**强大到无与伦比的数据结构splay-tree**

[上篇文章](http://www.notonlysuccess.com/?p=699)~研究了一下sbt(size balance tree)后,三鲜师傅评论说splay-tree功能更强大,并且有很多其他数据结构无法实现的功能

于是赶紧去学习,看了[这题](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1500)后发现sbt几乎完成不了其中的任何一个操作,而线段树也无法完成插入.删除.翻转的操作,但是splay却能很完美的解决

其他平衡二叉树是根据权值来限制树的结构的,没有splay这么灵活.splay的任意旋转可以对一整个区间进行操作,并且可以任意增加区间,任意删除区间,都是logn的复杂度.并且可以沿用线段树的延迟标记,每次不用急着传递给子树,遍历到时候传下去即可,然后更新上来(我写了push\_down和push\_up两个函数,其实线段树也是这个套路,不过我以前写线段树的时候直接把这两个函数写进update和query里的,现在回去看以前线段树的代码感觉很乱,没有单独写出来优美~)

献上[Crash撞神](http://www.docin.com/p-62465596.html)大牛的论文,里边就是将splay对一个区间的具体操作

一些其他二叉平衡树叶能完成的操作在这篇[和楼教主同个时代的美女神牛杨思雨的论文](http://www.docin.com/p-63165342.html)里有介绍

介绍一些练习splay的题目:  
  
【splay入门】用其他平衡二叉树能解决:(理论复杂度没有sbt好,但我用splay却比写sbt还快=.=)  
[[HNOI2002]营业额统计](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1588)  
[[NOI2004]郁闷的出纳员](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1503)  
[[HNOI2004]宠物收养所](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1208)

【splay热身】其他平衡二叉树不能解决,但是线段树和splay能解决(效率是线段树的1~5倍.)  
[I Hate It](http://acm.hdu.edu.cn/showproblem.php?pid=1754)(更新节点,区间最值)  
[A Simple Problem with Integers](http://acm.pku.edu.cn/JudgeOnline/problem?id=3468)(成段更新,区间求和)

[Memory Control](http://acm.hdu.edu.cn/showproblem.php?pid=2871)(较为复杂的区间操作)  
更多热身题可以去[这里](http://www.notonlysuccess.com/?p=59)找

【splay进阶】其他平衡二叉树和线段树都无法解决:  
[Robotic Sort](http://acm.hdu.edu.cn/showproblem.php?pid=1890)  
比较有意思的题目,结点处没有任何权值,只要记录每个数字对应的结点位子,然后从小到打把相对应的位子旋转到根节点,输出**i+左子树**的个数,接着给左子树一个翻转的延迟标记,最后删除该节点.注意把结点旋到根部的时候要先从根部把延迟标记push\_down下去.  
[Queue-jumpers](http://acm.hdu.edu.cn/showproblem.php?pid=3436)  
可以用各种数据结构解决的题,用splay的话和上题差不多,记录结点的位子..Top的操作有点麻烦,记得每次操作后都要把结点splay到根部,不然会超时=.=  
[Play with Chain](http://acm.hdu.edu.cn/showproblem.php?pid=3487)  
此题有splay的最独特的操作 区间旋转和切割  
[[AHOI2006]文本编辑器edito](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1267)r(较为简单的插入删除旋转,小数据版)  
[[AHOI2006]文本编辑器edito](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1269)r(较为简单的插入删除旋转,大数据版)  
这题很恶心,最后一个是结束符号也会让你输出来,我纠结了很久很久  
[[NOI2005]维修数列](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1500)(在插入删除基础上增加区间统计,较难)  
区间统计如果做了上边三个热身题的话应该没问题了  
旋转和区间统计的时候要特别注意,两者结合产生的trick留给大家自己去发现,这个trick整了我一个下午  
50M的数据量,用静态的数组开不下,动态的太浪费时间,怎么办呢?自己手动压个内存池吧

总结.splay的区间操作看上去很复杂,其实就是函数多,略数一下有以下几个  
void Rotate(int x,int f)  
void Splay(int x,int goal)  
void RotateTo(int k,int goal)  
void NewNode(int &x,int c)  
void push\_down(int x)  
void push\_up(int x)  
void makeTree(int &x,int l,int r,int f)  
void init(int n)  
这几个函数是必不可少的,并且还要再根据题目意思加一些函数  
但是真正理解了之后会发现…写了最上边的三个函数后 其实剩下的就和线段树差不多,这样看来编程复杂度也不是很大,手抽筋一下花个5分钟敲下这几个函数也就OK了  
所以splay的题完全可以套上线段树的马甲再加上线段树无法完成的操作变成更难的题目XD

好了,献上模板:(包括了回收内存,虽然模板上这题不需要用这个操作)

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[[Copy to clipboard]View Code](javascript:;) CPP

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176 | */\**  *http://acm.pku.edu.cn/JudgeOnline/problem?id=3468*  *区间跟新,区间求和*  *\*/*  #include <cstdio>  #define keyTree (ch[ ch[root][1] ][0])  const int maxn = 222222;  struct SplayTree{  int sz[maxn];  int ch[maxn][2];  int pre[maxn];  int root , top1 , top2;  int ss[maxn] , que[maxn];    inline void Rotate(int x,int f) {  int y = pre[x];  push\_down(y);  push\_down(x);  ch[y][!f] = ch[x][f];  pre[ ch[x][f] ] = y;  pre[x] = pre[y];  if(pre[x]) ch[ pre[y] ][ ch[pre[y]][1] == y ] = x;  ch[x][f] = y;  pre[y] = x;  push\_up(y);  }  inline void Splay(int x,int goal) {  push\_down(x);  while(pre[x] != goal) {  if(pre[pre[x]] == goal) {  Rotate(x , ch[pre[x]][0] == x);  } else {  int y = pre[x] , z = pre[y];  int f = (ch[z][0] == y);  if(ch[y][f] == x) {  Rotate(x , !f) , Rotate(x , f);  } else {  Rotate(y , f) , Rotate(x , f);  }  }  }  push\_up(x);  if(goal == 0) root = x;  }  inline void RotateTo(int k,int goal) {//把第k位的数转到goal下边  int x = root;  push\_down(x);  while(sz[ ch[x][0] ] != k) {  if(k < sz[ ch[x][0] ]) {  x = ch[x][0];  } else {  k -= (sz[ ch[x][0] ] + 1);  x = ch[x][1];  }  push\_down(x);  }  Splay(x,goal);  }  inline void erase(int x) {//把以x为祖先结点删掉放进内存池,回收内存  int head = 0 , tail = 0;  for (que[tail++] = x ; head < tail ; head ++) {  ss[top2 ++] = que[head];  if(ch[ que[head] ][0]) que[tail++] = ch[ que[head] ][0];  if(ch[ que[head] ][1]) que[tail++] = ch[ que[head] ][1];  }  }  //以上一般不修改//////////////////////////////////////////////////////////////////////////////  void debug() {printf("%d**\n**",root);Treaval(root);}  void Treaval(int x) {  if(x) {  Treaval(ch[x][0]);  printf("结点%2d:左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d ,val = %2d**\n**",x,ch[x][0],ch[x][1],pre[x],sz[x],val[x]);  Treaval(ch[x][1]);  }  }  //以上Debug      //以下是题目的特定函数:  inline void NewNode(int &x,int c) {  if (top2) x = ss[--top2];//用栈手动压的内存池  else x = ++top1;  ch[x][0] = ch[x][1] = pre[x] = 0;  sz[x] = 1;    val[x] = sum[x] = c;*/\*这是题目特定函数\*/*  add[x] = 0;  }    //把延迟标记推到孩子  inline void push\_down(int x) {*/\*这是题目特定函数\*/*  if(add[x]) {  val[x] += add[x];  add[ ch[x][0] ] += add[x];  add[ ch[x][1] ] += add[x];  sum[ ch[x][0] ] += (long long)sz[ ch[x][0] ] \* add[x];  sum[ ch[x][1] ] += (long long)sz[ ch[x][1] ] \* add[x];  add[x] = 0;  }  }  //把孩子状态更新上来  inline void push\_up(int x) {  sz[x] = 1 + sz[ ch[x][0] ] + sz[ ch[x][1] ];  */\*这是题目特定函数\*/*  sum[x] = add[x] + val[x] + sum[ ch[x][0] ] + sum[ ch[x][1] ];  }    */\*初始化\*/*  inline void makeTree(int &x,int l,int r,int f) {  if(l > r) return ;  int m = (l + r)>>1;  NewNode(x , num[m]); */\*num[m]权值改成题目所需的\*/*  makeTree(ch[x][0] , l , m - 1 , x);  makeTree(ch[x][1] , m + 1 , r , x);  pre[x] = f;  push\_up(x);  }  inline void init(int n) {*/\*这是题目特定函数\*/*  ch[0][0] = ch[0][1] = pre[0] = sz[0] = 0;  add[0] = sum[0] = 0;    root = top1 = 0;  //为了方便处理边界,加两个边界顶点  NewNode(root , -1);  NewNode(ch[root][1] , -1);  pre[top1] = root;  sz[root] = 2;      for (int i = 0 ; i < n ; i ++) scanf("%d",&num[i]);  makeTree(keyTree , 0 , n-1 , ch[root][1]);  push\_up(ch[root][1]);  push\_up(root);  }  */\*更新\*/*  inline void update( ) {*/\*这是题目特定函数\*/*  int l , r , c;  scanf("%d%d%d",&l,&r,&c);  RotateTo(l-1,0);  RotateTo(r+1,root);  add[ keyTree ] += c;  sum[ keyTree ] += (long long)c \* sz[ keyTree ];  }  */\*询问\*/*  inline void query() {*/\*这是题目特定函数\*/*  int l , r;  scanf("%d%d",&l,&r);  RotateTo(l-1 , 0);  RotateTo(r+1 , root);  printf("%lld**\n**",sum[keyTree]);  }      */\*这是题目特定变量\*/*  int num[maxn];  int val[maxn];  int add[maxn];  long long sum[maxn];  }spt;      int main() {  int n , m;  scanf("%d%d",&n,&m);  spt.init(n);  while(m --) {  char op[2];  scanf("%s",op);  if(op[0] == 'Q') {  spt.query();  } else {  spt.update();  }  }  return 0;  } |