**[总结下最近做的些题目](http://www.cnblogs.com/DiaoCow/archive/2010/05/19/1739618.html)**

**动态规划**

1.Max Sequence ，Maximum sum

这两题都是求最大m子段和（m = 2）

最大m子段和(最大子段和在子段个数上的推广)：设b[i][j]表示从前j项中i个子段和的最大值，且第i个子段和包含a[j];

因此状态转移方程为：b[i][j] = Max(b[i][j-1] + a[j] , b[i-1][t] + a[j]);   (  i-1 <= t < j)

b[i][j-1] + a[j]表示a[j]并入第i个子段，b[i-1][t] + a[j]表示a[j]独立为第i个子段

Max Sequence:

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

#include <stdio.h>  
#define Max(a,b) ((a) > (b) ? (a) : (b))  
#define N 100001  
  
int b[3][N] , a[N] , c[3][N] , n;  
  
int MaxSum()  
{   
 int i , j , k , max;   
   
 for(i = 0 ; i <= 2 ; i++) b[i][0] = c[i][0] = 0;   
 for(j = 0 ; j <= n ; j++) b[0][j] = c[0][j] = 0;   
   
 for(i = 1 ; i <= 2 ; i++)   
 {   
 //i == j   
 b[i][i] = b[i-1][i-1] + a[i];   
 c[i][i] = b[i][i];   
 for(j = i + 1; j <= n ; j++)   
 {   
 if(b[i][j-1] > c[i-1][j-1]) //并入   
 {   
 b[i][j] = b[i][j-1] + a[j];   
 }   
 else //独立   
 {   
 b[i][j] = c[i-1][j-1] + a[j];   
 }   
 c[i][j] = Max(c[i][j-1] , b[i][j]);   
 }   
 }   
 return c[2][n];  
}  
  
int main(void)  
{   
 int i;   
   
 while(scanf("%d", &n) && n)   
 {   
 for(i = 1 ; i <= n ; i++)   
 {   
 scanf("%d", a + i);   
 }   
 printf("%d\n", MaxSum());   
 }   
 return 0;  
}

优化下空间复杂度：

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

#include <stdio.h>  
#define N 100001  
#define Max(a,b) ((a) > (b) ? (a) : (b))  
  
int b[N] , c[2][N] , a[N] , n;  
  
int MaxSum(int m)  
{  
 int i , j , p , sum;  
  
 for(j = 0 ; j <= n ; j++) c[0][j] = 0;  
 b[0] = 0;  
   
 for(i = p = 1 ; i <= m ; i++ , p = 1 ^ p)  
 {  
 c[p][i] = b[i] = b[i-1] + a[i];  
 for(j = i + 1 ; j <= n ; j++)  
 {  
 b[j] = b[j-1] > c[1-p][j-1] ? b[j-1] + a[j] : c[1-p][j-1] + a[j];  
 c[p][j] = Max(b[j] , c[p][j-1]);  
 }  
 }  
  
 sum = -2002;  
 return Max(sum , c[1-p][n]);  
}  
   
int main(void)  
{   
 int i;   
   
 while(scanf("%d", &n) && n)   
 {   
 for(i = 1 ; i <= n ; i++)   
 {   
 scanf("%d", a + i);   
 }   
 printf("%d\n", MaxSum(2));   
 }   
 return 0;  
}

后来我上网看了看别人的解法，发现很巧妙（针对m =2)也就是找分割点。因此从前往后求一次最大子段和（记录以a[i]为结尾的最大子段和），然后从后往前求一次，不断更新最大值即可

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

#include <stdio.h>  
#define N 100001  
  
int a[N] , m[N] , n;  
  
int main(void)  
{  
 int i , b , max;  
  
 while(scanf("%d", &n) && n)  
 {  
 //正向DP  
 scanf("%d" , a);  
 max = b = m[0] = a[0];  
 for(i = 1 ; i < n ; i++)  
 {  
 scanf("%d" , a + i);  
 b > 0 ? b += a[i] : b = a[i];  
 if(b > max) max = b;  
 m[i] = max;  
 }  
  
 b = -1001; max = -2002;  
 for(i = n - 1 ; i > 0 ; i--)  
 {  
 b > 0 ? b += a[i] : b = a[i];  
 if(b + m[i-1] > max) max = b + m[i-1];  
 }  
 printf("%d\n",max);  
 }  
 return 0;  
}

Maximum sum用任一种方法即可解决

2.To the Max

这题是最大子段和在维数上的推广（一维变成二维）

我们只要把第i行和第j行之间的每一列元素看成一个整体，即：

a[0] = c[i][0] + c[i+1][0] + ... + c[j][0],

a[1] = c[i][1] + c[i+1][1] + ... + c[j][1],

......

a[n-1] = c[i][n-1] + c[i+1][n-1] + ... + c[n-1][0]

那么变成了求a[]的最大子段和，退化成了一维情况。

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

#include <stdio.h>  
#include <string.h>  
#define N 101  
  
int c[N][N] , a[N] , n;  
  
void SubSum(int i)  
{  
 int k;  
  
 for(k = 0 ; k < n ; k++)  
 {  
 a[k] += c[i][k];  
 }  
}  
  
int CalMaxSum()  
{  
 int i , j , k , b , max;  
  
 max = -128;  
 for(i = 0 ; i < n ; i++)  
 {  
 memset(a , 0 , sizeof(a));  
 for(j = i ; j < n ; j++)  
 {  
 SubSum(j);  
 b = a[0];  
 for(k = 1 ; k < n ; k++)  
 {  
 if(b > 0)  
 b += a[k];  
 else  
 b = a[k];  
  
 if(b > max) max = b;  
 }  
 }  
 }  
 return max;  
}  
int main(void)  
{  
 int i , j;  
  
 while(scanf("%d", &n) != EOF)  
 {  
 for(i = 0 ; i < n ; i++)  
 {  
 for(j = 0 ; j < n ; j++)  
 {  
 scanf("%d", &c[i][j]);  
 }  
 }  
 printf("%d\n", CalMaxSum());  
 }  
  
 return 0;  
}

**字典树**

1.Babelfish

这题就是实现单词的查询，我们可以在字典树的每个节点中添加char  word[]域，如果该节点字符是一个单词的结尾，则在它的word[]域中填写对应的单词。（这题实现方式很多，快排 + 二分也是可以的）

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

#include <stdio.h>  
#include <string.h>  
#define MAX 26  
  
typedef struct TrieNode  
{   
 char word[11];   
 struct TrieNode \*next[MAX];  
}TrieNode;  
  
TrieNode Memory[300010];  
int allocp = 0;  
  
void InitTrieRoot(TrieNode \*\*pRoot)  
{   
 \*pRoot = NULL;  
}  
  
TrieNode \*CreateTrieNode()  
{   
 int i;   
 TrieNode \*p;   
  
 p = &Memory[allocp++];   
 for(i = 0 ; i < MAX ; i++)   
 {   
 p->next[i] = NULL;   
 }   
 return p;  
}  
  
void InsertTrie(TrieNode \*\*pRoot , char \*s1 , char \*s2)  
{   
 int i , k;   
 TrieNode \*p;   
   
 if(!(p = \*pRoot))   
 {   
 p = \*pRoot = CreateTrieNode();   
 }   
 i = 0;   
 while(s1[i])   
 {   
 k = s1[i++] - 'a'; //确定branch   
 if(!p->next[k])   
 {  
 p->next[k] = CreateTrieNode();   
 }  
 p = p->next[k];   
 }  
 strcpy(p->word , s2);  
}  
  
char \*SearchTrie(TrieNode \*\*pRoot , char \*s)  
{   
 TrieNode \*p;   
 int i , k;   
   
 if(!(p = \*pRoot))   
 {   
 return 0;   
 }   
 i = 0;   
 while(s[i])   
 {   
 k = s[i++] - 'a';   
 if(p->next[k] == NULL) return NULL;   
 p = p->next[k];   
 }   
 return p->word;  
}   
  
int main(void)  
{   
 char s1[11] , s2[11] , \*p;  
 int i;  
 TrieNode \*Root = NULL;   
   
 InitTrieRoot(&Root);   
 while(1)   
 {   
 i = 0;  
 if((s1[i] = getchar()) == '\n') break;  
 scanf(" %s %s", s1 + 1 , s2);  
 InsertTrie(&Root , s2 , s1);   
 getchar();  
 }   
  
 while(scanf(" %s", s1) != EOF)   
 {   
 p = SearchTrie(&Root , s1);  
 if(p == NULL)  
 printf("eh\n");  
 else  
 printf("%s\n", p);   
 }   
 return 0;  
}

2.IMMEDIATE DECODABILITY

这题和以前做的Phone list没有区别，添加nEndFlag域就可以了

**逆序数（用树状数组其求解）**

1.DNA Sorting

这题题意比较容易，就是求的每个序列的逆序数，然后根据逆序数从小大大排列

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

#include <stdio.h>   
#include <string.h>  
  
#define N 101  
int b[N];   
char s[N][51];  
  
typedef struct  
{  
 int k;  
 int num;  
 char \*str;  
}Node;  
Node p[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;  
}  
  
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].num < x.num || (arr[i].num == x.num && arr[i].k < x.k));  
 do j--; while(arr[j].num > x.num || (arr[j].num == x.num && arr[j].k > x.k));  
 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 c , i , j , k , n , m;  
  
 while(scanf("%d%d", &n ,&m) != EOF)  
 {  
 getchar();  
 for(i = 0 ; i < m ; i++)  
 {  
 memset(b , 0 , sizeof(b));  
 k = 0; j = 0;  
 while((c = getchar()) != '\n')  
 {  
 s[i][j] = c;  
 k += j - Getsum(c);   
 Update(c , 1);  
 j++;  
 }  
 s[i][j] = '\0';  
 //记录逆序数k  
 p[i].k = i; p[i].num = k; p[i].str = s[i];  
 }  
 QuickSort(p , 0 , m - 1);  
 for(i = 0 ; i < m ; i++)  
 printf("%s\n", p[i].str);  
  
   
 }  
 return 0;  
}

2.Brainman ，Ultra-QuickSort

这两题意思就是，通过交换相邻两个元素，使得序列最终为ascending order，求至少要交换几次。

这实际上就是求这个序列的逆序数，因为逆序数表示序列中有几对数字逆序的，只要把这几对数字给顺序了，序列自然就顺序了。

另外，由于Ultra-QuickSort这题数字范围比较大，而使用树状数组需要开辟一个b[n]的数组，因此必须预先把数据离散化。

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

#include <stdio.h>   
#include <string.h>  
#define N 500010  
  
typedef struct  
{  
 unsigned val;  
 int k;  
}Node;  
Node a[N];  
int hash[N] , b[N] , M;   
  
int Lowbit(int x)  
{  
 return x & (-x);  
}  
  
void Update(int x, int c)   
{  
 int i;  
  
 for(i = x; i <= M ; i += Lowbit(i))  
 {  
 b[i] += c;  
 }  
}  
  
\_\_int64 Getsum(int x)   
{  
 int i;  
 \_\_int64 k = 0;  
  
 for(i = x; i >= 1 ; i -= Lowbit(i))  
 {  
 k += b[i];  
 }  
 return k;  
}  
  
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].val < x.val);  
 do j--; while(arr[j].val > x.val);  
 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;  
 \_\_int64 k;  
  
 while(scanf("%d",&n) && n)  
 {  
 for(i = 0 ; i < n ; i++)  
 {  
 scanf("%u", &a[i].val);  
 a[i].k = i;  
 }  
   
 //离散化  
 QuickSort(a , 0 , n - 1);  
 for(i = 0 ; i < n ; i++)  
 {  
 hash[a[i].k] = i + 1;  
 b[i+1] = 0;  
 }  
 M = n;  
  
 k = 0;  
 for(i = 0 ; i < n ; i++)  
 {  
 k += i - Getsum(hash[i]);   
 Update(hash[i],1);  
 }  
 printf("%I64u\n",k);  
 }  
 return 0;  
}

**BFS**

1.Catch That Cow

题目意思是：站在位置N，然后下一步可以跳跃到位置N - 1或者 N + 1或者2 \* N，问最少需要多少步就可以跳跃到位置K。

这题单向BFS就可以了，但要注意两处剪枝：

a.不要把同一位置p重复加入队列，因此需要哈希判重；

b.对于某些位置p , 就不要考虑跳跃到位置2\*p了,eg: n = 8 ,k = 10 ,第1步：2\*n = 16 , 然后需要6步跳回k，而最短步数显然是10 - 8 = 2步

（2\*p - k < k - p; p < 2k / 3）

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

#include <stdio.h>  
#define N 150000  
//队列  
typedef struct  
{  
 int val;  
 int step;  
}QNode;  
QNode Q[N];  
int front , rear;  
  
int n , k , hash[N] , lim;  
  
void EnQueue(int val , int step)  
{  
  
 Q[rear].val = val;  
 Q[rear++].step = step;  
}  
  
QNode DeleteQueue()  
{  
 return Q[front++];  
}  
  
int BFS()  
{  
 int i , j , c[3] ;  
 QNode temp;  
  
 if(n >= k) return n - k;  
 while(1)  
 {  
 temp = DeleteQueue();  
   
 c[0] = temp.val \* 2;  
 c[1] = temp.val + 1;  
 c[2] = temp.val - 1;  
 for(i = 0 ; i < 3 ; i++)  
 {  
 j = c[i];  
 if(j == k) return temp.step + 1;  
 if(j >= 0 && j < lim && !hash[j]) //哈希判重  
 {  
 hash[j] = 1;  
 EnQueue(j , temp.step + 1);  
 }  
 }  
 }  
 return -1;  
}  
  
  
int main(void)  
{  
 int i;  
  
 while(scanf("%d%d", &n ,&k) != EOF)  
 {  
 lim = (2 \* k / 3 + 1) \* 2;  
 for(i = 0 ; i <= lim ; i++) hash[i] = 0;  
   
 front = rear = 0;  
 hash[n] = 1;  
 EnQueue(n , 0);  
 printf("%d\n", BFS());  
 }  
 return 0;  
}

2.Knight Moves（两题同名）

这两题是BFS最直接的应用，求起点到终点的最短路径，可以选择单向BFS或者双向BFS（但我两者代码的时间差不多- -）

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

#include <stdio.h>  
#include <string.h>  
#define N 9  
//队列  
typedef struct  
{  
 int x , y;  
 int step;  
}QNode;  
QNode Q[N\*N];  
int front , rear , hash[N][N] , start\_x , end\_x , start\_y , end\_y;  
int dir[][2] = {  
 {2 , 1},  
 {2 ,-1},  
 {1 , 2},  
 {1 ,-2},  
 {-2,-1},   
 {-2, 1},   
 {-1,-2},   
 {-1, 2}  
};  
  
void EnQueue(int x , int y , int step)  
{  
 Q[rear].x = x;  
 Q[rear].y = y;  
 Q[rear++].step = step;  
}  
  
QNode DeleteQueue()  
{  
 return Q[front++];  
}  
  
int BFS()  
{  
 int x , y , k;  
 QNode temp;  
  
 if(start\_x == end\_x && start\_y == end\_y) return 0;  
 while(1)  
 {  
 temp = DeleteQueue();  
   
 for(k = 0 ; k < 8 ; k++)  
 {  
 x = temp.x + dir[k][0];  
 y = temp.y + dir[k][1];  
 if(x == end\_x && y == end\_y) return temp.step + 1;  
 if((x > 0 && x < 9 && y > 0 && y < 9) && !hash[x][y])  
 {  
 hash[x][y] = 1;  
 EnQueue(x , y , temp.step + 1);  
 }  
 }  
 }  
 return -1;  
}  
  
int main(void)  
{  
 char cStart\_y , cEnd\_y;  
  
 while(scanf(" %c%d %c%d", &cStart\_y, &start\_x, &cEnd\_y, &end\_x) != EOF)  
 {  
 memset(hash , 0 , sizeof(hash));  
 rear = front = 0;  
  
 start\_y = cStart\_y - 96;  
 end\_y = cEnd\_y - 96;  
 EnQueue(start\_x , start\_y , 0);  
 hash[start\_x][start\_y] = 1;  
   
 printf("To get from %c%d to %c%d takes %d knight moves.\n", cStart\_y, start\_x, cEnd\_y, end\_x, BFS());  
 }  
 return 0;  
}

3.Image Perimeters

这题就是计算所有相邻X（8个方向）连成区域的周长。这题首先需要从起始位置开始遍历X的连通区域，我们可以选择BFS，也可以选择DFS；

接着就是计算周长，这里有个技巧就是统计每个X周围”.”的个数即可（整个图要预先包含在一圈”.”内）

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

#include <stdio.h>  
#include <string.h>  
#define N 22  
//队列  
typedef struct  
{  
 int x , y;  
}QNode;  
QNode Q[N\*N];  
int front , rear , start\_x , start\_y , n , m;  
int dir[][2] = {  
 {-1, 0},  
 {1 , 0},  
 {0 ,-1},  
 {0 , 1},  
 {-1, 1},   
 {1 , 1},   
 {1, -1},   
 {-1, -1}  
};  
char b[N][N];  
  
void EnQueue(int x , int y)  
{  
 Q[rear].x = x;  
 Q[rear++].y = y;  
}  
  
QNode DeleteQueue()  
{  
 return Q[front++];  
}  
  
int EmptyQueue()  
{  
 if(front == rear) return 1;  
 return 0;  
}  
  
int Count(int x , int y)  
{  
 int c = 0;  
  
 if(b[x-1][y] == '.') c++;  
 if(b[x+1][y] == '.') c++;  
 if(b[x][y-1] == '.') c++;  
 if(b[x][y+1] == '.') c++;  
 return c;  
}  
  
int BFS()  
{  
 int x , y , k , cnt = 0;  
 QNode temp;  
  
 while(!EmptyQueue())  
 {  
 temp = DeleteQueue();  
   
 for(k = 0 ; k < 8 ; k++)  
 {  
 x = temp.x + dir[k][0];  
 y = temp.y + dir[k][1];  
  
 if((x > 0 && x <= n && y > 0 && y <= m) && b[x][y] == 'X')  
 {  
 cnt += Count(x , y);  
 b[x][y] = 'A'; //随意  
 EnQueue(x , y);  
 }  
 }  
 }  
 return cnt;  
}  
  
int main(void)  
{  
 int i , j;  
  
 while(scanf("%d%d%d%d", &n , &m , &start\_x, &start\_y) && n + m)  
 {  
 for(i = 1 ; i <= n ; i++)  
 {  
 for(j = 1 ; j <= m ; j++)  
 {  
 scanf(" %c", &b[i][j]);  
 }  
 }  
 //预先包含在一圈“.”内  
 for(j = 0 ; j <= m + 1; j++) b[0][j] = b[n+1][j]= '.';  
 for(i = 0 ; i <= n + 1; i++) b[i][0] = b[i][m+1]= '.';  
  
 rear = front = 0;  
 EnQueue(start\_x , start\_y);  
 b[start\_x][start\_y] = 'A';  
   
 printf("%d\n", BFS() + Count(start\_x, start\_y));  
 }  
 return 0;  
}

**线段树**

1.校门外的树

这题题意就是，给定一条线段，然后删除几个区间，最后求剩余的区间长度。

我们通过线段树的基本操作就可以完成了：删除：cover置为0 ，计算：统计cover为1的区间长度（注意删除一个节点时，总要连起子树一同删除）

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

#include <stdio.h>  
#define N 10001  
  
typedef struct  
{   
 int l , r;   
 int cover;  
}TreeNode;  
TreeNode seg\_tree[4\*N];  
  
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 DeleteSegTree(int p , int a , int b)  
{   
 int m;   
   
 if(seg\_tree[p].l == seg\_tree[p].r) //if(seg\_tree[p].l == a && seg\_tree[p].r == b)   
 {   
 seg\_tree[p].cover = 0;   
 return ;   
 }   
   
 m = (seg\_tree[p].l + seg\_tree[p].r) >> 1;   
 /\*原先完全覆盖\*/   
 if(seg\_tree[p].cover == 1)   
 {   
 seg\_tree[p].cover = 0;   
 seg\_tree[2\*p].cover = seg\_tree[2\*p+1].cover = 1;   
 }   
 if(b <= m)   
 {   
 DeleteSegTree(p << 1 , a , b);   
 }   
 else if(a > m)   
 {   
 DeleteSegTree((p << 1) + 1 , a , b);   
 }   
 else   
 {   
 DeleteSegTree(p << 1 , a , m);  
 DeleteSegTree((p << 1) + 1 , m + 1,b);   
 }  
}  
  
int Count(int p)  
{  
 if(seg\_tree[p].cover == 1)  
 {  
 return seg\_tree[p].r - seg\_tree[p].l + 1;  
 }  
 if(seg\_tree[p].l == seg\_tree[p].r)  
 {  
 return 0;  
 }  
 return Count(p << 1) + Count((p << 1) + 1);  
}  
  
int main(void)  
{  
 int k , m , l , r , i;  
  
 while(scanf("%d%d", &k , &m) != EOF)  
 {  
 CreateSegTree(1 , 1 , k);  
 seg\_tree[1].cover = 1;  
  
 for(i = 0 ; i < m ; i++)  
 {  
 scanf("%d%d", &l,&r);  
 l < r ? DeleteSegTree(1 , l , r) : DeleteSegTree(1 , r , l);  
 }  
 printf("%d\n",Count(1) + 1);  
 }  
 return 0;  
}

 这题打算尝试用快排+加合并区间做一下