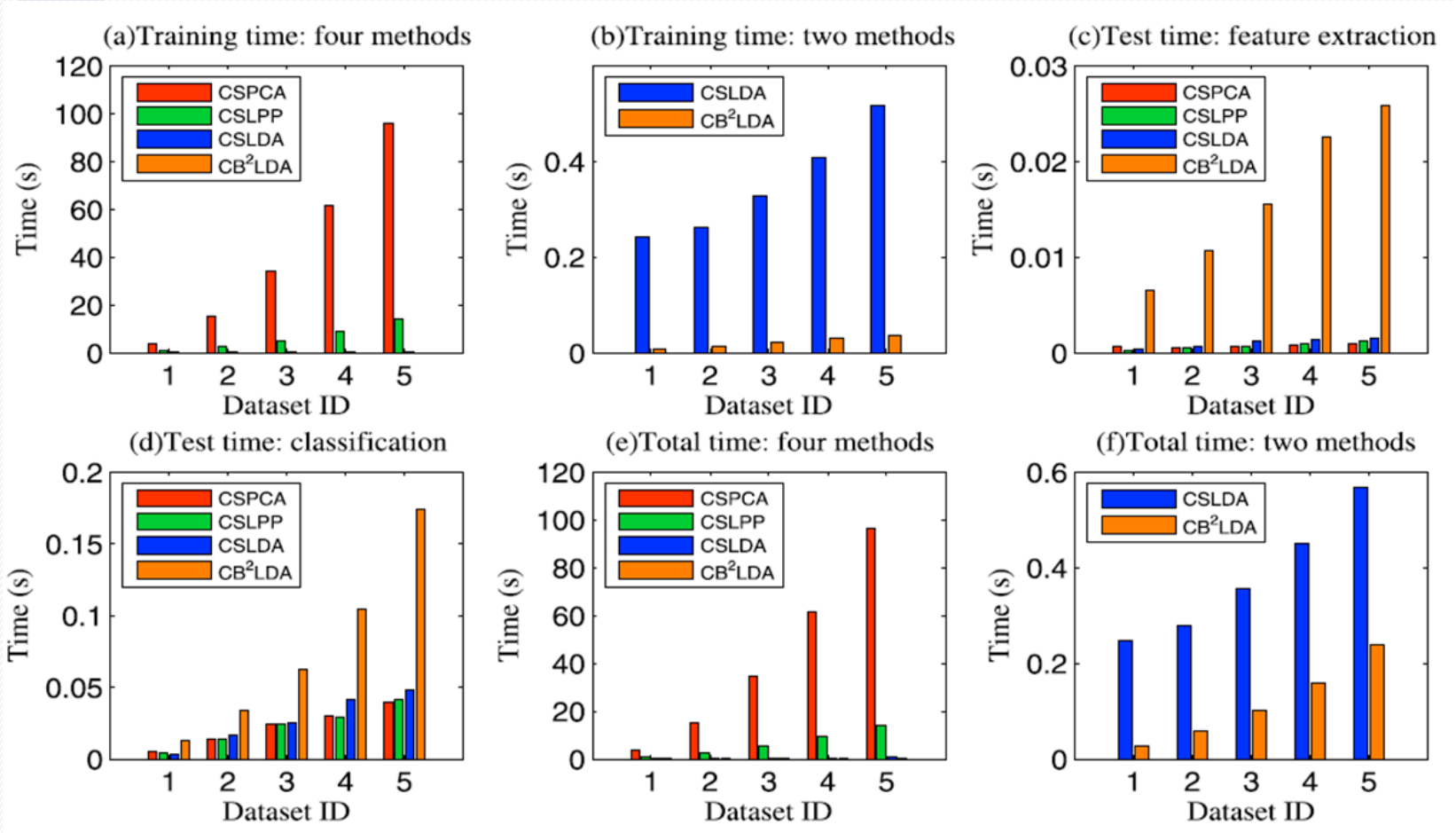
西南大學

算法的时间复杂度

张里博 lbzhang@swu.edu.cn

算法的时间效率分析

能否用算法在计算机上的运行时间作为度量标准?



算法的时间效率分析

实验结果

1. 与输入，软硬件环境以及具体实验设置有关；
2. 与程序开发质量有关（开发经验等）。

Table 2
Experimental parameter settings.

Data set	M	N_G	N_I	λ_{PN}: λ_{NP}: λ_{BN}: λ_{BP}
Training set	9	20	40	30 : 6 : 4 : 2
Test set	9	29	40	30 : 6 : 4 : 2

5. Experiments

To evaluate the performance of the proposed granular feature method and sequential 3WD model, we conducted an experimental analysis based on the well-known PIE (Pose, Illumination, and Expression) benchmark database [40]. The PIE database contains 41,368 face images of 68 subjects with variations in the pose, illumination, and expression. The front view subset of all the subjects was used in our experiments. All of the images were cropped to 64 × 64 pixels. Some examples of the facial images used in the experiment are shown in Fig. 3. All of the experiments were performed on a computer with an Intel i7-4790k (4G processor) and 16 GB RAM, and the method was programmed in MATLAB (version R2014a).

是否有独立于上述因素的评价指标？
即只跟问题的规模、算法的输入和算法本身相关的指标

算法的时间复杂度

2.6 时间复杂度分析

设 n 表示网络节点总数量，m 表示边的
K 表示网络中节点的平均度数，t 表示 Page
法中迭代的最大次数，T 表示标签传播过程
大迭代次数。

第一阶段的时间复杂度：首先初始化
PR 值的时间复杂度为 $O(n)$；接着在每一
中计算节点的 PR 值，直到达到最大迭代次
以迭代完成的时间复杂度为 $O(K \cdot n \cdot t)$
中使用已存在的基数排序算法按照节点重
升序对节点排序，所以时间复杂度为 $O(n)$
公式计算节点间影响力值的时间复
$O(2K^2 \cdot n)$。

D. Computational Analysis

Computational complexity is an important issue when estimating the performance of an algorithm [52], [49], [53]. In this section, we make a discussion on the computational cost of Algorithm 1. In EGSNR, the class weights $\mathbf{w}$ is computed once in advance, thus the major computational cost is spent number of training samples n, the complexity of SVD operation for updating $\mathbf{e}_1$ is $O(pq^2)$ assuming $p > q$. The cost of computing $\mathbf{e}_2$ is $O(m)$. The optimization of $\mathbf{x}$ involves matrix inverse and multiplication computation. Noting that $(\mathbf{D}^T\mathbf{D}+\mathbf{I})^{-1}$ is fixed in iterations, we can compute and store it by pseudo-inverse in advance. Thus, the cost of $\mathbf{x}$ is $O(n^2)$. For updating $\mathbf{g}$, the computational complexity is $O(ct^2)$, assuming each class has t training samples. Similar to $\mathbf{x}$, the computational consumption of $\mathbf{w}$ is $O(c^2)$. The time cost of computing $\mathbf{v}$ is $O(c)$. Thus, the total computational cost of Algorithm 1 is $O(k(pq^2 + m + n^2 + ct^2 + c^2))$ if there are k iterations.

目录

PART 01

基本运算次数

一、基本运算次数

- **基本运算：**算法运行过程中起主要作用且**花费时间最多**的运算。
  - 包括：加法，乘法，比较，元素的交换（移动）等；
  - 具体场景下的约定，例如排序和**检索问题**中，一般约定**比较运算**为基本运算

- **基本运算运行时间的差异可以忽略。因此，基本运算次数**常被用来**衡量算法的效率**。

一、基本运算次数

以比较为基本运算

输入：无序数组$L[1\cdots n]$；

输出：数组$L[1..n]$中最大的元素值；

顺序检索最大值算法

将第一个作为最大值，从前向后依次比较，遇见更大的就替换

规模不同的情况下，算法的基本运算次数唯一么？

一、基本运算次数

输入：无序数组$L[1\cdots n]$，元素 x;

输出：若元素 x 存在于数组 L 中，则输出 x 的位置下标；

若元素 x 不存在于数组 L 中，则输出 0；

顺序检索（查找）算法

从前向后依次比较数组 L 中的元素与 x 的大小

规模不同的情况下，算法的基本运算次数唯一么？

PART 02

时间复杂度函数

二、时间复杂度函数

时间复杂度函数（简称为时间复杂度）：将基本运算次数表示为问题规模的函数。

输入：无序数组$L[1\cdots n]$；

输出：数组$L[1..n]$中最大的元素值；

顺序检索最大值算法

将第一个作为最大值，从前向后依次比较，遇见更大的就替换

假设数组L有n个元素，顺序检索最大值算法的时间复杂度函数

共需要进行$n-1$次比较，时间复杂度函数为$T(n)=n-1$；

二、时间复杂度函数

算法1.2 **Search_max(*L*)**

输入：无序数组*L*[1..*n*];

输出：数组*L*[1..*n*]中最大的元素值;

伪码表示：

```
1.  x ← L[1];
2.  for  j ← 2  to  n do
3.      if L[j] > x then
4.          x ← L[j];
5.      end
6.  end
```

5	2	4	6	1	3

以比较为基本运算

有*n*个元素时，共需要进行*n*-1比较，基本运算次数可以表示为：*T*(*n*)=*n*-1。

同等问题规模下，同一个算法的时间复杂度函数唯一确定么？

输入：无序数组$L[1\cdots n]$，元素 x;

输出：若元素 x 存在于数组 L 中，则输出 x 的位置下标；

若元素 x 不存在于数组 L 中，则输出 0；

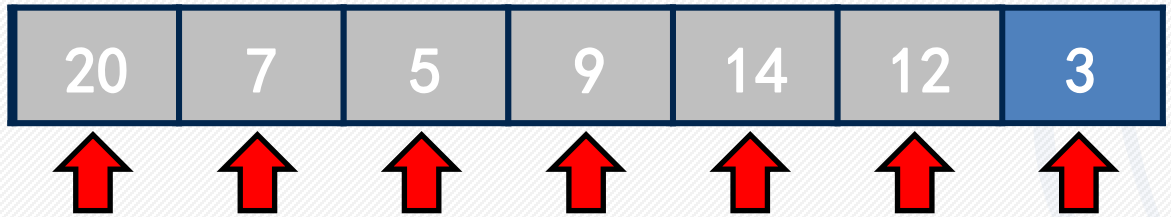
20	7	5	9	14	12	2

顺序检索

$x = 20$

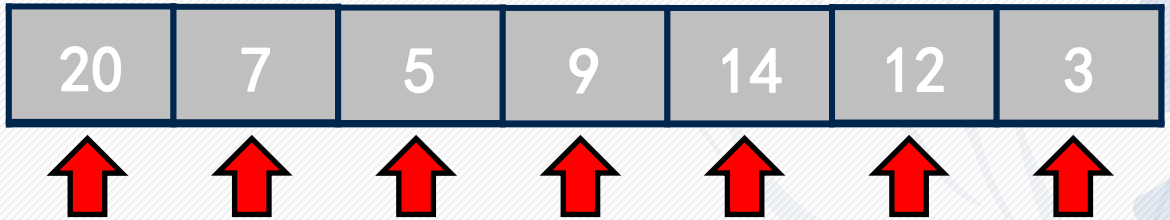
比较了 1 次，成功找到，输出下标为 1

$x = 3$



比较了 7 次，成功找到，输出下标为 7

$x = 8$



比较了 7 次，未成功找到，输出0；

二、时间复杂度函数

输入：无序数组$L[1\cdots n]$，元素 x;

输出：若元素 x 存在于数组 L 中，则输出 x 的位置下标；

若元素 x 不存在于数组 L 中，则输出 0；

20	7	5	9	14	12	2

相同问题规模下，算法的时间复杂度与输入有关

最好情况下的时间复杂度： x是L 的第一个元素，**时间复杂度为 1；**

最坏情况下的时间复杂度： x 是L 的最后一个元素或不在 L 中，

时间复杂度为 n;

平均情况下的时间复杂度： x**每种情况的概率**乘以对应的时间复杂度

二、时间复杂度函数

假如你委托一个公司开发算法，你希望对方的承诺最好情况下，最坏情况下，还是平均情况下的运行时间？

最好情况下？

平均情况下？能否预知每种合法输入出现的概率？

“最坏情况下的时间复杂度，代表了对用户的承诺”

麻省理工学院（MIT） 《Introduction To Algorithms》

因此，最常用的是最坏情况下的时间复杂度

算法1.2 **Search_max(*L*)**

输入：数组*L*[1..*n*]，其元素没有按照顺序排列;

输出：数组*L*[1..*n*]中最大的元素值;

伪码表示：

```
1.  x ← L[1];
2.  for  j ← 2  to  n do
3.      if L[j] > x then
4.          x ← L[j];
5.      end
6.  end
```

以比较为基本运算

有*n*个元素时，共需要进行*n*-1比较，基本运算次数可以表示为：*T*(*n*)=*n*-1。

二、时间复杂度函数

- 求解同一个问题时：
- 算法A（最坏情况下）的时间复杂度函数是$f(n)=n^2+100n$;
- 算法B （最坏情况下）的时间复杂度函数是$g(n)=2n^2$;
- 算法C（最坏情况下）的时间复杂度函数是$h(n)=0.5n^3$;

- 哪个算法效率高（$f(n)$、$g(n)$和$h(n)$的大小关系）？

- 函数大小关系与n的取值有关。
- 关注规模大的时候，还是规模小的时候？

故事：舍罕王赏麦

- 印度的舍罕王打算重赏“国际象棋的发明人”-宰相西萨·班·达依尔（Sissa Ben Dahir）。国王问他有何要求，这位聪明的大臣胃口看来并不大。
- 他跪在国王面前说：“陛下，请您在这张棋盘的第一个小格内，赏给我1粒麦子，在第二个小格内给2粒，第三格内给4粒，照这样下去，每一个小格内都比前一小格加一倍。陛下啊，把这样摆满棋盘上所有64格的麦粒都赏给您的仆人我吧！”国王一听，认为这区区赏金，微不足道。于是满口答应道：“爱卿，你所要求的并不多啊！你当然会如愿以偿。”说着，他令人把一袋麦子拿到宝座前。

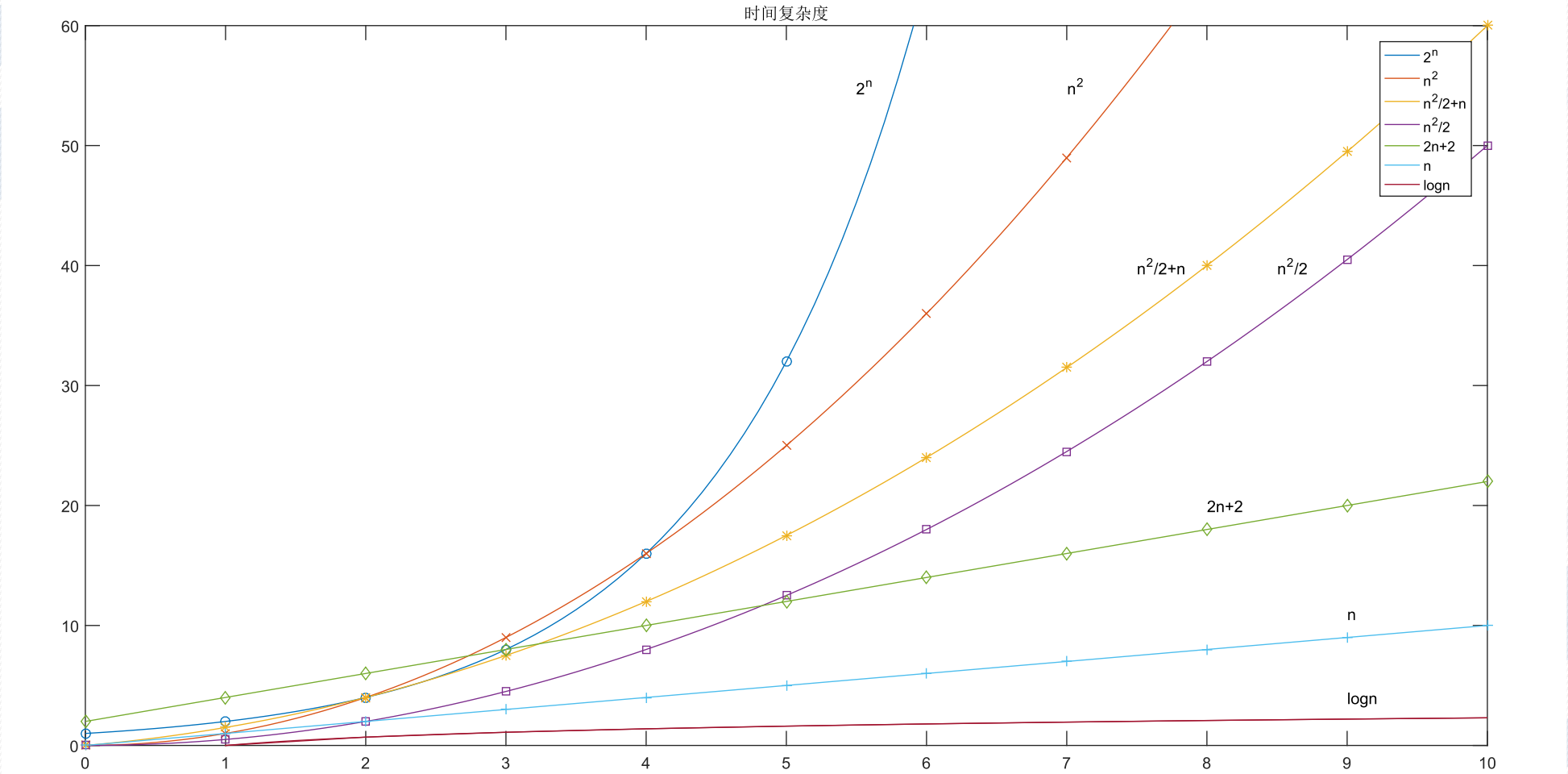
故事：舍罕王赏麦

按照第一格内放1粒，第二格内放2粒，第三格内放4粒，……还没有放到20格，一袋麦子就已经完了。于是，一袋又一袋的麦子被扛到国王面前来。但是，麦粒数一格接一格地增长得那样迅速，很快就可以看出，如果要计算到第64格，即使拿来全印度的粮食，国王也兑现不了他对西萨·班许下的诺言。

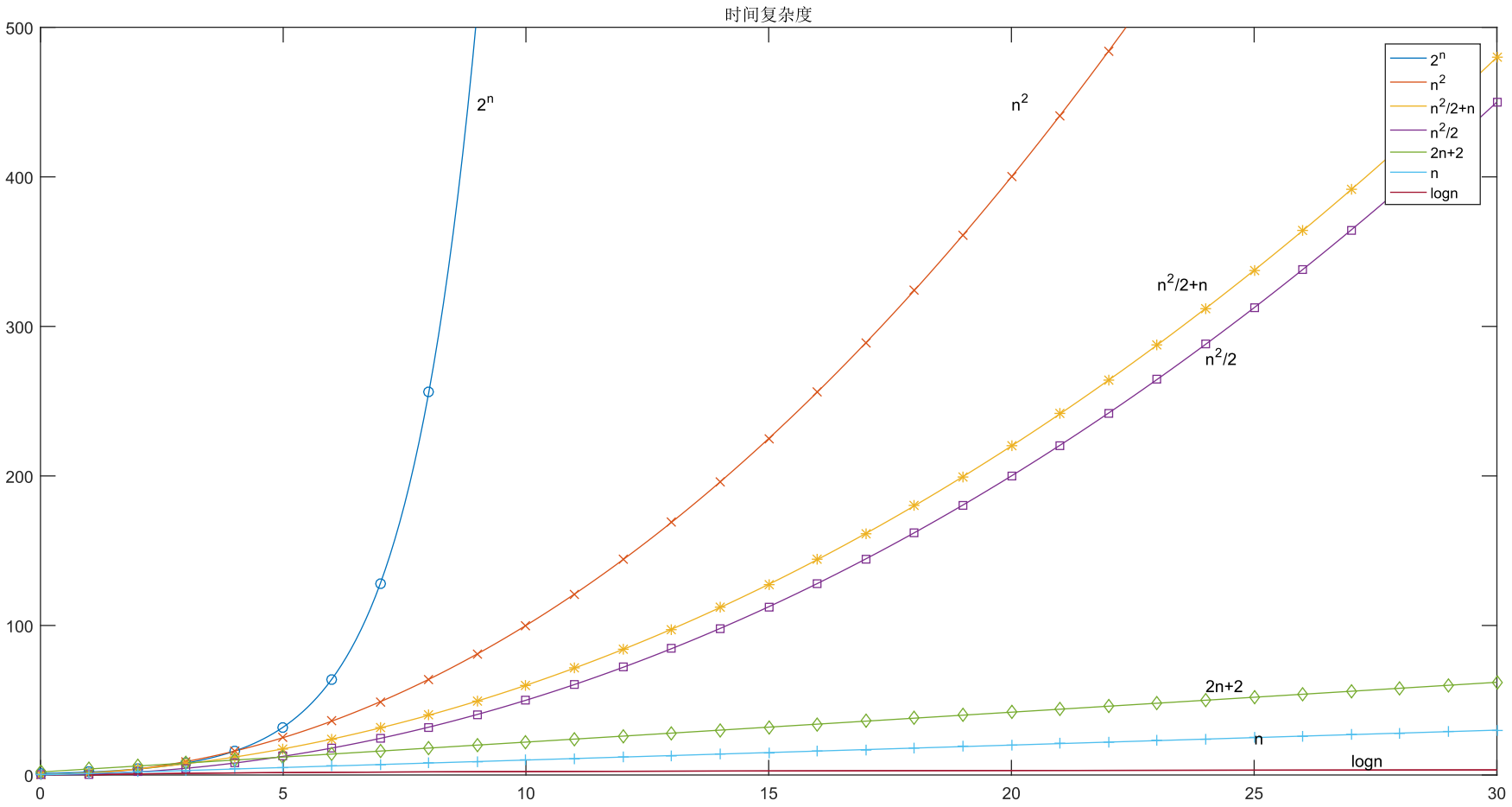
格子数	1	2	3	… …	23	… …	63	64
	2^1	2^2	2^3	… …	2^{23}	… …	2^{63}	2^{64}
小麦数量	2	4	8	… …	8388608	… …	9223372036854775808	18446744073709551616
累计数量	3	7	15	… …	16777215	… …		

2^{64}粒小麦是人类一百年的粮食产量

不同函数的增长速度



三、渐进时间复杂度



PART 03

渐进时间复杂度

三、渐进时间复杂度

- 时间复杂度函数的渐近表示，也被称为算法的渐进时间复杂度，是被广泛认同的衡量算法效率的方式。

- 目的：问题规模大的时候，算法的计算效率。即问题规模很大的时候，时间复杂度函数的增长速率。

- 实例规模大的时候，影响时间复杂度函数增长速率的关键因素是什么？

三、渐进时间复杂度

- 问题规模很大时，不同阶的函数差别很大。

- 因此，问题规模很大时，最高阶是影响函数增长速率的关键因素，而常数、系数、（除最高阶外的）低阶函数的影响不大。
  - 即在n很大的时候， $n^2/2$和n^3之间的差异是较大的，而n和5n之间的差异是微小的甚至可以忽悠不计的。
  - n^2，　$0.5n^2$，　n^2+4，　$10n^2-3n+6$

- 如何通过数学工具表示和实现呢？

定义1.1 设 f 和g是定义域为自然数集 N上的函数.

(1) 若存在正数 c 和 n_0使得对一切 $n \geq n_0$有$0 \leq f(n) \leq cg(n)$ 成立, 则称 $f(n)$ 的渐近上界是 $g(n)$，记作 $f(n) = O(g(n))$.

$$\boldsymbol{f(n) = 2n^2 + 100n}\ ,\ \boldsymbol{g(n)= n^3},\ \boldsymbol{h(n)= n^2}$$

$f(n) = O(g(n))$ 证明：令$c=2$， $n_0=100$，当$n \geq n_0$时，有$cg(n) - f(n)=2n(n^2 - n - 50)$

令$t(n)= n^2 - n - 50$，$t(n_0)>0$;

并且$t'(n)=2n - 1$, 当$n \geq n_0$时， $t'(n)>0$;

因此，当$n \geq n_0$时， $t(n)=cg(n) - f(n)= t(n_0)>0$，$f(n) = O(g(n))$.

$f(n) = O(h(n))$ 令$c=3$， $n_0=100$，证明过程与上面类似

定义1.1 设 f 和g是定义域为自然数集 N上的函数.

(4) 若对**任意正数 c** 都存在 n_0，使得当 $n \geq n_0$ 时有 $0 \leq f(n) < cg(n)$ 成立,则称 $f(n)$的**非紧上界(非渐近紧确上界)**是 $g(n)$，记作 $f(n)= o(g(n))$.

$f(n) = 2n^2 + 100n$ ， $g(n)= n^3$， $h(n)= n^2$　　$f(n) = o(g(n))$

证明：任意c，令$cg(n) - f(n)=cn^3 - 2n^2 - 100n=n*(cn^2 - 2n - 100)=n*t(n)$，$t'(n)=2cn-2$；

当$n \geq 1/c$时， $t'(n) >0$；

对于$t(n)=0$，$\Delta=400c+4>0$；假设$t(n)=0$两个根为x_1和x_2 ($x_1<x_2$)，

则$x_1+x_2=2/c>0$，$x_1*x_2=-100/c<0$。因此，$x_1<0<x_2$。

令$n_0=\max\{1/c, x_2\}+1$， $t(n_0)>0$， $t'(n_0) >0$

当$n \geq n_0$时， $t'(n) >0$， $t(n)\geq t(n_0)>0$。

因此，$f(n) = o(g(n))$.

定义1.1 设 f 和g是定义域为自然数集 N上的函数.

(4) 若对**任意正数 c** 都存在 n_0，使得当 $n \geq n_0$ 时有 $0 \leq f(n) < cg(n)$ 成立,则称 $f(n)$的**非紧上界(非渐近紧确上界)**是 $g(n)$，记作 $f(n)= o(g(n))$.

$$f(n) = 2n^2 + 100n \text{ ， } g(n)= n^3 \text{， } h(n)= n^2$$

$$f(n) = o(h(n))?$$

反例：

c=1，不存在n_0使得$n \geq n_0$ 时，

$ch(n) - f(n) = -n^2 - 100n = -n(n+100) > 0$恒成立；

定理1.1 设 f 和g是定义域为自然数集合的函数.

如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=0$，那么 $f(n)=o(g(n))$.

$f(n)=2n^2+100n$，$g(n)=n^3$，$h(n)=n^2$

$f(n)=o(g(n))$，$f(n)\neq o(h(n))$

- $f(n)=O(g(n))$，$f(n)=o(g(n))$，含义是什么？

$f(n)=O(g(n))$：$a_f \le a_g$;

$f(n)=o(g(n))$：$a_f < a_g$;

最高阶之间的关系

定义1.1 设 f 和g是定义域为自然数集 N上的函数.

(2) 若存在正数 c 和 n_0，使得对一切 $n \geq n_0$有 $0 \leq cg(n) \leq f(n)$ 成立, 则称 $f(n)$ 的**渐近的下界**是 $g(n)$，记作 $f(n) = \Omega(g(n))$.

$f(n)=n^2+100n$ ， $g(n)=n$ $\quad f(n)=\Omega(g(n))$. $\quad f(n)=\Omega(n^2)$.

(3) 若 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$, 则称 $f(n)$的**渐近紧界**是 $g(n)$，记作记作 $f(n)=\Theta(g(n))$.

$f(n)=n^2+100n$ ， $g(n)=n^2$ $\quad f(n)=\Theta(g(n))$.

(5) 若对任意正数 c 都存在 n_0，使得当 $n \geq n_0$ 时有 $0 \leq cg(n) < f(n)$成立, 则称 $f(n)$的**非紧下界**是 $g(n)$，记作 $f(n)=\omega(g(n))$.

$f(n)=n^2+100n$ ， $g(n)=n$ $\quad f(n)=\omega(g(n))$.

定理1.1 设 f 和g是定义域为自然数集合的函数.

(1)如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}$ 存在且等于某个常数$c>0$，那么

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

(2)如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=0$，那么 $f(n)=o(g(n))$.

(3)如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=+\infty$，那么 $f(n)=\omega(g(n))$.

例4 设 $f(n)=\frac{1}{2}n^2-3n$， 证明 $f(n)=\Theta(n^2)$.

证 因为

$$\lim_{n\to+\infty}\frac{f(n)}{n^2}=\lim_{n\to+\infty}\frac{\frac{1}{2}n^2-3n}{n^2}=\frac{1}{2}$$

根据定理1.1有 $f(n)=\Theta(n^2)$.

三、渐进时间复杂度

渐近上界 $f(n)= O(g(n))$ ： $a_f \le a_g$;

渐近下界 $f(n)= \Omega(g(n))$ ： $a_f \ge a_g$;

渐近紧界 $f(n)= \Theta(g(n))$ ： $a_f = a_g$;

非紧上界 $f(n)= o(g(n))$ ： $a_f < a_g$;

非紧下界 $f(n)= \omega(g(n))$ ： $a_f > a_g$.

问题规模很大时，最高阶是影响函数增长速率的关键因素，而常数、系数、（除最高阶外的）低阶函数影响不大。

界函数的定义和性质

定理1.2 （传递性）

设 f, g, h是定义域为自然数集合的函数，

(1) 如果 $f=O(g)$ 且 $g=O(h)$，那么 $f=O(h)$.

(2) 如果 $f=\Omega(g)$ 且 $g=\Omega(h)$，那么 $f=\Omega(h)$.

(3) 如果 $f=\Theta(g)$ 和 $g=\Theta(h)$，那么 $f=\Theta(h)$.

(4) 如果 $f= o(g)$且 $g=o(h)$ ，那么 $f(n)=o(h)$;

(5) 如果 $f= \omega(g)$且 $g=\omega(h)$ ，那么 $f(n)=\omega(h)$;

三、渐进时间复杂度

- 求解同一个问题时
- 算法A（最坏情况下）的时间复杂度函数是 $f(n)=n^2+100n$;
- 算法B （最坏情况下）的时间复杂度函数是$g(n)=2n^2$;
- 算法C（最坏情况下）的时间复杂度函数是$h(n)=0.5n^3$;

- $f(n) = \Theta(g(n))$， $f(n) = o(h(n))$
- 因此，从渐近时间复杂度的角度（问题规模足够大），**算法A与算法B时间效率没有明显差别，但都优于算法C。**

PART 04

分析与练习

算法最坏情况下的时间复杂度

输入：无序数组$L[1\cdots n]$，元素 x;

输出：若元素 x 存在于数组 L 中，则输出 x 的位置下标；

若元素 x 不存在于数组 L 中，则输出 0；

```
Search(x)
1.  for  i ← 1  to  n do
2.      if  L[i] == x then
3.          return i;
4.      end
5.  end
6.  return 0
```

只将比较作为基本运算

n次

比较、赋值都作为基本运算

n次
n次
1次

1次

渐近时间复杂度

$T(n)=n=O(n)$

$T(n)=2n+1=O(n)$

算法最坏情况下的时间复杂度

算法1.2 Search_max(*L*, *x*)
输入：无序数组*L*[1…*n*]
输出：数组*L*[1..*n*]中最大的元素值；

```
1.  x ← L[1];
2.  for  j ← 2  to  n do
3.      if L[j] > x then
4.          x ← L[j];
5.      end
6.  end
7.  return x
```

问题规模很大时，最高阶是影响函数增长速率的关键因素，而常数、系数、（除最高阶外的）低阶函数影响不大。

只将比较作为基本运算

n−1次

比较、赋值都作为基本运算

1次
n−1次
n−1次
n−1次

1次

渐近时间复杂度

$T(n)=n-1=O(n)$

$T(n)=3n-1=O(n)$

渐近时间复杂度

- 证明：多项式函数的阶低于指数函数的阶

$$n^d = o(r^n),\ \ r > 1,\ \ d > 0$$

证　　不妨设d为正整数，

$$\lim_{n\to\infty}\frac{n^d}{r^n} = \lim_{n\to\infty}\frac{dn^{d-1}}{r^n \ln r} = \lim_{n\to\infty}\frac{d(d-1)n^{d-2}}{r^n(\ln r)^2}$$

$$= \ldots = \lim_{n\to\infty}\frac{d!}{r^n(\ln r)^d} = 0$$

分子分母求导

几个常见的渐进时间复杂度

- 1. $n^d = o(r^n), r > 1,\ d > 0;$
- 2. $log_a n = o(n^d);$
- 3. $log_a n = \Theta(log_b n);$
- 4. $log(n!) = \Theta(n\ log\ n);$
- 5. $n = \Theta(r^{log\ n});$

函数的阶

- 阶的符号：$O, \Omega, \Theta, o, \omega$
- 阶的高低

  至少指数级：$2^n, 3^n, n!, \ldots$

  多项式级：$n, n^2, n\log n, n^{1/2}, \ldots$

  $\log n$的多项式级：$\log n, \log^2 n, \ldots$

- 注意

  阶反映的是大的n（$n>n_0$）的情况

  可以忽略前面的有限项

思考

- 1. A算法的渐近时间复杂度比B算法高，A算法在计算机上的运行时间一定比B多？

- 2. A算法的渐近时间复杂度比B算法高，（相同的软硬件环境和编程质量下）A算法的运行时间一定比B多？

最坏情况，n很大的时候

PART 05

NP问题

五、NP问题

- 与NP相关的总共有四类问题：P**问题**、NP**问题**、NPC**问题和**NPH**问题**，是计算复杂度理论中研究的主要内容之一。

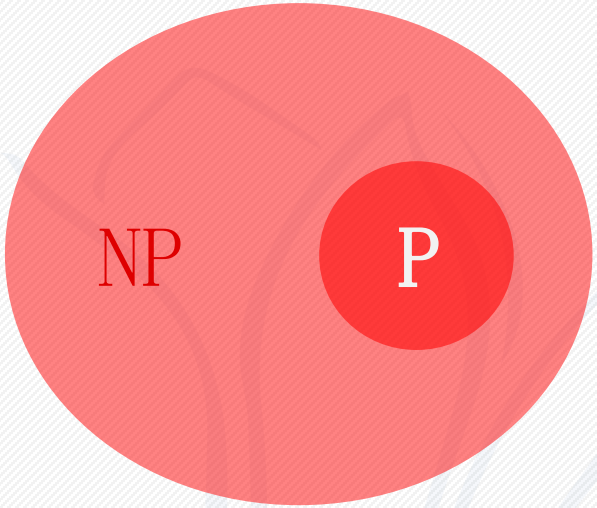
- P(Polynomial-time)**问题**
  - 能够在**多项式时间**内用算法**求解**的问题。

- **有效算法**
  - 渐进时间复杂度是问题规模的**多项式或者更低阶函数。**

五、NP问题

- P(Polynomial-time)**问题**
  - 能够在**多项式时间**内用算法求解的问题。

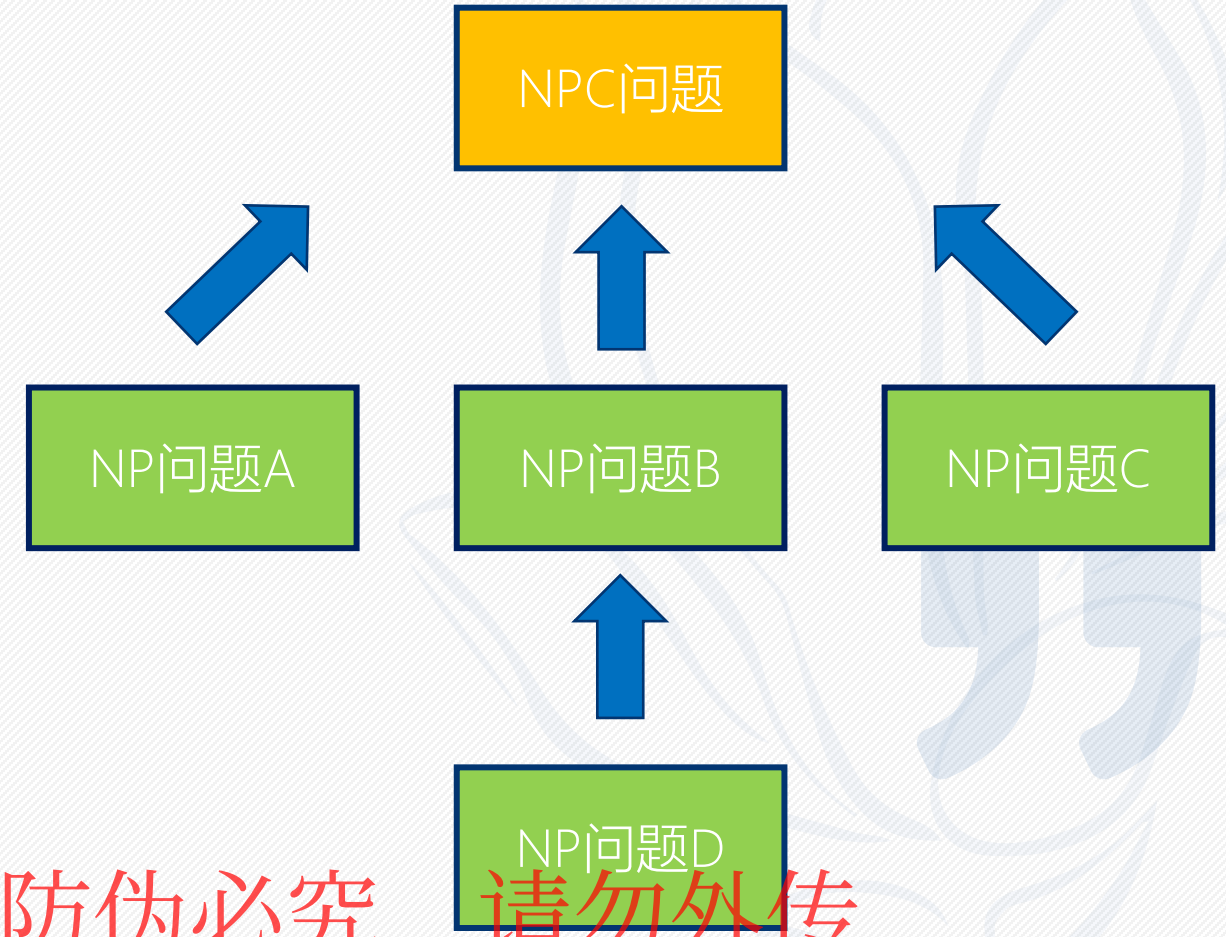
- NP(Nondeterministic polynomial-time)**问题**
  - **不确定是否存在**多项式时间的求解算法，但可以在**多项式时间**内**验证**一个猜测解的**正确性**的问题。



$P \subseteq NP$

五、NP问题

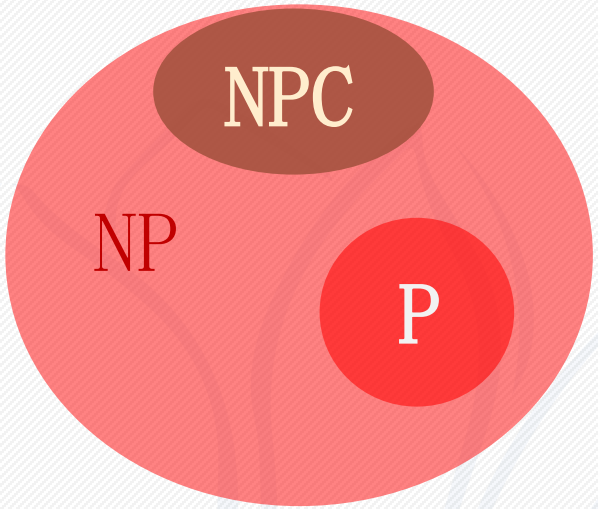
- NPC (Non-deterministic Polynomial complete problem)**问题**
  - 如果**所有NP问题**可在多项式时间内**归约成某个NP问题**，则该NP问题称为NP完全问题。



- NPC问题(2个条件)
  - ①自身是NP问题；
  - ②其他NP问题能约化到它；

五、NP问题

- NPC (Non-deterministic Polynomial complete problem)**问题**
  - 如果**所有NP问题**可在多项式时间内**归约成某个NP问题**，则该NP问题称为NP完全问题。



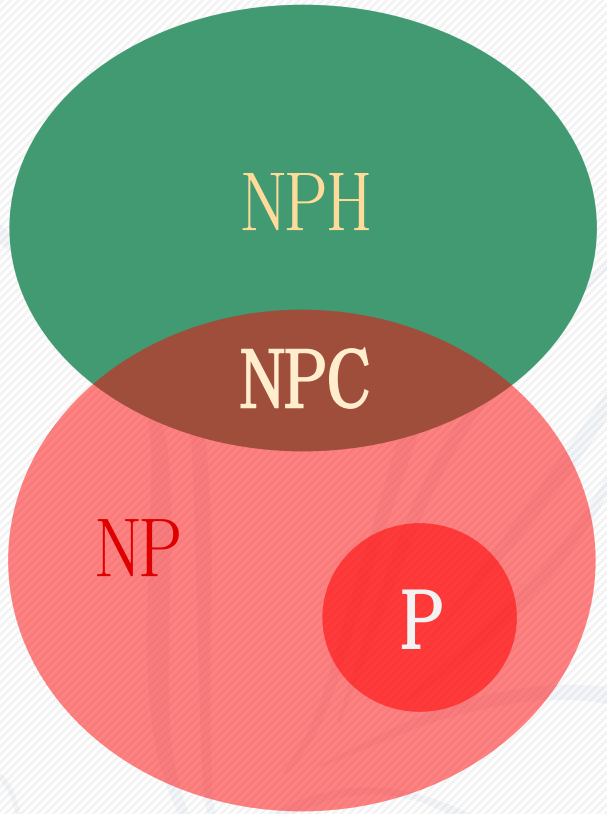
$P \subseteq NP$

$NPC \subseteq NP$

- NPC问题**确定没有多项式时间**复杂度的求解算法，只能用**指数级甚至更高阶时间**复杂度才能求解。

五、NP问题

- NPH(Non-deterministic Polynomial hard problem)问题
  - 如果所有NP问题可在多项式时间内转化（归约，意思是解决了后者也就相应的解决了前者）成某个问题，则该问题称为NP难问题。
- 满足NPC问题的条件②但不满足条件①。



- NPH问题不一定是NP问题，有可能是**不可判定问题**。即无法在多项式时间内验证一个解的正确性。

参考网址

- https://blog.csdn.net/sinat_21591675/article/details/86521190
- https://www.jianshu.com/p/3fff3dbe600e
- https://blog.csdn.net/weixin_34115824/article/details/91881076
- https://blog.csdn.net/weixin_37477009/article/details/109594975
- ……

总结

- 1. 基本运算次数
  - 基本运算次数与问题规模（或问题规模）有关
- 2.时间复杂度函数
  - 将基本运算次数表示为问题规模的函数
  - 最常用的是最坏情况下的时间复杂度函数
- 3.渐近时间复杂度(时间复杂度函数的渐近表示)
  - 目的：问题规模很大的时候，算法的计算效率
  - 函数的最高阶是影响（n很大时）增长速率的关键因素
  - 数学表示：界函数，它反映了函数最高阶的大小关系

NP问题

- P问题是指存在多项式时间求解算法；NP问题不确定是否存在多项式时间求解算法，但确定存在多项式时间验证解正确性的算法
- P问题是NP问题的子集，因为存在多项式时间求解算法的问题，一定能够在多项式时间内被验证。
- NPH问题不一定是NP问题，有可能是不可判定问题。这时候说明原问题也是不可判定的。
- NPC问题算法的时间复杂度是指数级甚至更高阶，该问题是NP问题和NPH问题的交集。