时间复杂度与时间限制

时间限制

一般会在题目中给出，常见的限制有 1/2/5/20 秒(s)，有的时候也会用毫秒(ms)的形式给出。我们一般会粗略估算计算机一秒钟可以执行 10^8 级别左右的基础运算（加减乘除等）。

假设一道题目时间限制为 1 秒，现在有两个算法都能实现这道题，根据题目输入的 n 的不同：

- 算法 1 要进行 n^2 次运算
- 算法 2 要进行 $10 \times n$ 次运算

那么显然，当 $n \leq 5000$ 时，两个算法都绰绰有余，而当 $n \leq 10^6$ 时，第一个算法就没法在时间限制内得到答案，评测结果就会是**超出时间限制**。

时间复杂度

表示方法

我们一般用时间复杂度这个概念来描述一个算法的耗时，这个概念详细定义比较复杂，可以简单理解为时间复杂度就是基础运算的规模，更准确的定义可以参照下面的扩展阅读部分。

表示时间复杂度的渐进符号有多种：大 Θ、大 Ω、小 ω、大 O、小 o 等等，各自有不同的规则。因为大写字母 O 比较容易打字和书写，所以经常会杂糅这些概念，用大 O 符号来描述算法的时间复杂度上限，通常**只保留多项式的最高次项，并省略系数**。

一些例子

比如上述的算法 1 与算法 2，我们通常会用 $O(n^2)$ 与 $O(n)$ 来描述。

再比如对于下述代码：

```
cin >> n;
for(int i = 1; i <= n; i++)
    cin >> a[i];
sum = 0;
for(int i = 1; i <= n; i++)
    for(int j = i; j <= n; j++)
        sum += a[i] * a[j];
cout << sum;
```

`cin >> a[i];` 执行了 n 次，`sum += a[i] * a[j];` 执行了 $n+(n-1)+\cdots+1=\frac{n\times(n-1)}{2}$，总的基础运算次数大概是 $\frac{1}{2}n^2+\frac{1}{2}n$ 次，只保留最高次项并省略常数后就是 $O(n^2)$。

有的时候某个算法在不同情况下时间复杂度不同，比如最坏情况下快速排序算法时间复杂度会很高，因此为了更准确的描述，我们会说快速排序算法的时间复杂度为：最坏情况下 $O(n^2)$，随机数据的一般情况下 $O(n\log n)$。

$\log n$ 表示对数，之所以不写底数，是因为我们可以轻松通过换根公式，加一个系数后把底数进行改变，所以 $\log_{10} n$、$\log_2 n$、$\ln n$ 都可以视作 $\log_2 n$，写作 $\log n$。

对数知识是数学中的难点，下面直接给出常见的 n 对应的 $\log_2 n$，也可以直接通过 `<cmath>` 库中的 `log2()` 函数得到。

n	$\log_2 n$
1000	≈ 10
5000	≈ 12
10^4	≈ 13
10^6	≈ 20
10^7	≈ 23

- `sort`：$O(n\log n)$
- `lower_bound/upper_bound/binary_search`：$O(\log n)$
- `priority_queue`：`.push()` $O(\log n)$，`.pop()` $O(\log n)$，`.top()` $O(1)$
- 判断 n 是否为质数
  - 从 $2\sim n-1$ 试除：$O(n)$
  - 从 $2\sim\sqrt{n}$ 试除：$O(\sqrt{n})$
  - 埃氏筛法：初始化 $O(n\log\log n)$，初始化后每次查询 $O(1)$
  - 欧拉筛法/线性筛法：初始化 $O(n)$，初始化后每次查询 $O(1)$
- 搜索枚举 n 个数的子集：$O(2^n)$
- 搜索枚举 n 的排列：$O(n!)$

常见时间复杂度能处理的数据规模

时间复杂度	1 秒时限下能处理的数据规模
$O(1)$	∞
$O(n)$	10^7（注意输入输出的耗时）
$O(n^2)$	5000
$O(n^3)$	200
$O(2^n)$	20
$O(n!)$	11
$O(\sqrt{n})$	10^{15}（注意使用合适的数据类型）
$O(n \log n)$	2×10^6

以上描述的都是常见的保险的数据范围，有时也可以通过题目的部分分数据范围反向推测可能用到的算法。

扩展阅读

- OI Wiki, 复杂度：https://oi-wiki.org/basic/complexity/
- 《骗分导论》——李博杰