# 树状数组（BIT）

## 引入问题

给出一个长度为n的数组，完成以下两种操作

- 将第i个数加上k
- 输出区间[x，y]内的区间和

朴素算法O（n^2）

- 单点修改：O（n）
- 区间查询：O（n）

引入树状数组O（nlog 2 n）

- 单点修改：O（log2 n）
- 区间查询：O（log 2 n）

## 前置知识 - lowbit（）运算

定义：非负整数n在二进制表示下最低位1及其后面的0构成的数值

例如：

- lowbit（44） = lowbit（（101100）2）= （100）2 = 4
    - 对于101100我们可以先将其取反010011之后再加上1（取反加一）

| 操作 | 数值 |
| --- | --- |
| A.原始 | 101100 |
| B.取反 | 010011 |
| C.末尾 + 1 | 010100 |

此时我们比较一下A和C的值，会发现最低为的1和后面的值相同，其余均不同，则我们按位与就可以i得到（100）2

| **1** | **0** | **1** | **1** | **0** | **0** |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |

        - 就是补码

因为计算机存储使用的是补码，取反加1后的值就是负的这个值，所以可以写出代码

```
lowbit(n){
    return n&(-n);
}
```

## 相关信息

# 全称

树状数组，Binary Indexed Tree(BIT), Fenwick Tree

# 起源与介绍

**树状数组**或**二元索引树**，又以其发明者命名为 FenwickFenwick 树。最早由 PeterM.FenwickPeterM.Fenwick 于1994年以《A New Data Structure for Cumulative Frequency Tables》为题发表在《SOFTWARE PRACTICE AND EXPERIENCE》。其初衷是解决数据压缩里的累积频率（Cumulative Frequency）的计算问题，现多用于高效计算数列的前缀和， 区间和。它可以以 **O(logn)** 的时间**得到任意前缀和（区间和）**。

按照 Peter M. Fenwick 的说法，BIT 的产生源自整数与二进制的类比。

*Each integer can be represented as sum of powers of two. In the same way, cumulative frequency can be represented as sum of sets of subfrequencies. In our case, each set contains some successive number of non-overlapping frequencies.*

简单翻一下：每个整数可以用二进制来进行表示，在某些情况下，序列累和（这里没有翻译为频率）也可以用一组子序列累和来表示。在本例子中，每个集合都有一些连续不重叠的子序列构成。

实际上， BIT 也是采用类似的想法，将序列累和类比为整数的二进制拆分，每个前缀和拆分为多个不重叠序列和，再利用二进制的方法进行表示。这与 Integer 的位运算非常相似。

之所以命名为： Binary Indexed Tree，在论文中 Fenwick 有如下解释：

*In recognition of the close relationship between the tree traversal algorithms and the binary representation of an element index,the name "binar indexed tree" is proposed for the new structure.*

也就是考虑到：树的遍历方法与二值表示之间的紧密联系，因此将其命名为二元索引树。

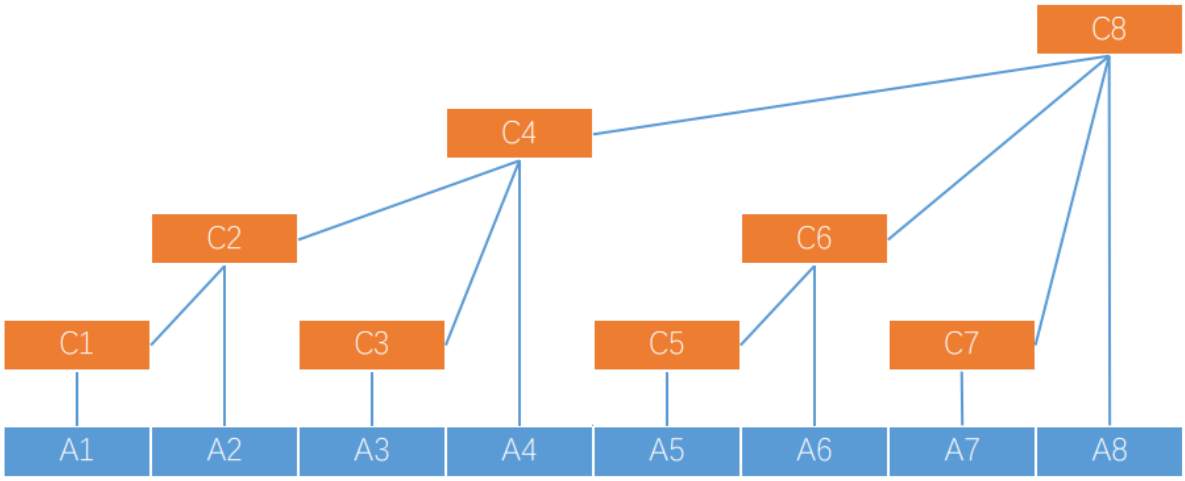
# 作用

树状数组是一个**查询**和**修改**复杂度都为**log(n)**的数据结构。主要用于查询任意两位之间的所有元素之和，但是每次只能修改一个元素的值；经过简单修改可以在log(n)的复杂度下进行范围修改，但是这时只能查询其中一个元素的值。

- 总结：支持单点修改，区间权值和查询，查询和修改复杂度都为log(n)
- 解决区间动态前缀和的东西
    - 假设有数组a1,a2,a3.....an
    - 我们需要n次的去询问a1 + a2 + a3 + ..... + am（1<=m<=n）
    - 会修改ai（1 <= i <= n）
    - 暴力的方法去求解的话
        - 修改O（1）
        - 查询最坏是O（n）
        - 暴力结果：O（n^2）

# 基本概念

假设有数组a[n]，那么查询a[1]+...+a[n]的时间是log级别的，而且是一个在线的数据结构，支持随时修改某个元素的值，复杂度也为log级别。




来观察这个图：

令这棵树的结点编号为C1，C2...Cn。令每个结点的值为这棵树的值的总和，那么容易发现：

C1 = ***A1***

C2 = C1 + A1 = ***A1 + A2***

C3 = ***A3***

C4 = C2 + C3 + A4 = ***A1 + A2 + A3 + A4***

C5 = ***A5***

C6 = C5 + A6 = ***A5 + A6***

C7 = ***A7***

C8 = C4 + C6 + C7 + A8 = ***A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8***

设节点编号为x，那么这个节点管辖的区间为2^k（其中k为x二进制末尾0的个数）个元素。因为这个区间最后一个元素必然为Ax， 很明显：**Cn = A(x – 2^k + 1) + ... + Ax**

# 树状数组的思想与实现

## 思想总结

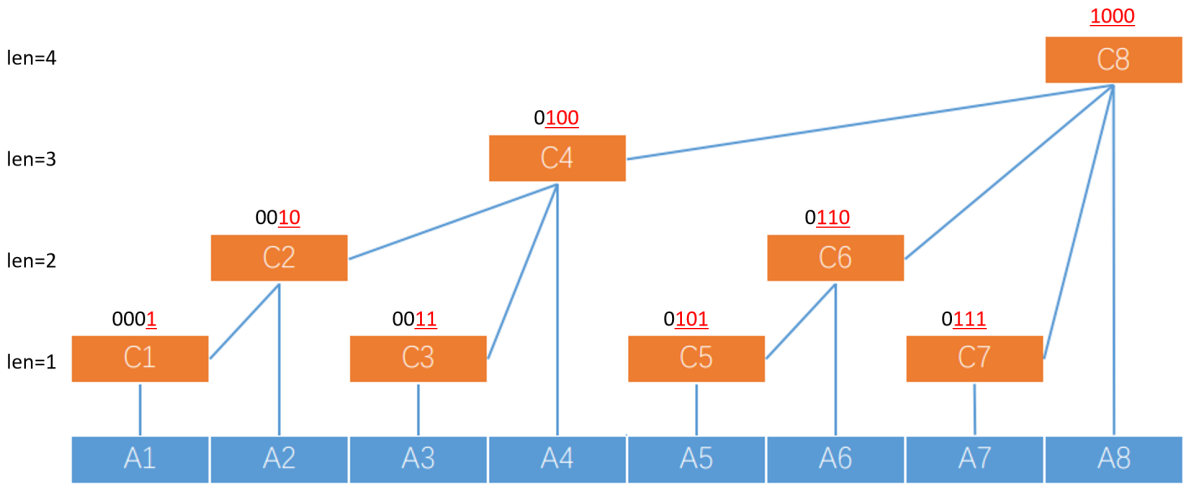
回到引入的问题

给出一个长度为n的数组，完成以下两种操作

- 将第i个数加上k
- 输出区间[x，y]内的区间和

解题思想：

- 区间和——》前缀和相减 ——》树结构维护（log2n）（就是基本概念里的那张图）




1. 每一个节点c[x]保存以x为根的子数叶中叶节点值的和
2. 每一个节点覆盖的长度len为lowbit的值
    - 我们发现每一层末尾的零的个数都是相同的
    - 0的个数与其覆盖的长度有关
    - 所以我们可以得出c[x]节点的长度就等于lowbit（x）
    - 还可以得出a[x]节点的父节点为c[x+lowbit(x)]
    - 树的深度为log2（n）+ 1
    - 构造这棵数

$$c[x]=\sum_{i=x-lowbit(x)+1}^{x}a[i]$$

```
lowbit( 1) =  1   lowbit( 2) =  2   lowbit( 3) =  1   lowbit( 4) =  4   lowbit( 5) =  1
lowbit( 6) =  2   lowbit( 7) =  1   lowbit( 8) =  8   lowbit( 9) =  1   lowbit(10) =  2
lowbit(11) =  1   lowbit(12) =  4   lowbit(13) =  1   lowbit(14) =  2   lowbit(15) =  1
lowbit(16) = 16   lowbit(17) =  1   lowbit(18) =  2   lowbit(19) =  1   lowbit(20) =  4
lowbit(21) =  1   lowbit(22) =  2   lowbit(23) =  1   lowbit(24) =  8   lowbit(25) =  1
lowbit(26) =  2   lowbit(27) =  1   lowbit(28) =  4   lowbit(29) =  1   lowbit(30) =  2
lowbit(31) =  1   lowbit(32) = 32   lowbit(33) =  1   lowbit(34) =  2   lowbit(35) =  1
lowbit(36) =  4   lowbit(37) =  1   lowbit(38) =  2   lowbit(39) =  1   lowbit(40) =  8
lowbit(41) =  1   lowbit(42) =  2   lowbit(43) =  1   lowbit(44) =  4   lowbit(45) =  1
lowbit(46) =  2   lowbit(47) =  1   lowbit(48) = 16   lowbit(49) =  1   lowbit(50) =  2
lowbit(51) =  1   lowbit(52) =  4   lowbit(53) =  1   lowbit(54) =  2   lowbit(55) =  1
lowbit(56) =  8   lowbit(57) =  1   lowbit(58) =  2   lowbit(59) =  1   lowbit(60) =  4
lowbit(61) =  1   lowbit(62) =  2   lowbit(63) =  1   lowbit(64) = 64   lowbit(65) =  1
lowbit(66) =  2   lowbit(67) =  1   lowbit(68) =  4   lowbit(69) =  1   lowbit(70) =  2
lowbit(71) =  1   lowbit(72) =  8   lowbit(73) =  1   lowbit(74) =  2   lowbit(75) =  1
lowbit(76) =  4   lowbit(77) =  1   lowbit(78) =  2   lowbit(79) =  1   lowbit(80) = 16
lowbit(81) =  1   lowbit(82) =  2   lowbit(83) =  1   lowbit(84) =  4   lowbit(85) =  1
lowbit(86) =  2   lowbit(87) =  1   lowbit(88) =  8   lowbit(89) =  1   lowbit(90) =  2
lowbit(91) =  1   lowbit(92) =  4   lowbit(93) =  1   lowbit(94) =  2   lowbit(95) =  1
lowbit(96) = 32   lowbit(97) =  1   lowbit(98) =  2   lowbit(99) =  1   lowbit(100) =  4
```

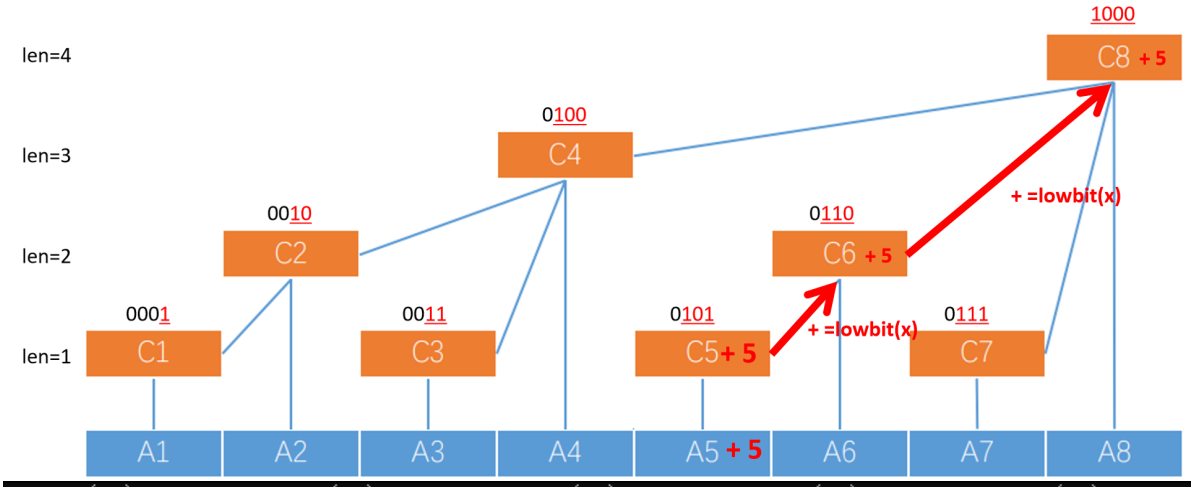
- 核心想法：只保存部分前缀和，这也是很多复杂度均摊的通用想法
- 二进制开分（lowbit（x）手算）
    - 假设我们要查询13（1-13的和），13的二进制为1101

| 二进制 | 对应数组 | 管辖范围 |
| --- | --- | --- |
| 1101 | d[13] | 2^0 |
| 1100 | d[12] | 2^2 |
| 1000 | d[8] | 2^3 |

# 两个操作

## add（i，k）

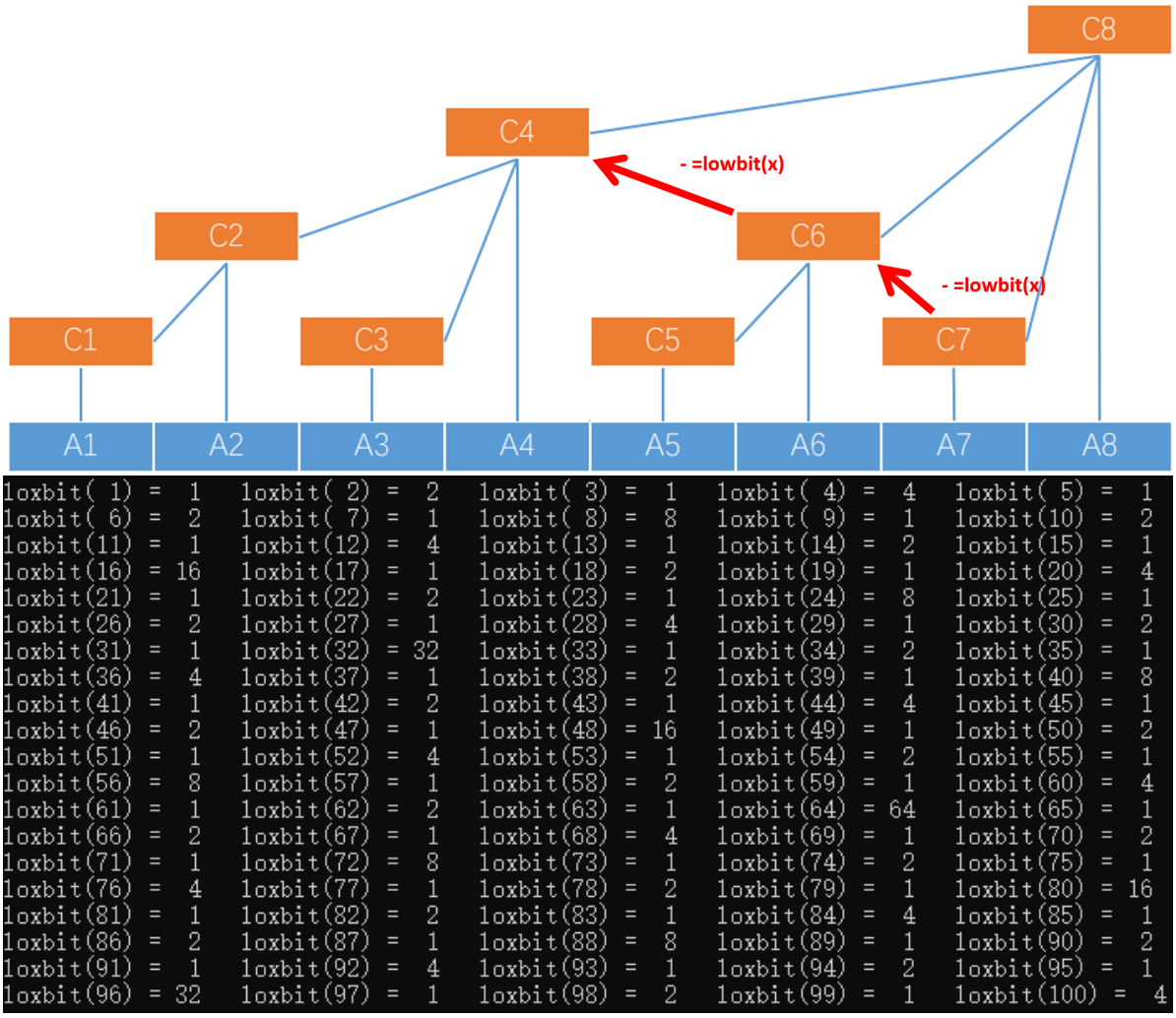
我们以add（5，5）为例子




```
loxbit( 1) =  1   loxbit( 2) =  2   loxbit( 3) =  1   loxbit( 4) =  4   loxbit( 5) =  1
loxbit( 6) =  2   loxbit( 7) =  1   loxbit( 8) =  8   loxbit( 9) =  1   loxbit(10) =  2
loxbit(11) =  1   loxbit(12) =  4   loxbit(13) =  1   loxbit(14) =  2   loxbit(15) =  1
loxbit(16) = 16   loxbit(17) =  1   loxbit(18) =  2   loxbit(19) =  1   loxbit(20) =  4
loxbit(21) =  1   loxbit(22) =  2   loxbit(23) =  1   loxbit(24) =  8   loxbit(25) =  1
loxbit(26) =  2   loxbit(27) =  1   loxbit(28) =  4   loxbit(29) =  1   loxbit(30) =  2
loxbit(31) =  1   loxbit(32) = 32   loxbit(33) =  1   loxbit(34) =  2   loxbit(35) =  1
loxbit(36) =  4   loxbit(37) =  1   loxbit(38) =  2   loxbit(39) =  1   loxbit(40) =  8
loxbit(41) =  1   loxbit(42) =  2   loxbit(43) =  1   loxbit(44) =  4   loxbit(45) =  1
loxbit(46) =  2   loxbit(47) =  1   loxbit(48) = 16   loxbit(49) =  1   loxbit(50) =  2
loxbit(51) =  1   loxbit(52) =  4   loxbit(53) =  1   loxbit(54) =  2   loxbit(55) =  1
loxbit(56) =  8   loxbit(57) =  1   loxbit(58) =  2   loxbit(59) =  1   loxbit(60) =  4
loxbit(61) =  1   loxbit(62) =  2   loxbit(63) =  1   loxbit(64) = 64   loxbit(65) =  1
loxbit(66) =  2   loxbit(67) =  1   loxbit(68) =  4   loxbit(69) =  1   loxbit(70) =  2
loxbit(71) =  1   loxbit(72) =  8   loxbit(73) =  1   loxbit(74) =  2   loxbit(75) =  1
loxbit(76) =  4   loxbit(77) =  1   loxbit(78) =  2   loxbit(79) =  1   loxbit(80) = 16
loxbit(81) =  1   loxbit(82) =  2   loxbit(83) =  1   loxbit(84) =  4   loxbit(85) =  1
loxbit(86) =  2   loxbit(87) =  1   loxbit(88) =  8   loxbit(89) =  1   loxbit(90) =  2
loxbit(91) =  1   loxbit(92) =  4   loxbit(93) =  1   loxbit(94) =  2   loxbit(95) =  1
loxbit(96) = 32   loxbit(97) =  1   loxbit(98) =  2   loxbit(99) =  1   loxbit(100) =  4
```

按图所示可以写出

```
void add(int x, int k){
    for(;x<=n;x+=lowbit(x))//n是数组a的总长度
        c[x] += k;
}
```




### ask（i)

我们以ask（7）为例子

```
int ask(int x){
    int ans = 0;
    for(;x;x-=lowbit(x))
        ans += c[x];
    return ans;
}
```

## 例题

树状数组 模板1（https://www.luogu.com.cn/problem/P3374)

```
#include<cstdio>
int n, m;
int c[1000000];
int lowbit(int x){
    return x&(-x);
}
void add(int x, int k){
    for(;x<=n;x+=lowbit(x))
        c[x] += k;
}
int ask(int x){
    int ans = 0;
```

```
    for(;x;x-=lowbit(x))
        ans += c[x];
    return ans;
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        int a;
        scanf("%d",&a);
        add(i,a);
    }
    for(int i=1;i<=m;i++){
        int ch,x,k;
        scanf("%d%d%d",&ch,&x,&k);
        if (ch == 1)
            add(x,k);
        else
            printf("%d\n",ask(k) - ask(x-1));
    }
    return 0;
}
```