线段树和树状数组

北京大学信息学院 郭炜
GWPL@PKU.EDU.CN

课程网页

http://acm.pku.edu.cn/JudgeOnline/summerschool/pku_acm_train.htm

上机地点：理科1号楼1235

线段树和树状数组

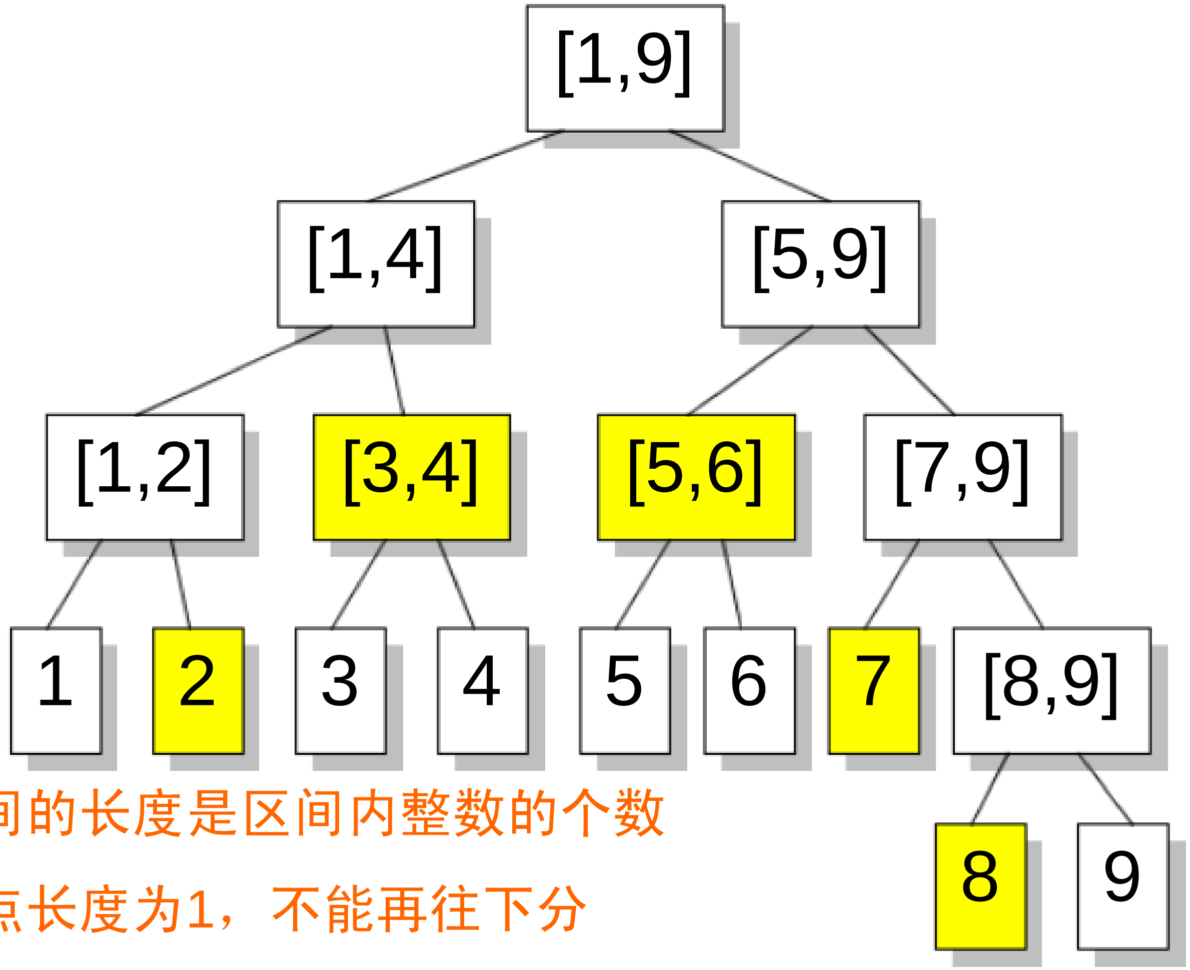
北京大学信息学院 郭炜
GWPL@PKU.EDU.CN

线段树(Interval Tree)

- 实际上还是称为区间树更好理解一些。
- 树：是一棵树，而且是一棵二叉树。
- 线段：树上的每个节点对应于一个线段(还是叫“区间”更容易理解，区间的起点和终点通常为整数)
- 同一层的节点所代表的区间，相互不会重叠。
- 叶子节点的区间是单位长度，不能再分了。

- 线段树是一棵二叉树，树中的每一个结点表示了一个区间[a,b]。a,b通常是整数。每一个叶子节点表示了一个单位区间。对于每一个非叶结点所表示的结点[a,b]，其左儿子表示的区间为[a,(a+b)/2]，右儿子表示的区间为[(a+b)/2,b](除法去尾取整）。

- 区间[1, 9]的线段树和子区间[2, 8]的分解



●每个区间的长度是区间内整数的个数

●叶子节点长度为1，不能再往下分

●若一个节点对应的区间是[a,b],则其子节点对应的区间分别是[a,(a+b)/2]和[(a+b)/2+1,b] （除法去尾取整）

● 线段树的平分构造，实际上是用了二分的方法。线段树是平衡树，它的深度为$\log_2(b-a+1)$。

线段树的特征

- I、线段树的深度不超过logL（L是最长区间的长度）。
- 2、线段树把区间上的任意一条线段都分成不超过2logL条线段。
- 这些结论为线段树能在O(logL)的时间内完成一条线段的插入、删除、查找等工作，提供了理论依据

线段树的构建

- function 以节点v为根建树、v对应区间为[l,r]
- {
-     对节点v初始化
-     if (l!=r)
-     {
-         以v的左孩子为根建树、区间为[l,(l+r)/2]
-         以v的右孩子为根建树、区间为[(l+r)/2+1,r]
-     }
- }

线段树的基本用途

- 线段树适用于和区间统计有关的问题。比如某些数据可以按区间进行划分，按区间动态进行修改，而且还需要按区间多次进行查询，那么使用线段树可以达到较快查询速度。

线段树应用举例

- 给你一个数的序列$A_1A_2 \ldots\ldots A_n$。 并且可能多次进行下列两个操作：

- 1、对序列里面的某个数进行加减
- 2、询问这个序列里面任意一个子序列$A_iA_{i+1} \ldots\ldots A_j$的和是多少。

- 希望第2个操作每次能在logn时间内完成

线段树应用举例

- 显然，[1,n]就是根节点对应的区间
- 可以在每个节点记录该节点对应的区间里面的数的和Sum。
- 对于操作1：因为序列里面A_i最多只会被线段树的Logn个节点覆盖。只要求对线段树覆盖A_i的节点的Sum进行加减操作。

- 对于操作2：同样只需要找到区间所覆盖的区域，然后把所找区域的Sum累加起来。

线段树应用举例

- 如果走到节点[L,R]时，如果要查询的区间就是[L,R]
- (求A_L到A_R的和）那么直接返回该节点的Sum
- 如果不是，则：
- 对于区间[L,R]，取mid=（L+R）/2；
- 然后看要查询的区间与[L,mid]或[mid+1,R]哪个有交集，就进入哪个区间进行进一步查询。
- 最后通过左右儿子区间的Sum值的维护调整当前区间Sum值。
- 因为这个线段树的深度最深的LogN，所以每次遍历操作都在LogN的内完成。但是常数可能很大。

线段树应用举例

- 如果是对区间所对应的一些数据进行修改，过程和查询类似。

- **用线段树解题，关键是要想清楚每个节点要存哪些信息（当然区间起终点，以及左右子节点指针是必须的），以及这些信息如何高效更新，维护，查询。不要一更新就更新到叶子节点，那样更新效率最坏就可能变成O(n)的了。**

- **先建树，然后插入数据，然后更新，查询。**

例题： POJ 3264 Balanced Lineup

给定**Q** $(1 \le Q \le 200{,}000)$个数**A_1,A_2 … A_Q**,，多次求任一区间**$A_i - A_j$**中最大数和最小数的差。

本题树节点结构是什么?

例题： POJ 3264 Balanced Lineup

给定**Q** ($1 \le Q \le 200,000$)个数$\mathbf{A_1}$,$\mathbf{A_2}$ … $\mathbf{A_Q}$,，多次求任一区间$\mathbf{A_i}$ – $\mathbf{A_j}$中最大数和最小数的差。

本题树节点结构：

```
struct CNode
{
	int L,R; //区间起点和终点
	int nMin,nMax;//本区间里的最大最小值
	CNode * pLeft, * pRight;
};
```

Sample Input

6 3
1
7
3
4
2
5
1 5
4 6
2 2

Sample Output

6
3
0

POJ 3468 A Simple Problem with Integers

给定**Q** (1 ≤ *Q* ≤ 100,000)个数**A_1,A_2 … A_Q**,，以及可能多次进行的两个操作:

1) 对某个区间**A_i … A_j**的个数都加**n(n**可变）
2) 求某个区间**A_i … A_j**的数的和

本题树节点要存哪些信息？只存该区间的数的和，行不行？

POJ 3468 A Simple Problem with Integers

只存和，会导致每次加数的时候都要更新到叶子节点，速度太慢，这是必须要避免的。

本题树节点结构：

```
struct CNode
{
        int L ,R; //区间起点和终点
        CNode * pLeft, * pRight;
        long long nSum; //原来的和
        long long Inc; //增量c的累加
}; //本节点区间的和实际上是nSum+Inc*(R-L+1)
```

POJ 3468 A Simple Problem with Integers

在增加时，如果要加的区间正好覆盖一个节点，则增加其节点的**Inc**值，不再往下走，否则要更新**nSum,**再将增量往下传

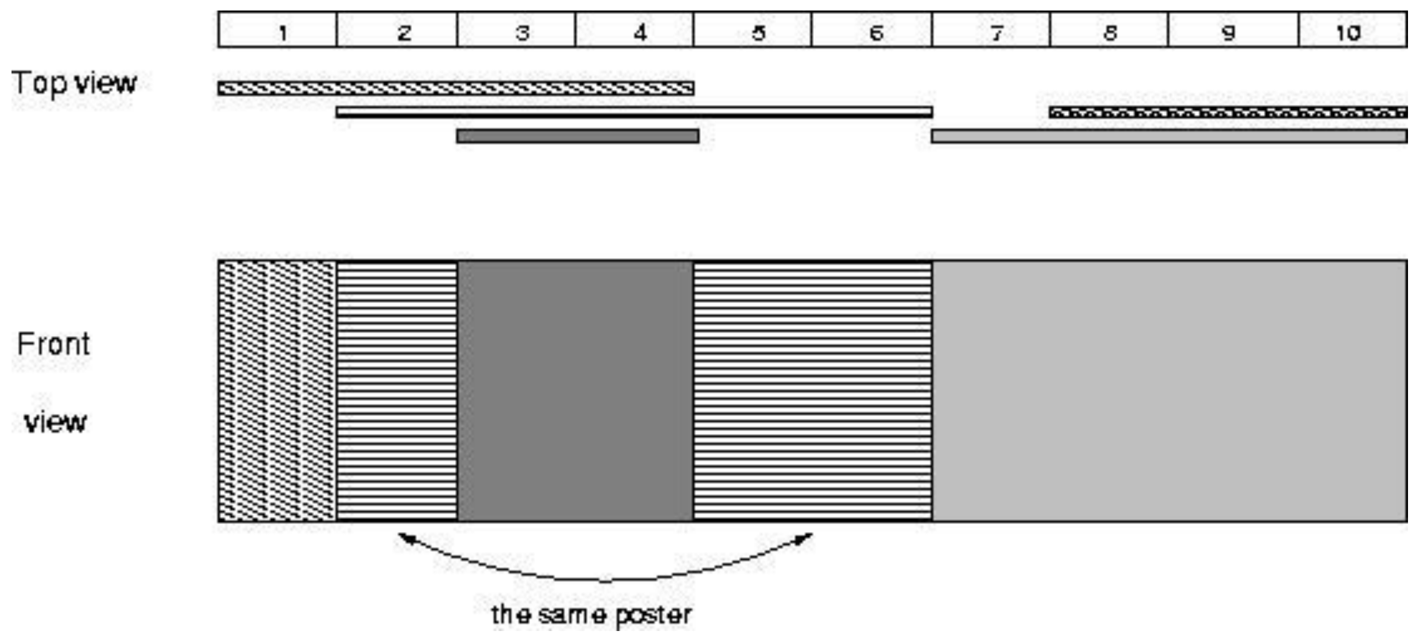
在查询时，如果待查区间不是正好覆盖一个节点，就将节点的**Inc**往下带，然后将**Inc**代表的所有增量累加到**nSum**上后将**Inc**清**0**，接下来再往下查询。

离散化

有时，区间的端点不是整数，或者区间太大导致建树内存开销过大MLE ,那么就需要进行“离散化”后再建树。

POJ 2528 Mayor's posters

给定一些海报，可能互相重叠，告诉你每个海报宽度（高度都一样）和先后叠放次序，问没有被完全盖住的海报有多少张。



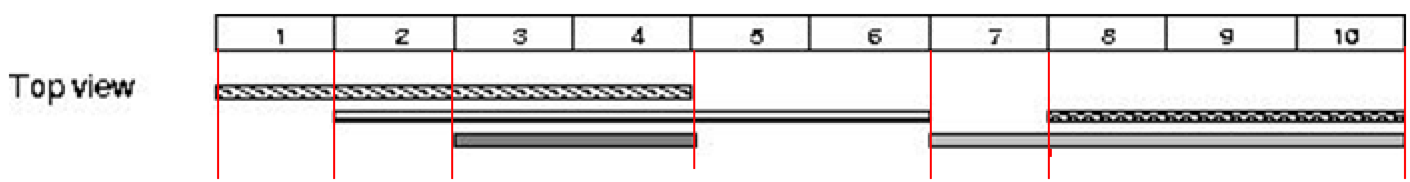
海报最多10,000张，但是墙有10,000,000块瓷砖长。海报端点不会落在瓷砖中间。

POJ 2528 Mayor's posters

如果每个叶子节点都代表一块瓷砖，那么线段树会导致MLE，即单位区间的数目太多。

实际上，由于最多10,000个海报，共计20,000个端点，这些端点把墙最多分成19,999个区间（题意为整个墙都会被盖到）

我们只要对这19,999个区间编号，然后建树即可。这就是离散化。



POJ 2528 Mayor's posters

如果海报端点坐标是浮点数，其实也一样处理。

树节点要保存哪些信息，而且这些信息该如何动态更新呢？

POJ 2528 Mayor's posters

```
struct CNode
{
        int L,R;
        bool bCovered;
         CNode * pLeft, * pRight;
};
```

bCovered表示本区间是否已经完全被海报盖住

关键： 插入数据的顺序 ------ 从底至上依次插入每张海报

POJ 1151 Atlantis

给定一些矩形，其顶点坐标是浮点数，可能互相重叠，问这些矩形覆盖到的面积是多大。

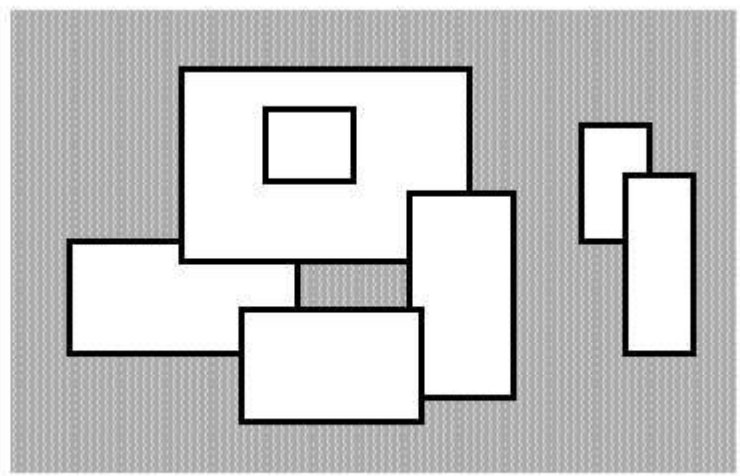


Figure 1. A set of 7 rectangles

用线段树做，先要离散化！！

POJ 1151 Atlantis

在Y轴进行离散化。n个矩形的2n个横边纵坐标共构成最多2n-1个区间的边界，对这些区间编号，建立起线段树。

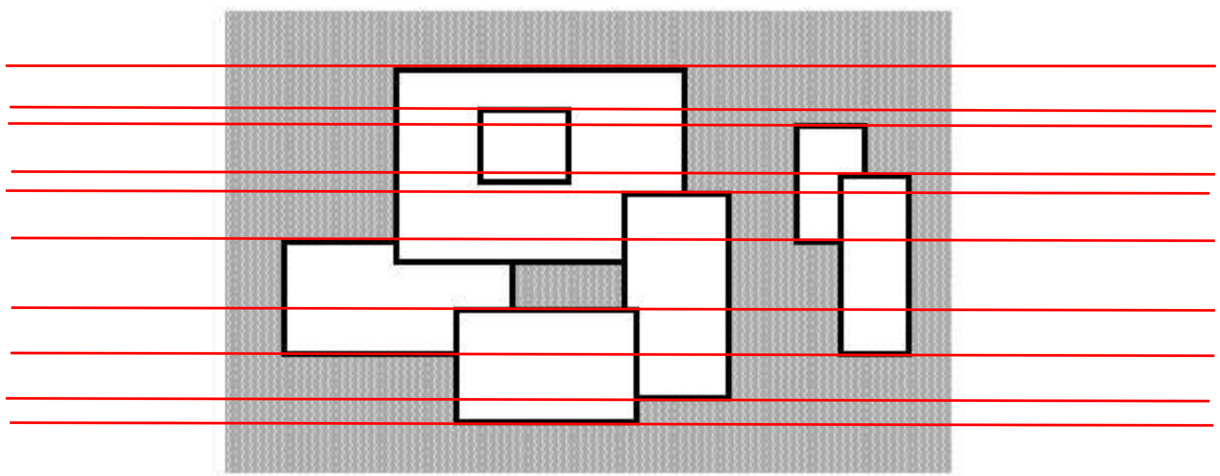


Figure 1. A set of 7 rectangles

POJ 1151 Atlantis

线段树的节点要保存哪些信息？如何将一个个矩形插入线段树？插入过程中这些信息如何更新？怎样查询？

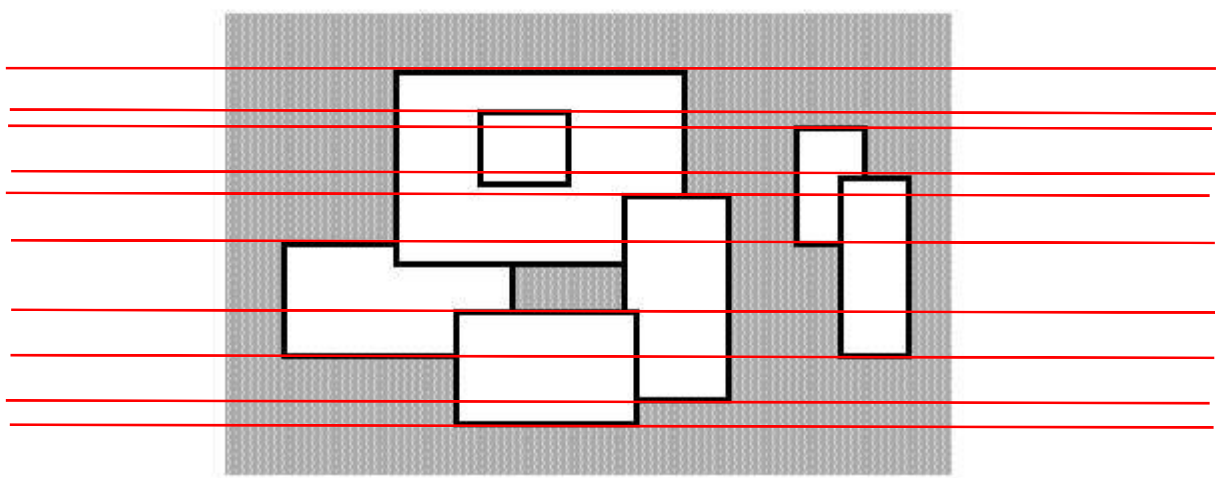


Figure 1. A set of 7 rectangles

POJ 1151 Atlantis

```
struct CNode
{
	int L,R;
	CNode * pLeft, * pRight;
	double Len; //落在本区间的线段总长度
	int Covers;//本区间被完全覆盖的重数
};
```

上面的“线段”指的是真实的线段，即由矩形的边构成的，不是“区间”的意思

POJ 1151 Atlantis

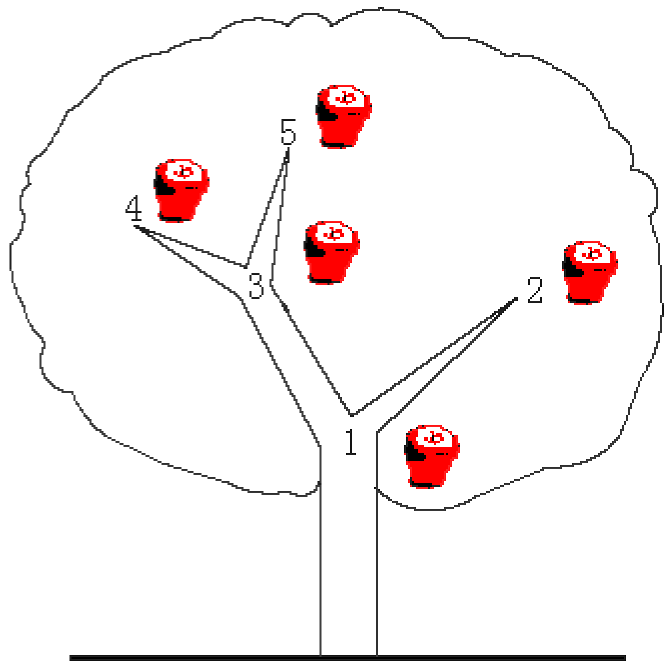
插入数据的顺序：

将矩形的纵边从左到右排序，然后依次将这些纵边插入线段树。要记住哪些纵边是一个矩形的左边(开始边)，哪些纵边是一个矩形的右边（结束边），以便插入时，对Len和Covers做不同的修改。

插入一条边后就更新总覆盖面积的值。

有时，不一定能够一眼看出什么是“区间”，这就要靠仔细观察，造出“区间”来。例如：

POJ 3321 Apple Tree



每个分叉点及末梢可能有苹果（最多1个），每次可以摘掉一个苹果，或有一个苹果新长出来，随时查询某个分叉点往上的子树里，一共有多少个苹果。

POJ 3321 Apple Tree

深度优先遍历整个苹果树，为每个节点标记一个开始时间和结束时间（所有时间都不相同），显然子树里面所有节点的开始和结束时间，都位于子树树根的开始和结束时间之间。

问题变成：

有n个节点，就有2n个开始结束时间，它们构成序列

$$A_1A_2....A_{2n}$$

序列里每个数是0或者1，可变化，随时查询某个区间里数的和。当然由于苹果树上每个放苹果的位置对应于数列里的两个数，所以结果要除以2

树状数组

- 对于序列a，我们设一个数组C
  - $C[i] = a[i - 2^k + 1] + \ldots + a[i]$
  - k为i在二进制下末尾0的个数
  - i从1开始算！
- C即为a的树状数组
- 对于i,如何求2^k？

- 2^k=i &(i^(i-1)) 也就是 i&(-i)

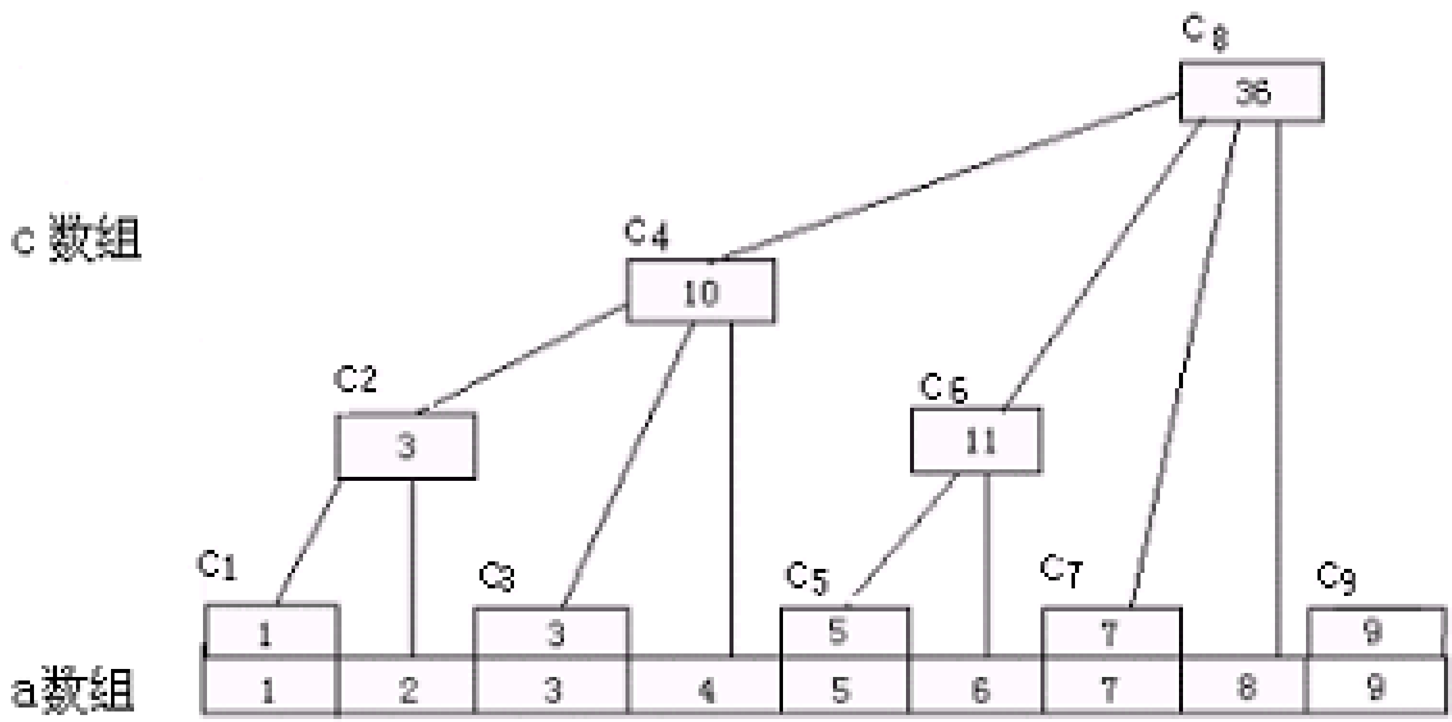
- 以6为例
- $(6)_{10}=(0110)_2$
- xor 6-1=$(5)_{10}=(0101)_2$
- $(0011)_2$
- and $(6)_{10}= (0110)_2$
- $(0010)_2 = (4)_{10}$
- 通常我们用lowbit(x)表示x对应的2^k ,
- lowbit(x) = x&(-x)
- lowbit(x) 实际上就是x的二进制表示形式留下最右边的1，其他位都变成0

C[i] = a[i-lowbit(i)+1] + …+ a[i]

C包含哪些项看上去没有规律

- C1=A1
- C2=A1+A2
- C3=A3
- C4=A1+A2+A3+A4
- C5=A5
- C6=A5+A6
- C7=A7
- C8=A1+A2+A3+A4+A5+A6+A7+A8
- …………
- C16=A1+A2+A3+A4+A5+A6+A7+A8+A9+A10+A11+A12+A13+A14+A15+A16

树状数组图示



树状数组的好处在于能快速求任意区间的和 a[i] + a[i+1] + … + a[j]

设sum(k) = a[1]+a[2]+…+a[k]

则 a[i] + a[i+1] + … + a[j] = sum(j)-sum(i-1)

有了树状数组，sum(k)就能在O(logN)时间内求出，N是a数组元素个数。而且更新一个**a**的元素所花的时间也是**O(logN)**的**(a**更新了**C**也得更新**)**。

为什么呢？

根据C的构成规律，可以发现sum(k)可以表示为：

$sum(k) = C[n_1]+C[n_2] + \ldots + C[n_m]$

其中 $n_m = k$

$n_{i-1} = n_i - lowbit(n_i)$ 而且 $n_1 - lowbit(n_1)$ 必须小于或等于0

如：$sum(6) = C[4]+C[6]$

lowbit(x) 实际上就是x的二进制表示形式留下最右边的1，其他位都变成0

那么，sum(k)最多有几项呢？这个决定了求区间和的时间复杂度

那么，sum(k)最多有几项呢？

$sum(k) = C[n_1]+C[n_2] + \ldots + C[n_m]$

其中 $n_m = k$

$n_{i-1} = n_i - lowbit(n_i)$

lowbit(x) 实际上就是x的二进制表示形式留下最右边的1，其他位都变成0

n_i - lowbit(n_i) 是什么样子？就是 n_i的二进制去掉最右边的1

k 的二进制里最多有几个1？ $\log_2 k$个

sum(k)最多$\log_2 k$项，所以本次求和的复杂度就是 $\log_2 k$

那么，为什么

$sum(k) = C[n_1]+C[n_2] + \ldots + C[n_m]$

其中 $n_m = k$

$n_{i-1} = n_i - lowbit(n_i)$

把每一项$C[n_i]$拆开细算，把n_i表示成2的整数次幂的和就能发现。证明略。

$C[i] = a[i-lowbit(i)+1] + \ldots + a[i]$

i - lowbit(i)+1 是什么？就是i把最右边的1去掉，然后再加1

更新一个a元素，C也要跟着更新，复杂度是多少呢？即C里有几项要更新呢？

- C1=A1
- C2=A1+A2
- C3=A3
- C4=A1+A2+A3+A4
- C5=A5
- C6=A5+A6
- C7=A7
- C8=A1+A2+A3+A4+A5+A6+A7+A8
- …………
- C16=A1+A2+A3+A4+A5+A6+A7+A8+A9+A10+A11+A12+A13+A14+A15+A16

更新一个a元素，C也要跟着更新，复杂度是多少呢？即C里有几项要更新呢？

如果a[i]更新了，那么以下的几项都需要更新：

$C[n_1], C[n_2], \ldots C[n_m]$

其中，$n_1 = i$，$n_{i+1} = n_i + \mathrm{lowbit}(n_i)$

$n_m + \mathrm{lowbit}(n_m)$ 必须大于 a 的元素个数 N

同理，总的来说更新一个元素的时间，也是logN的

初始状态下由a构建树状数组C的时间复杂度是多少？

显然是 O(N)的

因为

$C[k] = sum(k) – sum(k\text{-}lowbit(k))$

证：

$sum(k) = C[n_1]+C[n_2] + \ldots + C[n_{m-1}] + C[n_m]\ (n_m = k)$

$n_{m-1} = k\text{-}lowbit(k)$

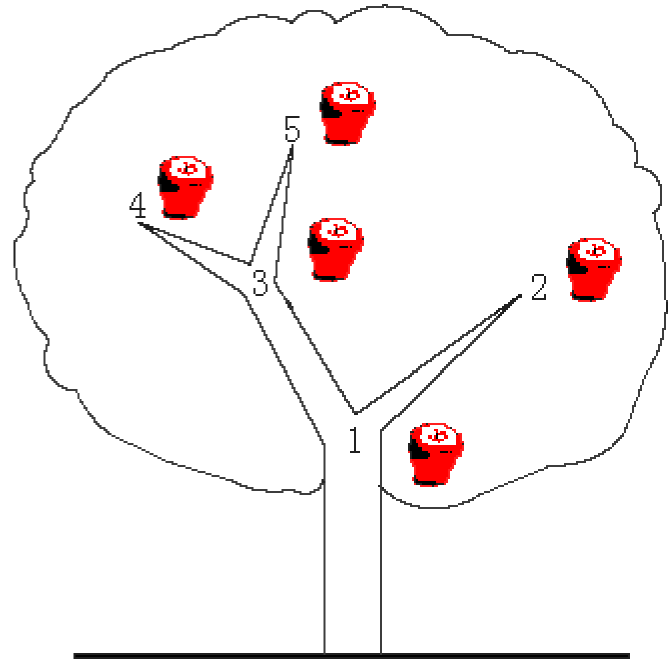
$sum(k\text{-}lowbit(k)) = C[n_1]+C[n_2] + \ldots + C[n_{m-1}]$

所以，树状数组适合单个元素经常修改而且还反复要求部分的区间的和的情况。

上述问题虽然也可以用线段树解决，但是用树状数组来做，编程效率和程序运行效率都更高

如果每次要修改的不是单个元素，而是一个区间，那就不能用树状数组了(效率过低）。

POJ 3321 Apple Tree



每个分叉点及末梢可能有苹果（最多1个），每次可以摘掉一个苹果，或有一个苹果新长出来，随时查询某个分叉点往上的子树里，一共有多少个苹果。

此题可用树状数组来做

根据题意，一开始时，所有能长苹果的地方都有苹果

Sample Input

3
1 2
1 3
3
Q 1
C 2
Q 1

Sample Output

3
2

二维树状数组

- 原始数组和树状数组都是二维的
- C[x][y] = Sum(a[i][j])
- x-lowbit[x]+1 <= i <=x
- y-lowbit[y]+1 <= j <=y
- 用于快速求数字子矩阵的和

POJ 1195 Mobile phones

一个由数字构成的大矩阵，能进行两种操作

1) 对矩阵里的某个数加上一个整数（可正可负）

2) 查询某个子矩阵里所有数字的和

要求对每次查询，输出结果

Instruction	Parameters	Meaning
0	S	Initialize the matrix size to S * S containing all zeros. This instruction is given only once and it will be the first instruction.
1	X Y A	Add A to the number of active phones in table square (X, Y). A may be positive or negative.
2	L B R T	Query the current sum of numbers of active mobile phones in squares (X, Y), where L <= X <= R, B <= Y <= T
3		Terminate program. This instruction is given only once and it will be the last instruction.

Sample Input

0 4
1 1 2 3
2 0 0 2 2
1 1 1 2
1 1 2 -1
2 1 1 2 3
3

Sample Output

3
4

POJ题目推荐：

2182, 2352, 1177, 3667,3067