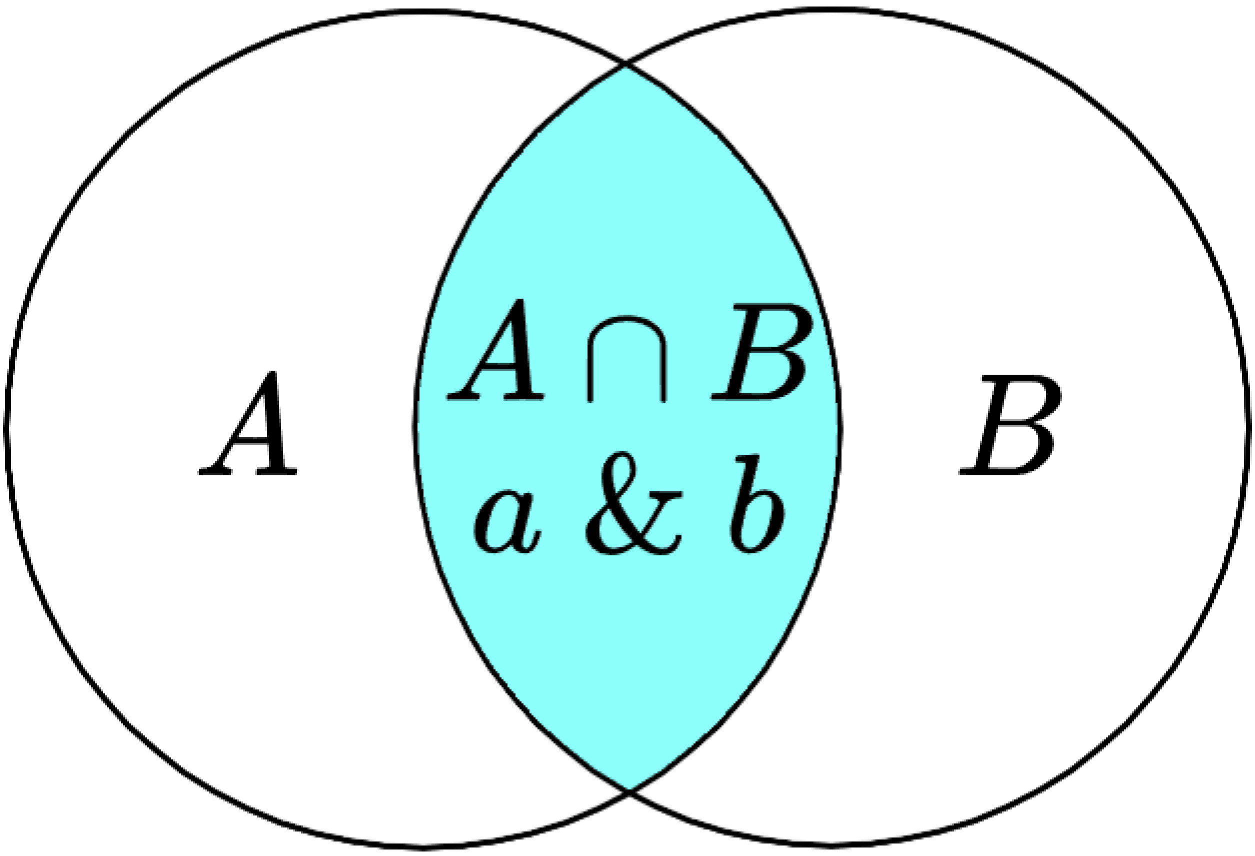
分享｜从集合论到位运算，常见位运算技巧分类总结！

已关注

灵茶山艾府 发起于 2023-06-16　最近编辑于 4 天前　来自浙江



> 图：集合交、按位与之间存在某种联系。

前言

本文将扫清位运算的迷雾，在集合论与位运算之间建立一座桥梁。

在高中，我们学了集合论（set theory）的相关知识。例如，包含若干整数的集合 $S=\{0,2,3\}$。在编程中，通常用哈希表（hash table）实现集合。例如 Java 中的 `HashSet`，C++ STL 中的 `unordered_set`。

在集合论中，有交集 $\cap$、并集 $\cup$、包含于 $\subseteq$ 等等概念。如果编程实现「求两个哈希表的交集」，需要一个个地遍历哈希表中的元素。那么，有没有效率更高的做法呢？

该二进制登场了。

集合可以用二进制表示，二进制**从低到高**第 i 位为 1 表示 i 在集合中，为 0 表示 i 不在集合中。例如集合 $\{0,2,3\}$ 可以用二进制数 1101 表示；反过来，二进制数 1101 就对应着集合 $\{0,2,3\}$。

正式地说，包含非负整数的集合 S 可以用如下方式「压缩」成一个数字：

$$f(S)=\sum_{i\in S} 2^i$$

上面举例的 $\{0,2,3\}$ 就可以压缩成 $2^0+2^2+2^3=13$，也就是二进制数 1101。

利用位运算「并行计算」的特点，我们可以高效地做一些和集合有关的运算。按照常见的应用场景，可以分为以下四类：

1. 集合与集合
2. 集合与元素
3. 遍历集合
4. 枚举集合

一、集合与集合

其中 $\&$ 表示按位与，$|$ 表示按位或，$\oplus$ 表示按位异或，$\sim$ 表示按位取反。

其中「对称差」指仅在其中一个集合的元素。

术语	集合	位运算	举例	举例
交集	$A\cap B$	$a\&b$	$\{0,2,3\}$ $\cap\{0,1,2\}$ $=\{0,2\}$	1101 $\&\,0111$ $=0101$
并集	$A\cup B$	$a\mid b$	$\{0,2,3\}$ $\cup\{0,1,2\}$ $=\{0,1,2,3\}$	1101 $\mid\ 0111$ $=1111$
对称差	$A\,\Delta\,B$	$a\oplus b$	$\{0,2,3\}$ $\Delta\ \{0,1,2\}$ $=\{1,3\}$	1101 $\oplus\,0111$ $=1010$
差	$A\setminus B$	$a\&\sim b$	$\{0,2,3\}$ $\setminus\{1,2\}$ $=\{0,3\}$	1101 $\&\,1001$ $=1001$
差（子集）	$A\setminus B$ （$B\subseteq A$）	$a\oplus b$	$\{0,2,3\}$ $\setminus\{0,2\}$ $=\{3\}$	1101 $\oplus\,0101$ $=1000$
包含于	$A\subseteq B$	$a\&b=a$ $a\mid b=b$	$\{0,2\}\subseteq\{0,2,3\}$	$0101\&1101=0101$ $0101\mid1101=1101$

> 注 1：按位取反的例子中，仅列出最低 4 个比特位取反后的结果，即 0110 取反后是 1001。
>
> 注 2：包含于的两种位运算写法是等价的，在编程时只需判断其中任意一种。
>
> 注 3：编程时，请注意运算符的优先级。例如 `==` 在某些语言中优先级更高。

二、集合与元素

通常会用到移位运算。

其中 $<<$ 表示左移，$>>$ 表示右移。

> 注：左移 i 位相当于乘 2^i，右移 i 位相当于除 2^i。

术语	**集合**	**位运算**	**举例**	**举例**
空集	$\varnothing$	0		
单元素集合	$\{i\}$	$1 << i$	$\{2\}$	$1 << 2$
全集	$U = \{0,1,2,\cdots n-1\}$	$(1 << n) - 1$	$\{0,1,2,3\}$	$(1 << 4) - 1$
补集	$\complement_U S = U \setminus S$	$\sim s$ 或者 $((1 << n) - 1) \oplus s$		
属于	$i \in S$	$(s >> i)\ \&\ 1 = 1$	$2 \in \{0,2,3\}$	$(1101 >> 2)\ \&\ 1 = 1$
不属于	$i \notin S$	$(s >> i)\ \&\ 1 = 0$	$1 \notin \{0,2,3\}$	$(1101 >> 1)\ \&\ 1 = 0$
添加元素	$S \cup \{i\}$	$s \mid (1 << i)$	$\{0,3\} \cup \{2\}$	$1001 \mid (1 << 2)$
删除元素	$S \setminus \{i\}$	$s \& \sim (1 << i)$	$\{0,2,3\} \setminus \{2\}$	$1101 \& \sim (1 << 2)$
删除元素（一定在集合中）	$S \setminus \{i\}$（$i \in S$）	$s \oplus (1 << i)$	$\{0,2,3\} \setminus \{2\}$	$1101 \oplus (1 << 2)$
删除最小元素		$s \& (s-1)$		见下

```
     s = 101100
   s-1 = 101011 // 最低位的 1 变成 0，同时 1 右边的 0 都取反，变成 1
s&(s-1) = 101000
```

特别地，如果 s 是 2 的幂，那么 $s \& (s-1) = 0$。

此外，某些数字可以借助标准库提供的函数算出：

术语	Python	Java	C++	Go
集合大小（元素个数）	`s.bit_count()`	`Integer.bitcount(s)`	`__builtin_popcount(s)`	`bits.OnesCount(s)`
二进制长度（减一得到集合中的最大元素）	`s.bit_length()`	`32-Integer.numberOfLeadingZeros(s)`	`32-__builtin_clz(s)`	`bits.Len(s)`
集合中的最小元素	`(s&-s).bit_length()-1`	`Integer.numberOfTrailingZeros(s)`	`__builtin_ctz(s)`	`bits.TrailingZeros(s)`

特别地，只包含最小元素的子集，即二进制最低 1 及其后面的 0，也叫 lowbit，可以用 `s & -s` 算出。举例说明：

```
     s = 101100
    ~s = 010011
(~s)+1 = 010100 // 根据补码的定义，这就是 -s  =>  s 的最低 1 左侧取反，右侧不变
s & -s = 000100 // lowbit
```

三、遍历集合

设元素范围从 0 到 $n-1$，挨个判断每个元素是否在集合 s 中：

Python3 | Java | C++ | Go

```
for (int i = 0; i < n; i++) {
    if ((s >> i) & 1) { // i 在 s 中
        // 处理 i 的逻辑
    }
}
```

四、枚举集合

设元素范围从 0 到 $n-1$，从空集 $\varnothing$ 枚举到全集 U：

Python3 | Java | C++ | Go

```
for (int s = 0; s < (1 << n); s++) {
    // 处理 s 的逻辑
}
```

设集合为 s，**从大到小**枚举 s 的所有**非空**子集 sub：

Python3 | Java | C++ | Go

```
for (int sub = s; sub; sub = (sub - 1) & s) {
    // 处理 sub 的逻辑
}
```

为什么要写成 `sub = (sub - 1) & s` 呢?

暴力做法是从 s 出发，不断减一，直到 0。但这样做，中途会遇到很多并不是 s 的子集的情况。例如 $s = 10101$ 时，减一得到 10100，这是 s 的子集。但再减一就得到 10011 了，这并不是 s 的子集，下一个子集应该是 10001。

把所有的合法子集按顺序列出来，会发现我们做的相当于「压缩版」的二进制减法，例如

$$10101 \rightarrow 10100 \rightarrow 10001 \rightarrow 10000 \rightarrow 00101 \rightarrow \cdots$$

如果忽略掉 10101 中的两个 0，数字的变化和二进制减法是一样的，即

$$111 \rightarrow 110 \rightarrow 101 \rightarrow 100 \rightarrow 011 \rightarrow \cdots$$

如何快速找到下一个子集呢？以 $10100 \rightarrow 10001$ 为例说明，普通的二进制减法会把最低位的 1 变成 0，同时 1 右边的 0 变成 1，即 $10100 \rightarrow 10011$。「压缩版」的二进制减法也是类似的，把最低位的 1 变成 0，但同时对于 1 右边的 0，只保留在 $s = 10101$ 中的 1，所以是 $10100 \rightarrow 10001$。怎么保留？$\& \ 10101$ 就行。

此外，如果要从大到小枚举 s 的所有子集 sub（从 s 枚举到空集 $\varnothing$），可以这样写：

Python3 | Java | C++ | Go

```
int sub = s;
do {
    // 处理 sub 的逻辑
    sub = (sub - 1) & s;
} while (sub != s);
```

原理是当 $sub = 0$ 时（空集），再减一就得到 -1，对应的二进制为 $111\cdots1$，再 $\&s$ 就得到了 s。所以当循环到 $sub = s$ 时，说明最后一次循环的 $sub = 0$（空集），s 的所有子集都枚举到了，退出循环。

注：还可以枚举全集 U 的所有大小**恰好**为 k 的子集，这一技巧叫做 Gosper's Hack，具体请看 视频讲解。

位运算题单

- 位运算（基础/性质/拆位/试填/恒等式/贪心/脑筋急转弯）

其它题单

- 滑动窗口（定长/不定长/多指针）
- 二分算法（二分答案/最小化最大值/最大化最小值/第K小）

- 单调栈（矩形系列/字典序最小/贡献法）
- 网格图（DFS/BFS/综合应用）
- 图论算法（DFS/BFS/拓扑排序/最短路/最小生成树/二分图/基环树/欧拉路径）
- 动态规划（入门/背包/状态机/划分/区间/状压/数位/数据结构优化/树形/博弈/概率期望）

欢迎关注 B站@灵茶山艾府