Binary Index Tree(Fenwick Tree)

树状数组

描述：

树状数组的典型应用场景是区间求和。对于包含n个数字的数组s，修改其中若干成员（其中）后，求数组s在区间（其中）上的所有成员的和。普通的算法是在修改了成员之后，求和时遍历区间相加求该区间的和。修改成员（其中）的时间复杂度为，求区间的和的时间复杂度为。而树状数组修改一个成员的时间复杂度为，求区间和的时间复杂度为，相比与普通算法的空间复杂度增加了。

函数LowBit用于计算一个数字的二进制形式下最低位的1代表的十进制的值。比如最低位的1代表的十进制值为，最低位的1代表的十进制值为，最低位的1代表的十进制值为，则有，，。

在C/C++中由于补码的原因，LowBit函数实现如下：

int LowBit(int x) { return x & (-x); }

内存中的数字按照补码存储（正整数的补码与原码相同，负整数的补码是原码取反加一，并且最高位bit设置为1）。比如，则；，则；，则。

对于非负整数x，x与-x进行位与操作，即可得到x中最低位的1所代表的十进制的值。比如；；。

额外需要注意的是，CPU架构中大端模式（Big-Endian）和小端模式（Little-Endian）的区别并不会影响该计算。因为大端和小端影响的是数据在内存中存放的顺序，当数据被CPU加载到寄存器中时，所有的位操作都是在寄存器上进行的，不会影响位操作，因此位操作可以从纯数学计算的角度来看。

所有的整数都可以表示为2的幂的和（即这种形式）

并查集的核心操作是查询父节点，这个操作实际上是查询祖宗节点。设为x节点的父节点，当时，称x为一个祖宗节点，设是x的祖宗节点。该方法可以压缩查询时搜索的节点数量，称为路径压缩技术。

将下面的集合，共10个成员，分成两个家庭A和B。每个成员都有父节点，初始时所有成员的父节点都指向自己。如图所示：



1. 声明0和4属于同一家庭，比较0和4的祖宗节点，设置（设置父节点的规则可以根据实际要求进行设计，但两个节点的其中一个的父节点必须设置为另一个节点的祖宗点），本文中我们取左节点的祖宗节点作为右节点的父节点；



1. 声明1和9节点属于同一家庭，设置；



1. 声明0和2节点属于同一家庭，设置；



1. 声明1和3节点属于同一家庭，设置；



1. 声明3和5节点属于同一家庭，设置；



1. 声明6和8节点属于同一家庭，设置；



1. 声明2和6节点属于同一家庭，设置；



1. 声明1和7节点属于同一家庭，设置；



合并两节点x和y时，根据固定规则设置（或者相反）；查询节点x的祖宗节点时，若则设置。并查集的合并、查询操作的时间复杂度接近。