操作：

1: push(stack,a);

2: push(stack,b);

.....

.....pop()

n: pop()

T(n)=O(n)

均摊分析-累计法

考虑栈的操作：

进栈和出栈是基本操作

增加一个操作：multipop(stack,k)

考虑栈的一系列操作：

1: push(stack,a);

2: multipop(stack,k);

.....

..... multipop(stack,k);

n: pop()　　最坏情况：n*k，当k==n时，$T(n)=\Theta(n^2)$

均摊分析-累计法

考虑栈的操作：

进栈和出栈是基本操作

增加一个操作：multipop(stack,k)

考虑栈的一系列操作：

1: push(stack,a);

2: multipop(stack,k);

.....

.....multipop(stack,k);

n: pop()

而实际上，栈里最多n个元素，所以不可能达到n^2，因此：$T(n)=\Theta(n)$是合理的

时间复杂度的分析技巧

直接估计、递推关系

均摊分析

- 累计法
- 记账法
- 势能法

例: k位二进制计数器问题

从0执行加1操作n次.

统计: 位反转次数

粗略估计 kn 或 $n\log n$.

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

时间复杂度的分析技巧-累计均摊法

累计法:

每个位反转的总次数

n+n/2+n/4+...≈2n

均摊: (累计后均摊)

每次操作平均反转2次

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

从0到n需要的时间-记账法

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

假定每一次置位的摊还代价为2元；
当进行置位的操作支付1元，
并将另外的1元保存为信用，
用来支付将来复位操作时的代价

每一行只有一个0变1
最终总的（n行），摊还代价为n

计数器从0到n需要的时间-势能法

势能法的原理
势能法是一种分析算法复杂度的方法，
通过定义一个势能函数，来衡量问题状态的变化，
并分析其变化的规律和代价。
在这个问题中，
可以定义势能为计数器中1的个数。
初始状态下，计数器中的1的个数为0，
势能为0。当进行计数操作时，
计数器中的1的个数会增加。
经过2^k次计数操作后，势能增加到了k，
表示所有位的值都达到了1。
总的时间复杂度为O(k)

A[4]	A[3]	A[2]	A[1]	A[0]
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0

计数器从0到n需要的时间-势能法

势能法的原理

- D_i : 第i次操作后的数据结构, i=0:n
- c_i : 第i次操作的实际耗费时间, i=1:n
- Φ : 势能函数, 将D_i映射为一个实数$\Phi(D_i)$
- $d_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$第i次操作的均摊时间
- $\Sigma_{i=1}^{n} d_i = \Sigma_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$

  $= \Sigma_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$
- 若$\Phi(D_i) \geq \Phi(D_0)$ (势能), 则 $\Sigma_{i=1}^{n} d_i \geq \Sigma_{i=1}^{n} c_i$.

应用到计数器问题.

- $\Phi(D_i)$ = D_i中1的个数.

$\Phi(D_i) - \Phi(D_{i-1}) <= (b_{i-1} - t_i + 1) - b_i - 1 = 1 - t_i$

$d_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) <= (t_i + 1) + (1 - t_i) = 2$

$\Sigma_{i=1}^{n} d_i <= 2n - b_n - b_0 = 2n - k$

A[4]	A[3]	A[2]	A[1]	A[0]
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0

复杂度f(n)的渐近记法

称g(n)是f(n)的渐进上界, 即$f(n) = O(g(n))$,
若 $\exists c>0, N>0, \forall n>N, f(n) \leq cg(n)$

例: $n^2+10000n = O(n^3)$, $n^2+10000n = O(n^2)$

注: $\forall n>0, n^2 \leq n^2+10000n \leq 10001n^2$.

复杂度f(n)的渐近记法

称g(n)是f(n)的渐进下界, 即$f(n) = \Omega(g(n))$,
若 $\exists c>0, N>0, \forall n>N, f(n) \geq c\ g(n)$

例: $n^2+10000n = \Omega(n)$, $n^2+10000n = \Omega(n^2)$

复杂度f(n)的渐近记法

$f(n) = \theta(g(n))$ 渐近同阶 (θ[王], Θ[C]),
若 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$

例: $n^2+10000n$

$n^2+10000n = O(n^2)$

$n^2+10000n = \Omega(n^2)$

因此： $n^2+10000n = \theta(n^2)$或 $\Theta(n^2)$

渐近记法的定价极限定义

称g(n)是f(n)的渐进上界, 即f(n) = O(g(n)),

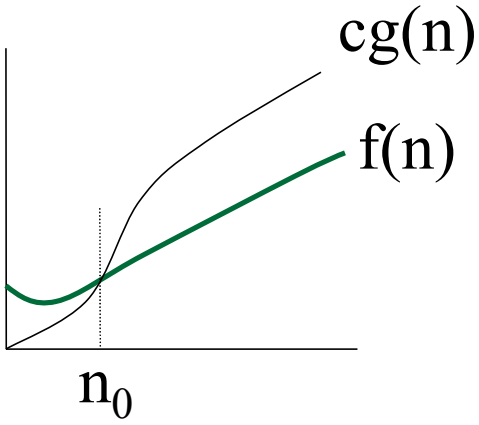
$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

称g(n)是f(n)的渐进下界, 即f(n) = Ω(g(n)),

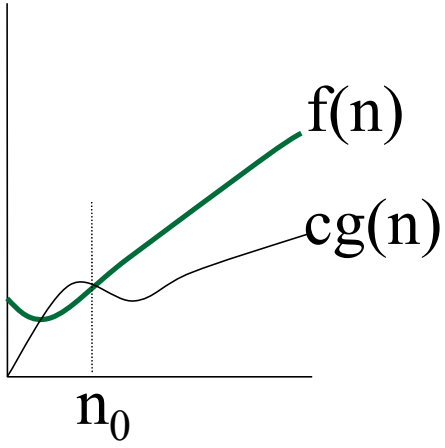
$$\liminf_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

f(n) = θ(g(n)) 渐近同阶 (θ[王], Θ[C])

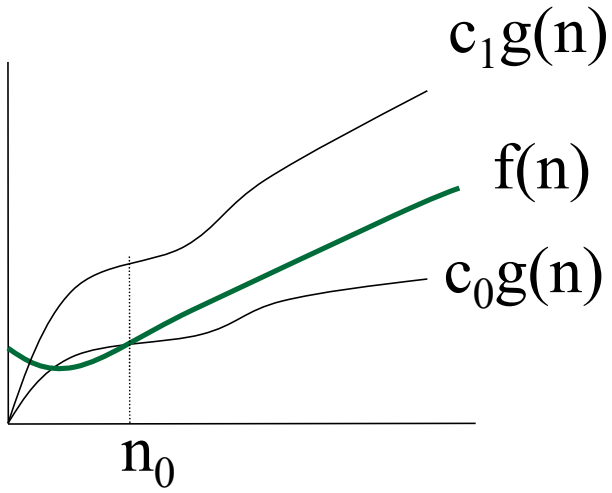
O, Ω, θ示意图



$f(n) = O(g(n))$



$f(n) = \Omega(g(n))$



$f(n) = \theta(g(n))$

常用渐近函数

$f(n) = O(g(n))$

为方便, g(n)通常取

n^k, //多项式

$n^k \log n$, $n \log\log n$ //含对数项

e^n 或 $e^{O(n)}$ //指数函数

例如: $\log n^3 = O(\log n)$, $\log 3^n = O(n)$

常用渐近函数

对素数进行筛选的算法，称为Eratosthenes(古希腊数学家埃拉托斯特尼斯)筛法。该算法的复杂度为nloglogn。

该算法的目标是找出小于等于给定上限n的所有素数。算法的步骤如下：

1.创建一个长度为n+1的布尔数组isPrime，并将所有元素初始化为true。

2.将isPrime[0]和isPrime[1]设置为false，因为0和1不是素数。

3.从2开始遍历数组，对于每个遇到的素数p，将isPrime[p]设置为true，并将大于p且能被p整除的所有索引i的isPrime[i]设置为false。

遍历完数组后，isPrime中为true的索引对应的数就是素数。

该算法的时间复杂度可以解释为nloglogn。遍历数组时，每个素数p将导致对isPrime数组的一次操作，而素数的数量约为n/logn。因此，算法的时间复杂度为O(n/logn * log(n/logn))，简化为O(nloglogn)。

常用渐近函数

快速乘法（Fast Multiplication）算法，它的复杂度为nloglogn

快速乘法算法用于高效地计算两个大整数的乘积。

算法的步骤如下：

1.将要相乘的两个整数拆分成多个小块，每个小块的大小为sqrt(n)。

2.对于每个小块，将其视为一个多项式，通过多项式乘法计算出其乘积。

3.将每个小块的乘积相加，得到最终的乘积结果。

拆分整数为小块的过程需要O(logn)次迭代。对于每个小块的乘积计算，可以使用传统的多项式乘法算法，其复杂度为O(sqrt(n)log(sqrt(n)))，即O(sqrt(n)logn)。因此，总体复杂度为O(logn * sqrt(n)logn)，简化为O(nloglogn)。

Contents

本章内容

- 什么是算法
- 算法复杂性分析
- P问题与NP完全问题
- 算法分析中的数学基础

可计算理论

有没有计算机无法解决的问题?

是什么使得某些问题难于计算，而另一些问题却容易计算?
在给定的计算模型下，来研究问题的复杂性
按照难度给问题分类
抽象复杂性研究

PCP问题没有算法(导引[S])

Post Corresponding Problem([S])

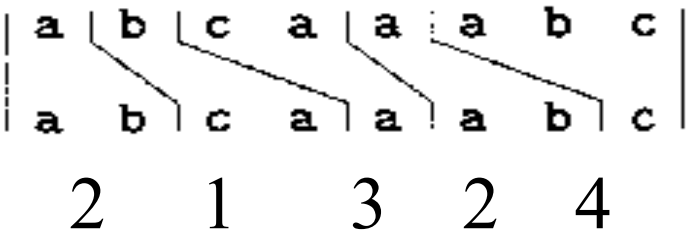
输入: 一簇骨牌

$$\left\{\left[\frac{t_1}{b_1}\right],\left[\frac{t_2}{b_2}\right],\cdots,\left[\frac{t_k}{b_k}\right]\right\}$$

定义其解(匹配)为一个序列 $i_1,i_2,\ldots,i_s\in\{1,\ldots,k\}$, 使得

$$t_{i_1}t_{i_2}\ldots t_{i_s}=b_{i_1}b_{i_2}\ldots b_{i_s}.$$

$$\left\{\left[\frac{b}{ca}\right],\left[\frac{a}{ab}\right],\left[\frac{ca}{a}\right],\left[\frac{abc}{c}\right]\right\}$$



输出: 给定骨牌簇是否有解.

PCP问题没有算法(导引[S])

t_1	t_2	t_3
a	ab	bba

b_1	b_2	b_3
baa	aa	bb

3，2，3，1

希尔伯特第十问题没有算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

Hilbert第十问题：“多项式是否有整数根”有没有算法?

1970’s被证明没有算法

- “算法”：对于输入p，p是k元多项式

1. 取k个整数的向量x（绝对值的和从小到大）.
2. 若p(x) = 0, 则接受,
3. 否则转1.

$x^2+y^2-3=0$死循环

3n+1问题目前不知道有没有算法

输入: 一个正整数n,

映射: f(n) = n/2,　若n是偶数;

　　f(n) = 3n+1, 若n是奇数.

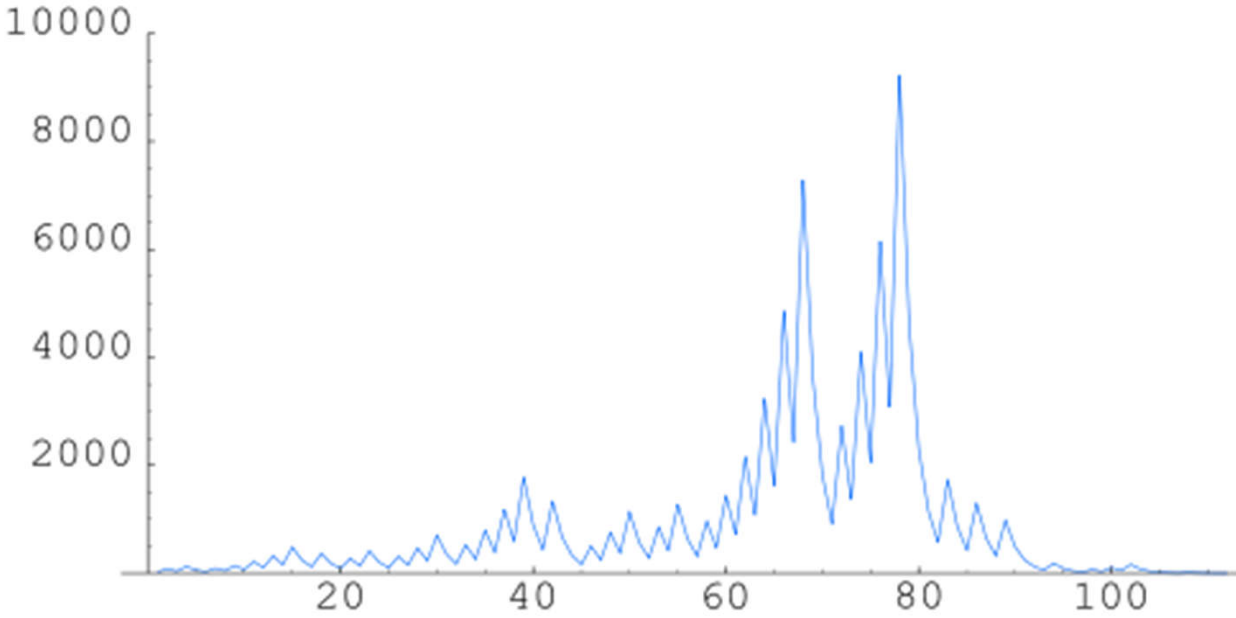
迭代: 5→16→8→…, 到1则停止

输出: n可在f迭代下是否能停止

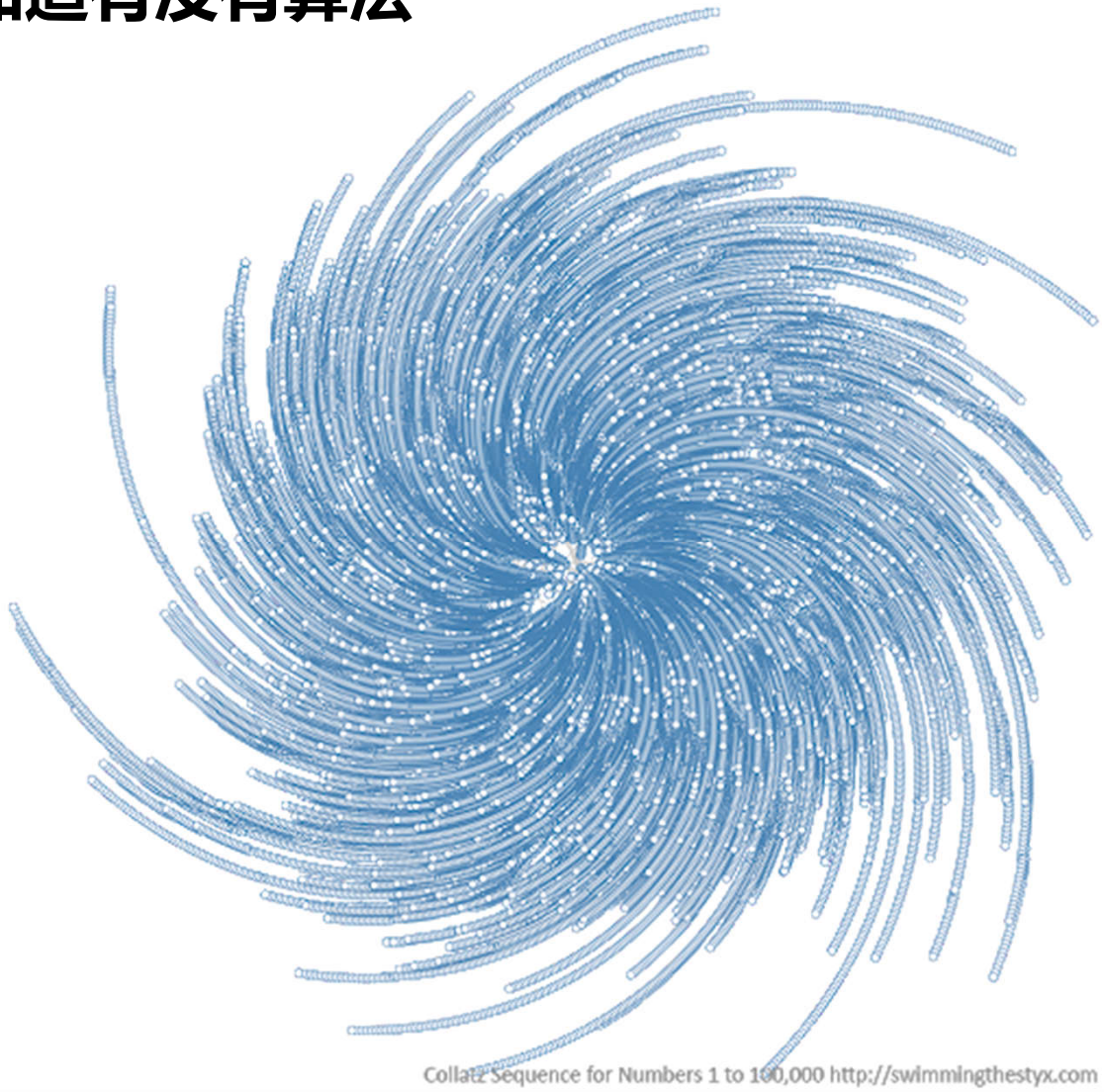
直接模拟是正确的算法吗?

27需迭代111步(见右图)

1~5×10^{18}都能到1.([wiki])



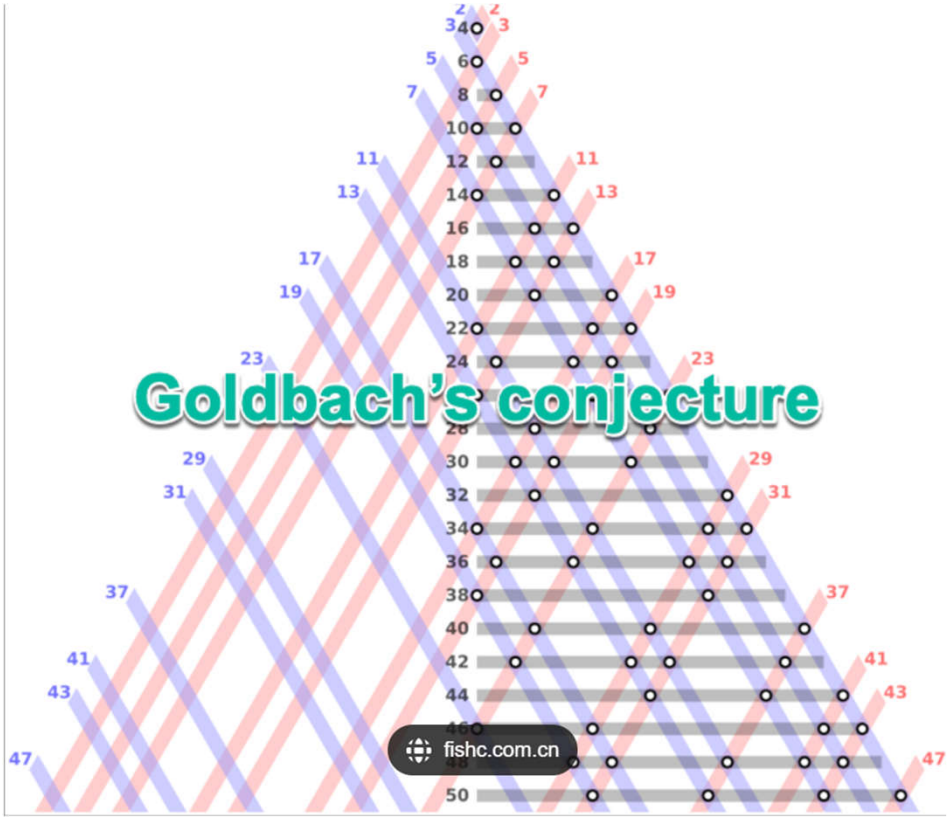
3n+1问题目前不知道有没有算法



哥德巴赫猜想至今尚未证明，没有算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

问题:任何一个大于2的偶数都可以表示为两个素数之和



需要给算法严格定义

问题的分类：

有的问题被解决了　　(找最大数)
有的问题没有被解决　(3n+1问题)
有的问题没有解决方案（PCP问题，希尔伯特第十问题)

将在计算理论部分给出算法的严格定义.

高效的算法

多项式时间算法(P类)在理论上是高效的算法

指数时间完全问题

例: 全量词化布尔公式是否真(TQBF): 全体博弈

$\exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots P(x_1, y_1, x_2, y_2, \ldots)$ 其中P是布尔公式

NP(非确定性多项式时间)完全问题

也称为可以多项式时间验证解正确性的问题.

例: 布尔公式是否有真赋值(CNF-SAT)

$\phi = ((\neg x) \wedge y) \vee (x \wedge (\neg z))$有真赋值(x,y,z)=(F,T,F)

例: 顶点覆盖, 哈密顿回路等

高效的算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

最小顶点覆盖问题要求找到一个顶点集合C，使得图中的每条边至少有一个端点属于C，并且C的大小最小。顶点覆盖是一种保证图中每条边至少与其中一个顶点相邻的方式。

最小顶点覆盖问题是一个优化问题，可以用数学模型来描述。假设图G有n个顶点，m条边，我们可以用一个长度为n的二进制向量x=(x1, x2, ..., xn)来表示一个候选顶点覆盖集合，其中xi=1表示顶点i属于候选集合，xi=0表示顶点i不属于候选集合。

我们的目标是最小化向量x的1的个数。

最小顶点覆盖问题是一个NP完全问题，这意味着目前没有已知的多项式时间算法可以解决该问题。

高效的算法

哈密顿回路问题要求找到一条路径，使得它从一个顶点出发，经过图中的每个顶点一次且仅一次，最后回到出发的顶点形成一个回路。

哈密顿回路问题可以用数学模型来描述。假设图G有n个顶点，我们可以用一个长度为n的排列（permutation）P=(v1, v2, ..., vn)来表示一个候选哈密顿回路，其中vi表示图中的第i个顶点。我们的目标是找到一个排列P，使得图中存在一条路径经过P中的每个顶点一次且仅一次，并且路径的起点和终点相同。

哈密顿回路问题是一个NP完全问题，这意味着目前没有已知的多项式时间算法可以解决该问题

德以明理 学以精工

容易解决的问题

通常把多项式时间算法(P类)在理论上看作是高效的算法

策略 与 技巧

规模 与 时间

NP完全问题

目前还没有找到多项式时间算法

七大千禧问题之一

密码系统设计

说明目前没有好算法

解决方案: 修改问题, 或使用近似算法

NP完全问题

七大千禧问题的简要描述：

1.P versus NP问题：该问题涉及计算机科学和计算复杂性理论。它探讨的是是否存在一种高效算法来解决所有问题的答案验证过程。

2.黎曼猜想：这个问题是关于素数分布的性质。它提出了一种关于黎曼ζ函数的假设，如果得到证明，将会对素数的分布和整数理论产生深远的影响。

3.黎曼猜想的证明：这个问题与第二个问题相关，要求给出黎曼猜想的证明。

4.Poincaré猜想：该问题是关于拓扑学中的三维球体的性质。它要求证明每个闭合的三维流形，如果它的每个紧致子流形都是同构于三维球体，那么该流形本身就是同构于三维球体。

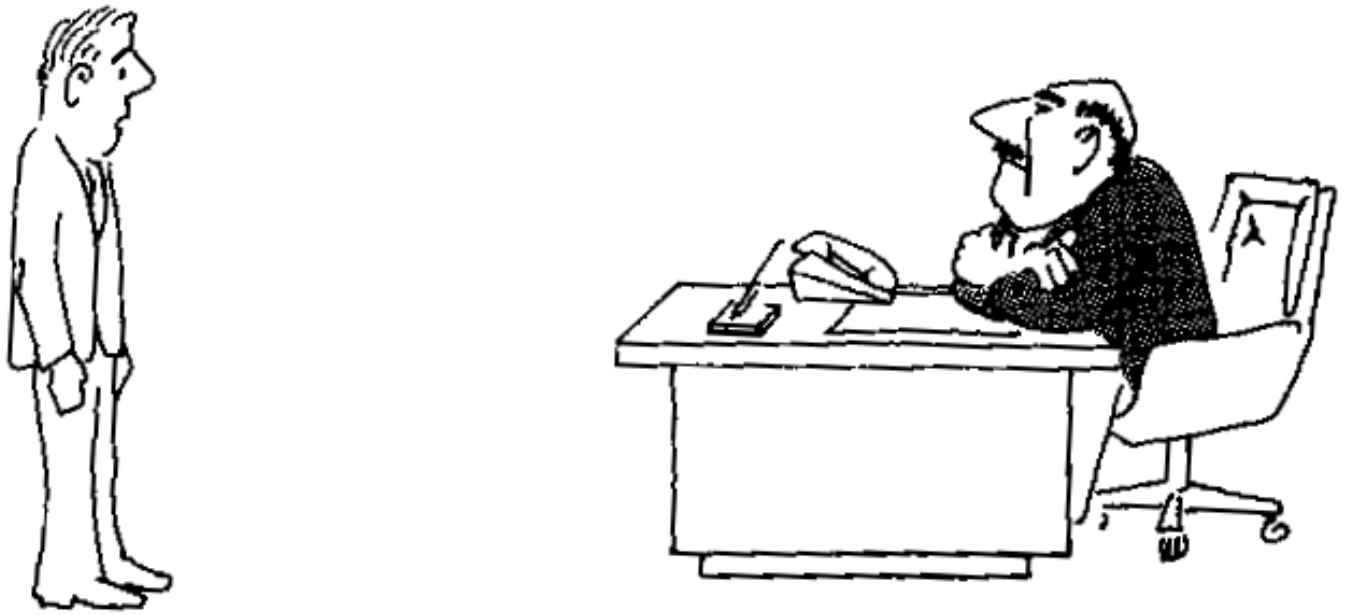
5.Navier-Stokes方程：这个问题涉及流体力学中的不可压缩流体。它要求解决Navier-Stokes方程的局部存在和光滑性问题，即证明在一定条件下是否始终存在唯一的光滑解。

6.Riemann假设的证明：这个问题要求给出黎曼假设的证明。黎曼假设是关于黎曼ζ函数的性质，如果得到证明，将会对数论和复分析产生重大影响。

7.Hodge猜想：该问题涉及代数几何中的Hodge理论。它要求证明Hodge猜想，该猜想是关于复代数簇上的Hodge结构的性质。

这些问题都是数学界的难题，至今尚未得到解决。解决任何一个问题都将对数学领域做出重大贡献，并获得丰厚的奖金。

NP完全问题



"I can't find an efficient algorithm, I guess I'm just too dumb."

Serious damage to your
position within the company !!!

NP完全问题

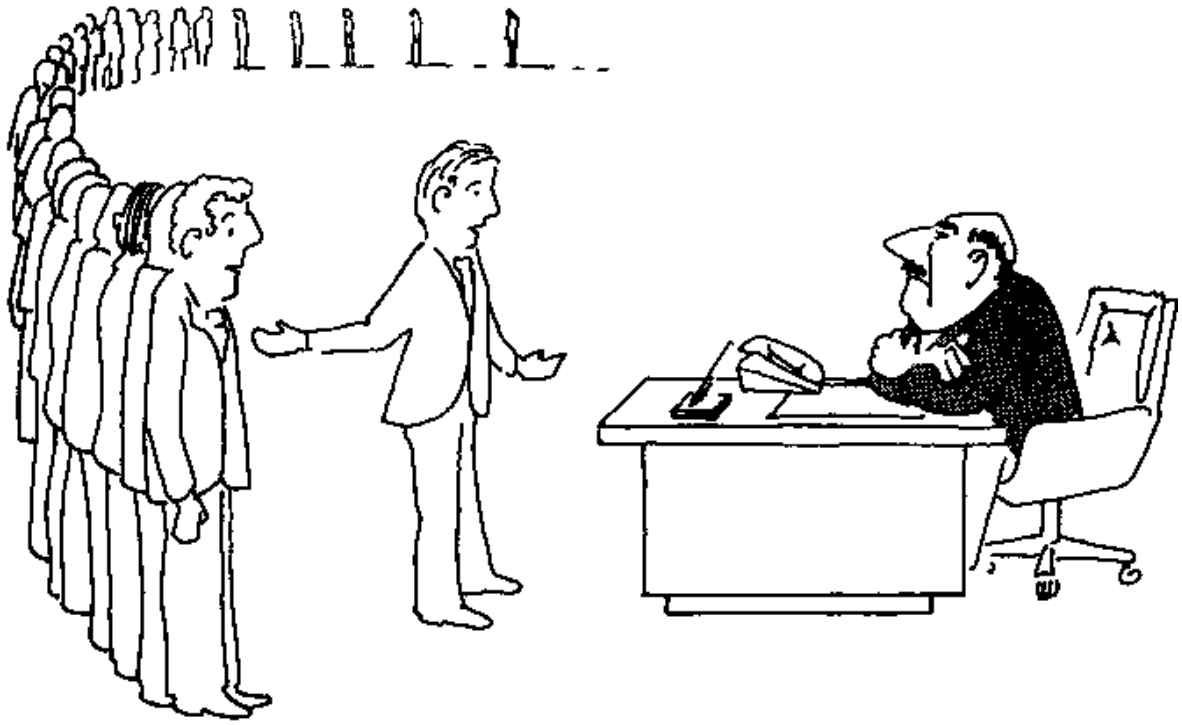
"I can't find an efficient algorithm, because no such algorithm is possible!"

Unfortunately, proving intractability can be just as hard as finding efficient algorithms !!!

∴ No hope !!!

$P \neq NP$

NP完全问题



"I can't find an efficient algorithm, but neither can all these famous people"

NP完全问题

“Anyway, we have to solve this problem.
Can we satisfy with a good solution ? ”

NP完全问题

A survey of algorithmic design techniques.

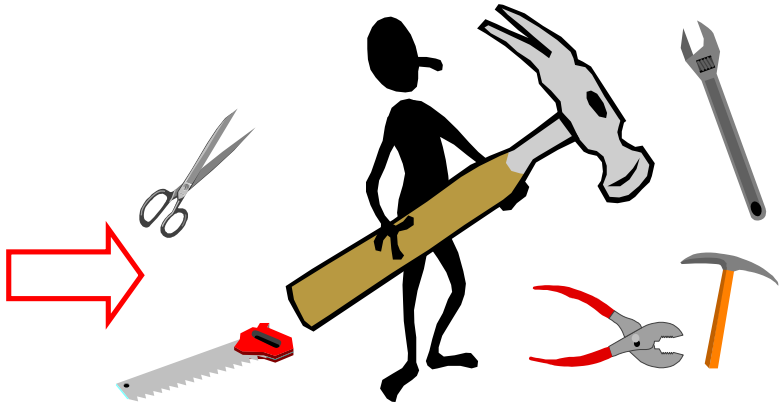
Abstract thinking.

How to develop new algorithms for any problem that may arise.

Be a great thinker and designer.

Not: A list of algorithms

- **Learn their code**
- **Trace them until work**
- **Implement them**
- **be a mundane programmer**



一些有趣的算法

旅行商问题Traveling Salesman Problem (TSP):
设有n个城市，已知任意两城市之间距离，现有一推销员想从某一城市出发巡回经过每一城市（且每城市只经过一次），最后又回到出发点，问如何找一条最短路径。

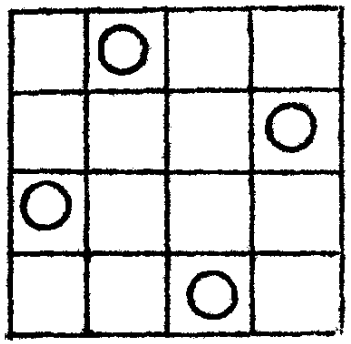
穷举法的时间复杂性为O(n!)

n = 21时，21! = 5.11×10^{19}, 设每条路径需CPU时间为1ns，则总共要1620多年才能完成。

一些有趣的算法

皇后问题:这是高斯1850年提出的一个著名问题：国际象棋中的“皇后”在横向、直向、和斜向都能走步和吃子，问在n×n 格的棋盘上如何能摆上n个皇后而使她们都不能互相攻击。

设n=4，试一试。



当n很大时，问题很难。对于n=8,现已知此问题，共有92种解，但只有12种是独立的，其余的都，可以由这12种利用对称或旋转而得到。

一些有趣的算法

背包问题1：有一旅行者要从n种物品中选取不超过b千克重的行李随身携带，要求总价值最大。

例：设背包的容量为50千克。物品1重10千克，价值60元；物品2重20千克，价值100元；物品3重30千克，价值120元。求总价值最大。

背包问题2：有一商人要从n种货物中选取不超过b千克重的行李随身携带，要求总价值最大。

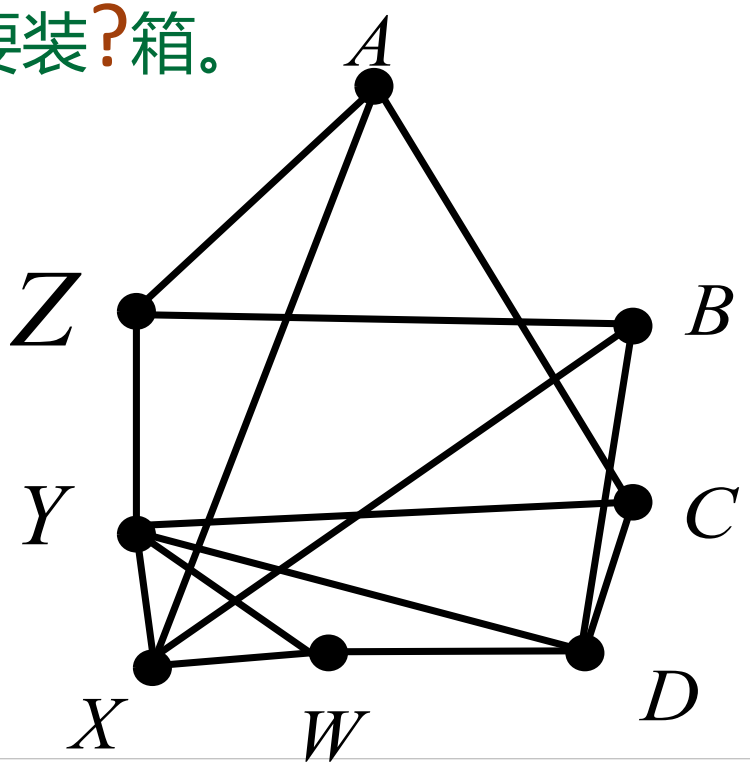
例：设背包的容量为50千克。物品1有60千克，每千克价值60元；物品2有20千克，每千克价值100元；物品3有40千克，每千克价值120元。求总价值最大。

一些有趣的算法

有4个人打算过桥，他们都在桥的某一端。时间是晚上，他们只有一只手电筒。最多只能有两个人同时过桥，而且必须携带手电筒。必须步行将手电筒带来带去。每个人走路的速度不同：甲过桥需要1分钟，乙2分钟，丙5分钟，丁10分钟。两个人一起走的速度等于其中较慢的人的速度。要求过桥总时间最短。

一些有趣的算法

有八种化学药品 A、B、C、D、W、X、Y、Z 要装箱运输。虽然量不大，仅装 1箱也装不满，但出于安全考虑，有些药品不能同装一箱。在下表中，符号“×”表示相应的两种药品不能同装一箱。运输这八种化学药品至少需要装?箱。



A								
B								
C	×							
D		×	×					
W				×				
X	×	×			×			
Y			×	×	×	×		
Z	×	×					×	
	A	B	C	D	W	X	Y	Z

rk.csai.cn

一些有趣的算法

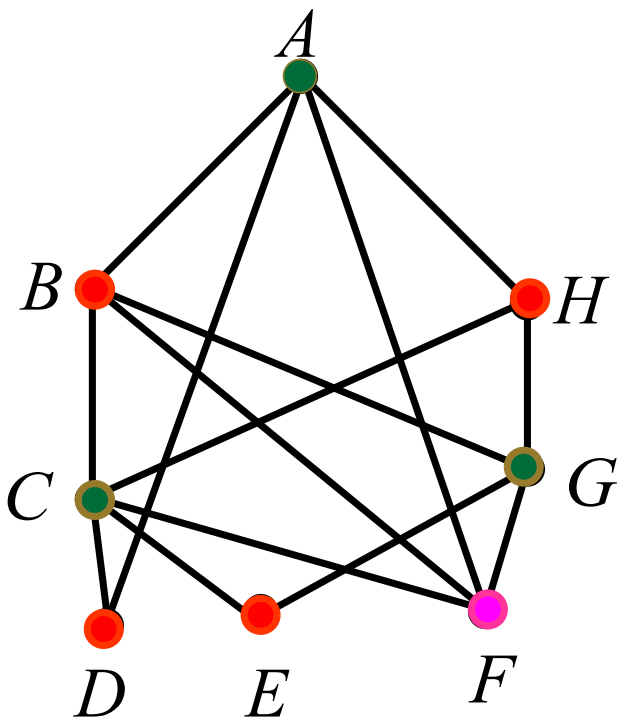
韦尔奇·鲍威尔法(Welch Powell)

1) 将图 G 中的结点按度数递减的次序进行排列(相同度数的结点的排列随意)。

2) 用第一种颜色，对第一点着色，并按排列次序对与前面结点不相邻的每一点着同样的颜色。

3) 用第二种颜色对尚未着色的点重复第 2 步，直到所有的点都着上颜色为止。

一些有趣的算法



解：

- 按度数递减次序排列各点

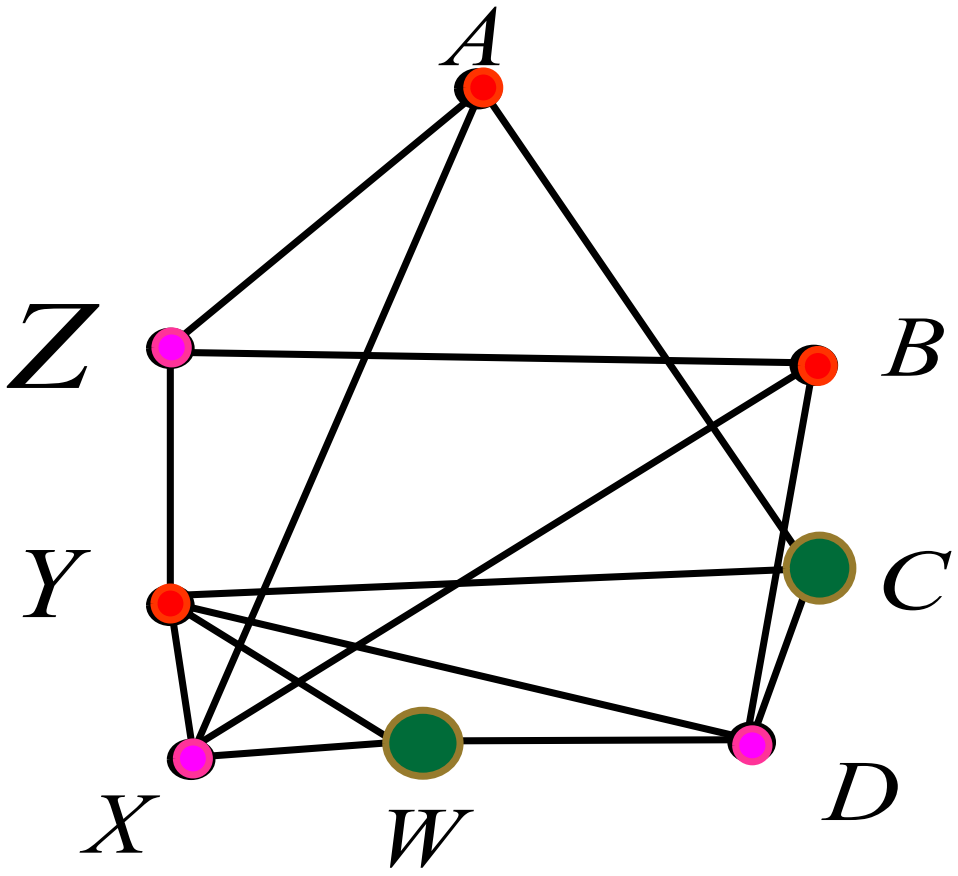
$$C\ A\ B\ F\ G\ H\ D\ E$$

- 第一种颜色： $C,\ A,\ G$
- 第二种颜色： $B,\ H,\ D,\ E$
- 第三种颜色： F

所以图是三色的。

另外图不能是两色的,因为图中有A,B,F两两相邻,所以 $x(G)=3$

一些有趣的算法



Y、X、D、A、B、C、W、Z

A								
B								
C	×							
D		×	×					
W				×				
X	×	×			×			
Y			×	×	×	×		
Z	×	×					×	
	A	B	C	D	W	X	Y	Z

rk.csai.cn

德以明理 学以精工

一些有趣的算法

The Drunk Jailer

描述A certain prison contains a long hall of n cells, each right next to each other. Each cell has a prisoner in it, and each cell is locked.

One night, the jailer gets bored and decides to play a game. For round 1 of the game, he takes a drink of whiskey, and then runs down the hall unlocking each cell. For round 2, he takes a drink of whiskey, and then runs down the hall locking every other cell (cells 2, 4, 6, ...). For round 3, he takes a drink of whiskey, and then runs down the hall. He visits every third cell (cells 3, 6, 9, ...). If the cell is locked, he unlocks it; if it is unlocked, he locks it.

一些有趣的算法

He repeats this for n rounds, takes a final drink, and passes out.

输入：The first line of input contains a single positive integer. This is the number of lines that follow. Each of the following lines contains a single integer between 5 and 100, inclusive, which is the number of cells n.

输出：For each line, print out the number.

样例输入	样例输出
2	2
5	10
100	

Halloween treats

Every year there is the same problem at Halloween: Each neighbor is only willing to give a certain total number of sweets on that day, no matter how many children call on him, so it may happen that a child will get nothing if it is too late. To avoid conflicts, the children have decided they will put all sweets together and then divide them evenly among themselves. From
last year's experience of Halloween they know how many sweets they get from each neighbor. Since they care more about justice than about the number of sweets they get, they want to select a subset of the neighbors to visit, so that in sharing every child receives the same number of sweets. They will not be satisfied if they have any sweets left which cannot be divided.

Halloween treats

Sample Input

4 5 (the number of children and the number of neighbors, respectively.)

1 2 3 7 5

3 6

7 11 2 5 13 17

0 0

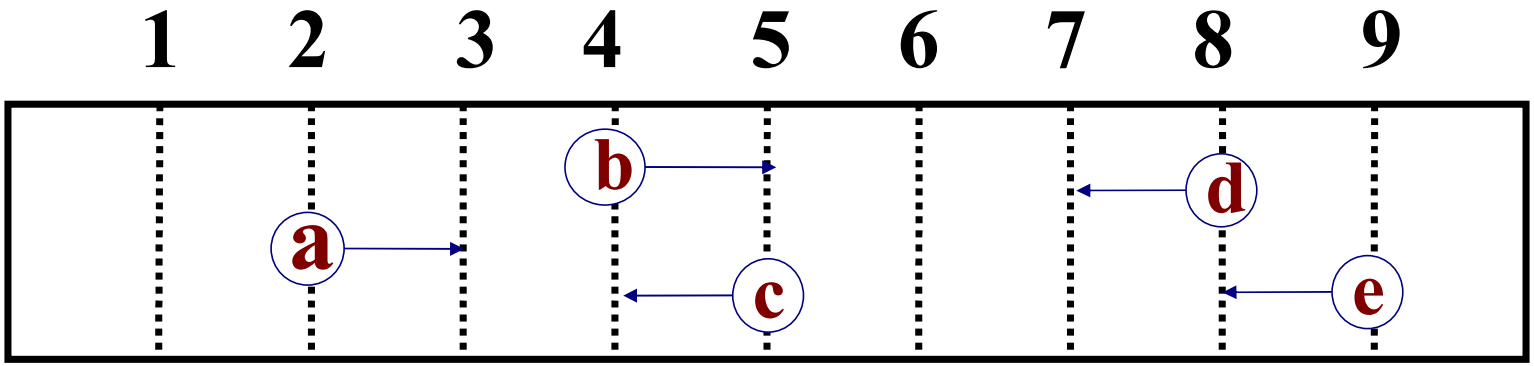
Sample Output

3 5

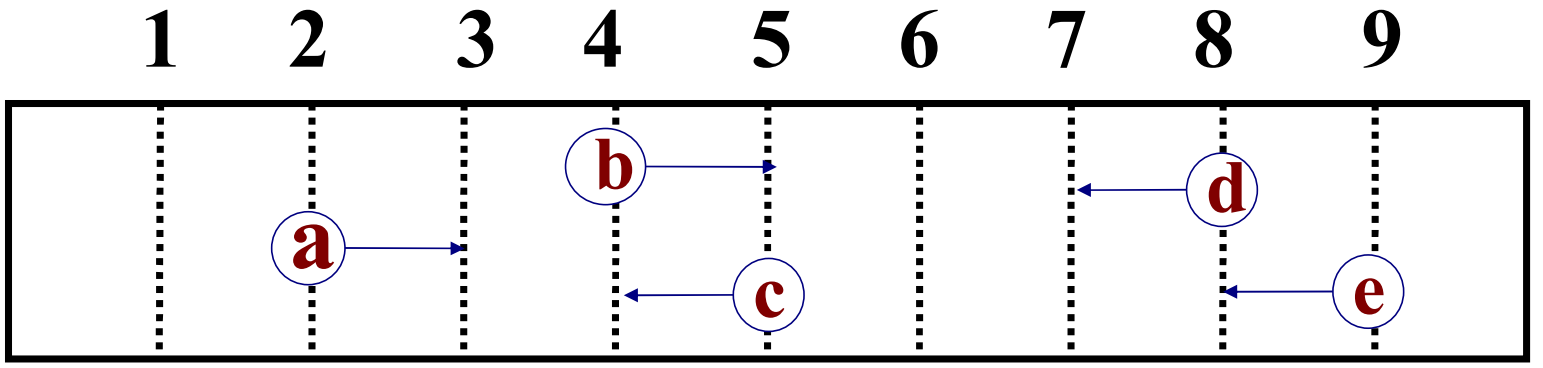
2 3 4

一些有趣的算法

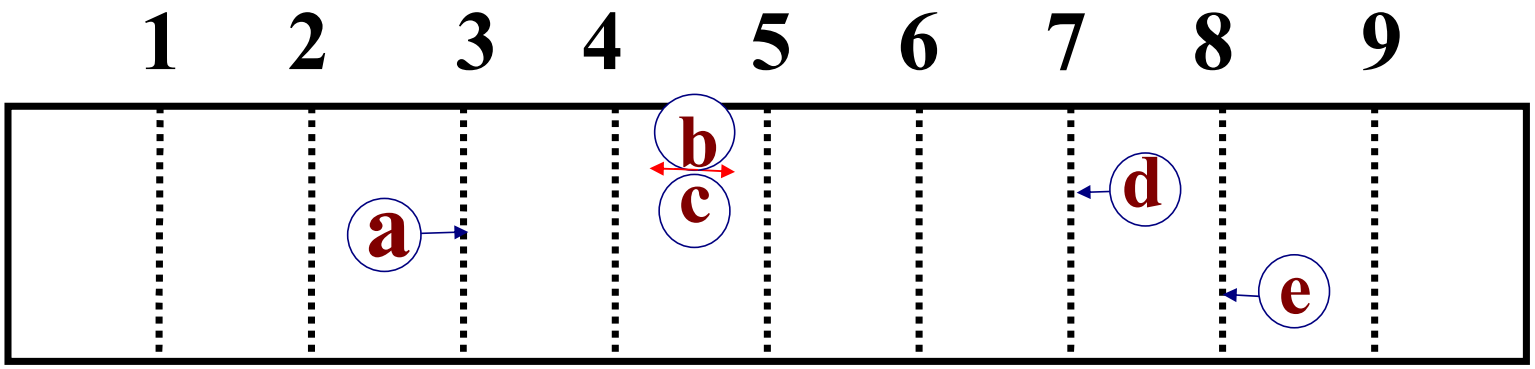
有一根10厘米的细木杆，在第2厘米、第4厘米、第5厘米、第8厘米、第9厘米这五个位置上各有一只蚂蚁。木杆很细，不能同时通过一只蚂蚁。开始时，蚂蚁的头朝左还是朝右是任意的，它们只能朝前走或调头，但不会后退。所有蚂蚁的速度都相同，均为1cm/s。当任意两只蚂蚁碰头时，两只蚂蚁会同时掉头并且仍然按1cm/s朝相反方向走。求蚂蚁掉下木杆的最小和最大时间。



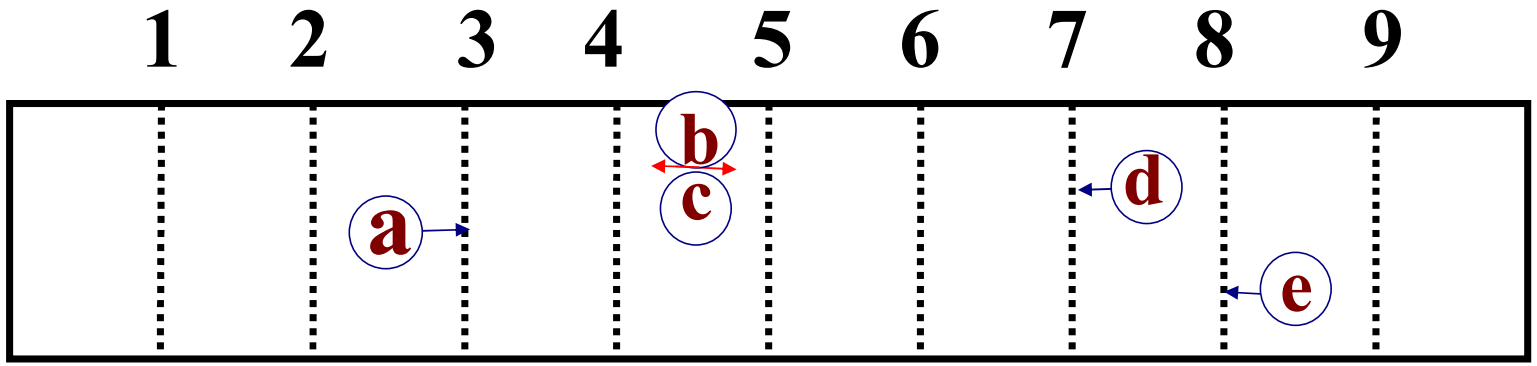
一些有趣的算法



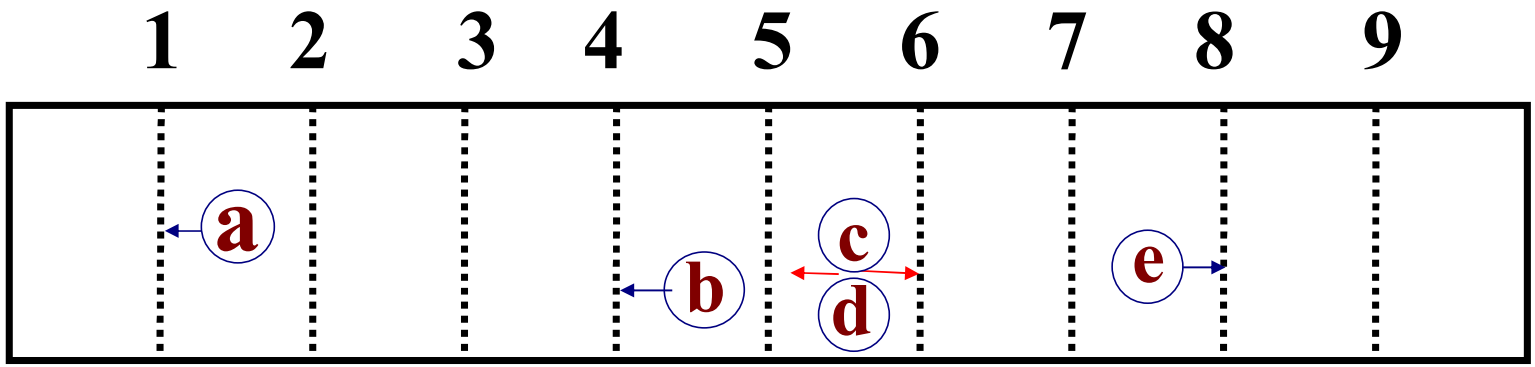
0.5s



一些有趣的算法



3.5s



5秒，a掉；6秒，e掉；8秒，b和d掉；9秒，c掉。

Contents

本章内容

什么是算法

算法复杂性分析

P问题与NP完全问题

算法分析中的数学基础

一些符号和公式

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

⌊x⌋: 小于等于x的最大整数, ⌈x⌉:大于等于x 的最小整数

$$a^{\log b^n} = n^{\log b^a}$$

Stirling公式

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

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

等比级数求和

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$

调和级数

$$\sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$$

一些符号和公式

$$\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n$$

$$\left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil$$

$$\left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$$

2020!末尾多少个0?

503个0

一些符号和公式

$$\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n$$

$$\left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil$$

$$\left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$$

2020!末尾多少个0?

503个0 = 404+80+16+3

一些符号和公式

鸽巢原理（又名抽屉原理）

若n+1只鸽子飞入n个鸽巢，那么至少有一个鸽巢至少有2只鸽子.

若n只鸽子飞入m (n>m)个鸽巢，那么至少有一个鸽巢至少有$\lfloor (n-1)/m \rfloor+1$只鸽子.

一些符号和公式

对数

$\log n = \log_2 n, \quad \lg n = \log_{10} n$

$\log^k n = (\log n)^k$

$\log \log n = \log(\log\ n)$

性质：

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

$\log_k n = c \log_l n$

lg 2 = 0.301

一些符号和公式

证明：

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

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

$$= (b^{\log_b n})^{\log_b a}$$

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

常用对数变换

$$\log_b N = \log_a N / \log_a b$$

$$\log_e N = \ln N$$

$$e = 2.71828$$

$$\log_a b = 1/\log_b a$$

一些符号和公式

阶乘

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

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

一些符号和公式

求和

等比级数的求和公式

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$

一些符号和公式

调和级数

$$\sum_{k=1}^{n}\frac{1}{k}=\ln n+O(1)$$

本章小结和作业

算法分析题 1-1, 1-2, 1-4

1-1 求下列函数的渐近表达式: $3n^2+10n$; $n^2/10+2^n$; $21+1/n$; $\log n^3$; $10\log 3^n$.

1-2 试论O(1)与O(2)的区别.

1-4 (1) 假设某算法在输入规模为n时的计算时间为$T(n)=3\times 2^n$. 在某台计算机上实现并完成该算法的时间为t秒. 现有另一台计算机, 其运行速度是第一台的64倍, 那么在这台新机器上用同一算法在t秒内能解输入规模为多大的问题?

(2) 若上述算法的计算时间改进为$T(n)=n^2$, 其余条件不变, 则在新机器上用t秒时间能解输入规模为多大的问题?

(3) 若上述算法的计算时间改进为$T(n)=8$, 其余条件不变, 那么在新机器上用t秒时间能解输入规模为多大的问题?

补充题: 构造输入使得求最大次大数问题的方法一和方法三比较次数不同.

附录:最大次大数问题分治法分析

已知$T(n) = 2T(n/2) + 2$， $T(2)=1, T(3)=3$.

若$n=2^k$，则 $T(n)=3n/2-2$; ([M])

$$
\begin{aligned}
T(2^k) &= 2T(2^{k-1})+2 \\
&= 2(2T(2^{k-2})+2)+2 \\
&= 2^2T(2^{k-2})+2^2+2 \\
&= 2^3T(2^{k-3})+2^3+2^2+2 \\
&= \ldots \\
&= 2^{k-1}\,T(2)+2^{k-1}+\ldots+2^2+2 \\
&= 2^{k-1}+2^{k-1}+\ldots+2^2+2 \\
&= 3\times 2^{k-1}-2 \\
&= 3n/2-2
\end{aligned}
$$

附录:最大次大数问题分治法分析

已知$T(n) = 2T(n/2) + 2$， $T(2)=1, T(3)=3$.

若$n=2^k$，则 $T(n)=3n/2-2$; ([M])

若$n=3\cdot 2^k$, 则$T(n)=5n/3-2$.

$$
\begin{aligned}
T(3\cdot 2^k) &= 2T(3\cdot 2^{k-1})+2 \\
&= 2(2T(3\cdot 2^{k-2})+2)+2 \\
&= 2^2T(3\cdot 2^{k-2})+2^2+2 \\
&= 2^3T(3\cdot 2^{k-3})+2^3+2^2+2 \\
&= \dots \\
&= 2^kT(3)+2^k+\dots+2^2+2 \\
&= 3\cdot 2^k+2^k+\dots+2^2+2 \\
&= 5\times 2^k-2 \\
&= 5n/3-2
\end{aligned}
$$

附录:最大次大数问题二分法分析

已知 $T(n) = 2T(n/2) + 2$, $T(2)=1, T(3)=3$.

若$n=2^k$, 则$T(n)=3n/2-2$; ([M])

若$n=3\cdot 2^k$, 则$T(n)=5n/3-2$;

对一般n, 下面归纳证明P(n): $3n/2-2 \le T(n) \le 5n/3-2$.

(1) 由定义，P(2)和P(3)成立

(2) 设$2^{k-1} \le n < 2^k$时P(n)成立，讨论$2^k \le n < 2^{k+1}$的情况

若n是偶数，则

$T(n) = 2\,T(n/2) + 2 \le 2\,(5/3\,(n/2) - 2\,) + 2 = 5n/3-2$

$T(n) = 2\,T(n/2) + 2 \ge 2\,(3/2\,(n/2) - 2\,) + 2 = 3n/2-2$

若n是奇数，则

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

$\le 5/3\,(n+1)/2 - 2 + 5/3\,(n-1)/2 - 2 + 2 = 5n/3-2$

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

$\ge 3/2\,(n+1)/2 - 2 + 3/2\,(n-1)/2 - 2 + 2 = 3n/2-2$

由(1)(2)知P(n)对所有n>1成立.