什么是动态树 (Dynamic Tree) 什么是 Link/cut tree

根据杨哲的论文，动态树是一类问题的总称，而解决这类问题的数据结构就是 Link/cut tree ，简称 LCT 。

根据个人理解，就是要：

1. 给你一些森林，维护他们的联通性
2. 在某棵树上的链（或链上节点）上做查询或修改。

很容易想到，如果没有第一点，那么第二点可以用[树链剖分](http://quartergeek.com/summary-of-heavy-light-decomposition)轻松解决。

Link/cut tree 的实现

原理我就不说了，可以参考上方的参考资料（Tarjan那篇不建议看）。主要说说实现。

大致上就是把某棵Splay的根节点指向这棵Splay所在实边最上方的点的父亲，也就是说，这是一条虚边。注意到这条虚边是单向的，由某棵Splay的根指向某一节点，但是那个节点的孩子里面并没有它。LCT的精髓就是通过这一条条的虚边将整棵大树串起来。

既然Splay中的根节点还有父亲，为了确定哪个节点是当前Splay的根，我们必须对正常的Splay tree进行修改。大致有两种方法：

* 在每个节点记下 Root 标记，以确定当前节点是否为它所在的Splay tree的根。然后旋转的时候通过标记下传修改Root标记。
* 优点：方便调试
* 缺点：LCT的操作写起来麻烦，而且容易出错
* 不记录Root标记，但是修改旋转过程。
* 优点：LCT**非常好写好理解**
* 缺点：需要修改splay操作

我个人还是偏向后面那种方法，后面具体说。

另外，LCT操作的实现也有两种方法：

* 固定根，通过修改过的 Access 操作 (Access2) 进行查询和修改。
* 动态根，通过换原树的根和原始的Access进行查询和修改。

很明显第一个由于少了换根，常数肯定要小的多。但是我觉得第二个理解比较容易，而且比较**好写**，**不容易出错**。

下面就介绍具体每个操作的写法。

Access(u)

这个操作是核心操作，我们保证做完之后，把u到根的路径打通成为实边，并且u的孩子节点的实边变成虚边。也就是这条Prefer Path一端是根，一端是u。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Node\* access(Node\* u)  {  Node\* v = EMPTY;  for (; u != EMPTY; u = u->parent)  {  splay(u);  u->child[1] = v;  update(v = u);  }  return v;  } |

大致的意思就是说，每次把一个节点旋到Splay的根，然后把上一次的Splay的根节点当作当前根的右孩子（也就是原树中的下方）。第一次初始 v = EMPTY 是为了清除u原来的孩子。

因为不是每次access都需要把最后节点旋到Splay的根，所以我就不在最后splay(v)了。

返回值是最后访问到的节点，也就是原树的根。

MakeRoot(x)

把x当作原树的根（因为是无向图，所以随便指定树根）。

|  |  |
| --- | --- |
| 1  2  3  4  5 | inline void makeroot(Node\* const x)  {  access(x)->rev ^= true;  splay(x);  } |

打通到原根的路径，打翻转标记（交换左右子树，也就是在原树中交换上下关系），splay清标记。

GetRoot(x)

这个操作用来得到当前Prefer Path在原树的最上方节点，常用来判断两个节点是否在同一棵树上。

|  |  |
| --- | --- |
| 1  2  3  4  5 | Node\* getroot(Node\* x)  {  for (x = access(x); clear(x), x->child[0] != EMPTY; x = x->child[0]);  return x;  } |

没什么好说的，就是旋到根，不断往左走（对应原树的往根走）。唯一要注意的是，要清除标记。

Link(x, y)

把 x 和 y 所在的子树连接起来（森林中树的连接）

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | inline void link(Node\* const x, Node\* const y)  {  makeroot(x);  x->parent = y;  access(x);  } |

最后那个access(x)可以不用，没什么影响。

Cut(x, y)

以x为树根，把y到x的路径分离出来。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | inline void cut(Node\* const x, Node\* const y)  {  makeroot(x);  access(y);  splay(y);  y->child[0]->parent = EMPTY;  y->child[0] = EMPTY;  update(y);  } |

Query(x, y)

询问x到y路径上的……以最大值为例：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | inline int query(Node\* x, Node\* y)  {  makeroot(x);  access(y), splay(y);  return y->max;  } |

Modify(x, y)

在x到y路径上做修改，以加上某一个值为例：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | inline void modify(Node\* x, Node\* y, const int w)  {  makeroot(x);  access(y), splay(y);  \_inc(y, w); //同splay中的清除标记  } |

\_splay\_parent(x, y)

看x的父亲y是否是x所在Splay的父亲。

|  |  |
| --- | --- |
| 1  2 | inline bool \_splay\_parent(Node \*x, Node\* (&y))  { return (y = x->parent) != EMPTY && (y->child[0] == x || y->child[1] == x); } |

splay过程中的终止条件就得改成这个。

例题：

后面的比较简单

[bzoj2631]tree(伍一鸣)

|  |  |
| --- | --- |
| 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  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214 | #include <cstdio>  #include <algorithm>  const int BUFSIZE = 20\*1024\*1024;  char \_stdin[BUFSIZE], \*p = \_stdin;  inline int readint()  {  char ch = \*p++;  for (; ch > '9' || ch < '0'; ch = \*p++);  int tmp = 0;  for (; '0' <= ch && ch <= '9'; ch = \*p++)  tmp = tmp\*10 + ch - '0';  return tmp;  }  inline char readchar()  {  char ch = \*p++;  for (; ch <= 32; ch = \*p++);  return ch;  }  const int MOD = 51061;  const int MAXN = 100002;  struct Edge  {  int v;  Edge \*next;  } g[MAXN\*2], \*header[MAXN];  inline void AddEdge(const int x, const int y)  {  static int LinkSize;  Edge\* const node = g+(LinkSize++);  node->v = y;  node->next = header[x];  header[x] = node;  }  struct Node  {  int value, sum, add, mul, size;  bool rev;  Node\* parent, \*child[2];  void clear\_mul(const long long m)  {  mul = mul \* m % MOD;  sum = sum \* m % MOD;  add = add \* m % MOD;  value = value \* m % MOD;  }  void clear\_add(const int a)  {  sum = (sum + static\_cast<long long>(a) \* size) % MOD;  add = add + a;  value = value + a;  }  } \_memory[MAXN], \*EMPTY = \_memory;  inline void clear(Node\* const x)  {  if (x == EMPTY) return;  if (x->rev)  {  x->child[0]->rev ^= true;  x->child[1]->rev ^= true;  std::swap(x->child[0], x->child[1]);  x->rev = false;  }  if (x->mul != 1)  {  if (x->child[0] != EMPTY) x->child[0]->clear\_mul(x->mul);  if (x->child[1] != EMPTY) x->child[1]->clear\_mul(x->mul);  x->mul = 1;  }  if (x->add)  {  if (x->child[0] != EMPTY) x->child[0]->clear\_add(x->add);  if (x->child[1] != EMPTY) x->child[1]->clear\_add(x->add);  x->add = 0;  }  }  inline void update(Node\* const x)  {  x->size = x->child[0]->size + 1 + x->child[1]->size;  x->sum = x->child[0]->sum + x->value + x->child[1]->sum;  }  inline void rotate(Node\* const x, const int c)  {  Node\* const y = x->parent;  y->child[c^1] = x->child[c];  if (x->child[c] != EMPTY) x->child[c]->parent = y;  x->parent = y->parent;  if (y->parent->child[0] == y) x->parent->child[0] = x; else  if (y->parent->child[1] == y) x->parent->child[1] = x;  y->parent = x;  x->child[c] = y;  update(y);  }  inline bool \_splay\_parent(Node \*x, Node\* (&y))  { return (y = x->parent) != EMPTY && (y->child[0] == x || y->child[1] == x); }  inline void splay(Node\* const x)  {  clear(x);  for (Node \*y, \*z; \_splay\_parent(x, y); )  if (\_splay\_parent(y, z))  {  clear(z), clear(y), clear(x);  const int c = y == z->child[0];  if (x == y->child[c]) rotate(x, c^1), rotate(x, c);  else rotate(y, c), rotate(x, c);  }  else  {  clear(y), clear(x);  rotate(x, x == y->child[0]); break;  }  update(x);  }  inline Node\* access(Node\* u)  {  Node\* v = EMPTY;  for (; u != EMPTY; u = u->parent)  {  splay(u);  u->child[1] = v;  update(v = u);  }  return v;  }  inline void makeroot(Node\* const x)  {  access(x)->rev ^= true;  splay(x);  }  inline void link(Node\* const x, Node\* const y)  {  makeroot(x);  x->parent = y;  //access(x);  }  inline void cut(Node\* const x, Node\* const y)  {  makeroot(x);  access(y), splay(y);  y->child[0]->parent = EMPTY;  y->child[0] = EMPTY;  update(y);  }  inline int getsum(Node\* const x, Node\* const y)  {  makeroot(x);  access(y), splay(y);  return y->sum % MOD;  }  inline void multiply(Node\* const x, Node\* const y, const int mul)  {  makeroot(x);  access(y), splay(y);  y->clear\_mul(mul);  }  inline void add(Node\* const x, Node\* const y, const int add)  {  makeroot(x);  access(y), splay(y);  y->clear\_add(add);  }  void maketree(int u, int parent)  {  Node\* const node = \_memory+u;  node->value = node->sum = node->size = node->mul = 1;  node->parent = \_memory+parent, node->child[0] = node->child[1] = EMPTY;  for (Edge \*e = header[u]; e; e = e->next)  if (e->v != parent)  maketree(e->v, u);  }  int n, q;  int main()  {  fread(\_stdin, 1, BUFSIZE, stdin);  EMPTY->parent = EMPTY->child[0] = EMPTY->child[1] = EMPTY;  n = readint(), q = readint();  for (int i = 1, x, y; i < n; ++i)  {  x = readint(), y = readint();  AddEdge(x, y);  AddEdge(y, x);  }  maketree(1, 0);  for (int i = 0, u, v, x, y; i < q; ++i)  {  char op;  op = readchar(), u = readint(), v = readint();  switch (op)  {  case '+':  x = readint();  add(\_memory+u, \_memory+v, x);  break;  case '-':  x = readint(), y = readint();  cut(\_memory+u, \_memory+v);  link(\_memory+x, \_memory+y);  break;  case '\*':  x = readint();  multiply(\_memory+u, \_memory+v, x);  break;  case '/':  printf("%d\n", getsum(\_memory+u, \_memory+v));  break;  }  }  } |

[bzoj2049][Sdoi2008]Cave 洞穴勘测

|  |  |
| --- | --- |
| 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 | #include <cstdio>  #include <algorithm>  const int MAXN = 10002;  struct Node  {  bool rev;  Node \*parent, \*child[2];  } \_memory[MAXN], \*EMPTY = \_memory;  inline void clear(Node\* const x)  {  if (x == EMPTY) return;  if (x->rev)  {  x->child[0]->rev ^= true;  x->child[1]->rev ^= true;  std::swap(x->child[0], x->child[1]);  x->rev = false;  }  }  void rotate(Node\* const x, const int c)  {  Node\* const y = x->parent;  y->child[c^1] = x->child[c];  if (x->child[c] != EMPTY) x->child[c]->parent = y;  x->parent = y->parent;  if (y->parent->child[0] == y) x->parent->child[0] = x; else  if (y->parent->child[1] == y) x->parent->child[1] = x;  x->child[c] = y;  y->parent = x;  }  inline bool \_splay\_parent(Node\* const x, Node\* (&y))  { return (y = x->parent) != EMPTY && (y->child[0] == x || y->child[1] == x); }  void splay(Node\* const x)  {  clear(x);  for (Node \*y, \*z; \_splay\_parent(x, y); )  if (\_splay\_parent(y, z))  {  clear(z), clear(y), clear(x);  const int c = y == z->child[0];  if (x == y->child[c]) rotate(x, c^1), rotate(x, c);  else rotate(y, c), rotate(x, c);  }  else  {  clear(y), clear(x);  rotate(x, x == y->child[0]);  }  }  Node\* access(Node\* u)  {  Node\* v = EMPTY;  for (; u != EMPTY; u = u->parent)  {  splay(u);  u->child[1] = v;  v = u;  }  return v;  }  inline Node\* getroot(Node\* x)  {  for (x = access(x); clear(x), x->child[0] != EMPTY; x = x->child[0]);  return x;  }  inline void makeroot(Node\* const x)  {  access(x)->rev ^= true;  splay(x);  }  void link(Node\* const x, Node\* const y)  {  makeroot(x);  x->parent = y;  access(x);  }  void cut(Node\* const x, Node\* const y)  {  makeroot(x);  access(y), splay(y);  y->child[0]->parent = EMPTY;  y->child[0] = EMPTY;  }  int n, m;  int main()  {  scanf("%d%d", &n, &m);  for (int i = 0; i <= n; ++i)  {  Node\* const node = \_memory+i;  node->child[0] = node->child[1] = node->parent = EMPTY;  }  for (int i = 0, x, y; i < m; ++i)  {  static char buf[10];  scanf("\n%s%d%d", buf, &x, &y);  Node \*ra, \*rb;  switch (buf[0])  {  case 'Q':  ra = getroot(\_memory+x), rb = getroot(\_memory+y);  printf(ra == rb && ra != EMPTY ? "Yes\n" : "No\n"); break;  case 'D': cut(\_memory+x, \_memory+y); break;  case 'C': link(\_memory+x, \_memory+y); break;  }  }  } |