[Splay伸展树模板总结](http://blog.csdn.net/snk1996/article/details/48210045)

1.基本点操作

**[cpp]** [view plain](http://blog.csdn.net/snk1996/article/details/48210045) [copy](http://blog.csdn.net/snk1996/article/details/48210045)

1. //Splay 基本操作 均摊复杂度O(lgN)
3. //POJ 1442
4. //基本点操作

7. // sp.init()    初始化


11. #include <iostream>
12. #include <algorithm>
13. #include <cmath>
15. **using** **namespace** std;
17. #define maxn 31000
18. #define min(a,b) ((a)<(b)?(a):(b))
19. **const** **int** oo = 0x3f3f3f3f;
20. **struct** Node{
21. **int** key, sz, cnt;
22. Node \*ch[2], \*pnt;//左右儿子和父亲
23. Node(){}
24. Node(**int** x, **int** y, **int** z){
25. key = x, sz = y, cnt = z;
26. }
27. **void** rs(){
28. sz = ch[0]->sz + ch[1]->sz + cnt;
29. }
30. }nil(0, 0, 0), \*NIL = &nil;
31. **struct** Splay{//伸展树结构体类型
32. Node \*root;
33. **int** ncnt;//计算key值不同的结点数，注意已经去重了
34. Node nod[maxn];
35. **void** init(){// 首先要初始化
36. root = NIL;
37. ncnt = 0;
38. }
39. **void** rotate(Node \*x, **bool** d){//旋转操作，d为true表示右旋
40. Node \*y = x->pnt;
41. y->ch[!d] = x->ch[d];
42. **if** (x->ch[d] != NIL)
43. x->ch[d]->pnt = y;
44. x->pnt = y->pnt;
45. **if** (y->pnt != NIL){
46. **if** (y == y->pnt->ch[d])
47. y->pnt->ch[d] = x;
48. **else**
49. y->pnt->ch[!d] = x;
50. }
51. x->ch[d] = y;
52. y->pnt = x;
53. y->rs();
54. x->rs();
55. }
56. **void** splay(Node \*x, Node \*target){//将x伸展到target的儿子位置处
57. Node \*y;
58. **while** (x->pnt != target){
59. y = x->pnt;
60. **if** (x == y->ch[0]){
61. **if** (y->pnt != target && y == y->pnt->ch[0])
62. rotate(y, **true**);
63. rotate(x, **true**);
64. }
65. **else**{
66. **if** (y->pnt != target && y == y->pnt->ch[1])
67. rotate(y, **false**);
68. rotate(x, **false**);
69. }
70. }
71. **if** (target == NIL)
72. root = x;
73. }
74. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*以上一般不用修改\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
75. **void** insert(**int** key){//插入一个值
76. **if** (root == NIL){
77. ncnt = 0;
78. root = &nod[++ncnt];
79. root->ch[0] = root->ch[1] = root->pnt = NIL;
80. root->key = key;
81. root->sz = root->cnt = 1;
82. **return**;
83. }
84. Node \*x = root, \*y;
85. **while** (1){
86. x->sz++;
87. **if** (key == x->key){
88. x->cnt++;
89. x->rs();
90. y = x;
91. **break**;
92. }
93. **else** **if** (key < x->key){
94. **if** (x->ch[0] != NIL)
95. x = x->ch[0];
96. **else**{
97. x->ch[0] = &nod[++ncnt];
98. y = x->ch[0];
99. y->key = key;
100. y->sz = y->cnt = 1;
101. y->ch[0] = y->ch[1] = NIL;
102. y->pnt = x;
103. **break**;
104. }
105. }
106. **else**{
107. **if** (x->ch[1] != NIL)
108. x = x->ch[1];
109. **else**{
110. x->ch[1] = &nod[++ncnt];
111. y = x->ch[1];
112. y->key = key;
113. y->sz = y->cnt = 1;
114. y->ch[0] = y->ch[1] = NIL;
115. y->pnt = x;
116. **break**;
117. }
118. }
119. }
120. splay(y, NIL);
121. }
122. Node\* search(**int** key){//查找一个值，返回指针
123. **if** (root == NIL)
124. **return** NIL;
125. Node \*x = root, \*y = NIL;
126. **while** (1){
127. **if** (key == x->key){
128. y = x;
129. **break**;
130. }
131. **else** **if** (key > x->key){
132. **if** (x->ch[1] != NIL)
133. x = x->ch[1];
134. **else**
135. **break**;
136. }
137. **else**{
138. **if** (x->ch[0] != NIL)
139. x = x->ch[0];
140. **else**
141. **break**;
142. }
143. }
144. splay(x, NIL);
145. **return** y;
146. }
147. Node\* searchmin(Node \*x){//查找最小值，返回指针
148. Node \*y = x->pnt;
149. **while** (x->ch[0] != NIL){//遍历到最左的儿子就是最小值
150. x = x->ch[0];
151. }
152. splay(x, y);
153. **return** x;
154. }
155. Node\* searchmax(Node \*x){
156. Node \*y = x->pnt;
157. **while**(x->ch[1] != NIL){
158. x = x->ch[1];
159. }
160. splay(x,y);
161. **return** x;
162. }
163. **void** del(**int** key){//删除一个值
164. **if** (root == NIL)
165. **return**;
166. Node \*x = search(key), \*y;
167. **if** (x == NIL)
168. **return**;
169. **if** (x->cnt > 1){
170. x->cnt--;
171. x->rs();
172. **return**;
173. }
174. **else** **if** (x->ch[0] == NIL && x->ch[1] == NIL){
175. init();
176. **return**;
177. }
178. **else** **if** (x->ch[0] == NIL){
179. root = x->ch[1];
180. x->ch[1]->pnt = NIL;
181. **return**;
182. }
183. **else** **if** (x->ch[1] == NIL){
184. root = x->ch[0];
185. x->ch[0]->pnt = NIL;
186. **return**;
187. }
188. y = searchmin(x->ch[1]);
189. y->pnt = NIL;
190. y->ch[0] = x->ch[0];
191. x->ch[0]->pnt = y;
192. y->rs();
193. root = y;
194. }
195. **int** rank(**int** key){//求结点高度
196. Node \*x = search(key);
197. **if** (x == NIL)
198. **return** 0;
199. **return** x->ch[0]->sz + 1/\* or x->cnt\*/;
200. }
201. Node\* findk(**int** kth){//查找第k小的值
202. **if** (root == NIL || kth > root->sz)
203. **return** NIL;
204. Node \*x = root;
205. **while** (1){
206. **if** (x->ch[0]->sz +1 <= kth && kth <= x->ch[0]->sz + x->cnt)
207. **break**;
208. **else** **if** (kth <= x->ch[0]->sz)
209. x = x->ch[0];
210. **else**{
211. kth -= x->ch[0]->sz + x->cnt;
212. x = x->ch[1];
213. }
214. }
215. splay(x, NIL);
216. **return** x;
217. }
219. Node\* proc(**int** key)
220. {
221. Node \*x = search(key);
222. **if**(x==NIL || x->cnt == 1 && x->ch[0]==NIL)
223. {
224. **return** NIL;
225. }
226. **if**(x->cnt > 1)
227. {
228. **return** x;
229. }
230. **else**
231. {
232. **return** searchmax(x->ch[0]);
233. }
234. }
236. Node\* succ(**int** key)
237. {
238. Node \*x = search(key);
239. **if**(x==NIL || x->cnt == 1 && x->ch[1]==NIL)
240. {
241. **return** NIL;
242. }
243. **if**(x->cnt > 1)
244. {
245. **return** x;
246. }
247. **else**
248. {
249. **return** searchmin(x->ch[1]);
250. }
251. }
253. **int** size()
254. {
255. **return** root->sz;
256. }
257. }sp;
258. **int** num[30010];
259. **int** cmd[30010];
260. **int** main(){
262. ios::sync\_with\_stdio(**false**);
263. **int** N,M,x,i;
264. **while**(cin >> N >> M)
265. {
266. sp.init();
267. **for**(**int** i=0;i<N;i++)
268. {
269. cin >> num[i];
270. }
271. **for**(**int** i=1;i<=M;i++)
272. {
273. cin >> cmd[i];
274. }
275. **int** j=1;
276. Node\* node;
277. **for**(**int** i=0;i<N;i++)
278. {
279. sp.insert(num[i]);
280. **while**(sp.size()==cmd[j])
281. {
282. node = sp.findk(j++);
283. cout << node->key << endl;
284. }
285. }
287. }
289. **return** 0;
290. }

2.基本区间操作

**[cpp]** [view plain](http://blog.csdn.net/snk1996/article/details/48210045) [copy](http://blog.csdn.net/snk1996/article/details/48210045)

1. /\*
2. \_http://acm.hdu.edu.cn/showproblem.php?pid=3487\_
3. 1.区间切割 区间翻转
4. 原序列:    1、2、3 ……n
5. rangeCut(a,b,c) 将[ath,bth]剪切掉 然后黏贴到新序列的第cth后面
6. rangeFlip(a,b)  翻转[ath,bth]


10. \_http://acm.hdu.edu.cn/showproblem.php?pid=1754\_
11. 2.单点更新 区间询问最值
12. query(a,b)  询问[a,b]最大值
13. update(a,c) 更新学生a的成绩为c


17. \_http://poj.org/problem?id=3468\_
18. 3.区间更新 区间求和
19. querySum(l,r)   询问[l,r]的和
20. updateInterval(l,r,c)   区间[l,r]的值+c
22. \*/
23. #include<iostream>
24. #include<cstdio>
25. #include<cstring>
26. #define LL long long
27. **using** **namespace** std;
28. **const** **int** MAXN = 333333;
29. #define m\_set(ptr,v,type,size) memset(ptr,v,sizeof(type) \* size)
30. #define loop(begin,end) for(int i=begin;i<end;i++)
31. #define debug   puts("here!")
32. **class** SplayTree
33. {
34. #define l(x) (ch[x][0])
35. #define r(x) (ch[x][1])
36. #define mid(x,y)    ((x+y)>>1)
37. **public**:
38. **int** ch[MAXN][2],pre[MAXN];
39. **int** sz[MAXN],val[MAXN],rev[MAXN],a[MAXN];
40. **int** root,tot;
42. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
44. **int** mx[MAXN];
46. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
48. LL sum[MAXN],add[MAXN];
50. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
52. **void** init()
53. {
54. m\_set(ch,0,**int**,MAXN\*2);
55. m\_set(pre,0,**int**,MAXN);
56. m\_set(sz,0,**int**,MAXN);
57. m\_set(val,0,**int**,MAXN);
58. m\_set(rev,0,**int**,MAXN);
60. /\*\*\*\*\*\*\*\*\*\*\*\*\*/
61. m\_set(mx,-1,**int**,MAXN);
62. /\*\*\*\*\*\*\*\*\*\*\*\*\*/
63. root = tot = 0;
64. }
66. **void** read(**int** n)        //0 1 2 3 .... n 0
67. {
68. a[1] = a[n+2] = -1;
69. //          loop(2,n+2){
70. //              a[i] = i - 1;
71. //          }
73. /\*\*\*\*\*\*\*\*\*\*\*\*\*/
74. **for**(**int** i=2;i<=n+1;i++)
75. {
76. scanf("%d",&a[i]);
77. }
78. /\*\*\*\*\*\*\*\*\*\*\*\*\*/
79. }
81. **void** push\_up(**int** rt)
82. {
83. sz[rt] = sz[l(rt)] + sz[r(rt)] + 1;
84. /\*\*\*\*\*\*\*\*\*\*\*\*\*/
85. mx[rt] = max(max(mx[l(rt)],mx[r(rt)]),val[rt]);
86. /\*\*\*\*\*\*\*\*\*\*\*\*\*/
87. sum[rt] = sum[l(rt)] + sum[r(rt)] + val[rt];
88. }
90. **void** push\_down(**int** rt)
91. {
92. //          if(rt&&rev[rt])
93. //          {
94. //              swap(l(rt),r(rt));
95. //              if(l(rt))   rev[l(rt)] ^= 1;
96. //              if(r(rt))   rev[r(rt)] ^= 1;
97. //              rev[rt] = 0;
98. //          }
100. **if**(add[rt])
101. {
102. **if**(l(rt))
103. {
104. val[l(rt)] += add[rt];
105. add[l(rt)] += add[rt];
106. sum[l(rt)] += add[rt] \* sz[l(rt)];
107. }
108. **if**(r(rt))
109. {
110. val[r(rt)] += add[rt];
111. add[r(rt)] += add[rt];
112. sum[r(rt)] += add[rt] \* sz[r(rt)];
113. }
114. add[rt] = 0;
115. }
116. }
118. **void** swap(**int** &x,**int** &y)
119. {
120. **int** tmp = x;
121. x = y;
122. y = tmp;
123. }

126. **void** rotate(**int** x,**int** f)    //将x旋转到其直接父节点 右旋 左旋
127. {
128. **int** y  = pre[x];
129. push\_down(x);
130. push\_down(y);
131. ch[y][!f] = ch[x][f];
132. **if**(ch[y][!f])   pre[ch[y][!f]] = y;
133. push\_up(y);
134. **if**(pre[y])  ch[pre[y]][r(pre[y])==y]= x;
135. pre[x] = pre[y];
136. ch[x][f] = y;
137. pre[y] = x;
138. }
140. **void** splay(**int** x,**int** goal)  //将x旋转到以goal节点为父节点
141. {
142. push\_down(x);
143. **while**(pre[x]!=goal)
144. {
145. **int** y = pre[x] , z = pre[pre[x]];
146. **if**(z==goal)
147. {
148. rotate(x,l(y)==x);
149. }
150. **else**
151. {
152. **int** f = (l(z) == y);
153. **if**(ch[y][!f] == x)
154. {
155. rotate(y,f);
156. rotate(x,f);
157. }
158. **else**
159. {
160. rotate(x,!f);
161. rotate(x,f);
162. }
163. }
164. }
165. push\_up(x);
166. **if**(goal==0) root = x;
167. }
169. **void** rotateTo(**int** k,**int** goal)
170. {
171. **int** x = root;
172. **while**(**true**)
173. {
174. push\_down(x);
175. **int** tmp = sz[l(x)] + 1;
176. **if**(k==tmp)  **break**;
177. **else** **if**(k<tmp)   x = l(x);
178. **else**
179. {
180. k -= tmp;
181. x = r(x);
182. }
183. }
184. splay(x,goal);
185. }
187. **void** buildTree(**int** l,**int** r,**int** &rt,**int** f)
188. {
189. **if**(l>r)  **return**;
190. **int** m = mid(l,r);
191. rt = ++tot;
192. val[rt] = a[m];
193. pre[rt] = f;
194. buildTree(l,m-1,l(rt),rt);
195. buildTree(m+1,r,r(rt),rt);
196. push\_up(rt);
197. }
199. **void** rangeCut(**int** l,**int** r,**int** goal)
200. {
201. rotateTo(l-1,0);
202. rotateTo(r+1,root);
203. push\_down(r(root));
204. **int** x = l(r(root)); //x子树代表[l,r]区间
205. //截取
206. l(r(root)) = 0;
207. pre[x] = 0;
208. push\_up(r(root));
209. push\_up(root);
210. rotateTo(goal,0);
211. rotateTo(goal+1,root);
212. push\_down(r(root));
213. l(r(root)) = x;
214. pre[x] = r(root);
215. push\_up(r(root));
216. push\_up(root);
217. }
219. **void** rangeFlip(**int** l,**int** r)
220. {
221. rotateTo(l-1,0);
222. rotateTo(r+1,root);
223. push\_down(r(root));
224. **int** x = l(r(root));
225. rev[x] ^= 1;
226. }
228. **void** dfs(**int** rt,**int** &size)
229. {
230. **if**(!rt) **return**;
231. push\_down(rt);
232. dfs(l(rt),size);
233. a[size++] = val[rt];
234. dfs(r(rt),size);
235. }

238. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
239. **int** query(**int** l,**int** r)
240. {
241. rotateTo(l-1,0);
242. rotateTo(r+1,root);
243. **return** mx[l(r(root))];
244. }
246. **void** update(**int** pos,**int** c)
247. {
248. rotateTo(pos,0);
249. val[root] = c;
250. push\_up(root);
251. }
253. LL querySum(**int** l,**int** r)
254. {
255. rotateTo(l-1,0);
256. rotateTo(r+1,root);
257. **return** sum[l(r(root))];
258. }
260. **void** updateInterval(**int** l,**int** r,**int** c)
261. {
262. rotateTo(l-1,0);
263. rotateTo(r+1,root);
264. val[l(r(root))] += c;
265. sum[l(r(root))] += c \* sz[l(r(root))];
266. add[l(r(root))] += c;
267. }
269. }spt;
271. //int main()
272. //{
273. //  int n,m;
274. //  while(~scanf("%d%d",&n,&m) && (n>=0 || m>=0))
275. //  {
276. //      spt.init();
277. //      spt.read(n);
278. //      spt.buildTree(1,n+2,spt.root,0);
279. //      char op[5];
280. //      int a,b,c;
281. //      while(m--)
282. //      {
283. //          scanf("%s%d%d",op,&a,&b);
284. //          if(op[0]=='C')
285. //          {
286. //              scanf("%d",&c);
287. //              spt.rangeCut(a+1,b+1,c+1);  //因为哨兵0 多加1
288. //          }
289. //          else
290. //          {
291. //              spt.rangeFlip(a+1,b+1);
292. //          }
293. //      }
294. //      n = 0;
295. //      spt.dfs(spt.root,n);    //中序遍历即为序列结果
296. //      loop(1,n-1)
297. //      {
298. //          if(i!=1)    printf(" ");
299. //          printf("%d",spt.a[i]);
300. //      }
301. //      printf("\n");
302. //  }
303. //  return 0;
304. //}




310. //int main()
311. //{
312. //  int n,m;
313. //  while(scanf("%d%d",&n,&m) == 2)
314. //  {
315. //      spt.init();
316. //      spt.read(n);
317. //      spt.buildTree(1,n+2,spt.root,0);
318. //      char op[2];
319. //      int a,b;
320. //      while(m--)
321. //      {
322. //          scanf("%s%d%d",op,&a,&b);
323. //          if(op[0]=='Q')
324. //          {
325. //              printf("%d\n",spt.query(a+1,b+1));
326. //          }
327. //          else
328. //          {
329. //              spt.update(a+1,b);
330. //          }
331. //      }
332. //  }
333. //  return 0;
334. //}


338. **int** main()
339. {
340. **int** n,m;
341. **while**(~scanf("%d%d",&n,&m))
342. {
343. spt.init();
344. spt.read(n);
345. spt.buildTree(1,n+2,spt.root,0);
346. **char** op[2];
347. **int** a,b,c;
348. **while**(m--)
349. {
350. scanf("%s%d%d",op,&a,&b);
351. **if**(op[0] == 'Q')
352. {
353. printf("%I64d\n",spt.querySum(a+1,b+1));
354. }
355. **else**
356. {
357. scanf("%d",&c);
358. spt.updateInterval(a+1,b+1,c);
359. }
360. }
361. }
362. **return** 0;
363. }