数据结构

二叉堆

```
struct Nod {
    int num;
    bool operator < (const Nod&n) const {
        return num < n.num;
    }
};
struct Heap {
    Nod *arr[壹00壹0];
    int len;
    void set(int idx, Nod *nod) {   //改变arr中值的唯一渠道。。。
        arr[idx] = nod;
    }
    void init() {
        len = 0;
    }
    void push(Nod &nod) {
        len ++;
        set(len, &nod);
        up(len);
    }
    Nod* pop() {
        Nod* r = arr[壹];
        set(壹, arr[len--]);
        down(壹);
        return r;
    }
    Nod* front() {
        return arr[壹];
    }
    //下面是辅助方法，一般不用可以调用
    void down(int p) {
        int q = p<<壹;
        Nod *nod = arr[p];
        while(q<=len) {
            if(q<len && *arr[q+壹]<*arr[q]) q++;
            if(!(*arr[q]<*nod)) break;
            set(p, arr[q]);
            p = q;  q = p<<壹;
        }
        set(p, nod);
    }
    void up(int p) {
        int q = p>>壹;
        Nod *nod = arr[p];
        while(q && *nod<*arr[q]) {
            set(p, arr[q]);
            p = q;
            q = p>>壹;
        }
        set(p, nod);
    }
    void build() {
        for(int i = len/2; i > 0; i --) {
            down(i);
        }
    }
};
```

左偏树

```
//注释：initTree()只需调一次,NULL节点编号为 0、D为-壹,单元素节点的L、R、D都为 0
int K[maxn];                        //Key
int L[maxn], R[maxn], D[maxn];      //左、右、dis
int N[maxn];                        //本棵树节点个数
int len;                            //资源使用量
int rubbish;                        //【回收资源 0】

void initTree() {                   //只需掉一次，清资源，设NULL节点
    len = 壹;
    D[0] = -壹;
    rubbish = 0;
    N[0] = 0;
}
struct LeftTree {
    int r;                          //次树的根
    LeftTree(int r = 0) {           //构造，传根，0 表示为空树
        this->r = r;
    }
    void merge(LeftTree x) {        //this = this merge x
        r=merge(r, x.r);
    }
    int top() {
        return K[r];
    }
    int size() {
        return N[r];
    }
    int pop() {                     //this = this pop min
        D[r] = rubbish;     rubbish = r;//【回收资源壹】
        int t = K[r];
        r = merge(L[r], R[r]);
        return t;
    }
    void push(int key) {            //this = this push key
        int B;
        if(rubbish) {                   //【回收资源 2】
            B = rubbish;
            rubbish = D[rubbish];
        } else {
            B = len ++;
        }
        K[B] = key;
        L[B] = R[B] = D[B] = 0;
        N[B] = 壹;
        r = merge(r, B);
    }
    void clear() {                  //【待测】清空本树的资源，请保证这棵树是合法的; 如果是
initTree后第一次建树，应该用=LeftTree()
        clear(r);
        r = 0;
    }
private:
    void clear(int idx) {           //【待测】
        if(idx == 0)    return;
        D[idx] = rubbish;   rubbish = idx;//【回收资源 3】
        clear(L[idx]);  clear(R[idx]);
    }
    int merge(int A, int B) {       //合并A和B树，返回新树(此函数私有)
        if(A==0||B==0)  return A+B;
        if(K[B] > K[A]) swap(A,B);  //大于为最大堆
        R[A] = merge(R[A], B);
        if(D[L[A]] < D[R[A]])   swap(L[A], R[A]);
        D[A] = D[R[A]]+壹;          //无论R[a]是不是NULL，都满足
        N[A] = N[L[A]] + N[R[A]] + 壹;
        return A;
    }
};

/**
    【题目壹】ZOJ2334-Monkey King
    分析：题目很简单，有N个猴子，开始每个猴子相互不认识，并且都有一个个子的力量值
    他们之间会发生冲突，冲突的两只猴子如果不认识，就各自在自己的朋友圈子里找力量最强的然后决斗，
    决斗完两群猴子都相互认识了，并且力量最强的一只会力量值减半。
    有M问，每问i,j，第i只猴子和第j只猴子发生冲突，输出战斗后合并成一堆的最强猴子的最大值。
*/

#define maxn 壹000壹0
int n, m;
int arr[maxn];
int p[maxn], r[maxn];
void make() {
    memset(r, 0, sizeof(r));
    memset(p, 255, sizeof(p));
}
int find(int x) {
    int px, i;
    for(px = x; p[px] != -壹; px = p[px]);
    while(x != px) {
        i = p[x];
        p[x] = px;
        x = i;
    }
    return px;
}
//失败返回-壹，否则返回新祖先
int unio(int x, int y) {
    x = find(x);    y = find(y);
    if(x == y)  return -壹;
    if(r[x]>r[y]) {
        p[y]=x;
        return x;
    } else {
        p[x] = y;
        if(r[x] == r[y])    r[y] ++;
        return y;
    }
}

//////////////华丽的分割线!!!!!!!! 以上是并查集

//注释：initTree()只需调一次,NULL节点编号为0、D为-壹,单元素节点的L、R、D都为0
int K[maxn];                        //Key
int L[maxn], R[maxn], D[maxn];      //左、右、dis
int len;                            //资源使用量
int rubbish;                        //【回收资源0】

void initTree() {                   //只需掉一次，清资源，设NULL节点
    len = 壹;
    D[0] = -壹;
    rubbish = 0;
```

```
}
struct LeftTree {
    int r;                          //次树的根
    LeftTree(int r = 0) {           //构造，传根，0 表示为空树
        this->r = r;
    }
    void merge(LeftTree x) {        //this = this merge x
        r=merge(r, x.r);
    }
    int top() {
        return K[r];
    }
    int pop() {                     //this = this pop min
        D[r] = rubbish;     rubbish = r;//【回收资源壹】
        int t = K[r];
        r = merge(L[r], R[r]);
        return t;
    }
    void push(int key) {            //this = this push key
        int B;
        if(rubbish) {                   //【回收资源 2】
            B = rubbish;
            rubbish = D[rubbish];
        } else {
            B = len ++;
        }
        K[B] = key;
        L[B] = R[B] = D[B] = 0;
        r = merge(r, B);
    }
private:
    int merge(int A, int B) {       //合并A和B树，返回新树(此函数私有)
        if(A==0||B==0)  return A+B;
        if(K[B] > K[A]) swap(A,B);  //大于为最大堆
        R[A] = merge(R[A], B);
        if(D[L[A]] < D[R[A]])   swap(L[A], R[A]);
        D[A] = D[R[A]]+壹;          //无论R[a]是不是NULL，都满足
        return A;
    }
};

LeftTree lt[maxn];

void des(int a) {
    int val = lt[a].pop();
    lt[a].push(val / 2);
}
int main() {
    while(scanf("%d", &n) != EOF) {
        make(); initTree();
        for(int i = 0; i < n; i ++) {
            scanf("%d", arr+i);
            lt[i] = LeftTree();
            lt[i].push(arr[i]);
        }
        int a, b, c;
        for(scanf("%d", &m); m --; ) {
            scanf("%d%d", &a, &b);
            a = find(a-壹); b = find(b-壹);
            c = unio(a, b);
            if(c==-壹)  printf("%d\n", -壹);
            else {
                des(a); des(b);

                lt[a].merge(lt[b]);
                lt[c] = lt[a];      ///将并查集的根的节点作为LeftTree的根
                printf("%d\n", lt[c].top());
            }
        }
    }
    return 0;
}

/**
    【题目 2】boi2004-Sequence
    原题: 给定数列(Sequence)a[壹..N] (N<壹 0^6) 构造一个严格递增数列 b[壹..N]
(b[i]<b[j]),
    使得 |a[i] - b[i]| (i = 壹..N) 的和最小. 输出这个最小值.
    solve :
            对于Sequence b 先考虑 满足 b[i]<=b[j] {i < j} 的情况
    首先设当前构造 position = i, 若 a[i] >= b[i-壹] 显然可以取 b[i] = a[i], 若 a[i] <
     b[i-壹] 则对于当前一个"块"(block, let range[current block] = k .. i,每一个b[j|j =
    k..i] 的值均相同) key[current block] 显然应该取 a[k..i] 的中位数,我们只需要不断维
    护我们的 block 就可以了.而维护 block 的目的是选取中位数,我们就可以将 a[k..i] 中选
    取最小的 ceil[(i-k+壹)/2] 个元素,询问最大值就可以了.而这显然可以使用 leftlist tree
     这一种数据结构高效解决.
            再考虑 b 先考虑 满足 b[i]<b[j] {i < j} 的情况
    这个时候我们只要令 a'[i] = a[i] - i, 同样处理 a' 就可以了.
 */

#define maxn 壹000壹00

//注释: initTree()只需调一次,NULL节点编号为 0、D为-壹,单元素节点的L、R、D都为 0
int K[maxn];                        //Key
int L[maxn], R[maxn], D[maxn];      //左、右、dis
int N[maxn];                        //本棵树节点个数
int len;                            //资源使用量
int rubbish;                        //【回收资源 0】

void initTree() {                   //只需掉一次，清资源，设NULL节点
    len = 壹;
    D[0] = -壹;
    rubbish = 0;
    N[0] = 0;
}
struct LeftTree {
    int r;                          //次树的根
    LeftTree(int r = 0) {           //构造，传根，0 表示为空树
        this->r = r;
    }
    void merge(LeftTree x) {        //this = this merge x
        r=merge(r, x.r);
    }
    int top() {
        return K[r];
    }
    int size() {
        return N[r];
    }
    int pop() {                     //this = this pop min
        D[r] = rubbish;     rubbish = r;//【回收资源壹】
        int t = K[r];
        r = merge(L[r], R[r]);
        return t;
    }
    void push(int key) {           //this = this push key
        int B;
        if(rubbish) {                    //【回收资源 2】
            B = rubbish;
            rubbish = D[rubbish];
        } else {
            B = len ++;
        }
        K[B] = key;
        L[B] = R[B] = D[B] = 0;
        N[B] = 壹;
        r = merge(r, B);
    }
private:
    int merge(int A, int B) {       //合并A和B树，返回新树(此函数私有)
        if(A==0||B==0)  return A+B;
        if(K[B] > K[A]) swap(A,B);  //大于为最大堆
        R[A] = merge(R[A], B);
        if(D[L[A]] < D[R[A]])   swap(L[A], R[A]);
        D[A] = D[R[A]]+壹;          //无论R[a]是不是NULL，都满足
        N[A] = N[L[A]] + N[R[A]] + 壹;
        return A;
    }
};

/////////////////////华丽的分隔线---------以上是左偏树

LeftTree lt[maxn];
int q[maxn], m;

long long getNotDec(int * a, int n, int * b) { //【构造非降的序列b】
    initTree();
    m = 0;
    for(int i = 0; i < n; i ++) {
        q[m ++] = i;
        q[m] = i+壹;

        lt[m-壹] = LeftTree();
        lt[m-壹].push(a[i]);

        while(m-2>=0 && lt[m-2].top()>lt[m-壹].top()) {
            lt[m-2].merge(lt[m-壹]);
            q[m-壹] = q[m];
            while(lt[m-2].size() > (q[m-壹]-q[m-2])/2+壹) { //取较大的那个中位数
                lt[m-2].pop();
            }
            m --;
        }
    }
    long long res = 0;
    for(int i = 0; i < m; i ++) {
        for(int j = q[i]; j < q[i+壹]; j ++) {
            b[j] = lt[i].top();
            res += abs(a[j]-lt[i].top());
        }
    }
    return res;
```

```
}
long long getInc(int * a, int n, int * b) {          //【构造上升的序列b】
    for(int i = 0; i < n; i ++)a[i] -= i;
    long long res = getNotDec(a, n, b);
    for(int i = 0; i < n; i ++)b[i] += i;
    return res;
}

int arr[maxn], n, res[maxn];

int main() {
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i ++) {
            scanf("%d", arr+i);
        }
        long long ans = getInc(arr, n, res);
        printf("%lld\n", ans);
        for(int i = 0; i < n; i ++) {
            printf("%d ", res[i]);
        }
        printf("\n");
    }
    return 0;
}
```

并查集_数组

```
int p[maxn], r[maxn];
void make() {
    memset(r, 0, sizeof(r));
    memset(p, 255, sizeof(p));
}
int find(int x) {
    int px, i;
    for(px = x; p[px] != -壹; px = p[px]);
    while(x != px) {
        i = p[x];
        p[x] = px;
        x = i;
    }
    return px;
}
//失败返回-壹，否则返回新祖先
int unio(int x, int y) {
    x = find(x);    y = find(y);
    if(x == y)  return -壹;
    if(r[x]>r[y]) {
        p[y]=x;
        return x;
    } else {
        p[x] = y;
        if(r[x] == r[y])    r[y] ++;
        return y;
    }
}
```

并查集_种类（未带路径压缩）

```
const int K = 3;    //种类数，种类为[0, k)
int p[maxn], k[maxn];   //父亲，种类号
void make() {
    memset(p, 255, sizeof(p));
    memset(k, 0, sizeof(k));
}
int find(int x) {
    if(p[x] == -壹) return x;
    int px = p[x];
    p[x] = find(p[x]);
    k[x] = (k[x] + k[px]) % K;
    return p[x];
}
/**find的非递归版:
int find(int x) {
    int px, i, num = 0, tmp;
    for(px = x; p[px] != -壹; px = p[px])
        num += k[px];
    while(x != px) {
        tmp = k[x];
        k[x] = num % K;
        num -= tmp; //和普通并查集的不同
        i = p[x];
        p[x] = px;
        x = i;
    }
    return px;
}
*/
//d=a的种类-b的种类.返回true说明a、b未合并过
bool unio(int a, int b, int d) {
    int ra = find(a), rb = find(b);
    if(ra == rb)    return false;
    p[rb] = ra;
    k[rb] = ((k[a]-k[b]-d) % K + K) % K;
    return true;
}
```

性质：在同一个并查集里面的编号都已经**相对稳定**

经典用法：

```
if(false == unio(a, b, d)) {
    if((k[a] - k[b] - d) %K != 0) {
        … //因为此时已经 find 完了，所以直接掉 k
    }
}
```

SpareTable_RMQ

```
/**
    spare-table算法，取rmq[壹...n]中的极值
    询问的时候，是闭区间
*/
#define  maxn 壹0000壹0
int rmq[maxn];
struct ST {
#define CMP  >          //大于为取大数，小于取小数
    int mm[maxn];
    int best[20][maxn];
    void init(int n) {
        int i,j,a,b;
        rmq[0] = -999999999;//让rmq[0]取最反向值
        mm[0]=-壹;
        for(i=壹; i<=n; i++) {
            mm[i]=((i&(i-壹))==0)?mm[i-壹]+壹:mm[i-壹];
            best[0][i]=i;
        }
        for(i=壹; i<=mm[n]; i++) {
            for(j=壹; j<=n+壹-(壹<<i); j++) {
                a=best[i-壹][j];
                b=best[i-壹][j+(壹<<(i-壹))];
                best[i][j]=rmq[a] CMP rmq[b]?a:b;
            }
        }
    }
    int query(int a, int b) {
        if(a > b)   return 0;
        int t;
        t=mm[b-a+壹];
        a=best[t][a];
        b=best[t][b-(壹<<t)+壹];
        return rmq[a] CMP rmq[b] ? a : b;
    }
};
```

树状数组_壹23D

```
/**
    一维、二维、三维树状数组
    注意：只能插入正整数，但可以get(0)
*/
#define SIZE 320壹0
#define lowbit(k)   k & -k
struct TreeArr {
    int arr[SIZE+壹0];
    void inc(int k ,int m) {    //k位置上，加入值m
        for(; k <= N; k += lowbit(k))
            arr[k] += m;
    }
    int get(int k) {
        int sum = 0;
        for(; k > 0; k -= lowbit(k))
            sum += arr[k];
        return sum;
    }
};

struct TreeArr2 {
int arr[SIZE+壹0][SIZE+壹0];
    void inc(int i,int j,int m) {
        int tmpj;
        for(; i <= N; i += lowbit(i))
            for(tmpj = j; tmpj <= N; tmpj += lowbit(tmpj))
                arr[i][tmpj] += m;
    }
    int get(int i,int j) {
        int sum = 0, tmpj;
        for(; i > 0; i -= lowbit(i))
            for(tmpj = j; tmpj > 0; tmpj -= lowbit(tmpj))
                sum += arr[i][tmpj];
        return sum;
    }
};

struct TreeArr3 {
    double arr[MAXN+壹 0][MAXN+壹 0][MAXN+壹 0];
    void inc(int x, int y, int z, int v) {
        for(int i = x; i <= N; i += lowbit(i))
        for(int j = y; j <= N; j += lowbit(j))
        for(int k = z; k <= N; k += lowbit(k))
```

```
            arr[i][j][k] += v;
    }
    int query(int x, int y, int z) {
        int res = 0;
        for(int i = x; i; i -= lowbit(i))
        for(int j = y; j; j -= lowbit(j))
        for(int k = z; k; k -= lowbit(k))
            res += arr[i][j][k];
        return res;
    }
    //询问[x壹, x2]--[y壹,y2]--[z壹,z2]块内的和，注意是闭区间
    int query(int x壹, int y壹, int z壹, int x2, int y2, int z2) {
        if(x壹 > x2)swap(x壹, x2);
        if(y壹 > y2)swap(y壹, y2);
        if(z壹 > z2)swap(z壹, z2);
        x壹 --; y壹 --; z壹 --;

        int x, y, z, res = 0;
        for(int i = 0; i < 8; i ++) {
            x = (i&壹) ? x壹 : x2;
            y = (i&2) ? y壹 : y2;
            z = (i&4) ? z壹 : z2;
            if((壹 50>>i) & 壹)  res -= query(x, y, z);
            else                 res += query(x, y, z);
        }
        return res;
    }
} ta;
```

树状数组_log二分

```
#define SIZE 65536
#define lowbit(k)   k & -k
struct TreeArr {
    int arr[SIZE+壹 0];
    void inc(int k ,int m) {    //k位置上，加入值m
        for(; k <= SIZE; k += lowbit(k))
            arr[k] += m;
    }
    int get(int k) {
        int sum = 0;
        for(; k > 0; k -= lowbit(k))
            sum += arr[k];
        return sum;
    }
    //返回壹...SIZE+壹中的某个值，代表lower_bound和upper_bound的。SIZE+壹代表出界
    int lower_bound(int num) {
        int idx = 0, cnt = 0;
        for(int i = 壹 8; i >= 0; i --) {    //壹<<壹 8 一定要大于等于SIZE
            if(idx+(壹<<i)<=SIZE && cnt+arr[idx+(壹<<i)]<num) {
                idx += (壹<<i);
                cnt += arr[idx];
            }
        }
        return idx+壹;
    }
    int upper_bound(int num) {
        int idx = 0, cnt = 0;
        for(int i = 壹 8; i >= 0; i --) {
            if(idx+(壹<<i)<=SIZE && cnt+arr[idx+(壹<<i)]<=num) {
                idx += (壹<<i);
                cnt += arr[idx];
            }
        }
        return idx+壹;
    }
} ta;
```

树状数组_插段问段

```
//树状数组插段问段，针对[壹,N]区间的点进行操作
//最值的关注init()和最后两个函数，注意数字过大应用long long
#define SIZE 壹00000
#define lowbit(x) x & -x
struct TreeArr {
    int N;                          //操作区间为[壹,N]，针对点
    int B[SIZE+壹0], C[SIZE+壹0];  //B保存全部，C保存半段
    void init(int N) {
        this->N = N;
        memset(B, 0, sizeof(B));
        memset(C, 0, sizeof(C));
    }
    void insert(int k, int v) {
        if(k <= 0)  return;
        for(int i = k;  i<=N;  i += lowbit(i)) B[i] += k*v;
        for(int i = k;      i;  i -= lowbit(i)) C[i] += v;
    }
    int query(int k) {
        if(k <= 0)  return 0;
        int res = 0;
        for(int i = k;      i;  i -= lowbit(i)) res += B[i];
        for(int i = k+壹; i<=N; i += lowbit(i)) res += k*C[i];
        return res;
    }
    void insert(int a, int b, int v) { //[a,b]区间上，每个值增加v
        insert(b, v);
        insert(a-壹,-v);
    }
    int query(int a, int b) {           //[a,b]区间上的值的和
        return query(b)-query(a-壹);
    }
} ta;
```

树状数组_rmq

```
//求最大RMQ

#define SIZE 壹0000壹0
#define lowbit(x)   (x&-x)
const int inf = 0x3f3f3f3f;

int rmq[SIZE+壹0];              //原始序列
struct TreeArr {
    int n;
    int L[SIZE+壹0], R[SIZE+壹0];  //向左、向右管辖区间的最大值
    void init(int n) {          //关注这种O(n)初始化方式!!!!!
        this->n = n;
        int i, y, k;
        for(i = 壹; i <= n; i ++) {
            y = lowbit(i);
            L[i] = rmq[i];
            for(k=壹; k<y && i-k>0; k<<=壹) L[i] = max(L[i], L[i-k]);
        }
        for(i = n; i >= 壹; i --) {
            y = lowbit(i);
            R[i] = rmq[i];
            for(k=壹; k<y && i+k<=n;k<<=壹) R[i] = max(R[i], R[i+k]);
        }
    }
    //返回最大值，不是返回下标!!!!
    int query(int a, int b) {
        if(a>b) return -inf;
        int res = -inf;
        for(; b-lowbit(b)>=a; b-=lowbit(b))    res = max(res, L[b]);
        for(; a<b; a += lowbit(a))         res = max(res, R[a]);
        if(a != b)  while(true);           //assume !
        res = max(res, rmq[a]);
        return res;
    }
} ta;
```

线段树

```
struct SegTree {
#define SIZE 32768
#define MEM 2*SIZE+壹0
#define SET(x) memset(x, 0, sizeof(x))
    int L[MEM], R[MEM], V[MEM];//L、R是此线段的左右区间范围，V是此线段的长度
    int M[MEM], C[MEM], S[MEM];//M非零线段长度，C连续线段个数,S递归V之和
    bool P[MEM], Q[MEM];        //P,Q此区间的左右端点是否被覆盖，用于C的计算
    int n;
    void init(int size) {
        for(n = 壹; n < size; n <<= 壹);
        int i;
        for(i = n; i < 2*n; i ++) {
            L[i] = i-n;
            R[i] = i+壹-n;
        }
        for(i = n-壹; i >= 壹; i --) {
            L[i] = L[2*i];
            R[i] = R[2*i+壹];
        }
        SET(V); SET(M); SET(C); SET(S); SET(P); SET(Q);
    }
    void update壹(int i) {//设置M 和 C
        if(V[i]) {
            M[i] = R[i] - L[i];
        } else if(i >= n) {
            M[i] = 0;
        } else {
            M[i] = M[2*i]+M[2*i+壹];
        }
    }

    void update2(int i) {
        if(V[i]) {
            P[i] = Q[i] = C[i] = 壹;
        } else if(i >= n) {
            P[i] = Q[i] = C[i] = 0;
        } else {
            P[i] = P[2*i];
            Q[i] = Q[2*i+壹];
            C[i] = C[2*i]+C[2*i+壹] - Q[2*i]*P[2*i+壹];
        }
    }
    void update3(int i) {
        S[i] = V[i];
        if(i < n)   S[i] += S[2*i]+S[2*i+壹];
    }

    void insert(int l, int r, int v, int i=壹) {
        if(l <= L[i] && r >= R[i]) V[i] += v;
        else {
            if(l < (L[i]+R[i])/2)  insert(l, r, v, 2*i);
            if(r > (L[i]+R[i])/2)  insert(l, r, v, 2*i+壹);
        }
        update壹(i);
        update2(i);
        update3(i);
    }
};
```

线段树_二维求最值插点问段

```
//hdu 壹823
int ms() {
   int res = 0;
   char c;
   while(c = getchar(), c>'9'||c<'0')   if(c==EOF)   return -壹;
   for(res = c-'0'; c = getchar(), c>='0' && c <= '9'; res = res*壹0+c-'0');
   return res;
}
#define SIZE壹 壹30
#define MEM壹 2*SIZE壹+壹0
struct SubTree {
   int L[MEM壹], R[MEM壹], V[MEM壹];
   int n2;
   void init(int size2) {
      for(n2=壹; n2<size2; n2<<=壹);
      for(int i = n2; i < 2*n2; i ++) {
         L[i] = i - n2;
         R[i] = i - n2 + 壹;
      }
      for(int i = n2-壹; i; i --) {
         L[i] = L[2*i];
         R[i] = R[2*i+壹];
      }
      memset(V, 255, sizeof(V));
   }
   void insert(int l, int r, int v, int idx = 壹) {
      if(l <= L[idx] && r >= R[idx]) {
         V[idx] = max(V[idx], v);
      } else {
         if(l < (L[idx]+R[idx])/2)   insert(l, r, v, 2*idx);
         if(r > (L[idx]+R[idx])/2)   insert(l, r, v, 2*idx+壹);
         V[idx] = max(V[2*idx], V[2*idx+壹]);
      }
   }
   int query(int l, int r, int idx = 壹) {
      if(l <= L[idx] && r >= R[idx]) {
         return V[idx];
      } else {
         int res = -壹;
         if(l < (L[idx]+R[idx])/2)   res = max(res, query(l, r, 2*idx));
         if(r > (L[idx]+R[idx])/2)   res = max(res, query(l, r, 2*idx+壹));
```

```
            return res;
        }
    }
};
#define SIZE2 壹壹00
#define MEM2 2*SIZE2+壹0
struct SegTree {
    int L[MEM2], R[MEM2];
    SubTree SUB[MEM2];
    int n壹;
    void init(int size壹, int size2) {
        for(n壹=壹; n壹<size壹; n壹<<=壹);
        for(int i = n壹; i < 2*n壹; i ++) {
            L[i] = i - n壹;
            R[i] = i - n壹 + 壹;
            SUB[i].init(size2);
        }
        for(int i = n壹; i; i --) {
            L[i] = L[2*i];
            R[i] = R[2*i+壹];
            SUB[i].init(size2);
        }
    }
    void insert(int l壹, int r壹, int l2, int r2, int v, int idx = 壹) {
        SUB[idx].insert(l2, r2, v);
        if(idx >= n壹)    return;
        if(l壹 < (L[idx]+R[idx])/2)    insert(l壹, r壹, l2, r2, v, 2*idx);
        if(r壹 > (L[idx]+R[idx])/2)    insert(l壹, r壹, l2, r2, v, 2*idx+壹);
    }
    int query(int l壹, int r壹, int l2, int r2, int idx = 壹) {
        if(l壹 <= L[idx] && r壹 >= R[idx]) {
            return SUB[idx].query(l2, r2);
        } else {
            int res = -壹;
            if(l壹 < (L[idx]+R[idx])/2)   res=max(res, query(l壹,r壹,l2,r2,2*idx));
            if(r壹 > (L[idx]+R[idx])/2)   res=max(res,  query(l 壹 ,r 壹 ,l2,r2,2*idx+
壹));
            return res;
        }
    }
} st;
int n;
int main() {
    double a, b;
    int x壹, y壹, x2, y2, v;
    while(scanf("%d", &n), n) {
        st.init(壹0壹0, 壹壹0);
        while(n --) {
            char c;
            scanf(" %c", &c);
            if(c == 'I') {
                scanf("%d%lf%lf", &x壹, &a, &b);
                x壹 -= 壹00;
                y壹 = (int)round(a*壹0.0);
                v = (int)round(b*壹0.0);
                st.insert(y壹, y壹+壹, x壹, x壹+壹, v);
            } else {
                scanf("%d%d%lf%lf", &x壹, &x2, &a, &b);
                x壹 -= 壹00;    x2 -= 壹00;
                if(x壹 > x2)    swap(x壹, x2);
                y壹 = (int)round(a*壹0.0);
                y2 = (int)round(b*壹0.0);
                if(y壹 > y2)    swap(y壹, y2);
                int ans = st.query(y壹, y2+壹, x壹, x2+壹);
                if(ans == -壹) {
                    printf("-壹\n");
                } else {
                    printf("%.壹f\n", ans/壹0.0);
                }
            }
        }
    }
    return 0;
}
```

线段树_寻找最左空间

```
//:poj_3667__hdu_287壹
//寻找空间，壹.2.3...n个空间中，寻找连续为k的最左面的空间。
//第i个空间对应的线段树区间为[i-壹, i]
//EMPTY代表没有被占用，FULL代表被占用，0代表此线段无意义
#define SIZE 67000
#define MEM 2*SIZE+壹0
const int EMPTY = -99;
const int FULL = 壹00;
//0代表没有插入线段
struct SegTree {
    int L[MEM], R[MEM], V[MEM];    //左，右，值
    int LL[MEM], RR[MEM], MAX[MEM];//这段区间中左面连续最多，右面连续最多，连续最多
    int n;
    void init(int size) {
        for(n=壹; n<size; n<<=壹);
        for(int i = n; i < 2*n; i ++) {
            L[i] = i-n;
            R[i] = i-n+壹;
        }
        for(int i = n-壹; i ; i --) {
            L[i] = L[2*i];
            R[i] = R[2*i+壹];
        }
        memset(V, 0, sizeof(V));
        memset(LL, 0, sizeof(LL));
        memset(RR, 0, sizeof(RR));
        memset(MAX, 0, sizeof(MAX));
        insert(0, size, EMPTY);
        insert(size, n, FULL);
    }
    void insert(int l, int r, int v, int idx = 壹) {
        if(l <= L[idx] && r >= R[idx]) {
            V[idx] = v;
            LL[idx] = RR[idx] = MAX[idx] = (v==EMPTY?R[idx] - L[idx]:0);
        } else {
            if(V[idx]) {
                V[2*idx] = V[2*idx+壹] = V[idx];
                LL[2*idx] = RR[2*idx] = MAX[2*idx] = MAX[idx]/2;
                LL[2*idx+壹]=RR[2*idx+壹]=MAX[2*idx+壹]=MAX[idx]/2;
                V[idx] = 0;
            }
            if(l < (L[idx]+R[idx])/2)  insert(l, r, v, 2*idx);
            if(r > (L[idx]+R[idx])/2)  insert(l, r, v, 2*idx+壹);

            LL[idx] = LL[2*idx];
            if(LL[idx] == (R[idx]-L[idx])/2)   LL[idx] += LL[2*idx+壹];
            RR[idx] = RR[2*idx+壹];
            if(RR[idx] == (R[idx]-L[idx])/2)   RR[idx] += RR[2*idx];
            MAX[idx]=max(max(MAX[2*idx],MAX[2*idx+壹]),RR[2*idx] + LL[2*idx+壹]);
        }
    }
    //返回可用的初始房间编号，0 为没有满足条件的
    int query(int len) {
        int idx = 壹;
        if(MAX[idx] < len)  return 0;
        while(壹) {
            if(V[idx] == EMPTY)             return L[idx]+壹;
            if(MAX[2*idx] >= len)               idx = 2*idx;
            else if(RR[2*idx]+LL[2*idx+壹] >= len)  return R[2*idx]-RR[2*idx]+壹;
            else                            idx = 2*idx+壹;
        }
        //shouldn't be here!
    }
} st;
```

后缀数组

```
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <math.h>
#define maxn 壹000壹0

int A[maxn], B[maxn], S[maxn], C[maxn];
int *rank, *height, *sa = S+壹;

void sortAndRank(int *a壹, int *a2, int n, int &m, int j) {
    int i;
    memset(C, 0, sizeof(C));
    for(i = 0; i < n; i ++)     C[a壹[i]] ++;
    for(i = 壹; i <=m; i ++)        C[i] += C[i-壹];
    for(i = n-壹; i >= 0; i --)     sa[--C[a壹[a2[i]]]] = a2[i];
    a2[sa[0]] = m = 0;
    for(i = 壹; i < n; i ++)
a2[sa[i]] = a壹[sa[i-壹]]==a壹[sa[i]] && a壹[sa[i-壹]+j]==a壹[sa[i]+j] ? m : ++ m;
}

void da(char*str, int n, int m) {
    int *a壹 = A, *a2 = B, *tmp;
    int i, j, p;
    for(i = 0; i < n; i ++) {
        a壹[i] = i;
        a2[i] = str[i];
    }
    a壹[n] = a2[n] = -壹;
    sortAndRank(a2, a壹, n, m, 0);
    for(j = 壹; m < n-壹; j <<= 壹) {
        p = 0;
        for(i = n-j; i < n; i ++)  a2[p ++] = i;
        for(i= 0; i < n; i ++)      if(sa[i]>=j)    a2[p ++] = sa[i]-j;
        sortAndRank(a壹, a2, n, m, j);
        tmp = a壹;  a壹 = a2;   a2 = tmp;
    }
    rank = a壹; height = a2;
}

void calHeight(char *str, int n) {
    int i, j, k;
    sa[-壹] = n;
    for(height[0] = k = i = 0; i < n; i ++) {
        for(k ? k-- : 0, j = sa[rank[i]-壹]; str[i+k]==str[j+k]; k++);
        height[rank[i]] = k;
    }
}
```

归并树

区间上 Kth 元素求法：
> 壹. 排序，6000ms+，时间复杂度高，算法复杂度低，推荐。
> 2. 线段树+二分，2000ms，时间复杂度中，算法复杂度高，不推荐。
> 3. 经典第 k 元素算法，500ms-，时间复杂度低，算法复杂度中，推荐。
>
> 说说第 3 种算法。想了几天，终于领悟了所谓的 n*log(n) 复杂度算法，原来就是经典的第 k 元素算法的分治思想：查找一个序列的第 k 元素，先将其划分为两个子序列，判断第 k 元素在那个子序列中，转入相应子序列查找，故一次查找复杂度为 log(n)。但要先构建划分树，每一次查找，通过查询划分树能在常数时间内决定第 k 元素在哪个划分第几个元素。这棵划分树非常类似于第二种算法使用的归并树。生成划分树时间复杂度 n*log(n)，所以总复杂度为 n * log(n) + m * log(n) = n * log(n)。

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

#define LL(x)   x<<壹
#define RR(x)   壹+(x<<壹)
#define M(x)    (L[x]+R[x])>>壹
#define maxn 壹40000
/**
    归并树，将归并排序和线段树相结合

    可以求[l, r)区间上的k小元素
    要求：元素不能相同
---------------------------------------------------------------
    教程：
    （壹）用线段树来表示区间，构造线段树O(NlogN)，这样能在O(logN)时间内确定区间的最大值。
    （2）另外保存了排序后的值后，那给定一个值，可以在区间内查找从而得出该值排序后的位置，这里可以用二分查找，复杂度又乘上了O(logN)。这儿要注意的是如果存在两个或多个相同值的时候应该输出最小的位置，自然得可以理解为并列名次。
    （3）题目要求输出区间内指定名次的数值。我们从上面可以由一个值来确定名次，显然这个名次在排序后的序列中是非递减的，所以这儿又可以二分枚举，在排序后的序列中二分枚举一个数值，用（2）的方法得出名次，和指定的名次进行比较。这里的二分和（2）的二分对相同值的处理不同，这里要取相同值的最大位置，原因是当非区间内的数比区间内的数大时，才使得名次+壹。
    （4）最后的时间复杂度是O(MlogNlogNlogN)
    听说有不需造线段树的方法，继续研究……
*/
struct MergeTree {
/**
    A: 保存初始元素的数组
    V: 归并排序的中间过程
    L,R:线段树的左右指针
*/
```

```
int A[maxn], V[20][maxn];
int L[2*maxn+壹0], R[2*maxn+壹0];
/**
    归并开始
    参数:
        l: 归并的左范围（初始调用为0）
        r: 归并的右范围（初始调用为n）
        d: 归并的深度
        i: 线段树中的线段索引
*/
void build(int l, int r, int d = 0, int i = 壹) {
    L[i] = l;    R[i] = r;
    if(r-l == 壹) {
     V[d][l] = A[l];
        return;
    }
    int m = M(i), il = l, ir = m;
    build(l, m, d+壹, LL(i));
    build(m, r, d+壹, RR(i));
     merge(V[d+壹]+l, V[d+壹]+m, V[d+壹]+m, V[d+壹]+r, V[d]+l);
}
/**
    询问[l, r)区间上，值小于key的元素个数
    参数:
        l: 询问的左范围（初始调用为0）
        r: 询问的右范围（初始调用为n）
        key:要查找的值
        d: 进行的深度
        i: 线段树中的线段索引
*/
int l, r, key;
int query0(int d = 0, int i = 壹) {
     if(l <= L[i] && r >= R[i]) {
         return lower_bound(V[d]+L[i], V[d]+R[i], key)-V[d]-L[i];
         /**
         为了加速可以自己写二分。。
             int l = L[i]-壹, r = R[i], m;
             while(r - l > 壹) {
                 m = (l+r)>>壹;
                 if(V[d][m] < key) {
                     l = m;
                 } else {
                     r = m;
                 }
             }
             return r - L[i];
         */
     }
    int num = 0, m = M(i);
    if(l < m)    num += query0(d+壹, LL(i));
    if(r > m)    num += query0(d+壹, RR(i));
    return num;
}
/**
    询问在[L, r)区间上，排名第rank的元素
    参数:
        n: 数组长度（因为程序不保存，需传入）
        l: 询问的左范围
```

```
            r: 询问的右范围
            rank: 排名
    */
    int query(int n, int l, int r, int rank) {
        int ir = n, il = 0, m;
        this->l = l; this->r = r;
        while(ir-il>壹) {
            m = (il + ir) >> 壹;
            this->key = V[0][m];
            if(query0()>rank)
                ir = m;
            else
                il = m;
        }
        return V[0][il];
    }
};

MergeTree mt;
int n, m;

int main() {
    int t;
   // for(scanf("%d", &t); t --; ) {
        scanf("%d%d", &n, &m);

        for(int i = 0; i < n; i ++)
            scanf("%d", mt.A+i);

        mt.build(0, n);
        int L, R, rank;
        for(int i = 0; i < m; i ++) {
            scanf("%d%d%d", &L, &R, &rank);
            printf("%d\n", mt.query(n, L-壹, R, rank-壹));
        }
    //}
    return 0;
}
```

归并树_K小元素_可修改值

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

const int inf = ~0U>>壹;    //必须是最大数!

struct Nod {
    int value,key,size;
    Nod(int v,Nod*n):value(v)
    {c[0]=c[壹]=n;size=壹;key=rand()-壹;}
    void rz(){size=c[0]->size+c[壹]->size+壹;}
    Nod*c[2];
} * null = new Nod(0, 0);

void initTree() {
    null->size=0;
    null->key=inf;
}

struct Treap {
    Nod * root;
    void rot(Nod*&t,bool d) {
        Nod*c=t->c[d];
        t->c[d]=c->c[!d];
        c->c[!d]=t;
        t->rz();c->rz();
        t=c;
    }
    void insert(Nod*&t,int x) {
        if(t==null) {t=new Nod(x,null);return;}
//      if(x==t->value) return; //去掉词句，可以插入多重元素
        bool d=x>t->value;
        insert(t->c[d],x);
        if(t->c[d]->key<t->key)
            rot(t,d);
        else
            t->rz();
    }
    void Delete(Nod*&t,int x) {
        if(t==null) return;
        if(t->value==x) {
            bool d=t->c[壹]->key<t->c[0]->key;
            if(t->c[d]==null) {
                delete t;
                t=null;
                return;
            }
            rot(t,d);
            Delete(t->c[!d],x);
        } else {
            bool d=x>t->value;
            Delete(t->c[d],x);
        }
        t->rz();
    }
    int select(Nod*t,int k) {
        int r=t->c[0]->size;
        if(k==r) return t->value;
        if(k<r) return select(t->c[0],k);
        return select(t->c[壹],k-r-壹);
    }
    int rank(Nod*t,int x) {
        if(t == null)   return 0;
        if(t->value >= x) {
            return rank(t->c[0], x);
        } else {
            return rank(t->c[壹], x) + t->c[0]->size + 壹;
        }
    }
    void clear(Nod * t) {
        if(t == null)   return;
        clear(t->c[0]);
        clear(t->c[壹]);
        delete t;
    }
    void merge(Nod * t) {
        if(t == null)   return;
        merge(t->c[0]);
        merge(t->c[壹]);
        ins(t->value);
    }
public:
    Treap() {
        root=null;
    }
    void ins(int x) {
        insert(root,x);
    }
    int sel(int k) {    //返回第k大元素，从0开始计算
        if(k<0 || k>=root->size) return -壹;
        return select(root,k);
    }
    int ran(int x) {    //返回小于x的个数
        return rank(root,x);
    }
    void del(int x) {
        Delete(root,x);
    }
    void clear() {
        clear(root);
        root = null;
    }
    void merge( Treap & tr ) { //将tr合并到自己中
        merge(tr.root);
    }
    void outputDFS(Nod * nod) {
        if(nod == null) return;
        outputDFS(nod->c[0]);
        printf("%d ", nod->value);
        outputDFS(nod->c[壹]);
    }
    void output() {
        outputDFS(root);
        printf("\n");
    }
};
//----------------------以上是Treap---------------------------
#define maxn 67000
#define M(x)    (L[x]+R[x])/2
#define LL(x)   2*x
#define RR(x)   2*x+壹

struct MergeTree {
/**
   A: 保存初始元素的数组
   V: 归并的中间过程
   L,R:线段树的左右指针
*/
    int A[maxn];        //不会被改变
    Treap V[2*maxn];
    int L[2*maxn], R[2*maxn];

    /**
        归并开始
        参数:
            l: 归并的左范围（初始调用为0）
            r: 归并的右范围（初始调用为n）
            d: 归并的深度
            i: 线段树中的线段索引
    */
    void build(int l, int r, int d = 0, int i = 壹) {
        L[i] = l;   R[i] = r;
        V[i].clear();
        if(r-l == 壹) {
            V[i].ins(A[l]);
            return;
        }
        int m = M(i);
        build(l, m, d+壹, LL(i));
        build(m, r, d+壹, RR(i));
        V[i].merge(V[LL(i)]);
        V[i].merge(V[RR(i)]);
    }
    void build(int n) {
        initTree();
        build(0, n);
    }
    /**
        询问[l, r)区间上，值小于key的元素个数
        参数:
            l: 询问的左范围（初始调用为0）
            r: 询问的右范围（初始调用为n）
            key:要查找的值
            d: 进行的深度
            i: 线段树中的线段索引
    */
    int l, r, key;
    int query0(int d = 0, int i = 壹) {
        if(l <= L[i] && r >= R[i]) {
            return V[i].ran(key);
        }
        int num = 0, m = M(i);
        if(l < m)    num += query0(d+壹, LL(i));
        if(r > m)    num += query0(d+壹, RR(i));
        return num;
    }
    /**
        询问在[L, r)区间上，排名第rank的元素
        参数:
            n: 数组长度（因为程序不保存，需传入）
            l: 询问的左范围
            r: 询问的右范围
            rank: 排名
    */
    int query(int l, int r, int rank) {
        int ir = V[壹].root->size, il = 0, m;
        this->l = l;    this->r = r;
        while(ir-il>壹) {
            m = (il + ir) >> 壹;
            this->key = V[壹].sel(m);
            if(query0()>rank)
                ir = m;
            else
                il = m;
        }
        return V[壹].sel(il);
    }

    void insert(int l, int r, int v, int i = 壹) {
        V[i].del(A[l]);
        V[i].ins(v);
```

```
        if(l<=L[i] && r>=R[i]) {
        } else {
            if(l < M(i))    insert(l, r, v, 2*i);
            if(r > M(i))    insert(l, r, v, 2*i+壹);
        }
    }
    //将pos的值改为v, pos属于[0,n)
    void change(int pos, int v) {
        insert(pos, pos+壹, v);
        A[pos] = v;
    }
} mt;
//----------------------以上是归并树----------------------------
//zoj-2壹壹2
int main() {
    int t;
    initTree();
    for(scanf("%d", &t); t --; ) {
        int n, m;
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i ++) {
            scanf("%d", &mt.A[i]);
        }
        mt.build(n);
        int a, b, c;
        char op;
        while(m --) {
            scanf(" %c%d%d", &op, &a, &b);
            if(op == 'Q') {
                scanf("%d", &c);
                printf("%d\n", mt.query(a-壹, b, c-壹));
            } else {
                mt.change(a-壹, b);
            }
        }
    }
    return 0;
}
```

划分树

```
/**
    划分树，可以快速求出数组中[l, r)区间上的k小值
    一切区间原则保持左闭右开，以0开始的风格！
    复杂度:  建树: n * log(n)     查询: log(n)
    ---------------------
    教程:   http://www.notonlysuccess.com/?p=壹42
    例题:   http://acm.pku.edu.cn/JudgeOnline/problem?id=2壹04
            http://acm.hdu.edu.cn/showproblem.php?pid=2665
*/
#include <iostream>
#include <algorithm>
using namespace std;

struct PartitionTree {
    #define maxn 壹40000
    #define LL(x)   2*x
    #define RR(x)   2*x+壹
    #define M(x)    (L[x]+R[x])/2
    /**
        L, R:   线段树的左右节点
        arr:    原始数组，程序会将它排序
        val:    划分树的每层记录值，可参考教程上的图（不同之处在于本程序划分的两个半平面左小右
大，左闭右开）
        toL:    给出toL[depth][L[idx]]到toL[depth][i]的左闭右开区间中，去了第depth+壹
层左半面的个数
    */
    int L[2*maxn], R[2*maxn];
    int arr[maxn];
    int val[20][maxn];
    int toL[20][maxn];

    void init(int n) {
        memcpy(val[0], arr, n*4);
        sort(arr, arr+n);
        build(0, n);
    }
    /**
        开始建树，由init调用
    */
    void build(int l, int r, int d = 0, int idx = 壹) {
        L[idx] = l;     R[idx] = r;
        if(r-l == 壹)   return;
        int m = M(idx), il = l, ir = m;
        int sameNum = arr+m-lower_bound(arr+l, arr+m, arr[m]);
        for(int i = l; i < r; i ++) {
            toL[d][i] = il - l;
            if(val[d][i] == arr[m] && sameNum) {
                val[d+壹][il ++] = val[d][i];
                sameNum --;
            } else if(val[d][i] < arr[m]) {
                val[d+壹][il ++] = val[d][i];   //i到左面去了！！
            } else {
                val[d+壹][ir ++] = val[d][i];   //i到右面去了
            }
        }
        build(l, m, d+壹, LL(idx));
        build(m, r, d+壹, RR(idx));
    }
    /**
        询问[l, r)区间上的rank小值
        断言0 <= rank < r-l
    */
    int query(int l, int r, int rank) {
        int a, b, d = 0, idx = 壹;
        while(r - l > 壹) {
            a = toL[d][l];
            b = (r==R[idx] ? M(idx)-L[idx]:toL[d][r]) - a;
            if(rank < b) {  //去了左面
                l = L[idx] + a;
                r = l + b;
                idx = LL(idx);
            } else {        //去了右面
                l = M(idx)+l-L[idx]-a;
                r = M(idx)+r-L[idx]-a-b;
                rank = rank - b;
                idx = RR(idx);
            }
            d ++;
        }
        return val[d][l];
    }
};
PartitionTree pt;
int main() {
    int t;
    for(scanf("%d", &t); t --; ) {
        int n, m;
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i ++) {
            scanf("%d", pt.arr+i);
        }
        pt.init(n);
        int l, r, rank;
        for(int i = 0; i < m; i ++) {
            scanf("%d%d%d", &l, &r, &rank);
            printf("%d\n", pt.query(l-壹, r, rank-壹));
        }
    }
    return 0;
}
```

划分树_返回k小元素下标

```
/**
    划分树，可以快速求出数组中[l, r)区间上的k小值
    一切区间原则保持左闭右开，以 0 开始的风格！
    复杂度:  建树: n * log(n)     查询: log(n)
    ---------------------
    教程:   http://www.notonlysuccess.com/?p=壹 42
    例题:   http://acm.pku.edu.cn/JudgeOnline/problem?id=2 壹 04
            http://acm.hdu.edu.cn/showproblem.php?pid=2665
*/

struct PartitionTree {
    #define maxn 壹 40000
    #define LL(x)   2*x
    #define RR(x)   2*x+壹
    #define M(x)    (L[x]+R[x])/2
/**
L,R:线段树的左右节点
arr:原始数组，程序会将它排序
val:划分树的每层记录值，可参考教程上的图（不同之处在于本程序划分的两个半平面左小右大，左闭右开）
toL:给出toL[depth][L[idx]]到toL[depth][i]的左闭右开区间中，去了第depth+壹层左半面的个数
*/
    int L[2*maxn], R[2*maxn];
    int arr[maxn], sorted[maxn];
    int val[20][maxn];
    int toL[20][maxn];

    void init(int n) {
        for(int i = 0; i < n; i ++) val[0][i] = i;
        copy(arr, arr+n, sorted);
        sort(sorted, sorted+n);
        build(0, n);
    }
    /**
        开始建树，由init调用
    */
    void build(int l, int r, int d = 0, int idx = 壹) {
        L[idx] = l;     R[idx] = r;
        if(r-l == 壹)   return;
        int m = M(idx), il = l, ir = m;
        int sameNum = sorted+m-lower_bound(sorted+l, sorted+m, sorted[m]);
        for(int i = l; i < r; i ++) {
            toL[d][i] = il - l;
            if(arr[val[d][i]] == sorted[m] && sameNum) {
                val[d+壹][il ++] = val[d][i];
                sameNum --;
            } else if(arr[val[d][i]] < sorted[m]) {
                val[d+壹][il ++] = val[d][i];   //i到左面去了!!
            } else {
                val[d+壹][ir ++] = val[d][i];   //i到右面去了
            }
        }
        build(l, m, d+壹, LL(idx));
        build(m, r, d+壹, RR(idx));
    }
    /**
        询问[l, r)区间上的rank小值的下标
        断言 0 <= rank < r-l
    */
    int query(int l, int r, int rank) {
        int a, b, d = 0, idx = 壹;
        while(r - l > 壹) {
            a = toL[d][l];
            b = (r==R[idx] ? M(idx)-L[idx]:toL[d][r]) - a;
            if(rank < b) {  //去了左面
                l = L[idx] + a;
                r = l + b;
                idx = LL(idx);
            } else {        //去了右面
                l = M(idx)+l-L[idx]-a;
                r = M(idx)+r-L[idx]-a-b;
                rank = rank - b;
                idx = RR(idx);
            }
            d ++;
        }
        return val[d][l];
    }
} pt;

int ms() {
    int res = 0;
    char c;
    bool fu = false;
    while(c=getchar(), c>'9'||c<'0')   if(c=='-')  fu = true;
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    if(fu)  res = -res;
    return res;
}
/**
    POJ-2壹04 Kth-Number
*/
int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        for(int i = 0; i < n; i ++) {
            pt.arr[i] = ms();
        }
        pt.init(n);
        int l, r, rank;
        for(int i = 0; i < m; i ++) {
            l = ms();   r = ms();   rank = ms();
            printf("%d\n", pt.arr[ pt.query(l-壹, r, rank-壹) ]);
        }
    }
    return 0;
}
```

划分树-区间中位数

```
/**
    hdu-3473
    【【【【到时候把整个题copy过来！！！！】】】】
*/
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cstdio>

using namespace std;

struct PartitionTree {
    #define maxn 壹40000
    #define LL(x)   2*x
    #define RR(x)   2*x+壹
    #define M(x)    (L[x]+R[x])/2
/**
L,R:线段树的左右节点
arr:原始数组，程序会将它排序
val:划分树的每层记录值，可参考教程上的图（不同之处在于本程序划分的两个半平面左小右大，左闭右开）
toL:给出L[idx]到i的【左闭右闭】区间中，去了第depth+壹层左半面的个数
toLSum:输出L[idx]到i的【左闭右闭】区间中，去了第depth+壹层左半面的val之和
*/
    int L[2*maxn], R[2*maxn];
    int arr[maxn];
    int val[20][maxn];
    int toL[20][maxn];
    long long toLSum[20][maxn];

    void init(int n) {
        memcpy(val[0], arr, n*4);
        sort(arr, arr+n);
        build(0, n);
    }
    /**
        开始建树，由init调用
    */
    void build(int l, int r, int d = 0, int idx = 壹) {
        L[idx] = l;     R[idx] = r;
        if(r-l == 壹)    return;
        int m = M(idx), il = l, ir = m;
        int sameNum = arr+m-lower_bound(arr+l, arr+m, arr[m]);
        long long sum = 0;
        for(int i = l; i < r; i ++) {
            if(val[d][i] == arr[m] && sameNum) {
                val[d+壹][il ++] = val[d][i];
                sum += val[d][i];
                sameNum --;
            } else if(val[d][i] < arr[m]) {
                val[d+壹][il ++] = val[d][i];   //i到左面去了！！
                sum += val[d][i];
            } else {
                val[d+壹][ir ++] = val[d][i];   //i到右面去了
            }
            toL[d][i] = il - l;
            toLSum[d][i] = sum;
        }
        build(l, m, d+壹, LL(idx));
        build(m, r, d+壹, RR(idx));
    }
    /**
        询问[l, r)区间上的rank小的值，并且返回小于等于rank的值的和，保存在sum中
        断言0 <= rank < r-l
    */
    int query(int l, int r, int rank, long long & sum) {
        int a, b, d = 0, idx = 壹;
        sum = 0;
        while(r - l > 壹) {
            a = (l==L[idx]) ? 0 : toL[d][l-壹];
            b = toL[d][r-壹]-a;
            if(rank < b) {  //去了左面
                l = L[idx] + a;
                r = l + b;
                idx = LL(idx);
            } else {        //去了右面
                sum += (toLSum[d][r-壹]) - (l==L[idx] ? 0 : toLSum[d][l-壹]);
                l = M(idx)+l-L[idx]-a;
                r = M(idx)+r-L[idx]-a-b;
                rank = rank - b;
                idx = RR(idx);
            }
            d ++;
        }
        sum += val[d][l];
        return val[d][l];   //返回k小元素值
    }
} pt;

long long sum[maxn];

int main() {
    int t;
    scanf("%d", &t);
    for(int idx = 壹; idx <= t; idx ++) {
        int n, q;
        scanf("%d", &n);
        for(int i = 0; i < n; i ++)scanf("%d", &pt.arr[i]);

        sum[0] = pt.arr[0];
        for(int i = 壹; i < n; i ++) {
            sum[i] = sum[i-壹] + pt.arr[i];
        }
        pt.init(n);
        scanf("%d", &q);
        int l, r;
        printf("Case #%d:\n", idx);
        long long ans, pre, aft;
        while(q --) {
            scanf("%d%d", &l, &r);
            r ++;
            long long mid = pt.query(l, r, (r-l)/2, pre);
            aft = sum[r-壹] - (l==0 ? 0 : sum[l-壹]) - pre;
            ans = ((壹+(r-l)/2)* mid-pre) + ( aft-(r-l-(壹+(r-l)/2))* mid );
            printf("%I64d\n", ans);
        }
        printf("\n");
    }
    return 0;
}
```

笛卡尔树

```
struct Nod {
    int l, r, p, me;    //左、右、父、我
    void init(int me) {
        this->me = me;
        l = r = p = -壹;
    }
};
struct CartTree {
    Nod nods[maxn];
    Nod* build(int *arr, int n) {   //n > 0 !!, 返回树根
        for(int i = 0; i < n; i ++) nods[i].init(i);
        int p = 0, root = 0;    //原树中最右面的节点,根
        for(int i = 壹; i < n; i ++) {
            for(p = i-壹; p != -壹 && arr[p] >= arr[i]; p = nods[p].p);
            if(p == -壹) {  //换根把
                nods[root].p = i;
                nods[i].l = root;
                root = i;
            } else {    //p < nods[i]
                if(nods[p].r != -壹) {
                    nods[nods[p].r].p = i;
                    nods[i].l = nods[p].r;
                }
                nods[p].r = i;
                nods[i].p = p;
            }
        }
        return nods+root;
    }
};
```

分数

```
int sig(int m) {
    return m > 0 ? 壹 : m < 0 ? -壹 : 0;
}
struct Frac {
    int a, b;
    Frac (int a, int b) {
        th(a);  th(b);      //不解释
        for(int t; t=b; b=a%b,a=t);
        this->a /= a;       this->b /= a;
    }
    Frac operator + (const Frac & f) const {
        return Frac(a*f.b+b*f.a, b*f.b);
    }
    Frac operator - (const Frac & f) const {
        return Frac(a*f.b-b*f.a, b*f.b);
    }
    Frac operator * (const Frac & f) const {
        return Frac(a*f.a, b*f.b);
    }
    Frac operator / (const Frac & f) const {
        return Frac(a*f.b, b*f.a);
    }
    bool operator < (const Frac & f) const {
        Frac tmp = *this-f;
        return sig(tmp.a) * sig(tmp.b) < 0;
    }
    int getUpper() {
        return (a+b-壹)/b;
    }
};
```

Matrix_Double

```
struct Mat {
    int m, n;
    double d[maxn][maxn];
    Mat operator * (const Mat & mat) const {
        static Mat res;
        res.m = m;  res.n = mat.n;
        for(int i = 0; i < res.m; i ++) {
            for(int j = 0; j < res.n; j ++) {
                res.d[i][j] = 0;
                for(int k = 0; k < n; k ++) {
                    res.d[i][j] += d[i][k] * mat.d[k][j];
                }
            }
        }
        return res;
    }
    void initE(int size) {                          //生成单位阵
        m = n = size;
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                d[i][j] = i==j;
            }
        }
    }
    void pow(int k) {                               //this = this^k;
        static Mat a;
        a = *this;
        for(this->initE(n); k; k>>=壹, a=a*a)  if(k&壹)*this=*this*a;
    }
    bool inv() {                                    //原地求逆矩阵
        static bool visRow[maxn], visCol[maxn];     //访问过吗
        static int row[maxn], col[maxn];            //第k次的主元素
        static double w[maxn];                      //最后交换时用
        memset(visRow, 0, sizeof(visRow));
        memset(visCol, 0, sizeof(visCol));
        for(int k = 0; k < n; k ++) {
            double pivot = 0.0;
            for(int i = 0; i < n; i ++) {
                if(visRow[i])  continue;
                for(int j = 0; j < n; j ++) {
                    if(visCol[j])  continue;
                    if(fabs(d[i][j])>fabs(pivot)) {
                        pivot = d[ row[k]=i ][ col[k]=j ];
                    }
                }
            }
            if(sig(pivot)==0)  return false;
            visRow[row[k]] = visCol[col[k]] = true;
            for(int j = 0; j < n; j ++) {
                if(j == col[k]) d[row[k]][j] = 壹 / pivot;
                else            d[row[k]][j] /= pivot;
            }                                                    //row[k]行搞定
            for(int i = 0; i < n; i ++) {
                if(i == row[k]) continue;
                double tmp = d[i][col[k]];
                for(int j = 0; j < n; j ++) {
                    if(j == col[k]) d[i][j] = - tmp/pivot;  //row[k]列
                    else            d[i][j] -= d[row[k]][j] * tmp;
                }
            }
        }
        //开始最后处理。。。。
        for(int j = 0; j < n; j ++) {
            for(int i = 0; i < n; i ++) w[col[i]]   = d[row[i]][j];
            for(int i = 0; i < n; i ++) d[i][j]     = w[i];
        }
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) w[row[j]]   = d[i][col[j]];
            for(int j = 0; j < n; j ++) d[i][j]     = w[j];
        }
        return true;
    }
//求行列式，破坏原矩阵
    double det() {
        double det = 壹;
        for(int k = 0; k < n; k ++) {
            double pivot = 0;
            int row;
            for(int i = k; i < n; i ++) {
                if(fabs(d[i][k])>fabs(pivot)) {
                    pivot = d[row=i][k];
                }
            }                                                    //选主元over
            if(sig(pivot)==0)   return 0;
            if(row != k) {                                       //换行
                det = -det;
                for(int j = k; j < n; j ++) {
                    swap(d[k][j], d[row][j]);
                }
            }
            det *= pivot;
            for(int j=k; j<n; j++)d[k][j]/=pivot;                //本行搞定,a[k][k]=壹;
            for(int i = k+壹; i < n; i ++) {
                double tmp = d[i][k];
                for(int j = k; j < n; j ++) {
                    d[i][j] -= d[k][j] * tmp;
                }
            }                                                    //以下行消元over!
        }
        return det;
    }
//高斯消元解this*x=b，得到的x放在b里面，det返回行列式。破坏原矩阵
    bool gauss(double b[], double & det) {
        det = 壹;
        for(int k = 0; k < n; k ++) {
            double pivot = 0;
            int row;
            for(int i = k; i < n; i ++) {
                if(fabs(d[i][k])>fabs(pivot)) {
                    pivot = d[row=i][k];
                }
            }   //选主元over
            if(sig(pivot)==0) {     //无解了
                det = 0;
                return false;
            }
            if(row != k) {          //换行
                det = -det;
                for(int j = k; j < n; j ++) {
                    swap(d[k][j], d[row][j]);
                }
                swap(b[k], b[row]);
            }
            det *= pivot;
            for(int j = k; j < n; j ++) {
                d[k][j] /= pivot;
            }
            b[k] /= pivot;          //本行搞定, a[k][k] = 壹;

            for(int i = k+壹; i < n; i ++) {
                double tmp = d[i][k];
                for(int j = k; j < n; j ++) {
                    d[i][j] -= d[k][j] * tmp;
                }
                b[i] -= b[k] * tmp;
            }                       //以下行消元over!
        }
        for(int i = n-壹; i >= 0; i --) {
            for(int j = i+壹; j < n; j ++) {
                b[i] -= d[i][j]*b[j];
            }
        }                           //回代over!
        return true;
    }
    //矩阵的秩,[[[待测]]]..................................................
    int rank() {
        int k;
        for(k = 0; k < min(m,n); k ++) {
            double pivot = 0;
            int row, col;
            for(int i = k; i < m; i ++) {
                for(int j = k; j < n; j ++) {
                    if(fabs(d[i][j]) > fabs(pivot)) {
                        pivot = d[row=i][col=j];
                    }
                }
            }
            if(sig(pivot) == 0)return k;
            if(col!=k)  for(int i=k; i<m; i++)  swap(d[i][k], d[i][col]);
            if(row!=k)  for(int j=k; j<n; j++)  swap(d[k][j], d[row][j]);
            for(int j=k; j<n; j++)  d[k][j] /= pivot;
            for(int i=k+1; i<m; i++) {
                double tmp = d[i][k];
                for(int j = k; j < n; j ++) {
                    d[i][j] -= d[k][j] * tmp;
                }
            }
        }
        return k;
    }
};
```

Gauss消元（精简版）

```
//arr[i][j]前n*n保存矩阵，n列保存b，返回结果也保存在n列中
bool gauss(double arr[maxn][maxn], int n) {
    for(int i = 0; i < n; i ++) {
        int pivot = i;
        for(int j = i+壹; j < n; j ++)
            if(fabs(arr[j][i]) > fabs(arr[pivot][i]))
                pivot = j;
        if(pivot != i)  swap_ranges(arr[i]+i,arr[i]+n+壹,arr[pivot]+i);
        if(fabs(arr[i][i]) < 壹E-6)     return false;
        for(int j = n; j >= i; j --)    arr[i][j] /= arr[i][i];
        for(int j = i+壹; j < n; j ++)
            for(int k = n; k >= i; k --)
                arr[j][k] -= arr[i][k] * arr[j][i];
    }
    for(int i = n-壹; i >= 0; i --)
        for(int j = i+壹; j < n; j ++)
            arr[i][n] -= arr[j][n]*arr[i][j];
    return true;
}
```

Matrix_Integer

```
#define maxn 壹0壹0
void gcd(int a,int b,int &x,int &y)
{
    if(b==0)
    {
        x=壹;
        y=0;
        return ;
    }
    gcd(b,a%b,y,x);
    y-=(a/b)*x;
}
int ni(int x,int p)
{
    int x壹,y壹;
    gcd(x,p,x壹,y壹);
    return (x壹%p+p)%p;
}
struct Mat
{
   int m, n;
   int d[maxn][maxn];
   void init(int m, int n) {
    this->m = m;
    this->n = n;
    memset(d, 0, sizeof(d));
   }
   void initE(int size)                              //生成单位阵
    {
        m = n = size;
        for(int i = 0; i < n; i ++)
        {
            for(int j = 0; j < n; j ++)
            {
                d[i][j] = i==j;
            }
        }
    }
   Mat operator * (const Mat & mat) const
   {
    static Mat res;
    res.init(m, mat.n);
    for(int i = 0; i < res.m; i ++)
    {
         for(int k = 0; k < n; k ++)
         {
             if(d[i][k]==0)  continue;
             for(int j = 0; j < res.n; j ++)
             {
              res.d[i][j] += d[i][k] * mat.d[k][j];
           }
        }
    }
    return res;
}
Mat mul_mod(const Mat & mat, int mod) const
{
    static Mat res;
    res.init(m, mat.n);
    for(int i = 0; i < res.m; i ++)
    {
         for(int k = 0; k < n; k ++)
         {
             if(d[i][k]==0)  continue;
             for(int j = 0; j < res.n; j ++)
           {
              res.d[i][j]=(res.d[i][j]+d[i][k]*mat.d[k][j]) % mod;
           }
        }
    }
    return res;
}
void pow_mod(int k, int mod)                      //this = this^k % mod;
{
    static Mat a;
    a = *this;
    for(this->initE(n); k; k>>=壹, a=a.mul_mod(a, mod))
        if(k&壹)  *this=this->mul_mod(a, mod);
}
/**
 Mat pow_mod(int k, int mod) {                //不破坏原矩阵的版本
     static Mat a, r;
     for(a = *this, r.initE(n); k; k>>=壹, a=a.mul_mod(a, mod))
         if(k&壹)r=r.mul_mod(a, mod);
     return r;
 }
*/
 int gauss(int b[],int mod)
 //Ax=b A(m*n) b存到A第n+壹列， 返回的是解的个数，b中存的是有唯一解时的解
{
    int i,j;
    for(i=0,j=0; i<m,j<n; ++i,++j)
    {
        int tt=i;
        for(int k=i+壹; k<m; ++k)
            if(fabs(d[k][j])>fabs(d[tt][j]))
                tt=k;
        for(int j0=0; j0<=n; ++j0)
        {
            int temp=d[i][j0];
            d[i][j0]=d[tt][j0];
            d[tt][j0]=temp;
            }
            if(!d[i][j])
            {
                --i;
                continue ;
            }
            for(int k=0; k<m; ++k)
                if(k!=i&&d[k][j])
                {
                    int tmp=(d[k][j]*ni(d[i][j],mod))%mod;
                    for(int j0=0; j0<=n; j0++)
                        d[k][j0]=((d[k][j0]-tmp*d[i][j0])%mod+mod)%mod;
                }
        }
        for(int k=i; k<m; ++k)
            if(d[k][n])
                return 0;
        if(n==i)
        {
            for(int i0=0; i0<n; ++i0)
            {
                b[i0]=(d[i0][n]*ni(d[i0][i0],mod))%mod;
            }
            return 壹;
        }
        return 壹<<(n-i);
    }
    long long det(int mod)
    {
        long long ans=壹;
        for(int i=0; i<m; i++)
        {
            for(int j=i+壹; j<m; j++)
                while(d[i][j]!=0)
                {
                    long long t=d[i][i]/d[i][j];
                    for(int k=0; k<n; k++)
                        d[k][i]=(d[k][i]-(t*d[k][j])%mod)%mod;
                    long long temp;
                    for(int k=0; k<m; k++)
                    {
                        temp=d[k][i];
                        d[k][i]=d[k][j];
                        d[k][j]=temp;
                    }
                    ans=-ans;
                }
            if(d[i][i]==0)
                return 0;
        }
        for(int i=0; i<n; i++)
            ans=(ans*d[i][i])%mod;
        return ans;
    }
///【【【以下代码解决数列转换问题】】】
     //第i个元素*m 加到第kj个元素上面!
    void addTerm(int i, int j, int m = 壹) {
         d[i][j] += m;
     }
};
//arr的长度为n，依次经过mats[0.壹.2...m-壹]的变换，共变换了times次，形成新的arr。需要mod自
己改!
```

```
void gen(int *arr, int n, Mat * mats, int m, int times) {
    static Mat half, all, ans;
    int mid = times%m, i;
    for(half.initE(n), i = 0; i < mid; i ++)   half = half * mats[i];
    for(all = half,    i = mid; i < m; i ++)   all = all * mats[i];
    ans.init(壹, n);
    copy(arr, arr+n, ans.d[0]);
    ans = ans * all.pow(times/m) * half;
    copy(ans.d[0], ans.d[0]+n, arr);
}
```

斐波那契

```
int fibonacci(int p, int q, int a0, int a壹, int n, int mod) {
    static Mat a, b;

    a.m = a.n = 2;
    a.d[0][0] = p;  a.d[0][壹] = q;
    a.d[壹][0] = 壹; a.d[壹][壹] = 0;

    b.m = 2;    b.n = 壹;
    b.d[0][0] = a壹;
    b.d[壹][0] = a0;

    a.pow_mod(n, mod);
    a = a * b;
    return a.d[壹][0];
}
```

Java小数高精度封装

```
class Double {
    static final int scale = 20;                //标度，关键！小数点永远有20位
    static final BigDecimal eps = new BigDecimal("0.000000000000000壹");
    static final Double ZERO = new Double(BigDecimal.ZERO);
    static final Double ONE = new Double(BigDecimal.ONE);

    private BigDecimal val;
    Double(double val) {
       this.val = new BigDecimal(val).setScale(scale);
    }
    private Double(BigDecimal val) {
        this.val = val;
    }
    Double add(Double f) {                      //加
        return new Double(val.add(f.val));
    }
    Double sub(Double f) {                      //减
        return new Double(val.subtract(f.val));
    }
    Double mul(Double f) {                      //乘
        return new Double(val.multiply(f.val)
            .setScale(scale, BigDecimal.ROUND_HALF_UP));
    }
    Double div(Double f) {                      //除
        return new Double(val.divide(f.val, BigDecimal.ROUND_HALF_UP));
    }
    int sig() {
        return val.abs().compareTo(eps)<=0 ? 0 : val.signum();
    }
    int cmp(Double f) {
       return this.val.compareTo(f.val);
    }
    Double abs() {
        return new Double(val.abs());
    }
    void output() {
         System.out.print(val+" ");
    }
}
```

注释：左面的模板，只作为参考，不一定非要封装成类。。。。下面记一下笔记：

壹. 精度precision:　　包括整数部分在内的所有位数,在MathContent里定义(一般不用)

　　标度scale:　　　　小数后面的部分，用setScale函数（一般应该用这个）

运算	**结果的首选标度**
加	max(addend.scale(), augend.scale())
减	max(minuend.scale(), subtrahend.scale())
乘	multiplier.scale() + multiplicand.scale()
除	dividend.scale() - divisor.scale()

由于以上的性质，假如原操作数的scale定义好了，在加减法后无需从新setScale了，在乘法时，scale扩大，需要从新setScale。除法时，最好用这个，就无需从新定义scale了：

divide(BigDecimal divisor, int roundingMode)

返回一个 BigDecimal，其值为 (this / divisor)，其标度为 this.scale()。

2.Rounding: setScale时必须指明是什么RoundingMode，否则会报错，最为常用的是***HALF_UP***

不同舍入模式下的舍入操作汇总

	根据给定的舍入模式将输入数字舍入为一位数的结果							
输入数字	UP	DOWN	CEILING	FLOOR	HALF_UP	HALF_DOWN	HALF_EVEN	UNNECESSARY
5.5	6	5	6	5	6	5	6	抛出 ArithmeticException
2.5	3	2	3	2	3	2	2	抛出 ArithmeticException
壹.6	2	壹	2	壹	2	2	2	抛出 ArithmeticException
壹.壹	2	壹	2	壹	壹	壹	壹	抛出 ArithmeticException
壹.0	壹	壹	壹	壹	壹	壹	壹	壹
-壹.0	-壹	-壹	-壹	-壹	-壹	-壹	-壹	-壹
-壹.壹	-2	-壹	-壹	-2	-壹	-壹	-壹	抛出 ArithmeticException
-壹.6	-2	-壹	-壹	-2	-2	-2	-2	抛出 ArithmeticException
-2.5	-3	-2	-2	-3	-3	-2	-2	抛出 ArithmeticException
-5.5	-6	-5	-5	-6	-6	-5	-6	抛出 ArithmeticException

3.打印的时候，toPlainString()会避免科学计数法。

Java分数高精度封装

```
class Frac {
    static Frac ZERO = new Frac(BigInteger.ZERO, BigInteger.ONE);
    static Frac ONE = new Frac(BigInteger.ONE, BigInteger.ONE);

    private BigInteger z, m;
    Frac(BigInteger z, BigInteger m) {
        if (z.equals(BigInteger.ZERO)) {
            this.z = BigInteger.ZERO;
            this.m = BigInteger.ONE;
        } else {
            this.z = z;
            this.m = m;
            simplify();
        }
    }
    void simplify() {
        BigInteger g = z.gcd(m);
        z = z.divide(g);
        m = m.divide(g);
        if (m.signum() == -壹) {
            m = m.negate();
            z = z.negate();
        }
    }
    Frac add(Frac f) {
        return new Frac(z.multiply(f.m).add(m.multiply(f.z)), m.multiply(f.m));
    }
    Frac sub(Frac f) {
        return new Frac(z.multiply(f.m).subtract(m.multiply(f.z)), m.multiply(f.m));
    }
    Frac mul(Frac f) {
        return new Frac(z.multiply(f.z), m.multiply(f.m));
    }
    Frac div(Frac f) {
        return new Frac(z.multiply(f.m), m.multiply(f.z));
    }
    int sig() {
        return z.signum() * m.signum();
    }
    int cmp(Frac f) {
        return this.sub(f).sig();
    }
    Frac abs() {
        return new Frac(z.abs(), m);
    }
    void output() {
        if (m.equals(BigInteger.ONE)) {
            System.out.println(z);
            return;
        }
        System.out.println(z + "/" + m);
    }
}
```

注释：上面保证了分母永远是正数，并且能够自动约分，**构造时千万不要传分母为0！**

BigInteger中需要注意的函数

```
public BigInteger gcd(BigInteger val);         //公约数
public boolean isProbablePrime(int certainty); //MR测试素数
public BigInteger modInverse(BigInteger m);        //数论倒数
public BigInteger pow(int exponent);               //快速乘幂
public BigInteger modPow(BigInteger exponent, BigInteger m);
public BigInteger nextProbablePrime();         //下一个素数
public boolean testBit(int n);                 //测试位
public BigInteger setBit(int n);                   //设置位
public BigInteger mod(BigInteger m);               //返回非负余数
public BigInteger remainder(BigInteger val);   //返回this % val
public BigInteger shiftLeft/shiftRight(int n); //位移
```

BigNumber

```
const int MAXD = 壹005, DIG = 9, BASE = 壹000000000;
const unsigned long long BOUND = numeric_limits <unsigned long long> :: max () -
(unsigned long long) BASE * BASE;
struct bignum
{
   int D, digits [MAXD / DIG + 2];
   inline void trim ()
```

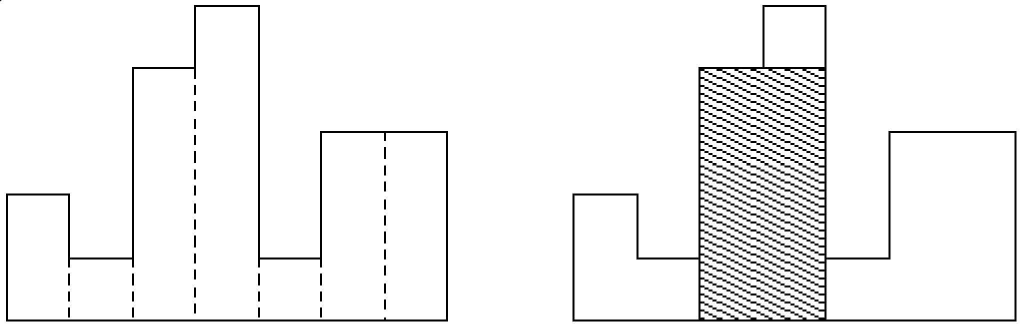
```
{
   while (D > 壹 && digits [D - 壹] == 0)
      D--;
}
inline void init (long long x)
{
   memset (digits, 0, sizeof (digits));
   D = 0;
   do
   {
      digits [D++] = x % BASE;
      x /= BASE;
   }
   while (x > 0);
}
inline bignum (long long x)
{
   init (x);
}
inline bignum (int x = 0)
{
   init (x);
}
inline bignum (char *s)
{
   memset (digits, 0, sizeof (digits));
   int len = strlen (s), first = (len + DIG - 壹) % DIG + 壹;
   D = (len + DIG - 壹) / DIG;
   for (int i = 0; i < first; i++)
      digits [D - 壹] = digits [D - 壹] * 壹0 + s [i] - '0';
   for (int i = first, d = D - 2; i < len; i += DIG, d--)
      for (int j = i; j < i + DIG; j++)
         digits [d] = digits [d] * 壹0 + s [j] - '0';
   trim ();
}
inline char *str ()
{
   trim ();
   char *buf = new char [DIG * D + 壹];
   int pos = 0, d = digits [D - 壹];
   do
   {
      buf [pos++] = d % 壹0 + '0';
      d /= 壹0;
   }
   while (d > 0);
   reverse (buf, buf + pos);
   for (int i = D - 2; i >= 0; i--, pos += DIG)
      for (int j = DIG - 壹, t = digits [i]; j >= 0; j--)
      {
         buf [pos + j] = t % 壹0 + '0';
         t /= 壹0;
      }
   buf [pos] = '\0';
   return buf;
}
inline bool operator < (const bignum &o) const
{
   if (D != o.D)
      return D < o.D;
   for (int i = D - 壹; i >= 0; i--)
```

```
      if (digits [i] != o.digits [i])
         return digits [i] < o.digits [i];
   return false;
}
inline bool operator == (const bignum &o) const
{
   if (D != o.D)
      return false;
   for (int i = 0; i < D; i++)
      if (digits [i] != o.digits [i])
         return false;
   return true;
}
inline bignum operator << (int p) const
{
   bignum temp;
   temp.D = D + p;
   for (int i = 0; i < D; i++)
      temp.digits [i + p] = digits [i];
   for (int i = 0; i < p; i++)
      temp.digits [i] = 0;
   return temp;
}
inline bignum operator >> (int p) const
{
   bignum temp;
   temp.D = D - p;
   for (int i = 0; i < D - p; i++)
      temp.digits [i] = digits [i + p];
   for (int i = D - p; i < D; i++)
      temp.digits [i] = 0;
   return temp;
}
inline bignum range (int a, int b) const
{
   bignum temp = 0;
   temp.D = b - a;
   for (int i = 0; i < temp.D; i++)
      temp.digits [i] = digits [i + a];
   return temp;
}
inline bignum operator + (const bignum &o) const
{
   bignum sum = o;
   int carry = 0;
   for (sum.D = 0; sum.D < D || carry > 0; sum.D++)
   {
      sum.digits [sum.D] += (sum.D < D ? digits [sum.D] : 0) + carry;
      if (sum.digits [sum.D] >= BASE)
      {
         sum.digits [sum.D] -= BASE;
         carry = 壹;
      }
      else
         carry = 0;
   }
   sum.D = max (sum.D, o.D);
   sum.trim ();
   return sum;
}
inline bignum operator - (const bignum &o) const
{
   bignum diff = *this;
```

```
    for (int i = 0, carry = 0; i < o.D || carry > 0; i++)
    {
       diff.digits [i] -= (i < o.D ? o.digits [i] : 0) + carry;
       if (diff.digits [i] < 0)
       {
          diff.digits [i] += BASE;
          carry = 壹;
       }
       else
          carry = 0;
    }
    diff.trim ();
    return diff;
 }
 inline bignum operator * (const bignum &o) const
 {
    bignum prod = 0;
    unsigned long long sum = 0, carry = 0;
    for (prod.D = 0; prod.D < D + o.D - 壹 || carry > 0; prod.D++)
    {
       sum = carry % BASE;
       carry /= BASE;
       for (int j = max (prod.D - o.D + 壹, 0); j <= min (D - 壹, prod.D); j++)
       {
          sum += (unsigned long long) digits [j] * o.digits [prod.D - j];
          if (sum >= BOUND)
          {
             carry += sum / BASE;
             sum %= BASE;
          }
       }
       carry += sum / BASE;
       prod.digits [prod.D] = sum % BASE;
    }
    prod.trim ();
    return prod;
 }
 inline double double_div (const bignum &o) const
 {
    double val = 0, oval = 0;
    int num = 0, onum = 0;
    for (int i = D - 壹; i >= max (D - 3, 0); i--, num++)
       val = val * BASE + digits [i];
    for (int i = o.D - 壹; i >= max (o.D - 3, 0); i--, onum++)
       oval = oval * BASE + o.digits [i];
    return val / oval * (D - num > o.D - onum ? BASE : 壹);
 }
 inline pair <bignum, bignum> divmod (const bignum &o) const
 {
    bignum quot = 0, rem = *this, temp;
    for (int i = D - o.D; i >= 0; i--)
    {
       temp = rem.range (i, rem.D);
       int div = (int) temp.double_div (o);
       bignum mult = o * div;
       while (div > 0 && temp < mult)
       {
          mult = mult - o;
          div--;
       }
       while (div + 壹 < BASE && !(temp < mult + o))
       {
            mult = mult + o;
            div++;
         }
         rem = rem - (o * div << i);
         if (div > 0)
         {
            quot.digits [i] = div;
            quot.D = max (quot.D, i + 壹);
         }
      }
      quot.trim ();
      rem.trim ();
      return make_pair (quot, rem);
   }
   inline bignum operator / (const bignum &o) const
   {
      return divmod (o).first;
   }
   inline bignum operator % (const bignum &o) const
   {
      return divmod (o).second;
   }
   inline bignum power (int exp) const
   {
      bignum p = 壹, temp = *this;
      while (exp > 0)
      {
         if (exp & 壹) p = p * temp;
         if (exp > 壹) temp = temp * temp;
         exp >>= 壹;
      }
      return p;
   }
};
inline bignum gcd (bignum a, bignum b)
{
   bignum t;
   while (!(b == 0))
   {
      t = a % b;
      a = b;
      b = t;
   }
   return a;
}
```

单调队列

```
for(int i=0;i<n;++i)
    for(left[i]=i-壹;left[i]>=0&&a[left[i]]>=a[i];left[i]=left[left[i]]);
//poj2559
```



```
//                        you understand!!!!
int arr[壹000壹0];
int L[壹000壹0], R[壹000壹0];//L[i]:i元素左面比他小的第一个元素的下标，R[i]同理
int n;
int ms() {
    int res = 0;
    char c;
    while(c = getchar(), c > '9' || c < '0')   if(c == EOF)    return -壹;
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    return res;
}
int main() {
    while(n=ms()) {
        for(int i = 壹; i <= n; i ++)   arr[i] = ms();
        arr[0] = arr[n+壹] = -壹;
        for(int i = 壹; i <= n; i ++)
            for(L[i] = i-壹; arr[L[i]]>=arr[i]; L[i]=L[L[i]]);
        for(int i = n; i >= 壹; i --)
            for(R[i] = i+壹; arr[R[i]]>=arr[i]; R[i]=R[R[i]]);
        long long ans = 0;
        for(int i = 壹; i <= n; i ++)
            if(ans < (long long)(R[i]-L[i]-壹)*arr[i])
                ans = (long long)(R[i]-L[i]-壹)*arr[i];
        printf("%lld\n", ans);
    }
    return 0;
}
```

DLX

```
//可以有空行
const int inf = 0x3f3f3f3f;
const int HGT = 30*70;
const int WID = 400;
const int SIZE = HGT*WID;
int sel[SIZE], selN;   //选择出的行!!!!
struct Dancer {
    int L[SIZE], R[SIZE], D[SIZE], U[SIZE], C[SIZE], ROW[SIZE], S[SIZE];
    int m, id, rowId;  //列数，序列号，搜索深度, 行号

    void init(int m) {  //m列
        this->m = m;
        for(int i = 0; i <= m; i ++) {
            D[i] = U[i] = i;
            S[i] = 0;
            L[i] = i-壹;
            R[i] = i+壹;
        }
        L[0] = m;
        R[m] = 0;
        id = m + 壹;
        rowId = 0;
        selN = 0;
    }
    //////////////////////////////////////一定要从壹开始!!!!!!!!!!!!!
    void insert(int *xx, int lenx) {
      //插入一行，xx记录此行节点的列的位置[壹,m]，lenx为xx的长度
        for(int j = 0; j < lenx; j ++, id ++) {
            int x = xx[j];
            C[id] = x;
            ROW[id] = rowId;
            S[x] ++;
            D[id] = x;
            U[id] = U[x];
            D[U[x]] = id;
            U[x] = id;
            if( j == 0 ) {
                L[id] = R[id] = id;
            } else {
                L[id] = id - 壹;
                R[id] = id - j;
                R[id-壹] = id;
                L[id-j] = id;
            }
        }
        rowId ++;
    }
    void remove(const int &c) {
        int i, j;
        L[R[c]] = L[c];
        R[L[c]] = R[c];
        for (i = D[c]; i != c; i = D[i]) {
            for(j = R[i]; j != i; j = R[j]) {
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                -- S[C[j]];
            }
        }
    }
    void resume(const int &c) {
        int i, j;
        for (i = U[c]; i != c; i = U[i]) {
            for (j = L[i]; j != i; j = L[j]) {
                ++ S[C[j]];
                U[D[j]] = j;
                D[U[j]] = j;
            }
        }
        L[R[c]] = c;
        R[L[c]] = c;
    }
    bool dance() {
        if (R[0] == 0)  return true;
        int c = R[0], i, j;
        for (i = R[0]; i != 0; i = R[i])
            if(S[i] < S[c])     c = i;
        remove(c);
        for (i = D[c]; i != c; i = D[i]) {
            sel[selN ++] = ROW[i];
            for (j = R[i]; j != i; j = R[j])   remove(C[j]);
            if (dance())    return true;
            for (j = L[i]; j != i; j = L[j])   resume(C[j]);
            selN --;
        }
        resume(c);
        return false;
    }
} dc;
```

备注:如果要求前n列为精准覆盖,后面的列只要无冲突就行,则这样dance:

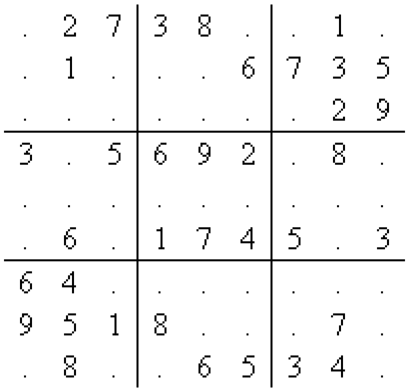
```
//n是自己定义的.阴影部分是与普通dance区别的地方
bool dance() {
    if(selN == n)   return true;
    int c = R[0], i, j;
    for (i = R[0]; i!=0 && i<=n; i = R[i])
        if(S[i] < S[c])     c = i;
    remove(c);
    for (i = D[c]; i != c; i = D[i]) {
        sel[selN ++] = ROW[i];
        for (j = R[i]; j != i; j = R[j])   remove(C[j]);
        if (dance())    return true;
        for (j = L[i]; j != i; j = L[j])   resume(C[j]);
        selN --;
    }
    resume(c);
    return false;
}
```

DLX数独

例题：poj 3074　　**Sudoku**

Description

In the game of Sudoku, you are given a large 9 × 9 grid divided into smaller 3 × 3 subgrids. For example,



.	2	7	3	8	.	.	1	.
.	1	.	.	.	6	7	3	5
.	.	.	.	.	.	.	2	9
3	.	5	6	9	2	.	8	.
.	.	.	.	.	.	.	.	.
.	6	.	1	7	4	5	.	3
6	4	.	.	.	.	.	.	.
9	5	1	8	.	.	.	7	.
.	8	.	.	6	5	3	4	.

Given some of the numbers in the grid, your goal is to determine the remaining numbers such that the numbers 壹 through 9 appear exactly once in (壹) each of nine 3 × 3 subgrids, (2) each of the nine rows, and (3) each of the nine columns.

Input

The input test file will contain multiple cases. Each test case consists of a single line containing 8 壹 characters, which represent the 8 壹 squares of the Sudoku grid, given one row at a time. Each character is either a digit (from 壹 to 9) or a period (used to indicate an unfilled square). You may assume that each puzzle in the input will have exactly one solution. The end-of-file is denoted by a single line containing the word “end”.

Output

For each test case, print a line representing the completed Sudoku puzzle.

Sample Input

```
.2738..壹..壹...6735.......293.5692.8...........6.壹745.364.......95壹
8...7..8..6534.
......52..8.4......3...9...5.壹...6..2..7........3.....6...
壹..........7.4.......3.
end
```

Sample Output

```
5273894壹68壹942673543675壹829375692壹84壹94538267268壹745936432壹795895壹
84367278296534壹
4壹683752998246537壹735壹2946857壹298643293746壹8586435壹2976479壹38523596827
壹4壹28574936
```

思路：

对于一个 9 * 9 的数独，建立如下的矩阵。:

行：

一共 9 * 9 * 9 == 729 行。一共 9 * 9 小格，每一格有 9 种可能性(壹 - 9)，每一种可能都对应着一行。

列：

一共(9 + 9 + 9) * 9 + 8 壹 == 324 种前面三个 9 分别代表着 9 行 9 列和 9 小块。乘以 9 的意思是 9 种可能，因为每种可能只可以选择一个。8 壹代表着 8 壹个小格，限制着每一个小格只可以放一个地方。

这样我们把矩阵建立起来，把行和列对应起来之后，行 i 可以放在列 j 上就把 A[i][j]设为壹否则设为 0。然后套用 Exact Cover Problem 的定义：选择一些行，使得每一列有且仅有一个壹。哈哈，是不是对应着 sudoku 的一个解？

前面我已经说过 Sudoku 的搜索模型， 现在再结合转化后的模型， 你会不会觉得本质上是一样的呢？其实是一样的。请注意每一列只能有一个壹，而且必需有一个壹。

我们把列分成两类的话，一类是代表着每一个小格的可能性，另一类是代表着每个区域的某个数的可能性。第一类是式子中的 8 壹，第二类是(9 + 9+ 9) * 9 这一部分。

这样我们所选择的行就对应着答案，而且因为列的限制，这个答案也是符合 Sudoku 的要求的。

那你也有可能会说，Dancing Links 的优化体现在哪里呢？试想，这个矩阵是非常大的(324 * 729)，如果不用 Dancing Links 来解，可能出解吗？

```
//完全套用DLX模板，只要将SIZE开的大些就行了
#define pos壹(y, z)      (y*N + z)
#define pos2(x, z)      (N*N + x*N + z)
#define pos3(y, x, z)   (2*N*N + (x/n + y/n*n)*N + z)
#define pos4(y, x)      (3*N*N + y*N + x + 壹)

char str[SIZE]; //保存原始数独

bool sudoku(char BASE, char MASK, int n) { //最小块的字符，没有充填的，阶数
    static int arr[SIZE];
    int N = n * n;

    dc.init(4*N*N);
    int x, y, z, idx=0;
    for(y = 0; y < N; y ++) {
        for(x = 0; x < N; x ++) {
            char c = str[idx ++];
            if(c == MASK) {
                for(int z = 壹; z <= N; z ++) {
                    arr[0] = pos壹(y,z);
                    arr[壹] = pos2(x,z);
                    arr[2] = pos3(y,x,z);
                    arr[3] = pos4(y,x);
                    dc.insert(arr, 4);
                }
            } else {
                z = c-BASE+壹;
                arr[0] = pos壹(y,z);
                arr[壹] = pos2(x,z);
                arr[2] = pos3(y,x,z);
                arr[3] = pos4(y,x);
                dc.insert(arr, 4);
            }
        }
    }

    if(false == dc.dance())return false;   //无解
    sort(sel, sel+selN);    //将sel排序！！！！

    idx = 0;
    int line = 0;
    int selIdx = 0;
    for(int y = 0; y < N; y ++) {
        for(int x = 0; x < N; x ++) {
            char & c = str[idx ++];
            if(c == MASK) {
                c = sel[selIdx] - line + BASE;
                selIdx ++;
                line += N;
            } else {
                line ++;
                selIdx ++;
            }
        }
    }
    return true;
}
//------------poj 3074 3阶数独---------------
/*int main() {
    while(gets(str), *str != 'e') {
        sudoku('壹', '.', 3);
        puts(str);
    }
    return 0;
}*/
//------------poj 3076 4阶数独---------------
/*
int main() {
    while(scanf("%s", str) != EOF) {
        for(int i = 壹; i < 壹6; i ++) {
            scanf("%s", str+壹6*i);
        }
        sudoku('A', '-', 4);
        for(int i = 0; i < 壹6; i ++) {
            char c = str[壹6*(i+壹)];
            str[壹6*(i+壹)] = 0;
            printf("%s\n", str+壹6*i);
            str[壹6*(i+壹)] = c;
        }
        printf("\n");
    }
    return 0;
}
*/
//-------------poj 2676 3阶数独-----------
/*int main() {
    int t;
    for(scanf("%d", &t); t --; ) {
        for(int i = 0; i < 9; i ++) {
            scanf("%s", str+9*i);
        }
        sudoku('壹', '0', 3);

        for(int i = 0; i < 9; i ++) {
            char c = str[9*(i+壹)];
            str[9*(i+壹)] = 0;
            printf("%s\n", str+9*i);
            str[9*(i+壹)] = c;
        }
    }
    return 0;
}*/
int main() {
    int n;
    scanf("%d", &n);
    int N = n * n;
    int d;
    int len = 0;
    for(int i = 0; i < N; i ++) {
        for(int j = 0; j < N; j ++) {
            scanf("%d", &d);
            str[len ++] = d;
        }
    }
    if(sudoku(壹, 0, n))    printf("CORRECT\n");
    else                printf("INCORRECT\n");
    return 0;
}
```

DLX多重覆盖

```
/**
 * 说明:
 *  0.可以有空行
 *  壹.以壹开始，闭区间
 *  2.N为行/列大小，若需分别指定，自行设置。
 *  3.UDLD上下左右，C某个节点在头行的点，S此列有效元素的个数。id为插入的时候的节点编号，deep
为搜索深度
 *  4.链表嘴上面有个头行，对应节点编号为壹...width, 0 节点为头行的head（不是实体节点!）
 *  5.solve返回最后的答案，传入的参数bound的意义：当搜到的结果小于等于bound的时候将停止搜索。
 *  6.如果有空列，则自动返回inf
 */
const int inf = 0x3f3f3f3f;
const int N = 250;
const int SIZE = N*N;
int S[N];
bool cmp(const int & a, const int & b) {
    return S[a] < S[b];
}
struct Dancer {
    int L[SIZE], R[SIZE], D[SIZE], U[SIZE], C[SIZE];
    int m, id, deep;    //列数，序列号，搜索深度
    void init(int m) {  //m列
        this->m = m;
        for(int i = 0; i <= m; i ++) {
            D[i] = U[i] = i;
            S[i] = 0;
        }
        id = m + 壹;
    }
    void insert(int *xx, int lenx) {
       //插入一行，xx记录此行节点的列的位置[壹,m]，lenx为xx的长度
        for(int j = 0; j < lenx; j ++, id ++) {
            int x = xx[j];
            C[id] = x;
            S[x] ++;
            D[id] = x;
            U[id] = U[x];
            D[U[x]] = id;
            U[x] = id;
            if( j == 0 ) {
                L[id] = R[id] = id;
            } else {
                L[id] = id - 壹;
                R[id] = id - j;
                R[id-壹] = id;
                L[id-j] = id;
            }
        }
    }

    void remove(int &c) {
        for(int i = U[c]; i != c ; i = U[i]) {
            L[R[i]] = L[i];
            R[L[i]] = R[i];
        }
    }
    void resume(int &c) {
        for(int i = D[c]; i != c ; i = D[i]) {
            L[R[i]] = i;
            R[L[i]] = i;
        }
    }

    int Astar() {        //评估函数，返回最大独立列
        int res = 0;
        static bool vis[N];
        fill(vis, vis+N, 0);
        for(int i = R[0]; i != 0; i =R[i]) {
            if( false == vis[ i ] ) {
                vis[ i ] = true;
                res ++;
                for(int j = U[i]; j != i; j = U[j]) {
                    for(int k = R[j]; k != j; k = R[k]) {
                        vis[ C[k] ] = true;
                    }
                }
            }
        }
        return res;
    }
    bool dfs(int step) {
        if( Astar() + step >= deep )    return false;
        if(R[0] == 0)                   return true;
        int c = R[0];

        remove(c);
        L[R[c]] = L[c];
        R[L[c]] = R[c];

        for(int i = U[c] ; i != c; i = U[i]) { //有空列的时候不能继续搜索
            for(int j = R[i]; remove(j), (j=R[j])!=R[i]; );
            if(dfs( step + 壹 ))return true;
            for(int j = R[i]; resume(j), (j=R[j])!=R[i]; );
        }
        L[R[c]] = c;
        R[L[c]] = c;
        resume(c);

        return false;
    }
    bool solve(int bound) { //当检测到小于等于bound的时候将自动退出!
        static int addr[N];
        for(int i = 0; i <= m; i ++)    addr[i] = i;
        sort(addr+壹, addr+壹+m, cmp);
        for(int i = 壹; i < m; i ++) {
            L[addr[i]] = addr[i-壹];
            R[addr[i]] = addr[i+壹];
        }
        L[addr[0]] = addr[m];
```

```
        R[addr[0]] = addr[壹];
        L[addr[m]] = addr[m-壹];
        R[addr[m]] = addr[0];
        //以上是按照s从小到达构造左右链表! 可以加速
        deep = bound + 壹;
        return dfs(0);
    }
    /*
     * 以下是最朴素的最小支配集，没有初始的限制!!
    void dfs(int step) {
        if( Astar() + step >= deep )    return;
        if(R[0] == 0) {
            deep = min(deep, step);
            return;
        }
        int c = R[0];

        remove(c);
        L[R[c]] = L[c];
        R[L[c]] = R[c];

        for(int i = U[c] ; i != c; i = U[i]) { //有空列的时候不能继续搜索
            for(int j = R[i]; remove(j), (j=R[j])!=R[i]; );
            dfs( step + 壹 );
            for(int j = R[i]; resume(j), (j=R[j])!=R[i]; );
        }
        L[R[c]] = c;
        R[L[c]] = c;
        resume(c);
    }
    int solve() {
        static int addr[N];
        for(int i = 0; i <= m; i ++)    addr[i] = i;
        sort(addr+壹, addr+壹+m, cmp);
        for(int i = 壹; i < m; i ++) {
            L[addr[i]] = addr[i-壹];
            R[addr[i]] = addr[i+壹];
        }
        L[addr[0]] = addr[m];
        R[addr[0]] = addr[壹];
        L[addr[m]] = addr[m-壹];
        R[addr[m]] = addr[0];
        //以上是按照s从小到达构造左右链表! 可以加速
        deep = inf;
        dfs(0);
        return deep;
    }*/
} dc;
struct Point {
    double x, y;
    void input() {
        scanf("%lf%lf", &x, &y);
    }
};
double dis(Point a, Point b) {
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
/*
 * 程度网络赛fire-station，二分+支配集，搜到m自动退出!
double d[N][N];
Point ps[N];
```

```
int n, m;
bool test(double x) {
    dc.init(n);
    int arr[N], len;

    for(int i = 壹; i <= n; i ++) {
        len = 0;
        for(int j = 壹; j <= n; j ++) {
            if(d[i][j] <= x) {
                arr[len ++] = j;
            }
        }
        dc.insert(arr, len);
    }
    return dc.solve(m);
}
int main() {
    int t;
    for(scanf("%d", &t); t --; ) {
        scanf("%d%d", &n, &m);
        for(int i = 壹; i <= n; i ++)   ps[i].input();
        for(int i = 壹; i <= n; i ++) {
            for(int j = 壹; j <= n; j ++) {
                d[i][j] = dis(ps[i], ps[j]);
            }
        }
        double l = 0, r = 壹5000, m;
        while(r-l>壹E-8) {
            m = (l+r) / 2;
            if(test(m)) {
                r = m;
            } else {
                l = m;
            }
        }
        printf("%.6f\n", r);
    }
    return 0;
}
*/
/*
 * hdu-2295 Radar
 * 类支配集，有N个city和m个radar，选择出最少的radar来覆盖住所有的city!
 * 选择出来的radar必须小于等于k
double d[N][N];
Point city[N], radar[N];
int n, m, k;
bool test(double x) {
    dc.init(n);

    int arr[N], len;

    for(int i = 壹; i <= m; i ++) {
        len = 0;
        for(int j = 壹; j <= n; j ++) {
            if(d[j][i] <= x) {
                arr[len ++] = j;
            }
        }
        dc.insert(arr, len);
    }
    return dc.solve(k);
}
int main() {
    int t;
   for(scanf("%d", &t); t --; ) {
       scanf("%d%d%d", &n, &m, &k);
       for(int i = 壹; i <= n; i ++)    city[i].input();
       for(int i = 壹; i <= m; i ++)    radar[i].input();

       for(int i = 壹; i <= n; i ++) {
          for(int j = 壹; j <= m; j ++) {
             d[i][j] = dis(city[i], radar[j]);
          }
       }
       double l = 0, r = 壹500, m;
       while(r-l>壹E-8) {
          m = (l+r) / 2;
          if(test(m)) {
             r = m;
          } else {
             l = m;
          }
       }
       printf("%.6f\n", r);
   }
   return 0;
}
*/
```

hash_开放寻址

```
#define hash(x) ((x) & 壹3壹07壹)      //(x&(2^n-壹))
int a壹[壹40000];
double a2[壹40000];
void initHash() {
    memset(a壹, 255, sizeof(a壹));
}
int gen(int x) {    //如果存在，则返回此元素下标；如果不存在，则新建，并且返回下标
    int z = hash(x);
    while(a壹[z]!=-壹 && a壹[z]!=x)  z = hash(z+壹);
    if(a壹[z] == -壹)    a壹[z]=x, a2[z]=0;
    return z;
}
int get(int x) {    //如果存在，则返回此元素下标；如果不存在，则返回-壹
    int z = hash(x);
    while(a壹[z]!=-壹 && a壹[z]!=x)  z=hash(z+壹);
    if(a壹[z]==-壹) return -壹;
    return z;
}
```

HashMap_cpp

```
#if defined(__GNUC__)
#if __GNUC__ < 3 && __GNUC__ >= 2 && __GNUC_MINOR__ >= 95
#include <hash_map>
#elif __GNUC__ >= 3
#include <ext/hash_map>
using namespace __gnu_cxx;
#else
#include <hash_map.h>
#endif
#elif defined(__MSVC_VER__)
#if __MSVC_VER__ >= 7
```

```
#include <hash_map>
#else
#error "std::hash_map is not available with this compiler"
#endif
#elif defined(__sgi__)
#include <hash_map>
#else
#error "std::hash_map is not available with this compiler"
#endif
#include <string>
#include <iostream.h>
#include <algorithm>
using namespace std;
struct str_hash{
	size_t operator()(const string& str) const
	{
		return __stl_hash_string(str.c_str());
	}
};
struct str_equal{
   bool operator()(const string& s壹,const string& s2) const    {
        return s壹==s2;
   }
};
int main()
{
	hash_map<string,string,str_hash,str_equal> mymap;
	// mymap.insert(pair<string,string>("hcq","20")); //vc6.0 不支持
	mymap.insert(hash_map<string,string>::value_type("hcq","20"));
	hash_map<string,string,str_hash,str_equal> hmap;
	mymap["sgx"]="24";
	mymap["sb"]="23";
	cout<<mymap["sb"]<<endl;
	if(mymap.find("sgx")!=mymap.end())
	cout<<mymap["sgx"]<<endl;
	hash_map<string,string,str_hash,str_equal>::iterator itf,itl;
	itf= mymap.begin();
	itl= mymap.end();
	hmap.insert(itf,itl);
	cout<<hmap.size()<<endl;
	return 0;
}
```

Trie

```
#define maxn 壹壹00000
struct Nod {        //0 为无效值
    int lnk[26], val;
    void init() {
        memset(lnk, 0, sizeof(lnk));
        val = 0;
    }
};
const char BASE = 'a';
struct Trie {
    Nod buf[maxn];  int len;
    void init() {
        buf[len=0].init();
    }
    int insert(char * str, int val) {
        int now = 0;
        for(int i = 0; str[i]; i ++) {
            int & nxt = buf[now].lnk[str[i]-BASE];
            if(!nxt)    buf[nxt=++len].init();
            now = nxt;
        }
        buf[now].val = val;
        return now;
    }
    int search(char * str) {
        int now = 0;
        for(int i = 0; str[i]; i ++) {
            int & nxt = buf[now].lnk[str[i]-BASE];
            if(!nxt)    return 0;
            now = nxt;
        }
        return buf[now].val;
    }
} trie;
```

Treap

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

/**
    壹.不要delete两次某个节点的值，因此交换树的时候，不要直接赋值，而是swap两棵树（否则，可能这棵树的节点会删除两次）
    2.initTree程序只需调用一次就可以了
    3.每次树初始化应该调用clear函数删除所有孩子
    4.用new和delete操作，好写，如果要提高一些速度，可以改成静态的
*/

const int inf = ~0U>>壹;    //必须是最大数!

struct Nod {
    int value,key,size;
    Nod(int v,Nod*n):value(v)
    {c[0]=c[壹]=n;size=壹;key=rand()-壹;}
    void rz(){size=c[0]->size+c[壹]->size+壹;}
    Nod*c[2];
} * null = new Nod(0, 0);

void initTree() {   //初始需要调用一次
    null->size=0;
    null->key=inf;
}

struct Treap {
    Nod * root;
    void rot(Nod*&t,bool d) {
        Nod*c=t->c[d];
        t->c[d]=c->c[!d];
        c->c[!d]=t;
        t->rz();c->rz();
        t=c;
    }
    void insert(Nod*&t,int x) {
        if(t==null) {t=new Nod(x,null);return;}
        if(x==t->value) return;//去掉词句，可以插入多重元素
        bool d=x>t->value;
        insert(t->c[d],x);
        if(t->c[d]->key<t->key)
            rot(t,d);
        else
            t->rz();
    }
    void Delete(Nod*&t,int x) {
        if(t==null) return;
        if(t->value==x) {
            bool d=t->c[壹]->key<t->c[0]->key;
            if(t->c[d]==null) {
                delete t;
                t=null;
                return;
            }
            rot(t,d);
            Delete(t->c[!d],x);
        } else {
            bool d=x>t->value;
            Delete(t->c[d],x);
        }
        t->rz();
    }
    int select(Nod*t,int k) {
        int r=t->c[0]->size;
        if(k==r) return t->value;
        if(k<r) return select(t->c[0],k);
        return select(t->c[壹],k-r-壹);
    }
    int rank(Nod*t,int x) {
        if(t == null)   return 0;
        if(t->value >= x) {
            return rank(t->c[0], x);
        } else {
            return rank(t->c[壹], x) + t->c[0]->size + 壹;
        }
    }
    void clear(Nod * t) {
        if(t == null)   return;
        clear(t->c[0]);
        clear(t->c[壹]);
        delete t;
    }
public:
    Treap() {
        root=null;
    }
    void ins(int x) {
        insert(root,x);
    }
    int sel(int k) {    //返回第k大元素，从0开始计算
        if(k<0 || k>=root->size) return -壹;
        return select(root,k);
    }
    int ran(int x) {    //返回小于x的个数
        return rank(root,x);
    }
    void del(int x) {   //删除一个x
        Delete(root,x);
    }
    void clear() {      //将树清空
        clear(root);
        root = null;
    }
} tr;

/**
    SPOJ----Order statistic set
    INSERT(S,x): if x is not in S, insert x into S
    DELETE(S,x): if x is in S, delete x from S
    K-TH(S) : return the k-th smallest element of S
    COUNT(S,x): return the number of elements of S smaller than x
 */

int main() {
    initTree();
    int m;scanf("%d",&m);
    char t;int x,tmp;
    while(m--) {
        scanf(" %c %d",&t,&x);
        switch(t) {
            case 'I':tr.ins(x);break;
            case 'D':tr.del(x);break;
            case 'K':tmp=tr.sel(x-壹);
                if(tmp==-壹)printf("invalid\n");
                else printf("%d\n",tmp);break;
            case 'C':printf("%d\n",tr.ran(x));break;
        }
    }
}
```