Binary Index Tree(Fenwick Tree)

树状数组

描述：

对于包含个数字的数组，修改其中若干成员（其中）后，求数组在区间（其中）上的所有成员之和。一般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。

而树状数组可以更快的进行区间求和，将该问题转化为前缀和，即。类似所有整数都可以表示成2的幂和，也可以把一串序列表示成一系列子序列的和。其中，子序列的个数是其二进制表示中1的个数，并且子序列代表的的个数也是2的幂。

1. 函数

函数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，函数实现如下：

int LowBit(int x) { return x & (x ^ (x-1)); }

或者利用计算机补码的特性，写成：

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，与进行位与操作，即可得到中最低位的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。

1. 维护前缀和的数组

对于长度为的数组（该算法需要数组下标从1开始，因此数组的范围为），数组中的元素。比如：

；

；

；

；

；

；

；

；

；

在数组的基础上，求数组中的和，只需累加所有，其中初始时，每累加一次，值减去，直到。（这里我暂时也没找到更好的讲解方法，解释的不是很清晰）

对于长度为的数组，构造树状数组的时间复杂度为，查询区域和的时间复杂度为，修改数组中一个值的时间复杂度为，空间复杂度为。

树状数组（Fenwick tree）：

<https://en.wikipedia.org/wiki/Fenwick_tree>

<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.8917>