**[线段树和树状数组题目总结（未完）](http://www.cnblogs.com/DiaoCow/archive/2010/05/08/1730469.html)**

1.Minimum Inversion Number

首先要知道如何求一个序列的逆序数，比如2514 ：

2之前没有数大于它，所以为0，

5之前也没有数大于它，所以为0，

1之前2,5都大于它，所以为2，

4之前只有5大于它，所以为1， 因此2514的逆序数为：0  + 0 + 2 + 1 = 3;

从前面的描述中，我们可以发现，只要依次累计前面有几个数大于当前数即可。于是我们可以用一个数组b[n]（初始化为0），然后每读取一个数a[i] 就标识b[a[i]] += 1 , 但是在标识之前我们先要统计有多少个数大于a[i],即累计b[a[i]+1] + ... +b[n-1]，很显然累计的时间复杂度为O(N)，我们能不能在快点呢? **当然，树状数组就可以，它求和的效率是log级别的。**

 那么如何求最小逆序数呢？（我这里假设一个序列中每个数字都不同）

若abcde...的逆序数为k，那么bcde...a的逆序数是多少？我们假设abcde...中小于a的个数为t-1 , 那么大于a的个数就是n - t，当把a移动左移一位时，原来比a大的现在都成了a的逆序对，即逆序数增加n - t,但是原来比a小的构成逆序对的数，现在都变成了顺序，因此逆序对减少 t - 1,所以新序列的逆序数为 k += n - t - t + 1，即k += n + 1 - 2 \* t , 于是我们只要不断移位（n次） 然后更新最小值就可以了

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#include <stdio.h>   
#include <string.h>  
  
#define N 5000   
  
int a[N+1] , b[N+1];   
  
int Lowbit(int x)  
{  
 return x & (-x);  
}  
  
void Update(int x, int c)   
{  
 int i;  
  
 for(i = x; i < N ; i += Lowbit(i))  
 {  
 b[i] += c;  
 }  
}  
  
int Getsum(int x)   
{  
 int i , k = 0;  
  
 for(i = x; i >= 1 ; i -= Lowbit(i))  
 {  
 k += b[i];  
 }  
 return k;  
}  
  
  
int main(void)  
{  
 int n , i , k , min;  
  
 while(scanf("%d",&n) != EOF)  
 {  
 for(i = 0 ; i < n ; i++)  
 {  
 scanf("%d", a + i);  
 a[i]++;  
 }  
  
 k = 0;  
 memset(b ,0 ,sizeof(b));  
 for(i = 0 ; i < n ; i++)  
 {  
 k += i - Getsum(a[i]); // 总数 - 小于(等于)a[i]的个数 = 大于a[i]的个数  
 Update(a[i],1);  
 }  
  
 min = k;  
 for(i = 0 ; i < n ; i++)  
 {  
 //Update min  
 k += 1 + n - 2 \* Getsum(a[i]);  
 min = min < k ? min : k ;  
 }  
 printf("%d\n",min);  
 }  
 return 0;  
}

2.Stars , Cows

Stars这题就是求一颗星星的左下方有多少星星。若我们把所有星星按照y坐标从小到大排序（若y坐标相同，按照x坐标从小到大排序），那么对于每一颗星星而言排在前面的就是他下方的星星或者说是左方的，**因此只要统计前面有多少星星的x坐标小于等于当前坐标即可，这显然是树状数组的看家本领。**

由于题目中已经排好序了，所以只要简单的统计更新即可

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#include <stdio.h>  
#include <string.h>  
#define N 32002   
  
int l[N] , b[N];   
  
int Lowbit(int x)  
{   
 return x & (-x);  
}  
  
void Update(int x, int c)  
{   
 int i;   
 for(i = x; i < N ; i += Lowbit(i))   
 {   
 b[i] += c;   
 }  
}  
  
int Getsum(int x)   
{   
 int i , k = 0;   
 for(i = x; i >= 1 ; i -= Lowbit(i))   
 {   
 k += b[i];   
 }   
 return k;  
}  
  
int main(void)  
{  
 int x , y , n , i;  
  
 while(scanf("%d",&n) != EOF)  
 {  
 memset(b,0,sizeof(b));  
 for(i = 0 ; i < n ; i++)  
 {  
 scanf("%d%d",&x ,&y);  
 l[Getsum(x+1)]++; //加1都是为了保证最小值为1  
 Update(x+1,1);  
 }  
 for(i = 0 ; i < n ; i++)  
 printf("%d\n",l[i]);  
 }  
 return 0;  
}

Cows这题和Stars几乎一模一样，只是排序工作需要自己做（按照e从大到小排序,若e相等按s从小到大排序），要特别注意区间重合情况

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#include <stdio.h>  
#include <string.h>  
#define N 100002   
  
typedef struct   
{  
 int s , e;  
 int ori;  
}Node;  
Node a[N];  
  
int b[N] , c[N];   
  
int Lowbit(int x)  
{   
 return x & (-x);  
}  
  
void Update(int x, int c)  
{   
 int i;   
 for(i = x; i < N ; i += Lowbit(i))   
 {   
 b[i] += c;   
 }  
}  
  
int Getsum(int x)   
{   
 int i , k = 0;   
 for(i = x; i >= 1 ; i -= Lowbit(i))   
 {   
 k += b[i];   
 }   
 return k;  
}  
  
//按照e从大到小排序,若e相等按s从小到大排序  
void QuickSort(Node \*arr , int left , int right)  
{  
 int i , j;  
 Node x , nTemp;  
  
 if(left >= right) //边界条件检查  
 return;  
 else  
 {   
 //Partition  
 i = left; j = right + 1; x = arr[i];  
 while(1)  
 {  
 do i++; while(i < j && arr[i].e > x.e || (arr[i].e == x.e && arr[i].s < x.s));  
 do j--; while(arr[j].e < x.e || (arr[j].e == x.e && arr[j].s > x.s));  
 if(i > j) break;  
 //swap(i,j)  
 nTemp = arr[i]; arr[i] = arr[j]; arr[j] = nTemp;  
 }  
 //swap(left,j)  
 nTemp = arr[left]; arr[left] = arr[j]; arr[j] = nTemp;  
  
 QuickSort(arr,left,j-1);  
 QuickSort(arr,j+1,right);  
 }  
}  
  
int main(void)  
{  
 int n , i;  
  
 while(scanf("%d",&n) && n) //一开始这里写的是!= EOF wa了好多次  
 {  
 memset(b , 0 , sizeof(b));  
 for(i = 0 ; i < n ; i++)  
 {  
 scanf("%d%d",&a[i].s ,&a[i].e);  
 a[i].e++; a[i].s++;  
 a[i].ori = i;  
 }  
  
 QuickSort(a , 0 , n - 1);  
   
 for(i = 0 ; i < n ; i++)  
 {  
 if(i && a[i].e == a[i-1].e && a[i].s == a[i-1].s) //区间重合  
 {  
 c[a[i].ori] = c[a[i-1].ori];  
 }  
 else  
 c[a[i].ori] = Getsum(a[i].s);   
 Update(a[i].s,1);  
 }  
  
 printf("%d",c[0]);  
 for(i = 1 ; i < n ; i++)  
 printf(" %d",c[i]);  
 printf("\n");  
 }  
 return 0;  
}

4.Mayor's posters

这题其实就是**线段染色问题**：每条线段都用某一种颜色绘画，且后画的线段会覆盖之前画的，求最后颜色数  
这里重新定义cover：  
cover = 0 ; //说明线段还没有被画  
cover = c ; //说明线段被覆盖且颜色为c  
cover = -1; //说明这条线段包含多种颜色

还有一个问题就是此题数据量比较大（10000000），因此直接建立线段树是不可能的，所以要先离散化。

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#include <stdio.h>  
#include <string.h>  
#define N 20002  
  
//线段树  
typedef struct  
{  
 int l , r;  
 int cover;  
}TreeNode;  
TreeNode seg\_tree[4\*N];  
  
//线段  
struct   
{  
 int l , r;  
}seg[N/2];  
  
int arr[N] , hash[10000001] , nFlag[N/2];  
  
void CreateSegTree(int p , int a , int b)  
{  
 int m;  
  
 seg\_tree[p].l = a;  
 seg\_tree[p].r = b;  
 seg\_tree[p].cover = 0;  
   
 if(a != b)  
 {  
 m = (a + b) >> 1;  
 CreateSegTree(p << 1 , a , m);  
 CreateSegTree((p << 1) + 1 , m + 1 , b);  
 }  
}  
  
void InsertSegTree(int p , int a , int b , int c)  
{  
 int m , k;  
  
 if(seg\_tree[p].l == a && seg\_tree[p].r == b)  
 {  
 seg\_tree[p].cover = c;  
 return ;  
 }  
  
 if(seg\_tree[p].cover > 0) //已经被覆盖  
 {  
 k = seg\_tree[p].cover;  
 seg\_tree[p].cover = -1;  
 seg\_tree[2\*p].cover =seg\_tree[2\*p+1].cover = k;  
 }  
  
 m = (seg\_tree[p].l + seg\_tree[p].r) >> 1;  
 if(b <= m)  
 {  
 InsertSegTree(p << 1 , a , b , c);  
 }  
 else if(a > m)  
 {  
 InsertSegTree((p << 1) + 1 , a , b , c);  
 }  
 else  
 {  
 InsertSegTree(p << 1 , a , m , c);  
 InsertSegTree((p << 1) + 1 , m + 1 , b , c);  
 }  
}  
  
  
int Count(int p)  
{  
 int k;  
  
 if((k = seg\_tree[p].cover) > 0)  
 {  
 if(!nFlag[k])  
 {  
 nFlag[k] = 1;  
 return 1;  
 }  
 return 0;  
 }  
 if(seg\_tree[p].l == seg\_tree[p].r)  
 {  
 return 0;  
 }  
 return Count(p << 1) + Count((p << 1) + 1);  
}  
  
//快排  
void QuickSort(int \*arr , int left , int right)  
{  
 int i , j , x , nTemp;  
  
 if(left >= right) //边界条件检查  
 return;  
 else  
 {   
 //Partition  
 i = left; j = right + 1; x = arr[i];  
 while(1)  
 {  
 do i++; while(i < j && arr[i] < x);  
 do j--; while(arr[j] > x);  
 if(i > j) break;  
 //swap(i,j)  
 nTemp = arr[i]; arr[i] = arr[j]; arr[j] = nTemp;  
 }  
 //swap(left,j)  
 nTemp = arr[left]; arr[left] = arr[j]; arr[j] = nTemp;  
  
 QuickSort(arr,left,j-1);  
 QuickSort(arr,j+1,right);  
 }  
}  
  
int main(void)  
{  
 int i , n , c , z , k;  
  
 scanf("%d", &z);  
 while(z-- > 0)  
 {  
 scanf("%d", &n);  
 for(i = 0 ; i < n ; i++)  
 {  
 scanf("%d%d", &seg[i].l , &seg[i].r);  
 arr[2\*i] = seg[i].l;  
 arr[2\*i+1] = seg[i].r;  
 nFlag[i+1] = 0;  
 }  
  
 //离散化(相同点离散化成同一值)  
 QuickSort(arr , 0 , 2 \* n - 1);  
 k = 1; hash[arr[0]] = 1;  
 for(i = 1 ; i < 2 \* n ; i++)  
 {  
 if(arr[i] != arr[i-1]) k++;  
 hash[arr[i]] = k;  
 }  
  
 CreateSegTree(1 , 1 , k);  
 for(i = 0 , c = 1 ; i < n ; i++ , c++)  
 {  
 InsertSegTree(1 , hash[seg[i].l] , hash[seg[i].r] , c);  
 }  
  
 printf("%d\n", Count(1));  
 }  
 return 0;  
}

5.I Hate It  
这道题是基本的线段树题目，所以比较容易，只要注意两点即可：  
1).(根据题目要求)线段树的叶子结点不再是[a,a+1],而是[a,a].因此我们建树的时候应该[a,m],[m+1,b]二分，而不是[a,m],[m,b]二分;  
2).递归返回的时候要更新父节点(seg\_tree[p].score = Max(seg\_tree[2\*p].score , seg\_tree[2\*p+1].score);)

我的虽然AC了，但耗时比较长，看到前面很多人100多MS就AC了，不知道是哪里优化的

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#include <stdio.h>  
#define N 200001  
#define Max(a,b) ((a) > (b) ? (a) : (b))  
  
typedef struct  
{  
 int score;   
}TreeNode;  
  
TreeNode seg\_tree[3 \* N];  
int s[N];  
  
void CreateSegTree(int p , int l , int r)   
{  
 int m;  
  
 if(l == r)  
 {  
 seg\_tree[p].score = s[l];  
 return;  
 }  
  
 m = (l + r) / 2;   
 CreateSegTree(2 \* p , l , m);   
 CreateSegTree(2 \* p + 1 , m + 1 , r);  
 //update root  
 seg\_tree[p].score = Max(seg\_tree[2\*p].score , seg\_tree[2\*p+1].score);  
}  
  
void ModifySegTree(int p , int l , int r , int a , int k) //a为要修改的“点”  
{  
 int m;  
  
 if(l == r && l == a)  
 {  
 seg\_tree[p].score = k;  
 return ;  
 }  
   
 m = (l + r) / 2;  
 if(a <= m)   
 ModifySegTree(2 \* p , l , m , a , k);  
 else   
 ModifySegTree(2 \* p + 1 , m + 1 , r , a , k);  
 //update root  
 seg\_tree[p].score = Max(seg\_tree[2\*p].score , seg\_tree[2\*p+1].score);   
}  
  
  
int SearchSegTree(int p , int l , int r , int a , int b) //(a,b)为所要查询的区间  
{  
 int i , j , m;  
  
 if(l == a && r == b)  
 {  
 return seg\_tree[p].score;  
 }  
  
 m = (l + r) / 2;  
 if(b <= m)   
 {  
 return SearchSegTree(2 \* p , l , m , a , b);  
 }  
 else if(a > m)   
 {  
 return SearchSegTree(2 \* p + 1 , m + 1, r , a , b);  
 }  
 else   
 {  
 i = SearchSegTree(2 \* p , l , m , a , m);  
 j = SearchSegTree(2 \* p + 1 , m + 1 , r , m + 1 , b);  
 return Max(i , j);  
 }  
}  
  
  
int main(void)  
{  
 int i , j , k , n , m;  
 char c;  
  
 while(scanf("%d%d", &n ,&m) != EOF)  
 {  
 for(i = 1 ; i <= n ; i++)  
 {  
 scanf("%d", s + i);  
 }  
  
 CreateSegTree(1 , 1 , n);  
  
 for(k = 1 ; k <= m ; k++)  
 {  
 scanf(" %c%d%d",&c ,&i ,&j);  
 if(c == 'Q')  
 {  
 printf("%d\n", SearchSegTree(1 , 1 , n , i , j));  
 }  
 else  
 {  
 ModifySegTree(1 , 1 , n , i , j);  
 }  
 }  
 }  
 return 0;  
}