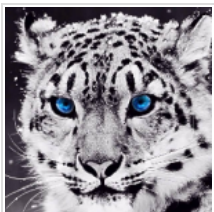


个人资料



MetalSeed

访问： 164651次

积分： 2740

等级： **BLOG > 5**

排名： 第5570名

原创： 104篇 转载： 20篇

译文： 1篇 评论： 106条

文章搜索

文章分类

嵌入式Android (5)

嵌入式linux (14)

STM32笔记 (13)

ACM回忆 (47)

MCU相关 (28)

随笔 小记 (2)

东西 小站 (5)

文章存档

2015年03月 (1)

2015年02月 (3)

2015年01月 (2)

2014年11月 (1)

2014年10月 (5)

展开

阅读排行

数据结构专题——线段树
(16085)主席树介绍
(11194)Android蓝牙串口通信模
(10787)[CSDN学院讲师招募，诚邀您加入！](#)[博客Markdown编辑器上线啦](#)[PMBOK第五版精讲视频教程](#)[火星入敏捷开发1001问](#)

数据结构专题——线段树

分类： **ACM回忆**

2012-10-04 14:18

16113人阅读

评论(22)

收藏

举报

build

query

存储

c

目录(?)

[+]

线段树

转载请注明出处，谢谢！<http://blog.csdn.net/metalseed/article/details/8039326>

持续更新中...

一：线段树基本概念

1：概述

线段树，类似区间树，是一个完全二叉树，它在各个节点保存一条线段（数组中的一段子数组），主要用于高效解决连续区间的动态查询问题，由于二叉结构的特性，它基本能保持每个操作的复杂度为 $O(\lg N)$ ！

性质：父亲的区间是 $[a, b]$ ， $(c = (a+b)/2)$ 左儿子的区间是 $[a, c]$ ，右儿子的区间是 $[c+1, b]$ ，线段树需要的空间为数组大小的四倍

2：基本操作（demo用的是查询区间最小值）

线段树的主要操作有：

(1)：线段树的构造 `void build(int node, int begin, int end);`

主要思想是递归构造，如果当前节点记录的区间只有一个值，则直接赋值，否则递归构造左右子树，最后回

```
#include <iostream>
using namespace std;

const int maxind = 256;
int segTree[maxind * 4 + 10];
int array[maxind];
/* 构造函数，得到线段树 */
void build(int node, int begin, int end)
{
    if (begin == end)
        segTree[node] = array[begin]; /* 只有一个元素,节点记录该元素 */
    else
    {
        /* 递归构造左右子树 */
        build(2*node, begin, (begin+end)/2);
        build(2*node+1, (begin+end)/2+1, end);

        /* 回溯时得到当前node节点的线段信息 */
        if (segTree[2 * node] <= segTree[2 * node + 1])
            segTree[node] = segTree[2 * node];
        else
            segTree[node] = segTree[2 * node + 1];
    }
}

int main()
{
    array[0] = 1, array[1] = 2, array[2] = 2, array[3] = 4, array[4] = 1, array[5] = 3;
    build(1, 0, 5);
    for(int i = 1; i <= 20; ++i)
        cout << "seg" << i << " = " << segTree[i] << endl;
    return 0;
}
```

此build构造的树如图：

NE555 + CD4017流水灯 (10126)
两款主流摄像头OV7620 (5683)
D/A与A/D转换器 (3297)
51单片机导论 (2999)
51单片机进阶 (2528)
POJ 2528 Mayor's poste (2314)
STM32F407 Discovery u (2307)

评论排行

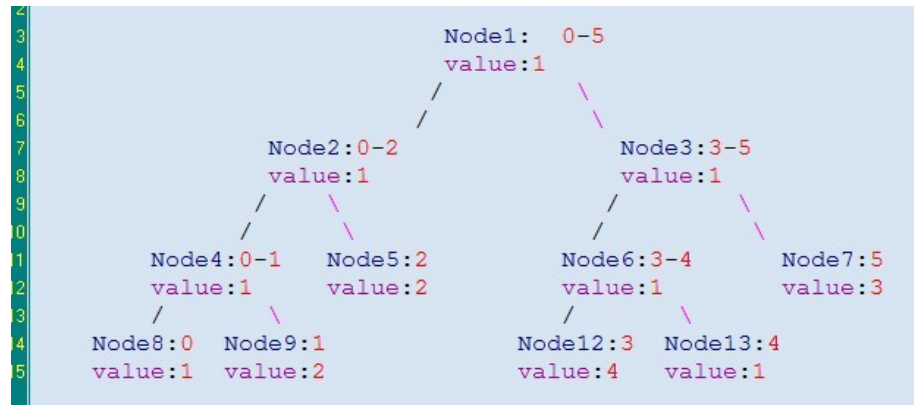
数据结构专题——线段树 (22)
Android蓝牙串口通信模块 (11)
主席树介绍 (9)
C代码优化方案 (4)
51单片机进阶 (4)
基于51的爱心流水灯源码 (4)
Video for linux 2 example (3)
悬挂运动控制系统 源代码 (3)
Log of Grade Two (3)
51单片机导论 (3)

推荐文章

* 【ShaderToy】开篇
* FFmpeg源代码简单分析:
avio_open2()
* 技能树之旅: 从模块分离到测试
* Qt5官方demo解析集36——Wiggly
Example
* Unity3d HDR和Bloom效果 (高动态范围图像和泛光)
* Android的Google官方设计指南

最新评论

天马行空的ACM现场赛回顾
MetalSeed: @chaoyueziji123:算是吧, 你是nupt的学弟?
天马行空的ACM现场赛回顾
chaoyueziji123: 666666, 学长现在是工作了吗
主席树介绍
Dash: 代码是极好的, 只是太有特色了。。。~
STM32F407 Discovery uart1串口
garryxin: @qzxf: 咨询: STM32F4 Discovery的串口怎么用? 是不是只要把USB线 (ST-LINK) ...
数据结构专题——线段树
楼上小字: 受教了。。。很详细
STM32F407 Discovery uart1串口
qzxf: 我串口调试助手接收到的总是乱码, 楼主能否把你的工程文件发给我, 万分感谢!
1290815000@qq....
OLED驱动 (51单片机)
MetalSeed: @u014787043: 如果确保硬件没问题, 看下液晶控制器的datasheet里初始化部分是否...
OLED驱动 (51单片机)
昵称都难找: 你好, 我参考了很多oled例程, 都差不多, 你这个很好。可是我用ATmega8驱动的时候却好像驱动不了...
两款主流摄像头OV7620与OV7670
dd_y: 您好, 我想请教几个我呢体, 做图像采集, 摄像头应该怎么选型呢, 我做图像的处理, 不要求实时, 对画面要求不...
数据结构专题——线段树
MetalSeed: 228003430



(2): 区间查询int query(int node, int begin, int end, int left, int right);

(其中node为当前查询节点, begin,end为当前节点存储的区间, left,right为此次query所要查询的区间) 主要思想是把所要查询的区间[a,b]划分为线段树上的节点, 然后将这些节点代表的区间合并起来得到所需值。比如前面一个图中所示的树, 如果询问区间是[0,2], 或者询问的区间是[3,3], 不难直接找到对应的节点回

```
int query(int node, int begin, int end, int left, int right)
{
    int p1, p2;

    /* 查询区间和要求的区间没有交集 */
    if (left > end || right < begin)
        return -1;

    /* if the current interval is included in the query interval return segTree[node] */
    if (begin >= left && end <= right)
        return segTree[node];

    /* compute the minimum position in the left and right part of the interval */
    p1 = query(2 * node, begin, (begin + end) / 2, left, right);
    p2 = query(2 * node + 1, (begin + end) / 2 + 1, end, left, right);

    /* return the expect value */
    if (p1 == -1)
        return p2;
    if (p2 == -1)
        return p1;
    if (p1 <= p2)
        return p1;
    return p2;
}
```

可见, 这样的过程一定选出了尽量少的区间, 它们相连后正好涵盖了整个[left,right], 没有重复也没有遗漏。同时, 考虑到线段树上每层的节点最多会被选取2个, 一共选取的节点数也是 $O(\log n)$ 的, 因此查询的时间复杂度也是 $O(\log n)$ 。

线段树并不适合所有区间查询情况, 它的使用条件是“相邻的区间的信息可以被合并成两个区间的并区间的信息”。即问题是可以被分解解决的。

(3): 区间或节点的更新 及 线段树的动态维护update (这是线段树核心价值所在, 节点中的标记域可以解动态维护需要用到标记域, 延迟标记等。

a: 单节点更新

```
void Updata(int node, int begin, int end, int ind, int add)/*单节点更新*/
{
    if( begin == end )
    {
        segTree[node] += add;
        return ;
    }
    int m = ( left + right ) >> 1;
    if(ind <= m)
        Updata(node * 2, left, m, ind, add);
    else
        Updata(node * 2 + 1, m + 1, right, ind, add);
    /*回溯更新父节点*/
    segTree[node] = min(segTree[node * 2], segTree[node * 2 + 1]);
}
```

b: 区间更新 (线段树中最有用的)

需要用到延迟标记, 每个结点新增加一个标记, 记录这个结点是否被进行了某种修改操作(这种修改操作会影响其子结点)。对于任意区间的修改, 我们先按照查询的方式将其划分成线段树中的结点, 然后修改这些结点的信息, 并给这些结点标上代表这种修改操作的标记。在修改和查询的时候, 如果我们到了一

个结点p，并且决定考虑其子结点，那么我们就看看结点p有没有标记，如果有，就要按照标记修改其子结点的信息，并且给子结点都标上相同的标记，同时消掉p的标记。（优点在于，不用将区间内的所有值都暴力更新，大大提高效率，因此区间更新是最优用的操作）

void Change来自dongxicheng.org

```
void Change(node *p, int a, int b) /* 当前考察结点为p，修改区间为(a,b]*/
{
    if (a <= p->Left && p->Right <= b)
        /* 如果当前结点的区间包含在修改区间内*/
        {
            ..... /* 修改当前结点的信息，并标上标记*/
            return;
        }
    Push_Down(p); /* 把当前结点的标记向下传递*/
    int mid = (p->Left + p->Right) / 2; /* 计算左右子结点的分隔点
    if (a < mid) Change(p->Lch, a, b); /* 和左孩子有交集，考察左子结点*/
    if (b > mid) Change(p->Rch, a, b); /* 和右孩子有交集，考察右子结点*/
    Update(p); /* 维护当前结点的信息（因为其子结点的信息可能有更改）*/
}
```

3: 主要应用

- (1): 区间最值查询问题 （见模板1）
- (2): 连续区间修改或者单节点更新的动态查询问题 （见模板2）
- (3): 多维空间的动态查询 （见模板3）

二：典型模板

模板1:

RMQ，查询区间最值下标---min

```
#include<iostream>
using namespace std;

#define MAXN 100
#define MAXIND 256 //线段树节点个数

//构建线段树,目的:得到M数组.
void build(int node, int b, int e, int M[], int A[])
{
    if (b == e)
        M[node] = b; //只有一个元素,只有一个下标
    else
    {
        build(2 * node, b, (b + e) / 2, M, A);
        build(2 * node + 1, (b + e) / 2 + 1, e, M, A);

        if (A[M[2 * node]] <= A[M[2 * node + 1]])
            M[node] = M[2 * node];
        else
            M[node] = M[2 * node + 1];
    }
}

//找出区间 [i, j] 上的最小值的索引
int query(int node, int b, int e, int M[], int A[], int i, int j)
{
    int p1, p2;

    //查询区间和要求的区间没有交集
    if (i > e || j < b)
        return -1;

    if (b >= i && e <= j)
        return M[node];

    p1 = query(2 * node, b, (b + e) / 2, M, A, i, j);
    p2 = query(2 * node + 1, (b + e) / 2 + 1, e, M, A, i, j);

    //return the position where the overall
    //minimum is
    if (p1 == -1)
        return M[node] = p2;
    if (p2 == -1)
        return M[node] = p1;
```

```

        if (A[p1] <= A[p2])
            return M[node] = p1;
        return M[node] = p2;
    }

int main()
{
    int M[MAXIND]; //下标1起才有意义,否则不是二叉树,保存下标编号节点对应区间最小值的下标.
    memset(M,-1,sizeof(M));
    int a[]={3,4,5,7,2,1,0,3,4,5};
    build(1, 0, sizeof(a)/sizeof(a[0])-1, M, a);
    cout<<query(1, 0, sizeof(a)/sizeof(a[0])-1, M, a, 0, 5)<<endl;
    return 0;
}

```

模板2:

连续区间修改或者单节点更新的动态查询问题 (此模板查询区间和)

```

#include <cstdio>
#include <algorithm>
using namespace std;

#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
#define root 1 , N , 1
#define LL long long
const int maxn = 111111;
LL add[maxn<<2];
LL sum[maxn<<2];
void PushUp(int rt) {
    sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void PushDown(int rt,int m) {
    if (add[rt]) {
        add[rt<<1] += add[rt];
        add[rt<<1|1] += add[rt];
        sum[rt<<1] += add[rt] * (m - (m >> 1));
        sum[rt<<1|1] += add[rt] * (m >> 1);
        add[rt] = 0;
    }
}
void build(int l,int r,int rt) {
    add[rt] = 0;
    if (l == r) {
        scanf("%lld",&sum[rt]);
        return ;
    }
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
    PushUp(rt);
}
void update(int L,int R,int c,int l,int r,int rt) {
    if (L <= l && r <= R) {
        add[rt] += c;
        sum[rt] += (LL)c * (r - l + 1);
        return ;
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
    PushUp(rt);
}
LL query(int L,int R,int l,int r,int rt) {
    if (L <= l && r <= R) {
        return sum[rt];
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    LL ret = 0;
    if (L <= m) ret += query(L , R , lson);
    if (m < R) ret += query(L , R , rson);
    return ret;
}
int main() {
    int N , Q;
    scanf("%d%d",&N,&Q);
    build(root);
    while (Q --) {
        char op[2];
        int a , b , c;
        scanf("%s",op);
        if (op[0] == 'Q') {
            scanf("%d%d",&a,&b);
            printf("%lld\n",query(a , b ,root));
        } else {
            scanf("%d%d%d",&a,&b,&c);
            update(a , b , c , root);
        }
    }
    return 0;
}

```

模板3:

多维空间的动态查询

三：练习题目

下面是hh线段树代码，典型练习哇~

在代码前先介绍一些我的线段树风格：

- maxn是题目给的最大区间，而节点数要开4倍，确切的来说节点数要开大于maxn的最小 2^x 的两倍
- lson和rson分辨表示结点的左儿子和右儿子，由于每次传参数的时候都固定是这几个变量，所以可以用预定义比较方便的表示
- 以前的写法是另外开两个数组记录每个结点所表示的区间，其实这个区间不必保存，一边算一边传下去就行，只需要写函数的时候多两个参数，结合lson和rson的预定义可以很方便
- PushUP(int rt)是把当前结点的信息更新到父结点
- PushDown(int rt)是把当前结点的信息更新给儿子结点
- rt表示当前子树的根(root)，也就是当前所在的结点

整理这些题目后我觉得线段树的题目整体上可以分成以下四个部分：

单点更新: 最最基础的线段树，只更新叶子节点，然后把信息用PushUP(int r)这个函数更新上来

- **hdu1166 敌兵布阵**
 - 题意: 0(-1)
 - 思路: 0(-1)
- 线段树功能: update: 单点增减 query: 区间求和

code:

```
#include<cstring>
#include<iostream>

#define M 50005
#define lson l,m,rt<<1
#define rson m+1,r,rt<<1|1
/*left,right,root,middle*/

int sum[M<<2];

inline void PushPlus(int rt)
{
    sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}

void Build(int l, int r, int rt)
{
    if(l == r)
    {
        scanf("%d",
            return ;
        }
    int m = ( l + r );

    Build(lson);
    Build(rson);
    PushPlus(rt);
}

void Udata(int p, int add, int l, int r, int rt)
{
    if( l == r )
    {
        sum[rt] += add;
        return ;
    }
    int m = ( l + r ) >> 1;
    if(p <= m)
        Udata(p, add, lson);
    else
        Udata(p, add, rson);

    PushPlus(rt);
}

int Query(int L,int R,int l,int r,int rt)
{
    if( L <= l && r <= R )
    {
        return sum[rt];
    }
}
```

```

int m = ( l + r ) >> 1;
int ans=0;
if(L<=m )
    ans+=Query(L,R,lson);
if(R>m)
    ans+=Query(L,R,rson);

return ans;
}
int main()
{
    int T, n, a, b;
    scanf("%d",&T);
    for( int i = 1; i <= T; ++i )
    {
        printf("Case %d:\n",i);
        scanf("%d",&n);
        Build(1,n,1);

        char op[10];

        while( scanf("%s",op) &&op[0]!='E' )
        {
            scanf("%d %d", &a, &b);
            if(op[0] == 'Q')
                printf("%d\n",Query(a,b,1,n,1));
            else if(op[0] == 'S')
                Updata(a,-b,1,n,1);
            else
                Updata(a,b,1,n,1);
        }
    }
    return 0;
}

```

hdu1754 I Hate It

题意:0(-1)

思路:0(-1)

线段树功能:update:单点替换 query:区间最值

```

#include <cstdio>
#include <algorithm>
using namespace std;

#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
const int maxn = 222222;
int MAX[maxn<<2];
void PushUP(int rt) {
    MAX[rt] = max(MAX[rt<<1] , MAX[rt<<1|1]);
}
void build(int l,int r,int rt) {
    if (l == r) {
        scanf("%d",&MAX[rt]);
        return ;
    }
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
    PushUP(rt);
}
void update(int p,int sc,int l,int r,int rt) {
    if (l == r) {
        MAX[rt] = sc;
        return ;
    }
    int m = (l + r) >> 1;
    if (p <= m) update(p , sc , lson);
    else update(p , sc , rson);
    PushUP(rt);
}
int query(int L,int R,int l,int r,int rt) {
    if (L <= l && r <= R) {
        return MAX[rt];
    }
    int m = (l + r) >> 1;
    int ret = 0;
    if (L <= m) ret = max(ret , query(L , R , lson));
    if (R > m) ret = max(ret , query(L , R , rson));
    return ret;
}
int main() {
    int n , m;
    while (~scanf("%d%d",&n,&m)) {
        build(1 , n , 1);
        while (m --) {
            char op[2];
            int a , b;
            scanf("%s%d%d",op,&a,&b);
            if (op[0] == 'Q') printf("%d\n",query(a , b , 1 , n , 1));
            else update(a , b , 1 , n , 1);
        }
    }
    return 0;
}

```

hdu1394 Minimum Inversion Number

题意:求Inversion后的最小逆序数

思路:用 $O(n\log n)$ 复杂度求出最初逆序数后,就可以用 $O(1)$ 的复杂度分别递推出其他解

线段树功能:update:单点增减 query:区间求和

```
#include <cstdio>
#include <algorithm>
using namespace std;

#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
const int maxn = 5555;
int sum[maxn<<2];
void PushUP(int rt) {
    sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void build(int l,int r,int rt) {
    sum[rt] = 0;
    if (l == r) return ;
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
}
void update(int p,int l,int r,int rt) {
    if (l == r) {
        sum[rt] ++;
        return ;
    }
    int m = (l + r) >> 1;
    if (p <= m) update(p , lson);
    else update(p , rson);
    PushUP(rt);
}
int query(int L,int R,int l,int r,int rt) {
    if (L <= l && r <= R) {
        return sum[rt];
    }
    int m = (l + r) >> 1;
    int ret = 0;
    if (L <= m) ret += query(L , R , lson);
    if (R > m) ret += query(L , R , rson);
    return ret;
}
int x[maxn];
int main() {
    int n;
    while (~scanf("%d",&n)) {
        build(0 , n - 1 , 1);
        int sum = 0;
        for (int i = 0 ; i < n ; i++) {
            scanf("%d",&x[i]);
            sum += query(x[i] , n - 1 , 0 , n - 1 , 1);
            update(x[i] , 0 , n - 1 , 1);
        }
        int ret = sum;
        for (int i = 0 ; i < n ; i++) {
            sum += n - x[i] - x[i] - 1;
            ret = min(ret , sum);
        }
        printf("%d\n",ret);
    }
    return 0;
}
```

hdu2795 Billboard

题意:h*w的木板,放进一些l*L的物品,求每次放空间能容纳且最上边的位子

思路:每次找到最大值的位子,然后减去L

线段树功能:query:区间求最大值的位子(直接把update的操作在query里做了)

```
#include <cstdio>
#include <algorithm>
using namespace std;

#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1
const int maxn = 222222;
int h , w , n;
int MAX[maxn<<2];
void PushUP(int rt) {
    MAX[rt] = max(MAX[rt<<1] , MAX[rt<<1|1]);
}
void build(int l,int r,int rt) {
    MAX[rt] = w;
    if (l == r) return ;
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
}
int query(int x,int l,int r,int rt) {
    if (l == r) {
        MAX[rt] -= x;
        return l;
    }
    int m = (l + r) >> 1;
    int ret = (MAX[rt<<1] >= x) ? query(x , lson) : query(x , rson);
    PushUP(rt);
    return ret;
}
int main() {
    while (~scanf("%d%d%d",&h,&w,&n)) {
        if (h > n) h = n;
        build(1 , h , 1);
        while (n--) {
            int x;
            scanf("%d",&x);
        }
    }
}
```

```

        if (MAX[1] < x) puts("-1");
        else printf("%d\n", query(x, 1, h, 1));
    }
    return 0;
}

```

成段更新 (通常这对初学者来说是一道坎), 需要用到延迟标记 (或者说懒惰标记), 简单来说就是每次更新的时候不要更新到底, 用延迟标记使得更新延迟到下次需要更新 or 询问到的时候

hdu1698 Just a Hook

题意: 0(-1)

思路: 0(-1)

线段树功能: update: 成段替换 (由于只 query 一次总区间, 所以可以直接输出 1 结点的信息)

```

#include <cstdio>
#include <algorithm>
using namespace std;

#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1
const int maxn = 111111;
int h, w, n;
int col[maxn << 2];
int sum[maxn << 2];
void PushUp(int rt) {
    sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}
void PushDown(int rt, int m) {
    if (col[rt]) {
        col[rt << 1] = col[rt << 1 | 1] = col[rt];
        sum[rt << 1] = (m - (m >> 1)) * col[rt];
        sum[rt << 1 | 1] = (m >> 1) * col[rt];
        col[rt] = 0;
    }
}
void build(int l, int r, int rt) {
    col[rt] = 0;
    sum[rt] = 1;
    if (l == r) return;
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
    PushUp(rt);
}
void update(int L, int R, int c, int l, int r, int rt) {
    if (L <= l && r <= R) {
        col[rt] = c;
        sum[rt] = c * (r - l + 1);
        return;
    }
    PushDown(rt, r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L, R, c, lson);
    if (R > m) update(L, R, c, rson);
    PushUp(rt);
}
int main() {
    int T, n, m;
    scanf("%d", &T);
    for (int cas = 1; cas <= T; cas++) {
        scanf("%d", &n);
        build(1, n, 1);
        while (m--) {
            int a, b, c;
            scanf("%d%d%d", &a, &b, &c);
            update(a, b, c, 1, n, 1);
        }
        printf("Case %d: The total value of the hook is %d.\n", cas, sum[1]);
    }
    return 0;
}

```

poj3468 A Simple Problem with Integers

题意: 0(-1)

思路: 0(-1)

线段树功能: update: 成段增减 query: 区间求和

```

#include <cstdio>
#include <algorithm>
using namespace std;

#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1
#define LL long long
const int maxn = 111111;
LL add[maxn << 2];
LL sum[maxn << 2];
void PushUp(int rt) {
    sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}
void PushDown(int rt, int m) {
    if (add[rt]) {
        add[rt << 1] += add[rt];
        add[rt << 1 | 1] += add[rt];
        sum[rt << 1] += add[rt] * (m - (m >> 1));
        sum[rt << 1 | 1] += add[rt] * (m >> 1);
        add[rt] = 0;
    }
}

```



```

    }
}
void build(int l,int r,int rt) {
    add[rt] = 0;
    if (l == r) {
        scanf("%lld",&sum[rt]);
        return ;
    }
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
    PushUp(rt);
}
void update(int L,int R,int c,int l,int r,int rt) {
    if (L <= l && r <= R) {
        add[rt] += c;
        sum[rt] += (LL)c * (r - l + 1);
        return ;
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
    PushUp(rt);
}
LL query(int L,int R,int l,int r,int rt) {
    if (L <= l && r <= R) {
        return sum[rt];
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    LL ret = 0;
    if (L <= m) ret += query(L , R , lson);
    if (m < R) ret += query(L , R , rson);
    return ret;
}
int main() {
    int N , Q;
    scanf("%d%d",&N,&Q);
    build(1 , N , 1);
    while (Q --) {
        char op[2];
        int a , b , c;
        scanf("%s",op);
        if (op[0] == 'Q') {
            scanf("%d%d",&a,&b);
            printf("%lld\n",query(a , b , 1 , N , 1));
        } else {
            scanf("%d%d%d",&a,&b,&c);
            update(a , b , c , 1 , N , 1);
        }
    }
    return 0;
}

```

poj2528 Mayor's posters

题意: 在墙上贴海报, 海报可以互相覆盖, 问最后可以看见几张海报

思路: 这题数据范围很大, 直接搞超时+超内存, 需要离散化:

离散化简单的来说就是只取我们需要的值来用, 比如说区间[1000,2000],[1990,2012] 我们用不到 $[-\infty, 999]$ [1001,1989][1991,1999][2001,2011][2013,+\infty]这些值, 所以我只需要

1000,1990,2000,2012就够了, 将其分别映射到0,1,2,3, 在于复杂度就大大的降下来了

所以离散化要保存所有需要用到的值, 排序后, 分别映射到 $1 \sim n$, 这样复杂度就会小很多很多

而这题的难点在于每个数字其实表示的是一个单位长度(并非一个点), 这样普通的离散化会造成许多错误(包括我以前的代码, poj这题数据奇弱)

给出下面两个简单的例子应该能体现普通离散化的缺陷:

例子一: 1-10 1-4 5-10

例子二: 1-10 1-4 6-10

普通离散化后都变成了[1,4][1,2][3,4]

线段2覆盖了[1,2], 线段3覆盖了[3,4], 那么线段1是否被完全覆盖掉了呢?

例子一是完全被覆盖掉了, 而例子二没有被覆盖

为了解决这种缺陷, 我们可以在排序后的数组上加些处理, 比如说[1,2,6,10]

如果相邻数字间距大于1的话, 在其中加上任意一个数字, 比如加成[1,2,3,6,7,10], 然后再做线段树就好了.

线段树功能: update: 成段替换 query: 简单hash

```

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

const int maxn = 11111;
bool hash[maxn];
int li[maxn] , ri[maxn];
int X[maxn*3];
int col[maxn<<4];
int cnt;

void PushDown(int rt) {
    if (col[rt] != -1) {
        col[rt<<1] = col[rt<<1|1] = col[rt];
        col[rt] = -1;
    }
}

void update(int L,int R,int c,int l,int r,int rt) {

```

```

    if (L <= 1 && r <= R) {
        col[rt] = c;
        return ;
    }
    PushDown(rt);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
}
void query(int l,int r,int rt) {
    if (col[rt] != -1) {
        if (!hash[col[rt]]) cnt ++;
        hash[ col[rt] ] = true;
        return ;
    }
    if (l == r) return ;
    int m = (l + r) >> 1;
    query(lson);
    query(rson);
}
int Bin(int key,int n,int X[]) {
    int l = 0 , r = n - 1;
    while (l <= r) {
        int m = (l + r) >> 1;
        if (X[m] == key) return m;
        if (X[m] < key) l = m + 1;
        else r = m - 1;
    }
    return -1;
}
int main() {
    int T , n;
    scanf("%d",&T);
    while (T --) {
        scanf("%d",&n);
        int nn = 0;
        for (int i = 0 ; i < n ; i ++) {
            scanf("%d%d",&li[i] , &ri[i]);
            X[nn++] = li[i];
            X[nn++] = ri[i];
        }
        sort(X , X + nn);
        int m = 1;
        for (int i = 1 ; i < nn; i ++) {
            if (X[i] != X[i-1]) X[m++] = X[i];
        }
        for (int i = m - 1 ; i > 0 ; i --) {
            if (X[i] != X[i-1] + 1) X[m++] = X[i-1] + 1;
        }
        sort(X , X + m);
        memset(col , -1 , sizeof(col));
        for (int i = 0 ; i < n ; i ++) {
            int l = Bin(li[i] , m , X);
            int r = Bin(ri[i] , m , X);
            update(l , r , i , 0 , m , 1);
        }
        cnt = 0;
        memset(hash , false , sizeof(hash));
        query(0 , m , 1);
        printf("%d\n",cnt);
    }
    return 0;
}

```

poj3225 Help with Intervals

题意: 区间操作, 交, 并, 补等

思路:

我们一个一个操作来分析: (用0和1表示是否包含区间, -1表示该区间内既有包含又有不包含)

U: 把区间 $[l, r]$ 覆盖成1

I: 把 $[-\infty, l)$ $(r, \infty]$ 覆盖成0

D: 把区间 $[l, r]$ 覆盖成0

C: 把 $[-\infty, l)$ $(r, \infty]$ 覆盖成0 , 且 $[l, r]$ 区间0/1互换

S: $[l, r]$ 区间0/1互换

成段覆盖的操作很简单, 比较特殊的就是区间0/1互换这个操作, 我们可以称之为异或操作

很明显我们可以知道这个性质: 当一个区间被覆盖后, 不管之前有没有异或标记都没有意义了

所以当节点得到覆盖标记时把异或标记清空

而当一个节点得到异或标记的时候, 先判断覆盖标记, 如果是0或1, 直接改变一下覆盖标记, 不然的话改变异或标记

开区间闭区间只要数字乘以2就可以处理 (偶数表示端点, 奇数表示两端点间的区间)

线段树功能: update: 成段替换, 区间异或 query: 简单hash

```

#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

const int maxn = 131072;
bool hash[maxn+1];
int cover[maxn<<2];
int XOR[maxn<<2];
void FXOR(int rt) {
    if (cover[rt] != -1) cover[rt] ^= 1;
    else XOR[rt] ^= 1;
}

```

```

}
void PushDown(int rt) {
    if (cover[rt] != -1) {
        cover[rt<<1] = cover[rt<<1|1] = cover[rt];
        XOR[rt<<1] = XOR[rt<<1|1] = 0;
        cover[rt] = -1;
    }
    if (XOR[rt]) {
        FXOR(rt<<1);
        FXOR(rt<<1|1);
        XOR[rt] = 0;
    }
}
void update(char op,int L,int R,int l,int r,int rt) {
    if (L <= l && r <= R) {
        if (op == 'U') {
            cover[rt] = 1;
            XOR[rt] = 0;
        } else if (op == 'D') {
            cover[rt] = 0;
            XOR[rt] = 0;
        } else if (op == 'C' || op == 'S') {
            FXOR(rt);
        }
        return ;
    }
    PushDown(rt);
    int m = (l + r) >> 1;
    if (L <= m) update(op , L , R , lson);
    else if (op == 'I' || op == 'C') {
        XOR[rt<<1] = cover[rt<<1] = 0;
    }
    if (m < R) update(op , L , R , rson);
    else if (op == 'I' || op == 'C') {
        XOR[rt<<1|1] = cover[rt<<1|1] = 0;
    }
}
void query(int l,int r,int rt) {
    if (cover[rt] == 1) {
        for (int it = l ; it <= r ; it ++ ) {
            hash[it] = true;
        }
        return ;
    } else if (cover[rt] == 0) return ;
    if (l == r) return ;
    PushDown(rt);
    int m = (l + r) >> 1;
    query(lson);
    query(rson);
}
int main() {
    cover[1] = XOR[1] = 0;
    char op , l , r;
    int a , b;
    while ( ~scanf("%c %d,%d%c\n",&op , &l , &a , &b , &r) ) {
        a <= 1 , b <= 1;
        if (l == '(') a ++;
        if (r == ')') b --;
        if (a > b) {
            if (op == 'C' || op == 'I') {
                cover[1] = XOR[1] = 0;
            }
        } else update(op , a , b , 0 , maxn , 1);
    }
    query(0 , maxn , 1);
    bool flag = false;
    int s = -1 , e;
    for (int i = 0 ; i <= maxn ; i ++ ) {
        if (hash[i]) {
            if (s == -1) s = i;
            e = i;
        } else {
            if (s != -1) {
                if (flag) printf(" ");
                flag = true;
                printf("%c%d,%d%c",s&1?'(':'[' , s>>1 , (e+1)>>1 , e&1?')':'']);
                s = -1;
            }
        }
    }
    if (!flag) printf("empty set");
    puts("");
    return 0;
}

```

练习

[poj1436 Horizontally Visible Segments](#)

[poj2991 Crane](#)

[Another LCIS](#)

[Bracket Sequence](#)

[区间合并](#)

这类题目会询问区间中满足条件的连续最长区间,所以PushUp的时候需要对左右儿子的区间进行合并

[poj3667 Hotel](#)

题意: 1 a: 询问是不是有连续长度为a的空房间,有的话住进最左边

2 a b: 将[a,a+b-1]的房间清空

思路: 记录区间中最长的空房间

线段树操作: update: 区间替换 query: 询问满足条件的最左断点

```

#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>

```

```

using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

const int maxn = 55555;
int lsum[maxn<<2] , rsum[maxn<<2] , msum[maxn<<2];
int cover[maxn<<2];

void PushDown(int rt,int m) {
    if (cover[rt] != -1) {
        cover[rt<<1] = cover[rt<<1|1] = cover[rt];
        msum[rt<<1] = lsum[rt<<1] = rsum[rt<<1] = cover[rt] ? 0 : m - (m >> 1);
        msum[rt<<1|1] = lsum[rt<<1|1] = rsum[rt<<1|1] = cover[rt] ? 0 : (m >> 1);
        cover[rt] = -1;
    }
}

void PushUp(int rt,int m) {
    lsum[rt] = lsum[rt<<1];
    rsum[rt] = rsum[rt<<1|1];
    if (lsum[rt] == m - (m >> 1)) lsum[rt] += lsum[rt<<1|1];
    if (rsum[rt] == (m >> 1)) rsum[rt] += rsum[rt<<1];
    msum[rt] = max(lsum[rt<<1|1] + rsum[rt<<1] , max(msum[rt<<1] , msum[rt<<1|1]));
}

void build(int l,int r,int rt) {
    msum[rt] = lsum[rt] = rsum[rt] = r - l + 1;
    cover[rt] = -1;
    if (l == r) return ;
    int m = (l + r) >> 1;
    build(lson);
    build(rson);
}

void update(int L,int R,int c,int l,int r,int rt) {
    if (L <= l && R <= r) {
        msum[rt] = lsum[rt] = rsum[rt] = c ? 0 : r - l + 1;
        cover[rt] = c;
        return ;
    }
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L , R , c , lson);
    if (m < R) update(L , R , c , rson);
    PushUp(rt , r - l + 1);
}

int query(int w,int l,int r,int rt) {
    if (l == r) return l;
    PushDown(rt , r - l + 1);
    int m = (l + r) >> 1;
    if (msum[rt<<1] >= w) return query(w , lson);
    else if (rsum[rt<<1] + lsum[rt<<1|1] >= w) return m - rsum[rt<<1] + 1;
    return query(w , rson);
}

int main() {
    int n , m;
    scanf("%d%d",&n,&m);
    build(1 , n , 1);
    while (m --) {
        int op , a , b;
        scanf("%d",&op);
        if (op == 1) {
            scanf("%d",&a);
            if (msum[1] < a) puts("0");
            else {
                int p = query(a , 1 , n , 1);
                printf("%d\n",p);
                update(p , p + a - 1 , 1 , 1 , n , 1);
            }
        } else {
            scanf("%d%d",&a,&b);
            update(a , a + b - 1 , 0 , 1 , n , 1);
        }
    }
    return 0;
}

```

练习

[hdu3308 LCIS](#)

[hdu3397 Sequence operation](#)

[hdu2871 Memory Control](#)

[hdu1540 Tunnel Warfare](#)

[CF46-D Parking Lot](#)

扫描线

这类题目需要将一些操作排序,然后从左到右用一根扫描线(当然是在我们脑子里)扫过去
最典型的的就是矩形面积并,周长并等题

[hdu1542 Atlantis](#)

题意: 矩形面积并

思路: 浮点数先要离散化; 然后把矩形分成两条边, 上边和下边, 对横轴建树, 然后从下到上扫描上去, 用 cnt 表示该区间下边比上边多几个, sum 代表该区间内被覆盖的线段的长度总和
这里线段树的一个结点并非是线段的一个端点, 而是该端点和下一个端点间的线段, 所以题目中 $r+1, r-1$ 的地方可以自己好好的琢磨一下

线段树操作: update: 区间增减 query: 直接取根节点的值

```

#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>

```

```

using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

const int maxn = 2222;
int cnt[maxn << 2];
double sum[maxn << 2];
double X[maxn];
struct Seg {
    double h , l , r;
    int s;
    Seg() {}
    Seg(double a, double b, double c, int d) : l(a) , r(b) , h(c) , s(d) {}
    bool operator < (const Seg &cmp) const {
        return h < cmp.h;
    }
} ss[maxn];
void PushUp(int rt, int l, int r) {
    if (cnt[rt]) sum[rt] = X[r+1] - X[l];
    else if (l == r) sum[rt] = 0;
    else sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void update(int L, int R, int c, int l, int r, int rt) {
    if (L <= l && r <= R) {
        cnt[rt] += c;
        PushUp(rt, l, r);
        return;
    }
    int m = (l + r) >> 1;
    if (L <= m) update(L, R, c, lson);
    if (m < R) update(L, R, c, rson);
    PushUp(rt, l, r);
}
int Bin(double key, int n, double X[]) {
    int l = 0, r = n - 1;
    while (l <= r) {
        int m = (l + r) >> 1;
        if (X[m] == key) return m;
        if (X[m] < key) l = m + 1;
        else r = m - 1;
    }
    return -1;
}
int main() {
    int n, cas = 1;
    while (~scanf("%d", &n) && n) {
        int m = 0;
        while (n--) {
            double a, b, c, d;
            scanf("%lf%lf%lf%lf", &a, &b, &c, &d);
            X[m] = a;
            ss[m++] = Seg(a, c, b, 1);
            X[m] = c;
            ss[m++] = Seg(a, c, d, -1);
        }
        sort(X, X + m);
        sort(ss, ss + m);
        int k = 1;
        for (int i = 1; i < m; i++) {
            if (X[i] != X[i-1]) X[k++] = X[i];
        }
        memset(cnt, 0, sizeof(cnt));
        memset(sum, 0, sizeof(sum));
        double ret = 0;
        for (int i = 0; i < m - 1; i++) {
            int l = Bin(ss[i].l, k, X);
            int r = Bin(ss[i].r, k, X) - 1;
            if (l <= r) update(l, r, ss[i].s, 0, k - 1, 1);
            ret += sum[1] * (ss[i+1].h - ss[i].h);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n", cas++, ret);
    }
    return 0;
}

```

hdu1828 Picture

题意: 矩形周长并

思路: 与面积不同的地方是还要记录竖的边有几个(numseg记录), 并且当边界重合的时候需要合并(用lbd和rbd表示边界来辅助)

线段树操作: update: 区间增减 query: 直接取根节点的值

```

#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
using namespace std;
#define lson l , m , rt << 1
#define rson m + 1 , r , rt << 1 | 1

const int maxn = 22222;
struct Seg {
    int l , r , h , s;
    Seg() {}
    Seg(int a, int b, int c, int d) : l(a) , r(b) , h(c) , s(d) {}
    bool operator < (const Seg &cmp) const {
        if (h == cmp.h) return s > cmp.s;
        return h < cmp.h;
    }
} ss[maxn];
bool lbd[maxn<<2], rbd[maxn<<2];
int numseg[maxn<<2];
int cnt[maxn<<2];
int len[maxn<<2];
void PushUP(int rt, int l, int r) {

```

```

    if (cnt[rt]) {
        lbd[rt] = rbd[rt] = 1;
        len[rt] = r - l + 1;
        numseg[rt] = 2;
    } else if (l == r) {
        len[rt] = numseg[rt] = lbd[rt] = rbd[rt] = 0;
    } else {
        lbd[rt] = lbd[rt<<1];
        rbd[rt] = rbd[rt<<1|1];
        len[rt] = len[rt<<1] + len[rt<<1|1];
        numseg[rt] = numseg[rt<<1] + numseg[rt<<1|1];
        if (lbd[rt<<1|1] && rbd[rt<<1]) numseg[rt] -= 2; //两条线重合
    }
}

void update(int L,int R,int c,int l,int r,int rt) {
    if (L <= l && r <= R) {
        cnt[rt] += c;
        PushUP(rt, l, r);
        return ;
    }
    int m = (l + r) >> 1;
    if (L <= m) update(L, R, c, lson);
    if (m < R) update(L, R, c, rson);
    PushUP(rt, l, r);
}

int main() {
    int n;
    while (~scanf("%d",&n)) {
        int m = 0;
        int lbd = 10000, rbd = -10000;
        for (int i = 0; i < n; i++) {
            int a, b, c, d;
            scanf("%d%d%d%d",&a,&b,&c,&d);
            lbd = min(lbd, a);
            rbd = max(rbd, c);
            ss[m++] = Seg(a, c, b, 1);
            ss[m++] = Seg(a, c, d, -1);
        }
        sort(ss, ss + m);
        int ret = 0, last = 0;
        for (int i = 0; i < m; i++) {
            if (ss[i].l < ss[i].r) update(ss[i].l, ss[i].r - 1, ss[i].s, lbd, rbd - 1, 1);
            ret += numseg[1] * (ss[i+1].h - ss[i].h);
            ret += abs(len[1] - last);
            last = len[1];
        }
        printf("%d\n",ret);
    }
    return 0;
}

```

练习

[hdu3265 Posters](#)

[hdu3642 Get The Treasury](#)

[poj2482 Stars in Your Window](#)

[poj2464 Brownie Points II](#)

[hdu3255 Farming](#)

[ural1707 Hypnotoad's Secret](#)

[uva11983 Weird Advertisement](#)

多颗线段树问题

此类题目主用特点是区间不连续，有一定规律间隔，用多棵树表示不同的偏移区间

[hdu 4288 coder](#)

题意：

维护一个有序数列 $\{A_n\}$ ，有三种操作：

- 1、添加一个元素。
- 2、删除一个元素。
- 3、求数列中下标 $\%5 = 3$ 的值的和。

由于有删除和添加操作，所以离线离散操作，节点中cnt存储区间中有几个数，sum存储偏移和

```

#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int maxn=100002;

#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1

__int64 sum[maxn<<2][6];
int cnt[maxn << 2];

char op[maxn][20];
int a[maxn];

int X[maxn];

void PushUp(int rt)
{
    cnt[rt] = cnt[rt<<1] + cnt[rt<<1|1];
}

```

```

int offset = cnt[rt<<1];
for(int i = 0; i < 5; ++i)
{
    sum[rt][i] = sum[rt<<1][i];
}
for(int i = 0; i < 5; ++i)
{
    sum[rt][(i + offset) % 5] += sum[rt<<1][i];
}
}

void Build(int l, int r, int rt)
{
    /*此题Build完全可以用一个memset代替*/
    cnt[rt] = 0;
    for(int i = 0; i < 5; ++i)    sum[rt][i] = 0;
    if( l == r ) return;
    int m = ( l + r ) >> 1;
    Build(lson);
    Build(rson);
}

void Udata(int p, int op, int l, int r, int rt)
{
    if( l == r )
    {
        cnt[rt] = op;
        sum[rt][1] = op * X[l-1];
        return ;
    }
    int m = ( l + r ) >> 1;
    if(p <= m)
        Udata(p, op, lson);
    else
        Udata(p, op, rson);

    PushUp(rt);
}

int main()
{
    int n;
    while(scanf("%d", &n) != EOF)
    {
        int nn = 0;
        for(int i = 0; i < n; ++i)
        {
            scanf("%s", &op[i]);
            if(op[i][0] != 's')
            {
                scanf("%d", &a[i]);
                if(op[i][0] == 'a')
                {
                    X[nn++] = a[i];
                }
            }
        }

        sort(X,X+nn);/*unique前必须sort*/
        nn = unique(X, X + nn) - X; /*去重并得到总数*/

        Build(1, nn, 1);

        for(int i = 0; i < n; ++i)
        {
            int pos = upper_bound(X, X+nn, a[i]) - X; /* hash */
            if(op[i][0] == 'a')
            {
                Udata(pos, 1, 1, nn, 1);
            }
            else if(op[i][0] == 'd')
            {
                Udata(pos, 0, 1, nn, 1);
            }
            else printf("%I64d\n",sum[1][3]);
        }
    }
    return 0;
}

```

2: hdu 4267 A simple problem with integers

题目：给出n个数，每次将一段区间内满足 $(i-l)\%k==0$ ($r>i>l$) 的数 a_i 增加c，最后单点查询。

这种题目更新的区间是零散的，如果可以通过某种方式让离散的都变得连续，那么问题就可以用线段树完美解决。解决方式一般也是固定的，那就是利用题意维护多颗线段树。此题虚维护55颗，更新最终确定在一颗上，查询则将查询点被包含的树全部叠加。

```

#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
#include<algorithm>
#include<set>
#include<vector>
#include<string>
#include<map>
#define eps 1e-7
#define LL long long
#define N 500005
#define zero(a) fabs(a)<eps
#define lson step<<1

```

```

#define rson step<<1|1
#define MOD 1234567891
#define pb(a) push_back(a)
using namespace std;
struct Node{
    int left,right,add[55],sum;
    int mid(){return (left+right)/2;}
}L[4*N];
int a[N],n,b[11][11];
void Bulid(int step ,int l,int r){
    L[step].left=l;
    L[step].right=r;
    L[step].sum=0;
    memset(L[step].add,0,sizeof(L[step].add));
    if(l==r) return ;
    Bulid(lson,l,L[step].mid());
    Bulid(rson,L[step].mid()+1,r);
}
void push_down(int step){
    if(L[step].sum){
        L[lson].sum+=L[step].sum;
        L[rson].sum+=L[step].sum;
        L[step].sum=0;
        for(int i=0;i<55;i++){
            L[lson].add[i]+=L[step].add[i];
            L[rson].add[i]+=L[step].add[i];
            L[step].add[i]=0;
        }
    }
}
void update(int step,int l,int r,int num,int i,int j){
    if(L[step].left==l&&L[step].right==r){
        L[step].sum+=num;
        L[step].add[b[i][j]]+=num;
        return;
    }
    push_down(step);
    if(r<=L[step].mid()) update(lson,l,r,num,i,j);
    else if(l>L[step].mid()) update(rson,l,r,num,i,j);
    else {
        update(lson,l,L[step].mid(),num,i,j);
        update(rson,L[step].mid()+1,r,num,i,j);
    }
}
int query(int step,int pos){
    if(L[step].left==L[step].right){
        int tmp=0;
        for(int i=1;i<=10;i++) tmp+=L[step].add[b[i][pos%i]];
        return a[L[step].left]+tmp;
    }
    push_down(step);
    if(pos<=L[step].mid()) return query(lson,pos);
    else return query(rson,pos);
}
int main(){
    int cnt=0;
    for(int i=1;i<=10;i++) for(int j=0;j<i;j++) b[i][j]=cnt++;
    while(scanf("%d",&n)!=EOF){
        for(int i=1;i<=n;i++) scanf("%d",&a[i]);
        Bulid(1,1,n);
        int q,d;
        scanf("%d",&q);
        while(q--){
            int k,l,r,m;
            scanf("%d",&k);
            if(k==2){
                scanf("%d",&m);
                printf("%d\n",query(1,m));
            }
            else{
                scanf("%d%d%d%d",&l,&r,&d,&m);
                update(1,l,r,m,d,1%d);
            }
        }
    }
    return 0;
}

```

线段树与其他结合练习(欢迎大家补充):

- [hdu3954 Level up](#)
- [hdu4027 Can you answer these queries?](#)
- [hdu3333 Turing Tree](#)
- [hdu3874 Necklace](#)
- [hdu3016 Man Down](#)
- [hdu3340 Rain in ACStar](#)
- [zju3511 Cake Robbery](#)
- [UESTC1558 Charitable Exchange](#)
- [CF85-D Sum of Medians](#)
- [spojGSS2 Can you answer these queries II](#)

上一篇 [hdu 2502月之数](#)
下一篇 [POJ 2528 Mayor's posters 线段树+离散化](#)

顶 16
踩 0

主题推荐

[数据结构](#) [二叉树](#) [递归](#) [存储](#) [合并](#)

猜你在找

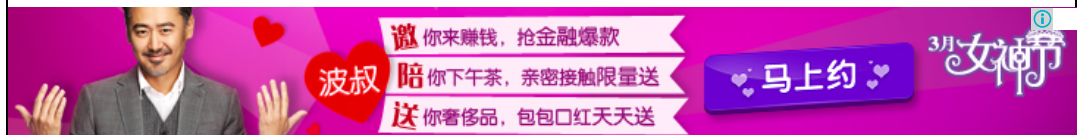
[叉姐的魔法训练第一课—— 初级魔法练习](#)
[大数斐波那契数列+取余](#)
[CC++2014年7月华为校招机试真题一](#)
[中文分词之Trie树](#)
[高级程序员的必学](#)

[参加ACM比赛所需的基础知识](#)
[PAT 1045 Favorite Color Stripe 最长公共子序列](#)
[Codeforces Round #292 Div 2 -- A Drazil and](#)
[#翻译# 介绍后缀树suffix tree](#)
[POJ1007-DNA Sorting](#)

准备好了么？跳吧！

更多职位尽在 **CSDN JOB**

招募IT猎头顾问——技术人员转行的新仕	我要跳槽	数据架构师	我要跳槽
上海科锐福克斯人才顾问有限公司	0.5-1K/月	北京华医网科技股份有限公司	10-15K/月
大数据高级研发工程师	我要跳槽	数据挖掘工程师	我要跳槽
上海奔耀信息科技有限公司	4-16K/月	上海晶赞科技发展有限公司	10-30K/月



查看评论

12楼 [楼上小宇](#) 2015-01-27 23:43发表



受教了。。。很详细

11楼 [MetalSeed](#) 2014-11-12 16:09发表



228003430

10楼 [累了泪了funny](#) 2014-07-25 20:53发表



为什么都不喜欢定义结构体呢，表示用数组模拟，看不懂，不要嘲笑我，我是菜鸟

Re: [MetalSeed](#) 2014-11-07 09:46发表



回复累了泪了funny: 那个时候强迫症，数组写起来方便一些，速度稍微快一点

9楼 [MetalSeed](#) 2013-01-01 23:50发表



http://blog.csdn.net/shiqi_614/article/details/8228102

8楼 [GodHunter47](#) 2012-12-31 15:16发表



弱弱的屌丝一枚，不过勉为其难的给你顶一下

Re: [MetalSeed](#) 2013-01-01 23:47发表



回复GodHunter47: 。。。~~呼呼 谢谢大神

7楼 [Zr777111](#) 2012-10-12 15:11发表



高深。。。

6楼 [wd339092886](#) 2012-10-12 14:09发表



不行不行，我还是倾向于编程。。

5楼 [dyx心心](#) 2012-10-12 11:32发表



楼主是搞ACM的？

Re: [MetalSeed](#) 2012-10-12 23:38发表



回复 [dyx心心](#): ACM弱菜一枚.. 正在努力中..

Re: [dyx心心](#) 2012-11-19 09:37发表



回复 [MetalSeed](#): 我感觉最后那道题，[hdu4267](#)那个sum就是个标记而已，不需要 `L[step].sum+=num;` 只要 `L[step].sum=1`，就行了吧

4楼 [c_006](#) 2012-10-11 11:31发表



mark一下。楼主的编译软件是什么的？这么花花绿绿的vs2010？

Re: [hellosijian](#) 2012-10-11 19:46发表



回复 [c_006](#): 应该是vim

Re: [hello_world_ww](#) 2013-01-26 09:34发表



回复 [hellosijian](#): GCC/G++

3楼 [jiangkaii](#) 2012-10-11 07:32发表



进首页了！！

Re: [MetalSeed](#) 2012-10-11 10:51发表



回复 [jiangkaii](#): 是蒋凯么。。。

Re: [jiangkaii](#) 2012-10-12 01:51发表



回复 [MetalSeed](#): 是的，

2楼 [pl__](#) 2012-10-09 16:16发表



手动实现std::map？

Re: [MetalSeed](#) 2012-10-09 17:36发表



回复 [pl__](#): 非也。

map是用红黑树实现的，特点是插入时按照键值自动排序

1楼 [Dstnoe](#) 2012-10-08 23:22发表



请问，线段树和B树在应用上有什么特别之处？

Re: [MetalSeed](#) 2012-10-09 17:38发表



回复 [Dstnoe](#): 不好意思..B树只有耳闻...没详细研究过... 还得努力呢

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide
Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase
Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

