# 1、区间第K大+取后缀

Hdu1890

|  |
| --- |
| /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Author :kuangbin  Created Time :2013/8/24 23:28:43  File Name :F:\2013ACM练习\专题学习\splay\_tree\_2\HDU1890.cpp  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/  #include <stdio.h>  #include <string.h>  #include <iostream>  #include <algorithm>  #include <vector>  #include <queue>  #include <set>  #include <map>  #include <string>  #include <math.h>  #include <stdlib.h>  #include <time.h>  using namespace std;  #define Key\_value ch[ch[root][1]][0]  const int MAXN = 100010;  int pre[MAXN],ch[MAXN][2],root,tot1;  int size[MAXN];//子树规模  int rev[MAXN];//反转标记  int s[MAXN],tot2;//内存池和容量  //debug部分\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void Treavel(int x)  {  if(x)  {  Treavel(ch[x][0]);  printf("结点：%2d: 左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d rev = %2d\n",x,ch[x][0],ch[x][1],pre[x],size[x],rev[x]);  Treavel(ch[x][1]);  }  }  void debug()  {  printf("root:%d\n",root);  Treavel(root);  }  //以上是debug部分\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void NewNode(int &r,int father,int k)  {  r = k;  pre[r] = father;  ch[r][0] = ch[r][1] = 0;  size[r] = 1;  rev[r] = 0;  }  //反转的更新  void Update\_Rev(int r)  {  if(!r)return;  swap(ch[r][0],ch[r][1]);  rev[r] ^= 1;  }  inline void push\_up(int r)  {  size[r] = size[ch[r][0]] + size[ch[r][1]] + 1;  }  inline void push\_down(int r)  {  if(rev[r])  {  Update\_Rev(ch[r][0]);  Update\_Rev(ch[r][1]);  rev[r] = 0;  }  }  void Build(int &x,int l,int r,int father)  {  if(l > r)return;  int mid = (l+r)/2;  NewNode(x,father,mid);  Build(ch[x][0],l,mid-1,x);  Build(ch[x][1],mid+1,r,x);  push\_up(x);  }  int n;  void Init()  {  root = tot1 = tot2 = 0;  ch[root][0] = ch[root][1] = pre[root] = size[root] = rev[root] = 0;  NewNode(root,0,n+1);  NewNode(ch[root][1],root,n+2);  Build(Key\_value,1,n,ch[root][1]);  push\_up(ch[root][1]);  push\_up(root);  }  //旋转，0为左旋，1为右旋  inline void Rotate(int x,int kind)  {  int y = pre[x];  push\_down(y);  push\_down(x);//先把y的标记下传，在把x的标记下传  ch[y][!kind] = ch[x][kind];  pre[ch[x][kind]] = y;  if(pre[y])  ch[pre[y]][ch[pre[y]][1]==y] = x;  pre[x] = pre[y];  ch[x][kind] = y;  pre[y] = x;  push\_up(y);  }  //Splay调整，将r结点调整到goal下面  inline void Splay(int r,int goal)  {  push\_down(r);  while(pre[r] != goal)  {  if(pre[pre[r]] == goal)  {  //有反转操作，需要先push\_down,再判断左右孩子  push\_down(pre[r]);  push\_down(r);  Rotate(r,ch[pre[r]][0]==r);  }  else  {  //有反转操作，需要先push\_down  push\_down(pre[pre[r]]);  push\_down(pre[r]);  push\_down(r);  int y = pre[r];  int kind = ch[pre[y]][0]==y;  if(ch[y][kind] == r)  {  Rotate(r,!kind);  Rotate(r,kind);  }  else  {  Rotate(y,kind);  Rotate(r,kind);  }  }  }  push\_up(r);  if(goal == 0) root = r;  }  //得到第k个结点(需要push\_down)  inline int Get\_kth(int r,int k)  {  push\_down(r);  int t = size[ch[r][0]] + 1;  if(t == k)return r;  if(t > k)return Get\_kth(ch[r][0],k);  else return Get\_kth(ch[r][1],k-t);  }  //找前驱(需要push\_down)  inline int Get\_pre(int r)  {  push\_down(r);  if(ch[r][0] == 0)return -1;//不存在  r = ch[r][0];  while(ch[r][1])  {  r = ch[r][1];  push\_down(r);  }  return r;  }  //找后继(需要push\_down)  inline int Get\_next(int r)  {  push\_down(r);  if(ch[r][1] == 0)return -1;  r = ch[r][1];  while(ch[r][0])  {  r = ch[r][0];  push\_down(r);  }  return r;  }  struct Node  {  int id,val;  }node[MAXN];  bool cmp(Node a,Node b)  {  if(a.val != b.val)return a.val < b.val;  else return a.id < b.id;  }  int main()  {  //freopen("in.txt","r",stdin);  //freopen("out.txt","w",stdout);  while(scanf("%d",&n) == 1 && n)  {  for(int i = 1;i <= n;i++)  {  scanf("%d",&node[i].val);  node[i].id = i;  }  sort(node+1,node+n+1,cmp);  Init();  for(int i = 1; i <= n;i++)  {  Splay(node[i].id,0);  printf("%d",size[ch[root][0]]);  if(i < n)printf(" ");  else printf("\n");  Splay(Get\_kth(root,i),0);  Splay(Get\_next(node[i].id),root);  Update\_Rev(Key\_value);  }  }  return 0;  } |