
算法常识
算法预备

什么是算法 (Algorithm)

- 算法是解决特定问题的准确而完整的描述。
- 算法是指令（规则）的有限序列。
- 算法是一种策略机制，能够对一定规范的输入，在有限时间内获得所要求的输出。

算法的特征

有穷性
(Finiteness)

确定性
(Definiteness)

输入项
(Input)

输出项
(Output)

可行性
(Effectiveness)

算法的特征：有穷性（Finiteness）

- 算法必须能在执行有限个步骤之后终止。
- 算法的每个操作步骤都能在有限的时间内完成。
- 执行时间应该是合理的。
- 如果执行时间太长，即使最终得到正确结果，也是没有意义的。

算法的特征：确定性 (Definiteness)

- 算法的每一步骤必须有确切的定义，不允许有歧义性和多义性。
- 在任何条件下，算法都只有一条执行路径。

算法的特征：输入项（Input）

- 一个算法有零个或多个输入，描述运算对象的初始情况。
- 零个输入是指算法本身就能确定初始条件，而无须人为输入。

算法的特征：输出项（Output）

- 一个算法有一个或多个输出，以反映对输入数据加工后的结果。
- 没有输出的算法是毫无意义的。

算法的特征：可行性（Effectiveness）

- 算法必须遵循特定条件下的解题规则。
- 算法描述中的每一个操作步骤都是规则允许使用的、可执行的。

算法的评价标准

同一问题可用不同算法解决，算法分析的目的在于选择合适算法和改进算法。

正确性	• 评价算法优劣的前提标准。
时间复杂度	• 执行算法所需要的计算工作量。
空间复杂度	• 执行算法所需要的内存空间。
可读性	• 算法供人阅读的容易程度。
健壮性/容错性	• 算法对不合理数据输入的反应能力和处理能力。

在正确性的前提下，一个算法的评价主要从时间复杂度和空间复杂度来考虑。

时间复杂度（Time Complexity）

- 算法的时间复杂度是指执行算法所需要的计算工作量。
- 一般来说，计算机算法是问题规模n的函数$f(n)$。
- 问题的规模n越大，算法执行的时间也就越长。
- 算法的时间复杂度也因此记做

$$T(n) = O\big(f(n)\big)$$

时间复杂度的估计方式

找出算法的基本操作，根据相应的语句确定它的执行次数，最高阶的执行次数就是时间复杂度。

时间复杂度的估计方式

【C++】

```cpp
for (int i = 1; i <= n; i++)
{
    for (int j = 1; j <= n; j++)
    {
        c[i][j] = 0;
        for (int k = 1; k <= n; k++)
        {
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

【Python】

```python
for i in range(1, n+1):
    for j in range(1, n+1):
        c[i][j] = 0
        for k in range(1, n+1):
            c[i][j] += a[i][k] * b[k][j]
```

找出算法的基本操作，根据相应的语句确定它的执行次数，最高阶的执行次数就是时间复杂度。

时间复杂度的估计方式

【C++】

```
1.  for (int i = 1; i <= n; i++)
2.  {
3.      for (int j = 1; j <= n; j++)
4.      {
5.          c[i][j] = 0;
6.          for (int k = 1; k <= n; k++)
7.          {
8.              c[i][j] += a[i][k] * b[k][j];
9.          }
10.     }
11. }
```

该步骤属于基本操作执行次数：n^2次

【Python】

```
1.  for i in range(1, n+1):
2.      for j in range(1, n+1):
3.          c[i][j] = 0
4.          for k in range(1, n+1):
5.              c[i][j] += a[i][k] * b[k][j]
```

该步骤属于基本操作执行次数：n^2次

找出算法的基本操作，根据相应的语句确定它的执行次数，最高阶的执行次数就是时间复杂度。

时间复杂度的估计方式

找出算法的基本操作，根据相应的语句确定它的执行次数，最高阶的执行次数就是时间复杂度。

【C++】

```
1.  for (int i = 1; i <= n; i++)
2.  {
3.      for (int j = 1; j <= n; j++)
4.      {
5.          c[i][j] = 0;
6.          for (int k = 1; k <= n; k++)
7.          {
8.              c[i][j] += a[i][k] * b[k][j];
9.          }
10.     }
11. }
```

该步骤属于基本操作执行次数：n^2次

该步骤属于基本操作执行次数：n^3次

【Python】

```
1.  for i in range(1, n+1):
2.      for j in range(1, n+1):
3.          c[i][j] = 0
4.          for k in range(1, n+1):
5.              c[i][j] += a[i][k] * b[k][j]
```

该步骤属于基本操作执行次数：n^2次

该步骤属于基本操作执行次数：n^3次

时间复杂度的估计方式

找出算法的基本操作，根据相应的语句确定它的执行次数，最高阶的执行次数就是时间复杂度。

【C++】

```
1.  for (int i = 1; i <= n; i++)
2.  {
3.      for (int j = 1; j <= n; j++)
4.      {
5.          c[i][j] = 0;
6.          for (int k = 1; k <= n; k++)
7.          {
8.              c[i][j] += a[i][k] * b[k][j];
9.          }
10.     }
11. }
```

该步骤属于基本操作执行次数：n^2次

该步骤属于基本操作执行次数：n^3次

【Python】

```
1.  for i in range(1, n+1):
2.      for j in range(1, n+1):
3.          c[i][j] = 0
4.          for k in range(1, n+1):
5.              c[i][j] += a[i][k] * b[k][j]
```

该步骤属于基本操作执行次数：n^2次

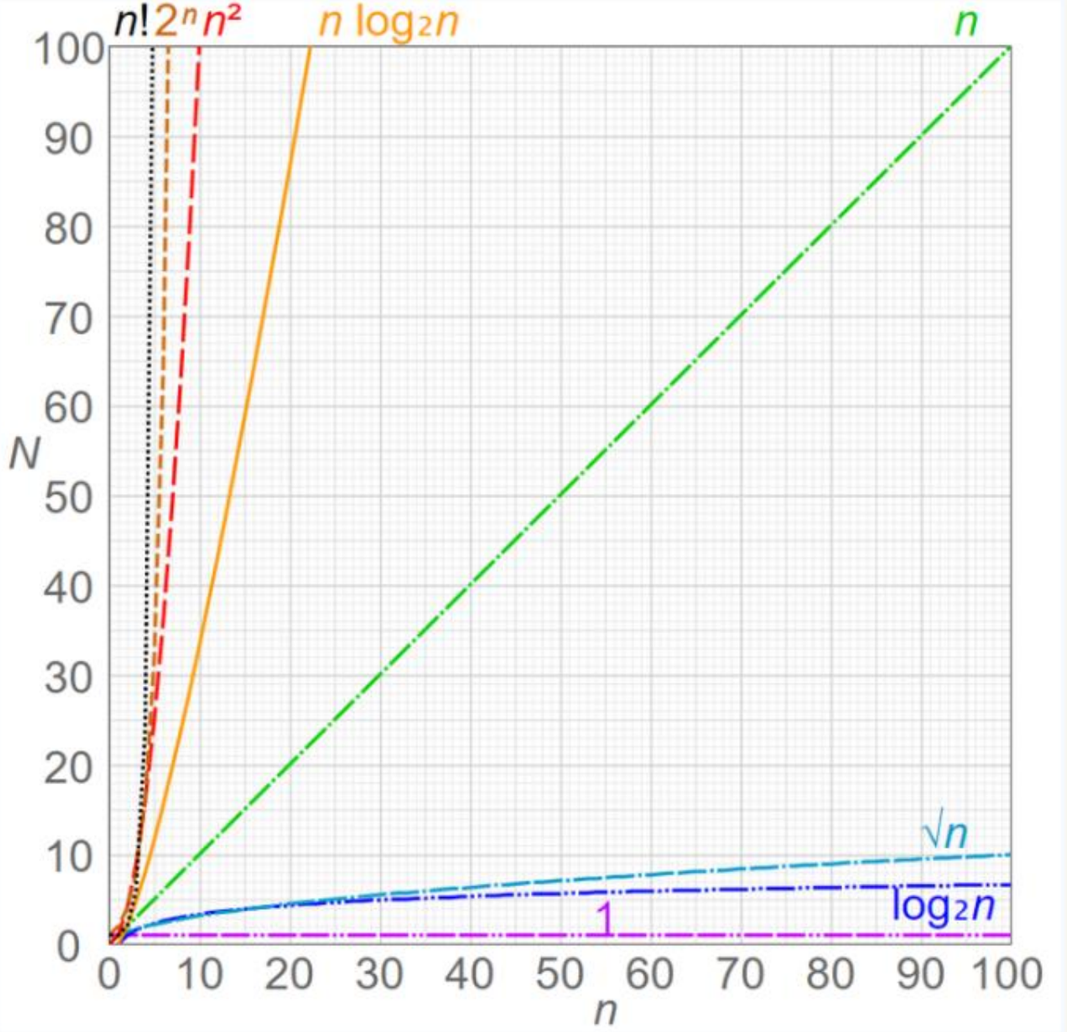
该步骤属于基本操作执行次数：n^3次

$O(n^3)$

常见的时间复杂度

名称	字母表示	典型问题
常数阶	$O(1)$	连续自然数求和
对数阶	$O(\log n)$	二分搜索
线性阶	$O(n)$	无序数组搜索
线性对数阶	$O(n \times \log n)$	归并排序
平方阶	$O(n^2)$	冒泡排序
立方阶	$O(n^3)$	矩阵乘法
指数阶	$O(2^n)$	旅行推销员的动态规划解法

对于相同规模的问题，算法的执行效率越来越低。



空间复杂度（Space Complexity）

- 算法执行期间所需要的存储空间，主要包括三部分：
  - ✓输入数据所占的存储空间
  - ✓程序本身所占的空间
  - ✓算法执行过程中需要临时占用的存储空间
- 算法的空间复杂度是指对一个算法在运行过程中临时占用存储空间大小的量度。
- 算法的空间复杂度记作

$$S(n) = O(f(n))$$

时间复杂度与空间复杂度的关系

- 算法的时间复杂度和空间复杂度合称为算法的复杂度。
- 对于一个算法，其时间复杂度和空间复杂度往往是相互影响的。
  - ✓当追求一个较好的时间复杂度时，可能会使空间复杂度的性能变差，即可能导致占用较多的存储空间。
  - ✓当追求一个较好的空间复杂度时，可能会使时间复杂度的性能变差，即可能导致占用较长的运行时间。

设计优秀算法需要考虑的因素

- 因此，设计一个算法（特别是大型算法）时，要综合考虑算法各方面的性能因素，才能够设计出比较好的算法
  - 算法的使用频率
  - 算法处理的数据量的大小
  - 算法描述语言的特性
  - 算法运行的机器系统环境
  - ……

算法的基本结构

一个算法可以用顺序、选择、循环三种基本结构组合而成。

顺序结构

- 语句与语句之间按从上到下的顺序执行

选择结构

- 先根据条件做出判断，再决定执行哪一种操作
- 必须包含判断语句

循环结构

- 从某处开始，按照一定条件，反复执行某一处理步骤
- 反复执行的步骤为循环体

算法思想

模拟

枚举

递推

递归

迭代

贪心

分治

试探

数据结构

向量

栈

队列

链表

树

图

堆/优先队列

森林

集合

映射

算法实践

高精度计算

排序

二分查找

字符串匹配

深度优先搜索与回溯

广度优先搜索与分枝限界

动态规划

哈希

数学概念

质数

因数

幂

同余

矩阵

组合数学

离散数学

博弈论

问题的难度

P

NP

NPC

NPH

P问题

- P（Deterministic Polynomial，确定多项式）问题是可以在多项式时间内被确定机（通常意义的计算机）解决的问题。
- 典型的P问题
  - ✓找最大数
  - ✓排序

NP问题

- NP（Non-Deterministic Polynomial，非确定多项式）问题，是指可以在多项式时间内被非确定机解决的问题。
  - ✓也是可以在多项式的时间里验证一个解的问题。
  - ✓也是可以在多项式的时间里猜出一个解的问题。
- 典型的NP问题
  - ✓大整数分解：比如把9938550分解成两个数的乘积。如果告诉你这两个数是1123和8850，那么很容易进行验证。
  - ✓哈密尔顿环：在一个无向图里，由指定的起点前往指定的终点，途中经过所有其它结点且只经过一次。

NP问题与P问题的关系

- 所有的P类问题都是NP问题。
- 也就是说，能多项式地解决一个问题，必然能多项式地验证一个问题的解。
- 千禧难题之首：P问题是否等于NP问题？

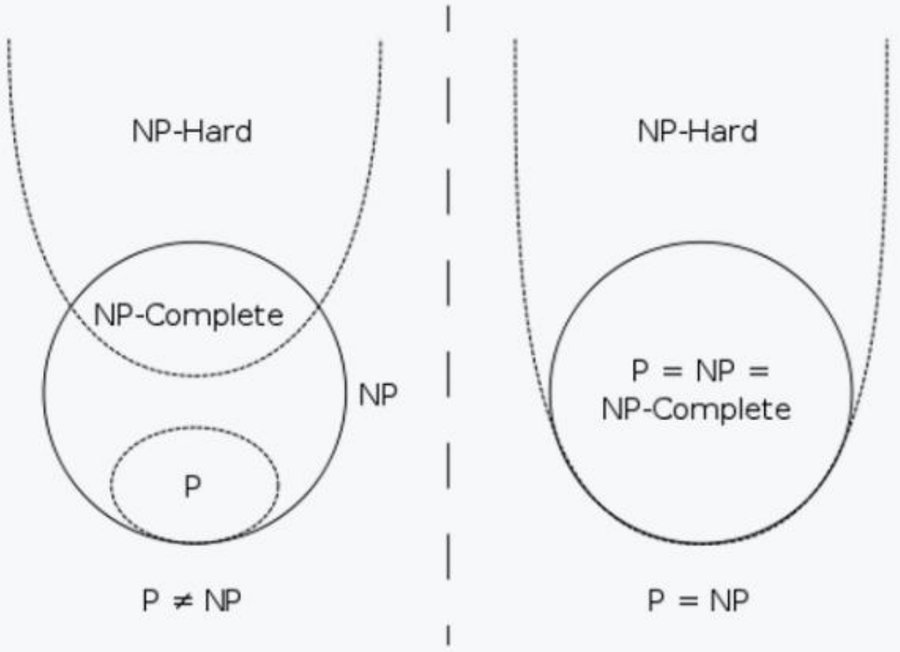
NPC问题

- NPC（NP Complete，NP完全）问题是最有可能不是P问题的问题类型。
- 所有的NP问题都可以用多项式时间归约到一个NPC问题。
- 如果NPC可以在多项式时间内求解，那么P问题就等于NP问题。
- 典型NPC问题：
  - ✓有向哈密尔顿环问题：给定有向图H，判定其是否有经过图中每个顶点且仅一次的回路。
  - ✓划分问题：给定一个具有n个整数的集合S，是否能把S划分成两个子集S_1和S_2，使得S_1中的整数之和等于S_2中的整数之和。

NPH问题

- NPH（NP-Hard，NP难）问题：至少难度是NP问题的问题，也就是可能比NP问题还难的问题。
- 典型问题
  - ✓围棋

P/NP/NPC/NPH之间的关系




WELL DONE