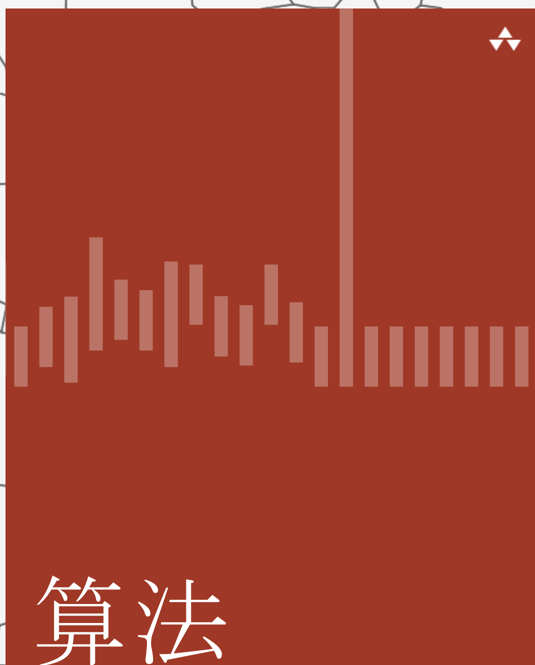
算法

罗伯特-塞奇威克 | 凯文-韦恩



Robert Sedgewick | Kevin Wayne

http:// algs
4.cs.princeton.edu

1.4 算法的分析

- *介绍*
- *观察*
- *数学模型*
- *增长顺序的分类*
- *算法理论*

- *记忆*

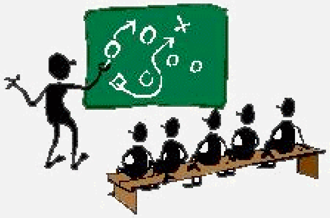
角色阵容

程序员需要开发一个工作解决方案
。

学生有一天可能会扮演这些角色中的任何一个或全部。

客户希望有效地解决问题。

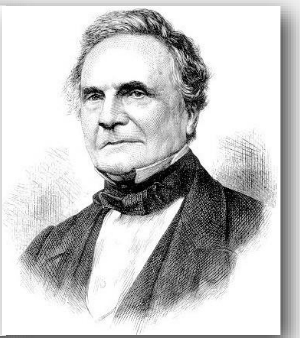
理论家想要了解。

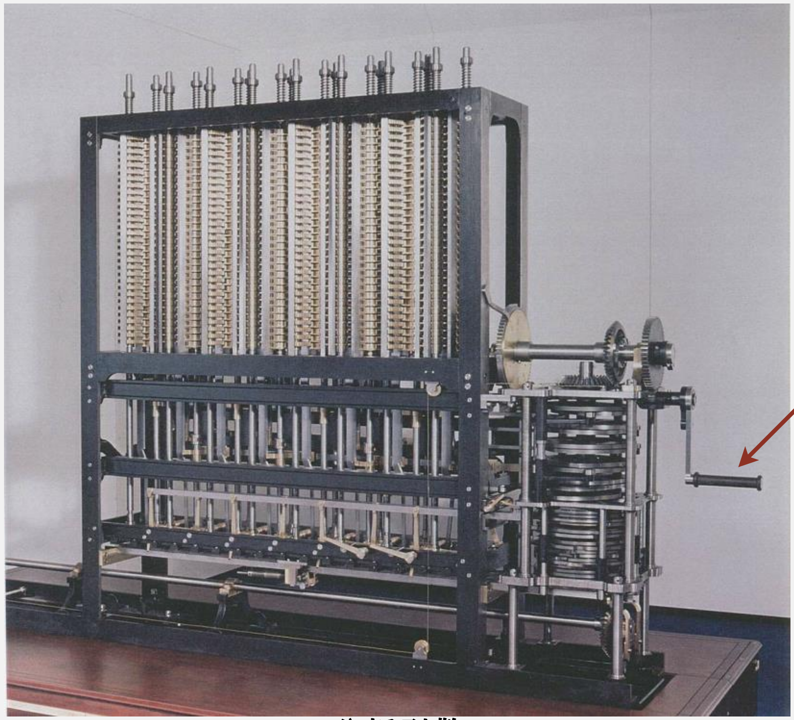


基本的阻挡和对付有时是必要的。[本讲座]。

运行时间

"一旦有了分析引擎，它就必然会指导科学的未来进程。每当在它的帮助下寻求任何结果时，就会出现这样的问题：通过什么样的计算过程，机器可以在最短的时间内得出这些结果？"--查尔斯-巴贝奇（1864年）





分析引擎

你要转多少次曲柄？

分析算法的原因

预测业绩。

比较算法。提供保证。

本课程

理解理论基础。

算法理论

主要的实际原因：避免性能错误。

由于程序员不了解性能特点，客户得到的性能很差

一些算法上的成功

离散傅里叶变换。

- 将N个样本的波形分解为周期性成分。
- 应用。DVD、JPEG、MRI、天体物理学、....

砖头的力量。N^2步。

- FFT算法。N对数N步，启用新技术。

弗里德里希-高斯 1805年

时间

二次方

64T

32T

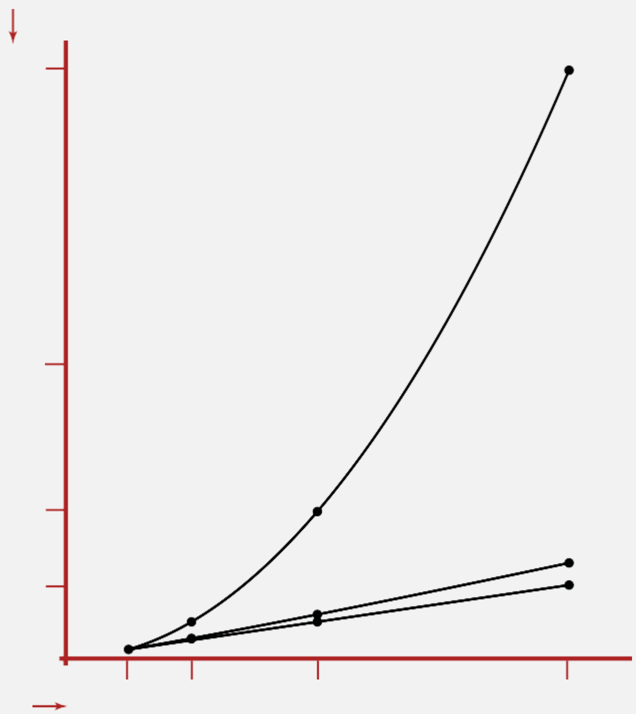
16T

线性思维线性

8T

思维

尺寸 1K 2K 4K 8K



一些算法上的成功

N体模拟。

- 模拟N个物体之间的引力相互作用。

* 蛮力。N^2步。

巴恩斯-胡特算法。$N \log N$步，实现了新的研究。

安德鲁 Appel
81年的PU

时
间

二次方

64T

32T

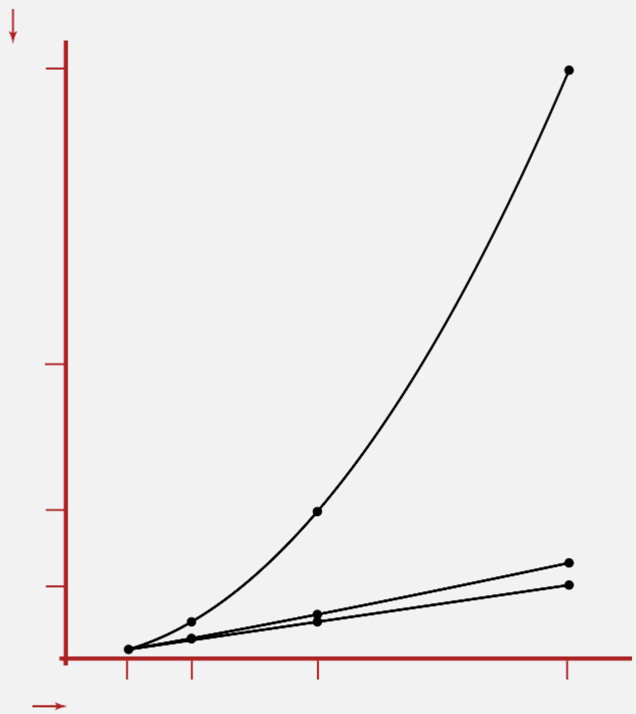
16T

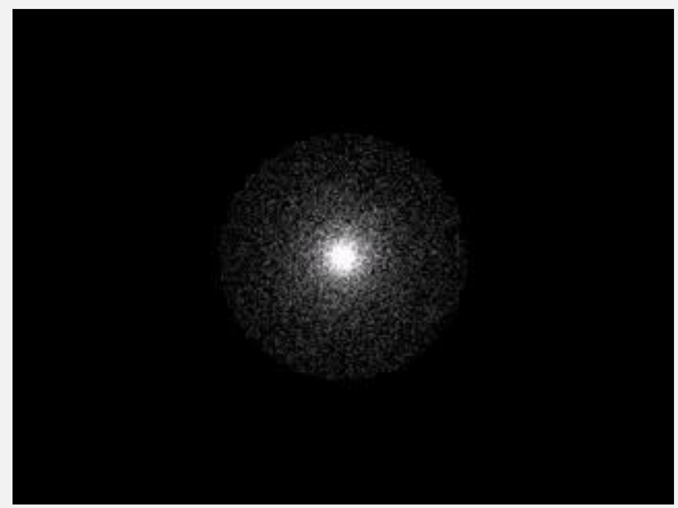
线性思维线性

8T

思维

尺寸 1K 2K 4K 8K





挑战

Q.我的程序能够解决大量的实际输入吗？

洞察力。[Knuth 1970s]使用科学方法来理解性能。

应用于算法分析的科学方法

预测性能和比较算法的框架。科学方法。

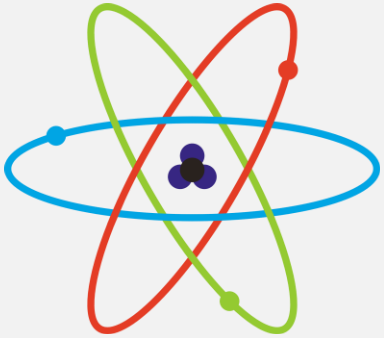
- 观察自然界的一些特征。
- 假设一个与观察结果一致的模型。
- 用假说预测事件。
- 通过进一步观察验证预测。

* 通过重复验证，直到假设和观察结果一致。

原则。

- 实验必须是可重复的。

* 假设必须是可证伪的。



自然界的特征。计算机本身。

http:// algs
4.cs.princeton.edu

1.4 算法的分析

- 介绍
- *观察*
- 数学模型
- 增长顺序的分类
- 算法理论
- 记忆

例子。3-SUM

3-SUM。给出N个不同的整数，有多少个三倍数的总和正好为零？

```
% 更多 8ints.txt
8
30 -40 -20 -10 40 0 10 5

% java ThreeSum 8ints.txt
4
```

	a[i]	a[j]	a[k]	总数
1	30	-40	10	0
2	30	-20	-10	0
3	-40	40	0	0
4	-10	0	10	0

背景。与计算几何学中的问题有很深的关系。

3-SUM：粗暴的算法

```
public class ThreeSum
{
   public static int count(int[] a)
   {
      int N = a.length;
      int count = 0;
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N;
            k++)
               如果（a[i] + a[j] + a[k] ==
                  0） count++。

      返回计数。
   }

   public static void main(String[] args)
   {
      int[] a = In.readInts(args[0]); StdOut.println (
      count(a)) 。
   }
```

检查每一个三联体

为简单起见，忽略整数溢出。

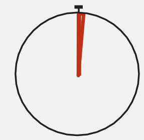
```
}
```

测量运行时间

Q. 如何为一个程序计时？

A. 手册。

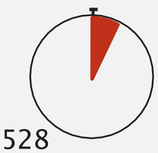
% java ThreeSum 1Kints.txt



滴答滴答

70

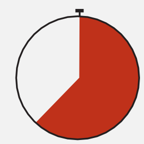
% java ThreeSum 2Kints.txt



滴答滴答 滴答滴答 滴答滴答
滴答滴答 滴答滴答 滴答滴答

528

% java ThreeSum 4Kints.txt



滴答滴答 滴答滴答 滴答滴答
滴答滴答 滴答滴答 滴答滴答
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒滴滴答答嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒滴滴答答嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒嘀嗒
嘀嗒滴滴答答

4039

测量运行时间

Q.如何为一个程序计时？

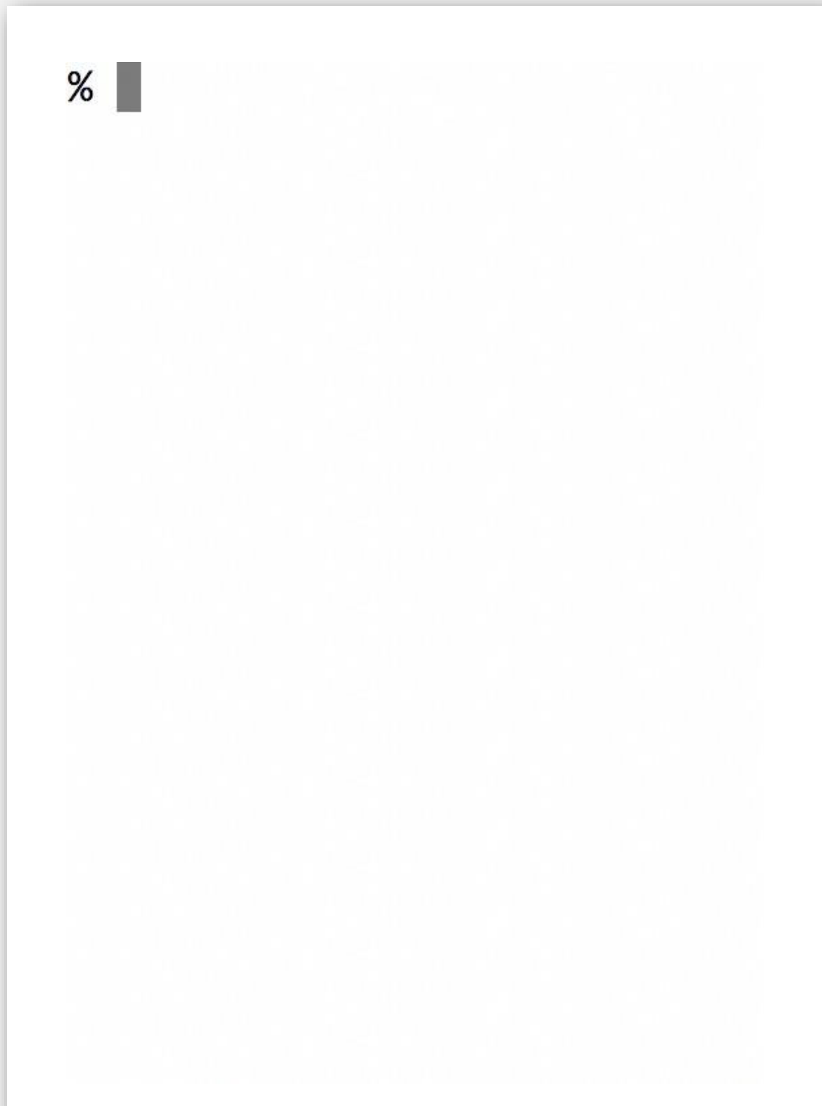
A.自动。

公共课	秒表(属于	stdlib.jar)
	秒表()	*创建一个新的秒表*
双	elapsedTime()	*创建以来的时间（秒）。*

```
public static void main(String[] args)
{
   int[] a = In.readInts(args[0]);
   Stopwatch stopwatch = new Stopwatch();
   StdOut.println(ThreeSum.count(a));
   double time = stopwatch.elapsedTime()
   。
}
```

实证分析

对不同的输入尺寸运行程序，并测量运行时间。



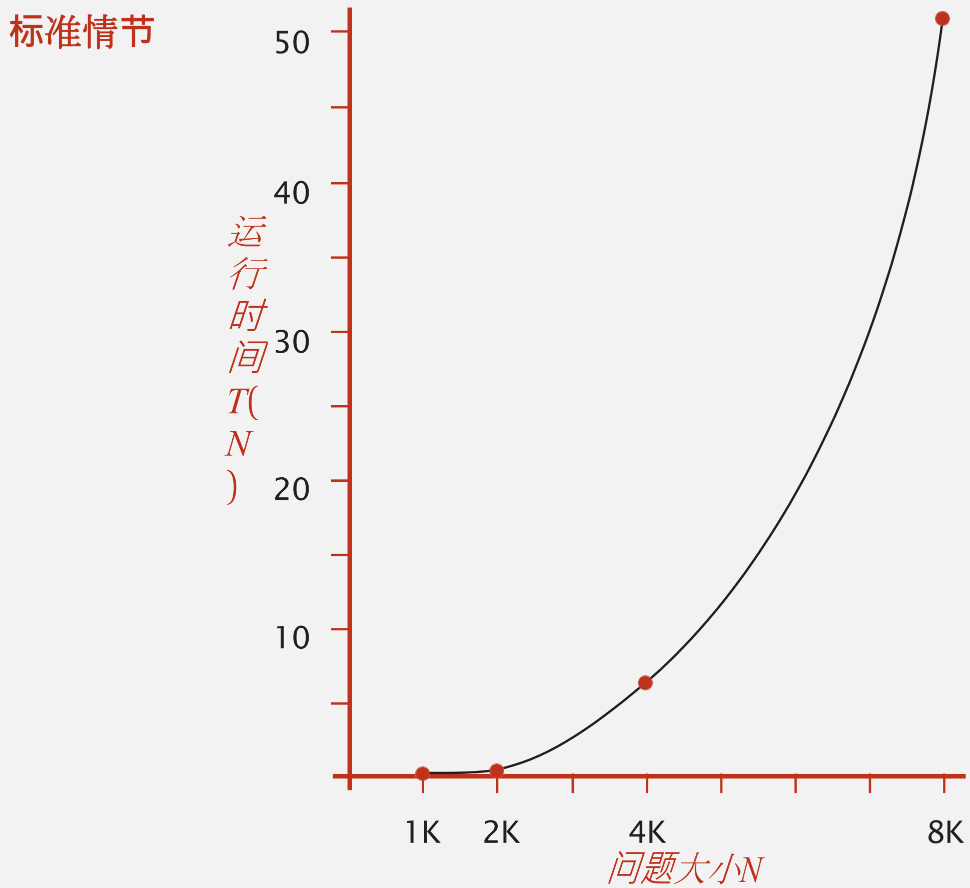
实证分析

对不同的输入尺寸运行程序，并测量运行时间。

N	时间（秒） †
250	0.0
500	0.0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

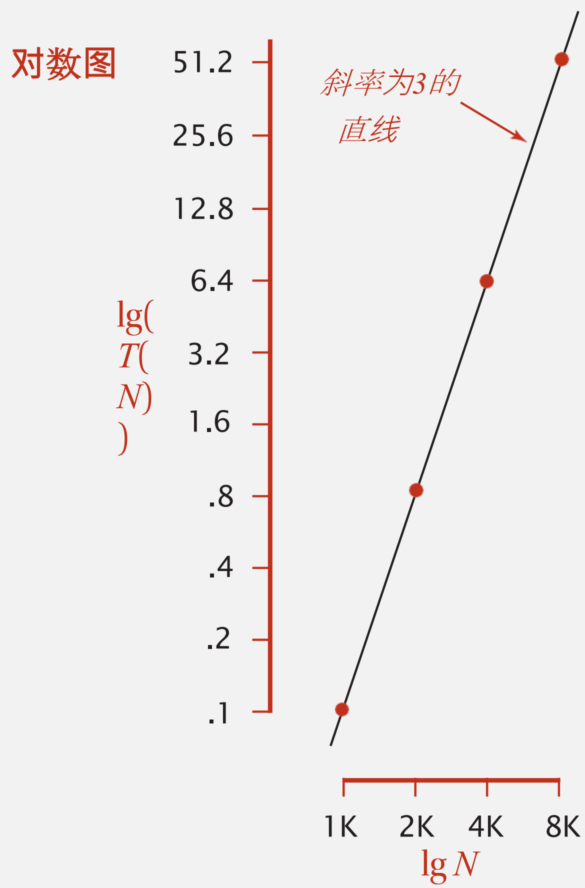
数据分析

标准图。绘制运行时间T（N）与输入大小N的关系。



数据分析

对数图。用对数比例绘制运行时间T（N）与输入大小N的关系图。



$$\lg(T(N)) = b \lg N + c$$
$$b = 2.999$$
$$c = -33.2103$$

$T(N) = a N^b$，其中$a = 2^c$

权力法

坡度

回归。通过数据点拟合直线：$a N^b$。

假设。运行时间约为$1.006\ 10^{-10} N^{2.999}$秒。

预测和验证

假设。运行时间约为$1.006\ 10^{-10} N^{2.999}$秒。

运行时间的 "增长顺序 "约为 N3[敬请关注]

预测。

- 51.0秒，N=8,000。
- 408.1秒，N=16,000。

观察到的情况。

N	时间（秒） †
8,000	51.1
8,000	51.0
8,000	51.1
16,000	410.8

验证了假说!

倍增假说

倍增假说。估计幂律关系中b的快速方法。运行程序，将输入的大小加倍。

N	时间（秒） †	比例	lg比率
250	0.0		–
500	0.0	4.8	2.3
1,000	0.1	6.9	2.8
2,000	0.8	7.7	2.9
4,000	6.4	8.0	3.0
8,000	51.1	8.0	3.0

≈3

假设。运行时间约为$a\,N^{\,b}$，b = lg比率。

注意事项。不能用加倍的假设来确定对数因素。

倍增假说

倍增假说。估计幂律关系中b的快速方法。

Q.如何估计a（假设我们知道b）？

A.运行该程序（对于足够大的N值）并求解a。

N	时间（秒）†
8,000	51.1
8,000	51.0
8,000	51.1

$51.1 = a\ 80003$

$a = 0.998\ 10^{-10}$

假设。运行时间约为$0.998\ 10^{-10} N^3$秒。

几乎相同的假说
与通过线性回归得到的

实验性的算法

系统独立效应。

- 算法。
- 输入数据

决定了幂律中的指数b

。

系统依赖效应。

- 硬件。CPU、内存、缓存、…
- 软件：编译器、解释器、垃圾收集器，……。
- 系统：操作系统、网络、其他应用程序、…

坏消息。难以获得精确的测量。

好消息。比其他科学容易得多，也便宜得多。

决定了幂律中的常数a

算法

ROBERT SEDGEWICK | KEVIN WAYNE

http:// algs 4.cs.princeton.edu

1.4 算法的分析

- 介绍
- *观察*
- 数学模型
- 增长顺序的分类
- 算法理论
- 记忆

http:// algs
4.cs.princeton.edu

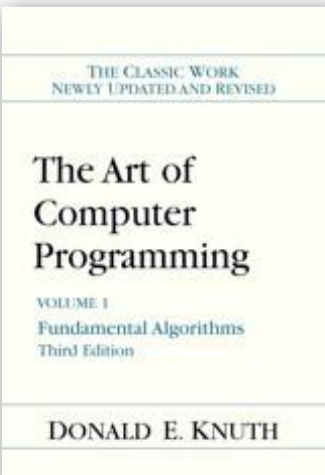
1.4 算法的分析

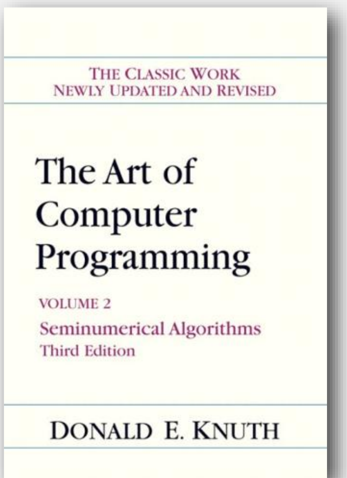
- 介绍
- 观察
- *数学模型*
- 增长顺序的分类
- 算法理论
- 记忆

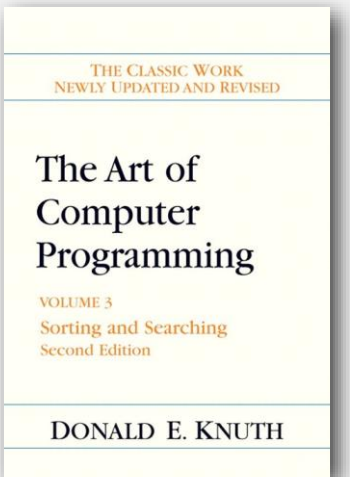
运行时间的数学模型

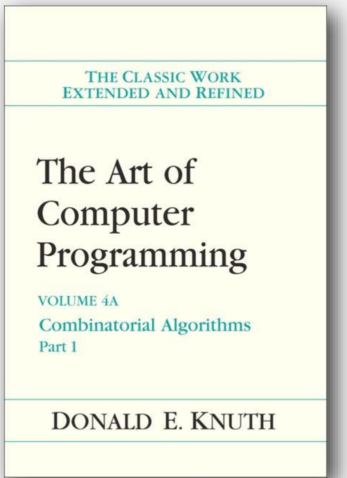
总运行时间：所有操作的成本频率之和。

- 需要对程序进行分析以确定操作集。
- 成本取决于机器、编译器。
- 频率取决于算法、输入数据。









Donald Knuth

1974年图灵奖

原则上，准确的数学模型是可以得到的。

基本业务的成本

操作	例子	纳秒 †
整数加	`a + b`	2.1
整数乘法	`a * b`	2.4
整数除法	`a / b`	5.4
浮点加法	`a + b`	4.6
浮点乘法	`a * b`	4.2
浮点除法	`a / b`	13.5
正弦	`Math.sin(theta)`	91.3
正切线	`Math.atan2(y, x)`	129.0
...	...	...

† 运行OS X的Macbook Pro 2.2GHz，2GB内存

基本业务的成本

操作	例子	纳秒 †
变量声明	ĀĀĀ	c1
任务说明	a = b	c2
整数比较	a < b	c3
阵列元素访问	a[i]	c4
阵列长度	a.长度	c5
1D阵列分配	新的int[N]。	c6 N
二维阵列分配	新的int[N][N]	c7 N^2
字符串长度	s.length()	c8
子串提取	s.substring(N/2, N)	c9
字符串串联	s + t	c10 N

新手的错误。滥用字符串连接法。

例如：1-SUM

Q.有多少条指令是输入大小N的函数？

```
int count = 0。
for (int i = 0; i < N; i++)
   if (a[i] == 0)
      count++。
```

操作	频率
变量声明	2
任务说明	2
少于比较	N + 1
相当于	N
阵列访问	N
增量	N至2 N

例如：2-SUM

Q.有多少条指令是输入大小*N*的函数？

```
int count = 0。
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++。
```

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2}N(N-1) = \binom{N}{2}$$

操作	频率
变量声明	N + 2
任务说明	N + 2
少于比较	½ (n + 1) (n + 2)
相当于	½ N (N - 1)
阵列访问	N (N - 1)
增量	½ N (N - 1) to N (N - 1)

准确计数

简化计算

"有一个衡量计算过程中所涉及的工作量的标准是很方便的，即

使它是一个非常粗略的标准。我们可以

计算各种基本操作在整个过程中的应用次数，然后给予它们不同的

权重。

例如，我们可以计算加法、减法、乘法、除法、记录数字和提取的

数量。

表中的数字。在用矩阵计算的情况下，大部分工作包括乘法

和写下数字，因此我们将只尝试计算乘法和记录的数量。"--

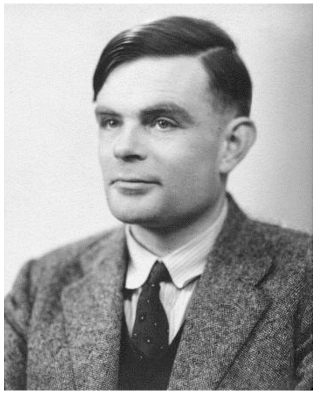
艾伦-图灵

ROUNDING-OFF ERRORS IN MATRIX PROCESSES

By A. M. TURING

(*National Physical Laboratory, Teddington, Middlesex*)

[Received 4 November 1947]



SUMMARY

A number of methods of solving sets of linear equations and inverting matrices are discussed. The theory of the rounding-off errors involved is investigated for some of the methods. In all cases examined, including the well-known 'Gauss elimination process', it is found that the errors are normally quite moderate: no exponential build-up need occur.

简化1：成本模式

成本模型。使用一些基本操作作为运行时间的代理。

```
int count = 0。
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++。
```

$$0 + 1 + 2 + \ldots + (N - 1) = \frac{1}{2} N (N - 1)$$

$N \rightarrow$

2

操作	频率
变量声明	N + 2
任务说明	N + 2
少于比较	½ (n + 1) (n + 2)
相当于	½ N (N - 1)
阵列访问	N (N - 1)
增量	½ N (N - 1) to N (N - 1)

成本模型=数组访问

(我们假设编译器/JVM没有对数组访问进行优化！)

简化2：倾斜符号法

- 估计运行时间（或内存）与输入大小N的关系。

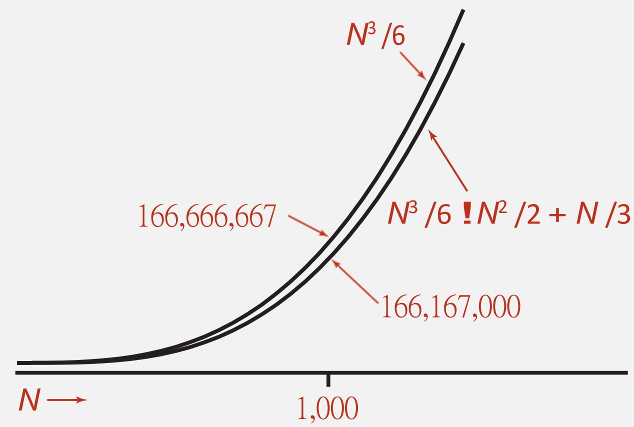
忽略低阶项。
  - 当N很大时，条款可以忽略不计
  - 当N小的时候，我们不关心

前1。 $\frac{1}{6} n^3 + 20 n + 16 \sim \frac{1}{6} N^3$

前2。 $\frac{1}{6} n^3 + 100 n^{4/3} + 56 \sim \frac{1}{6} N^3$

前3。 $\frac{1}{6} N^3 - \underbrace{\frac{1}{2} N^2 + \frac{1}{3} N} \sim \frac{1}{6} N^3$

摒弃低阶术语
(例如，N=1000：50万对1.66亿)



前导期近似值

技术定义：$f(N) \sim g(N)$意味着

肢
体
N $\frac{f(N)}{g(N)}$ 1

简化2：倾斜符号法

- 估计运行时间（或内存）与输入大小*N*的关系。
忽略低阶项。
  - 当*N*很大时，条款可以忽略不计
  - 当*N*小的时候，我们不关心

操作	频率	倾斜符号
变量声明	N + 2	~ N
任务说明	N + 2	~ N
少于比较	½ (n + 1) (n + 2)	~ ½ N2
相当于	½ N (N - 1)	~ ½ N2
阵列访问	N (N - 1)	~ N2
增量	½ N (N - 1) to N (N - 1)	~ ½ N2 到 ~ N2

例如：2-SUM

Q.作为输入大小N的函数，大约有多少个数组访问？

```
int count = 0。
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++。
```

"内循环"

$$0 + 1 + 2 + \ldots + (N - 1) = \frac{1}{2} N (N - 1) \sim \frac{N^2}{2}$$

A.$\sim N^2$个阵列访问。

底线。使用成本模型和tilde符号来简化计数。

例子。3-SUM

Q.作为输入大小N的函数，大约有多少个数组访问？

```
int count = 0。
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N;
      k++)
         如果（a[i] + a[j] + a[k] ==
            0） count++。
```

"内循环"

$$\binom{N}{3} = \frac{n(n-1)(n-2)}{3!} \sim \frac{1}{6}N^3$$

A.~ ½ N^3个阵列访问。

底线。使用成本模型和tilde符号来简化计数。

估计一个离散的总和

Q.如何估计一个离散的总和？A1.学习离散数学课程。

A2.用积分代替总和， 并使用微积分!

例1 ：1+2+...+N。

$$\sum_{i=1}^{N} i \sim \int_{x=1}^{N} x\,dx \sim \frac{1}{2}N^2$$

例2. 1k + 2k + ... + N^k。

$$\sum_{i=1}^{N} i^k \sim \int_{x=1}^{N} x^k\,dx \sim \frac{1}{k+1}N^{k+1}$$

例3. 1 + 1/2 + 1/3 + ... + 1/N.

$$\sum_{i=1}^{N} \frac{1}{i} \sim \int_{x=1}^{N} \frac{dx}{x} = \ln N$$

N N N N N N 1

例4. 3-sum三重循环。

$$\underset{i=1\ \ j=i\ \ k=j}{} 1 \sim \underset{x=1\quad y=x\quad z=y}{} dz\, dy\, dx \sim 6\, N^3$$

运行时间的数学模型

原则上，准确的数学模型是可以得到的。

在实践中。

*公式可能很复杂。

可能需要高级数学。

- 最好把精确的模型留给专家。

成本（取决于机器和编译器）

$$T_N = c_1\,A + c_2\,B + c_3\,C + c_4\,D + c_5\,E$$

A = 阵列访问

B=整数加法

C = 整数比较

D = 增量

E = 变量分配

频次
(取决于算法、输入)

一句话。我们在本课程中使用近似的模型。$T(N) \sim c\,N^3$.

http:// algs 4.cs.princeton.edu

1.4 算法的分析

- 介绍
- 观察
- **数学模型**
- 增长顺序的分类
- 算法理论
- 记忆

http:// algs
4.cs.princeton.edu

1.4 算法的分析

- 介绍
- 观察
- 数学模型
- *增长顺序的分类*
- 算法理论
- 记忆

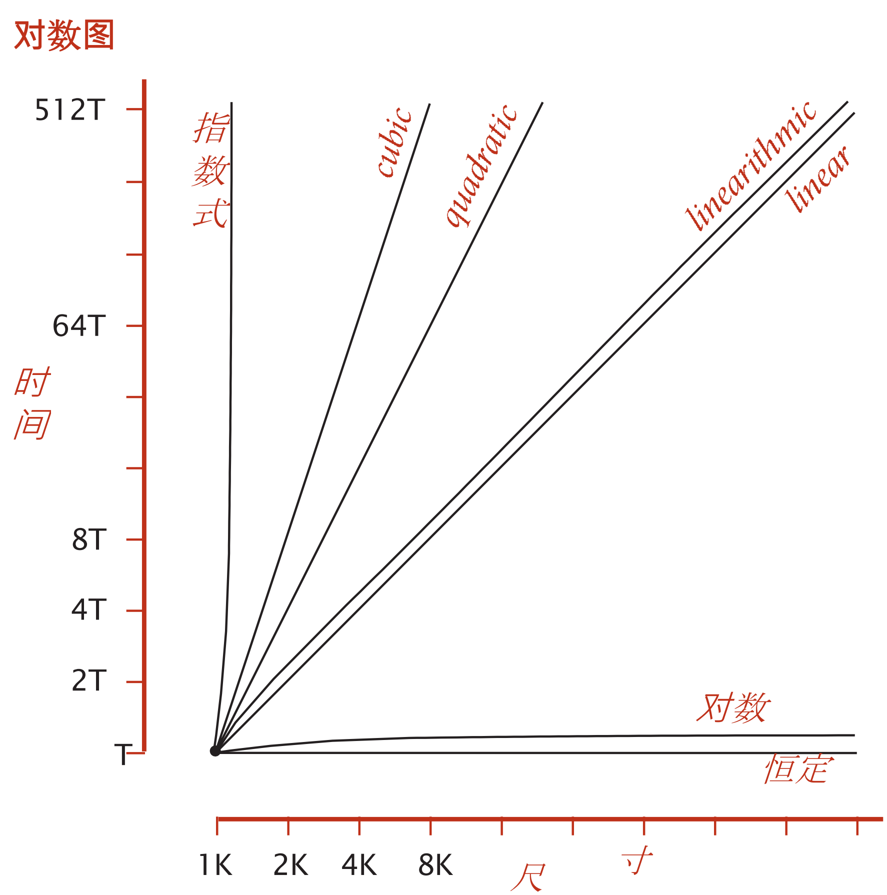
常见的增长顺序分类法

增长顺序 抛弃领先系数

好消息。小编为大家带来了一组函数

1， 对数N, N, N对数N, N^2, N^3,
以及2N

就足以描述典型算法的增长顺序。



512K

典型的增长顺序

常见的增长顺序分类法

增长顺序	名称	典型的代码框架	描述	例子	T(2N) / T(N)
1	恒定的	`a = b + c。`	声明	两个数字相加	1
对数N	对数	`while (N > 1)` `{ N = N / 2; ... }`	一分为二	二进制搜索	~ 1
N	线型	`for (int i = 0; i < N; i++)` `{ ... }`	循环	找到最大	2
N对数N	线性思维	[见mergesort讲座]	分而治之	合并排序	~ 2

N2	二次方	`for (int i = 0; i < N; i++)`
		`for (int j = 0; j < N; j++)`
		`{ ... }`
			双环	检查所有配对	4

N3	立体	`for (int i = 0; i < N; i++)`
		`for (int j = 0; j < N; j++)`
		`for (int k = 0; k < N; k++)`
		`{ ... }`
			三重环	检查所有的三倍体	8

2N	指数式	[见组合搜索讲座]	详尽的搜索	检查所有子集	T(N)

增长顺序的实际意义

增长率	可在几分钟内解决的问题大小			
	1970s	1980s	1990s	2000s
1	任何	任何	任何	任何
对数N	任何	任何	任何	任何
N	数百万	数以千万计的	数以亿计的	十亿
N对数N	数以十万计	数百万	数百万	数以亿计的
N2	几百个	千人	数千人	数以万计
N3	一百个	几百个	千人	数千人
2N	20	20s	20s	30

一句话。需要线性或线性算法来跟上摩尔定律的步伐。

二进制搜索演示

目标。给出一个排序的数组和一个键，找到该键在数组中的索引？二进制搜索。比

较键和中间的条目。

- 太小了，向左走。
- 太大，向右走。
- 相等，发现。

成功寻找33

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

掠夺 嗨

二进制搜索。 Java实现

实施起来很琐碎？

- 第一个二进制搜索发表于1946年；第一个无错误的搜索发表于1962年。

2006年发现的Java `Arrays.binarySearch()`中的漏洞。

```
public static int binarySearch(int[] a, int key)
{
    int lo = 0, hi = a.length-1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        如果（键<a[mid]）hi = mid - 1；否则如果
         （键>a[mid]）lo = mid + 1；否则返回mid
        。
}   }
    返回-1。
```

一个 "三向比较"

不变性。如果键出现在数组a[]中，那么a[lo]键α[hi]。

二进制搜索：数学分析

命题。 二进制搜索最多使用$1+\lg N$键比较来搜索一个大小为N的排序数组。

定义。$T(N)$键比较二进制搜索一个大小≤N的排序子数， 二进制搜索递归。$T(N) \le T$

$(N/2) + 1$对于 $N > 1$， $T(1) = 1$。

左半边或右半边

可以用一个实现
双向比较（而不是三向）。

Pf草图。

$$
\begin{aligned}
T(N) &\le T(N/2) + 1 \\
&\le T(N/4) + 1 + 1 \\
&\le T(N/8) + 1 + 1 + 1 \\
&\ \ \dots \\
&\le T(N/N) + 1 + 1 + \dots + 1 \\
&\ \ 1 + \lg N
\end{aligned}
$$

鉴于

对第一项应用递归法 对第一项应

用递归法

停止应用，T(1) = 1

3-SUM的N2 log N算法

基于排序的算法。

- 第1步：对N个（不同的）数字进行排序。
- 第2步：对于每一对数字a[i]来说和a[j]，二进制搜索-（a[i]+a[j]）。

输入

30 -40 -20 -10 40 0 10 5

分类

-40 -20 -100 5 10 30 40

二进制搜索

(-40, -20)	60
(-40, -10)	50
(-40, 0)	40
(-40, 5)	35
(-40, 10)	30
⋮	⋮
(-40, 40)	~~0~~
⋮	⋮
(-20, -10)	30
⋮	⋮
(-10, 0)	10
⋮	⋮
(10, 30)	~~-40~~
(10, 40)	-50
(30, 40)	-70

才算 $A[i] < A[j] < A[k]$ 以避免重复计算

分析。增长的顺序是$N^2 \log N$。

- 第1步：N^2与插入式排序。
- 第2步。 $N^2 \log N$与二进制搜索。

比较方案

假设。基于排序的3-SUM的$N^2 \log N$算法在实践中明显快于粗暴的N^3算法。

N	时间（秒）
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1

ThreeSum.java

N	时间（秒）
1,000	0.14
2,000	0.18
4,000	0.34
8,000	0.96
16,000	3.67
32,000	14.88
64,000	59.16

ThreeSumDeluxe.java

指导原则。通常情况下，更好的增长顺序在实践中更快。

http:// algs
4.cs.princeton.edu

1.4 算法的分析

- 介绍
- 观察
- 数学模型
- *增长顺序的分类*
- 算法理论
- 记忆

http:// algs
4.cs.princeton.edu

1.4 算法的分析

- 介绍
- 观察
- 数学模型
- 增长顺序的分类
- *算法理论*
- 记忆

分析的类型

最好的情况。成本的下限。

- 由 "最简单 "的输入决定。
- 为所有输入提供一个目 标。

最坏的情况。成本的上限值。

- 由 "最难 "的输入决定的 。
- 为所有输入提供担保。

平均案例。随机投入的预期成本。

- 需要一个 "随机 "输入的模型。
- 提供了一种预测性能的方法。

例1.阵列访问的暴力3-SUM。最好。 $\sim \frac{1}{2} N^3$

平均值。 $\sim \frac{1}{2} N^3$

最坏的情况。 $\sim \frac{1}{2} N^3$

例2.二进制搜索的比较。最好的。

~ 1

平均值。 $\sim \lg N$

最坏的情况。 $\sim \lg N$

分析的类型

最好的情况。成本的下限。最坏的情况。
成本的上限值。平均情况。"预期 "成本。

实际数据可能与输入模型不符？

- 需要理解输入，以有效地处理它。
- 方法1：为最坏情况设计。
- 方法2：随机化，取决于概率保证。

算法理论

目标。

- 确定问题的 "难度"。
- 开发 "最佳 "算法。

办法。

- 在分析中压制细节：分析 "在一个恒定系数内"。
- 通过关注最坏情况，消除输入模型的变异性。

最佳的算法。

- 对任何输入的性能保证（在一个恒定系数内）。
- 没有任何算法可以提供更好的性能保证。

算法理论中常用的记号

记号	提供	例子	简称为	用于
大Theta	渐进式增长顺序	Θ(N2)	½ N2 10 N2 5 N2 + 22 N log N + 3N ⋮	分类算法
大哦	Θ(N2)和更小	O(N2)	10 N2 100 N 22 N log N + 3 N ⋮	设定上限
大欧米茄	Θ(N2)和较大	Ω(N2)	½ N2 N5 N3 + 22 N log N + 3 N ⋮	制定下限

算法理论：实例1

目标。

- 确定问题的 "难度 "并开发 "最优 "算法。
- 例：1-SUM = "*数组中是否有一个0？*"

上限。一个具体的算法。

- 例证。1-SUM的强硬算法：查看每个数组条目。

1-SUM的最佳算法的运行时间为O(N)。

下限。证明没有算法能做得更好。

- 前。必须检查所有N个条目（任何未检查的条目可能是0）。

1-SUM的最佳算法的运行时间为Ω(N)。

最佳的算法。

- 下限等于上限（在一个恒定系数内）。

＊＊＊＊＊＊＊＊＊＊。1-SUM的强行算法是最优的：其运行时间为Θ(N)

。

算法理论：实例2

目标。

- 确定问题的 "难度 "并开发 "最优 "算法。
- 前。3-SUM。

上限。一个具体的算法。

- 前。3-SUM的强硬算法。

3-SUM的最佳算法的运行时间为$O(N^3)$。

算法理论：实例2

目标。

- 确定问题的 "难度 "并开发 "最优 "算法。
- 前。3-SUM。

上限。一个具体的算法。

- 前。3-SUM的改进算法。

3-SUM的最佳算法的运行时间为O(N^2 logN)。

下限。证明没有算法能做得更好。

- 例证。必须检查所有*N*个条目以解决3-SUM。
- 解决3-SUM的最佳算法的运行时间为Ω(*N*)。

未解决的问题。

＊3-SUM的最佳算法？

- 3-SUM的次方程算法？
- 3-SUM的二次方下限？

算法设计方法

开始。

- 开发一种算法。

证明了一个下限。

差距？

- 降低上限（发现一种新的算法）。
- 提高下限（更困难）。

算法设计的黄金时代。

- 1970s-·
- 许多重要问题的上限值稳步递减。
- 许多已知的最优算法。

注意事项。

- 过于悲观地关注最坏的情况？
- 需要比 "在一个恒定系数内 "更好的方法来预测性能。

常用的记号

记号	提供	例子	简称为	用于
Tilde	领导层	~ 10 N2	10 N2 10 N2 + 22 N log N 10 n2 + 2 n + 37	提供近似的模型
大Theta	渐近增长率	Θ(N2)	½ N2 10 N2 5 N2 + 22 N log N + 3N	分类算法
大哦	Θ(N2)和更小	O(N2)	10 N2 100 N 22 N log N + 3 N	设定上限
大欧米茄	Θ(N2)和较大	Ω(N2)	½ N2 N5 N3 + 22 N log N + 3 N	制定下限

常见的错误。把大奥解释为近似模型。本课程。注重近似模型：使用Tild–

notation

http:// algs
4.cs.princeton.edu

1.4 算法的分析

- 介绍
- 观察
- 数学模型
- 增长顺序的分类
- 算法理论
- 记忆

http:// algs
4.cs.princeton.edu

1.4 算法的分析

- 介绍
- 观察
- 数学模型
- 增长顺序的分类
- 算法理论
- 记忆

基础知识

位。0或1。

NIST　　大多数计算机科学家

字节。8位。

兆字节（MB）。100万或2^{20}字节。吉字节（GB）。10亿或2^{30}字节。

64位机器。我们假设一台64位机器有8个字节的指针。

- 可以寻址更多的内存。
- 指针使用更多的空间。

一些JVM将普通的对象指针 "压缩 "到4个字节，以避免这种成本。

原始类型和数组的典型内存用量

类型	字节
布尔型	1
字节	1
炭	2
黠黠	4
浮动	4
长	8
双	8

用于原始类型

类型	字节
char[]	2N + 24
卫星	4N + 24
双倍[]	8N + 24

适用于一维数组

类型	字节
char[][]	~ 2 M N
ĀĀĀĀ	~ 4 M N
双倍[][]	~ 8 M N

用于二维数组

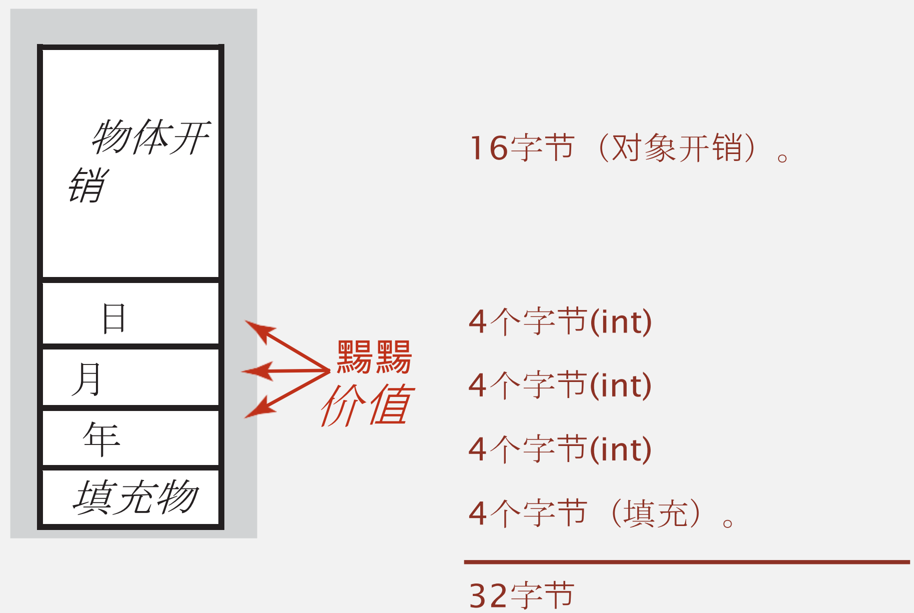
Java中对象的典型内存用量

对象开销。16个字节。参考资料。8个字节。

填充。每个对象使用8个字节的倍数。

例1.一个Date对象使用32字节的内存。

```
public class Date
{
        私营      黬黬   天。
        私营      黬黬   月。
...     私营      黬黬   年。
}
```



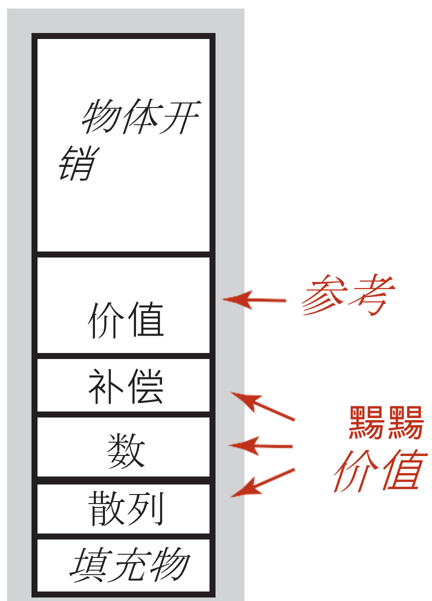
Java中对象的典型内存用量

对象开销。16个字节。参考资料。8个字节。

填充。每个对象使用8个字节的倍数。

例2.一个长度为*N*的原始字符串使用~ 2N字节的内存。

```
public class String
{
   private char[] value;
   private int offset;
   private int count;
   private int hash;
...
}
```



16字节（对象开销）。

8字节（对数组的引用） 2N +
24字节（char[]数组）

4个字节(int)

4个字节(int)

4个字节(int)

4字节（填充） 2N

+ 64字节

典型的内存使用总结

一个数据类型值的总内存使用量。

- 原始类型。4字节为`int`，8字节为`double`，...。
- 对象参考。8个字节。
- 数组：24字节+每个数组条目的内存。
- 对象：16字节+每个实例变量的内存
  + 8字节，如果是内部类（用于指向包围类的指针）。

填充：四舍五入到8字节的倍数。

浅显的内存使用。不要计算被引用的对象。

深度使用内存。如果数组条目或实例变量是一个引用，为被引用的对象增加内存（递归）。

例子

Q.`WeightedQuickUnionUF`使用多少内存作为N的函数？使用**tilde**符号来简化你的答案。

```
public class WeightedQuickUnionUF
{
   private int[] id;
   private int[] sz;
   私有的int count。

   public WeightedQuickUnionUF(int N)
   {
      id = new int[N];
      sz = new int[N];
      for (int i = 0; i < N; i++) id[i] = i
      。
   }  for (int i = 0; i < N; i++) sz[i] = 1;
   ...
}
```

16个字节
(物体开销)

8 + (4N + 24) 每个引用

+ int[] 阵列 4字节(int)

4个字节（填充）。

A.$8\,N + 88 \sim 8\,N$字节。

http:// algs 4.cs.princeton.edu

1.4 算法的分析

- *介绍*
- *观察*
- *数学模型*
- *增长顺序的分类*
- *算法理论*
- *记忆*

转动曲柄：总结

实证分析。

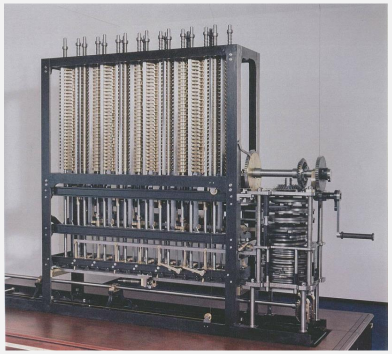
- 执行程序以进行实验。

假设为幂律，并提出运行时间的假设。

- 模型使我们能够进行预测。

数学分析。

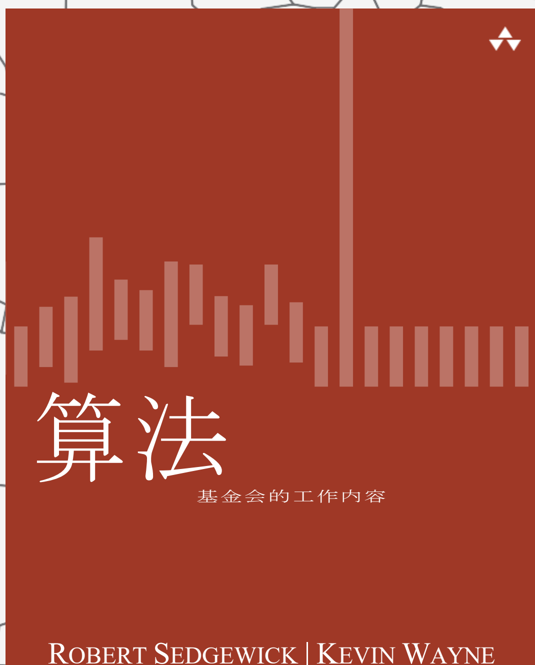
- 分析算法，计算操作的频率。
- 使用倾斜符号来简化分析。
- 模型使我们能够解释行为。



科学方法。

- 数学模型独立于某一特定系统；适用于尚未建成的机器。

* 经验分析是验证数学模型和进行预测的必要条件。



http:// algs 4.cs.princeton.edu

1.4 算法的分析

- *介绍*
- *观察*
- *数学模型*
- *增长顺序的分类*
- *算法理论*
- *记忆*