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$s[i]$（其中$1 \le i \le n$）后，求数组 s 在区间$[p,q]$（其中$1 \le p \le q \le n$）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$[p,q]$相加求该区间的和。修改成员$s[i]$（其中$1 \le i \le n$）的时间复杂度为$O(1)$，求$s[p,q]$区间的和的时间复杂度为$O(N)$。

区间和问题可以转化为前缀和，即$s[p,q] = s[1,q] - s[1,q]$。根据 Peter M. Fenwick，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$s[i]$的个数也是 2 的幂。

（1）LowBit 函数

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$34_{10} = 10{,}0010_2$最低位的 1 代表的十进制值为2_{10}，$12_{10} = 1100_2$最低位的 1 代表的十进制值为4_{10}，$8_{10} = 1000_2$最低位的 1 代表的十进制值为8_{10}，则有$LowBit(34) = 2$，$LowBit(12) = 4$，$LowBit(8) = 8$。

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 ^ (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：

$34_{10} = 0010{,}0010_2$，则$-34_{10} = 1101{,}1110_2$；

$12_{10} = 0000{,}1100_2$，则$-12_{10} = 1111{,}0100_2$；

$8_{10} = 0000{,}1000_2$，则$-8_{10} = 1111{,}1000_2$。

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：

$34_{10} \& (-34_{10}) = 0010{,}0010_2 \& 1101{,}1110_2 = 10_2 = 2_{10}$；

$12_{10} \& (-12_{10}) = 0000{,}1100_2 \& 1111{,}0100_2 = 100_2 = 4_{10}$；

$8_{10} \& (-8_{10}) = 0000{,}1000_2 \& 1111{,}1000_2 = 1000_2 = 8_{10}$。

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。

（2）维护 s 前缀和的数组 bit

对于长度为 n 的数组 s（该算法需要数组下标从 1 开始，因此数组 s 的范围为$[1,n]$），数组 bit 中的元素$bit[i] = \sum_{j=i-LowBit(i)+1}^{i} s_j$。比如：

$bit[1] = \sum_{j=1-1+1}^{1} s_j = s[1]$；

$bit[2] = \sum_{j=2-2+1}^{2} s_j = s[1] + s[2]$；

$bit[3] = \sum_{j=3-1+1}^{3} s_j = s[3]$；

$bit[4] = \sum_{j=4-4+1}^{4} s_j = s[1] + s[2] + s[3] + s[4]$；

$bit[5] = \sum_{j=5-1+1}^{5} s_j = s[5]$；

$bit[6] = \sum_{j=6-2+1}^{6} s_j = s[5] + s[6]$；

$bit[7] = \sum_{j=7-1+1}^{7} s_j = s[7]$；

$bit[8] = \sum_{j=8-8+1}^{8} s_j = s[1] + s[2] + s[3] + s[4] + s[5] + s[6] + s[7] + s[8]$；

$bit[9] = \sum_{j=9-9+1}^{9} s_j = s[9]$；

在数组 bit 的基础上，求数组 s 中$[0, p]$的和，只需累加所有$bit[i]$，其中初始时$i = p$，每累加一次$bit[i]$，i值减去$LowBit(i)$，直到$i \leq 0$。（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 n 的数组 s，构造树状数组的时间复杂度为$O(N\, log_2 N)$，查询区域和的时间复杂度为$O(log_2 N)$，修改数组 s 中一个值的时间复杂度为$O(log_2 N)$，空间复杂度为$O(N)$。