

# Link Cut Tree

📅 2021-04-20 | 📅 2022-08-24 | 📁 OI & ACM | 👁 12

📄 6k | ⌚ 5 分钟

## 简介



对于一棵有根树的每个点连向它的儿子的所有边，我们选择一条边，令其为实边，其它边都为虚边

实边相连的儿子称为实儿子，虚边相连的儿子称为虚儿子

对于若干条相连的实边，我们用 *Splay* 维护

*LCT* 大概就是用一些 *Splay* 维护的动态剖分，但这样说并不准确

我们需要另外引入一个叫辅助树的东西，注意到在之前的定义中我们只说明了 *Splay* 用来维护实链

但原树上的虚边也需要维护，所以这些 *Splay* 之间是有联系的，而这些联系就是原树上的虚边

我们将这些联系在一起的 *Splay* 所构成的东西称作辅助树，注意到一个 *Splay* 的根节点是没有父亲节点的

但是在辅助树上一个 *Splay* 的根节点的父亲节点即为原树上这个实链的父亲节点，并且这个父亲节点并没有相应的这个儿子，换句话说叫认父不认子

大概就是这样

## 模板

### 一般模板

```

1  #define lc T[i].ch[0]
2  #define rc T[i].ch[1]
3  struct LinkCutTree {
4      int v, val, ch[2];
5      bool rev;
6  } T[maxn]; int f[maxn];
7  inline int get(int i) {
8      if (T[f[i]].ch[0] == i) return 0;
9      if (T[f[i]].ch[1] == i) return 1;
10     return -1;
11 }
12 inline void maintain(int i) { // maintain 操作需要保证 lc 或 rc 等于 0 无影响
13     T[i].v = T[i].val ^ T[lc].v ^ T[rc].v;
14 }
15 inline void setr(int i) { // 所有的标记下放操作都需要判断 i 是否为 0
16     if (!i) return ;
17     T[i].rev ^= 1; swap(lc, rc);
18 }
19 inline void push(int i) {
20     bool &rev = T[i].rev;
21     if (rev) setr(lc), setr(rc);
22     rev = 0;
23 }
24 inline void rotate(int x) {
25     int fa = f[x], ffa = f[f[x]], wx = get(x);
26     if (~get(fa)) T[ffa].ch[T[ffa].ch[1] == fa] = x;
27     f[x] = ffa; f[fa] = x; f[T[x].ch[wx ^ 1]] = fa;
28     T[fa].ch[wx] = T[x].ch[wx ^ 1]; T[x].ch[wx ^ 1] = fa;
29     maintain(fa); maintain(x);
30 }
31 void clt(int i) {
32     static int st[maxn], top;
33     st[top = 1] = i;
34     while (~get(i)) st[++top] = i = f[i];
35     while (top) push(st[top--]);
36 }
37 void Splay(int i) {
38     clt(i);
39     for (int fa = f[i]; ~get(i); rotate(i), fa = f[i])
40         if (~get(fa)) rotate(get(i) == get(fa) ? fa : i);
41 }
42 void access(int i) { for (int p = 0; i; i = f[p = i]) Splay(i), rc = p, maintain(i)
43 inline void make_rt(int i) { access(i); Splay(i); setr(i); }
44 inline void split(int x, int y) { make_rt(x); access(y); Splay(y); }
45 int find_rt(int i) { access(i); Splay(i); while (lc) i = lc; return Splay(i), i; }
46 inline void link(int x, int y) {

```

```

47     if (find_rt(x) == find_rt(y)) return ;
48     split(x, y); f[x] = y;
49 }
50 inline void cut(int x, int y) {
51     if (find_rt(x) != find_rt(y)) return ;
52     split(x, y);
53     if (T[y].ch[0] == x && !T[x].ch[1])
54         T[y].ch[0] = f[x] = 0, maintain(y);
55 }

```

## 维护生成树

对于边  $i$ , 其连接  $(u, v)$ , 我们将其当做点  $i + n$ 。需要注意总点数为  $n + m$

```

1  struct Edge {
2      int u, v, w;
3  } e[maxm];
4
5  #define lc T[i].ch[0]
6  #define rc T[i].ch[1]
7  struct LinkCutTree {
8      int v, val, ch[2];
9      bool rev;
10 } T[maxn + maxm]; int f[maxn + maxm];
11 inline int get(int i) {
12     if (T[f[i]].ch[0] == i) return 0;
13     if (T[f[i]].ch[1] == i) return 1;
14     return -1;
15 }
16 inline void maintain(int i) {
17     T[i].v = T[i].val;
18     if (e[T[lc].v].w > e[T[i].v].w) T[i].v = T[lc].v;
19     if (e[T[rc].v].w > e[T[i].v].w) T[i].v = T[rc].v;
20 }
21 inline void setr(int i) {
22     if (!i) return ;
23     T[i].rev ^= 1; swap(lc, rc);
24 }
25 inline void push(int i) {
26     bool &rev = T[i].rev;
27     if (rev) setr(lc), setr(rc);
28     rev = 0;
29 }
30 inline void rotate(int x) {

```

```

31     int fa = f[x], ffa = f[f[x]], wx = get(x);
32     if (~get(fa)) T[ffa].ch[T[ffa].ch[1] == fa] = x;
33     f[x] = ffa; f[fa] = x; f[T[x].ch[wx ^ 1]] = fa;
34     T[fa].ch[wx] = T[x].ch[wx ^ 1]; T[x].ch[wx ^ 1] = fa;
35     maintain(fa); maintain(x);
36 }
37 void clt(int i) {
38     static int st[maxn + maxm], top;
39     st[top = 1] = i;
40     while (~get(i)) st[++top] = i = f[i];
41     while (top) push(st[top--]);
42 }
43 void Splay(int i) {
44     clt(i);
45     for (int fa = f[i]; ~get(i); rotate(i), fa = f[i])
46         if (~get(fa)) rotate(get(i) == get(fa) ? fa : i);
47 }
48 void access(int i) { for (int p = 0; i; i = f[p = i]) Splay(i), rc = p, maintain(i); }
49 inline void make_rt(int i) { access(i); Splay(i); setr(i); }
50 inline void split(int x, int y) { make_rt(x); access(y); Splay(y); }
51 int find_rt(int i) { access(i); Splay(i); while (lc) i = lc; return Splay(i), i; }
52 inline void link(int x, int y) {
53     if (find_rt(x) == find_rt(y)) return ;
54     split(x, y); f[x] = y;
55 }
56 inline void cut(int x, int y) {
57     if (find_rt(x) != find_rt(y)) return ;
58     split(x, y);
59     if (T[y].ch[0] == x && !T[x].ch[1])
60         T[y].ch[0] = f[x] = 0, maintain(y);
61 }

```

## 维护子树

LCT 维护子树需要保证信息满足可减性，对于 LCT 上的每个点我们不仅需要维护

```

1  #define lc T[i].ch[0]
2  #define rc T[i].ch[1]
3  struct LinkCutTree { // sv 表示 u 的虚儿子的信息，v 表示 u 所在 Splay 上的子树的 v
4      int v, sv, val, ch[2];
5      bool rev;
6  } T[maxn]; int f[maxn];
7  void init_LCT() {
8      for (int i = 1; i <= n; ++i) T[i].val = 1;

```

```

9  }
10 inline int get(int i) {
11     if (T[f[i]].ch[0] == i) return 0;
12     if (T[f[i]].ch[1] == i) return 1;
13     return -1;
14 }
15 inline void maintain(int i) {
16     T[i].v = T[i].val + T[i].sv + T[lc].v + T[rc].v;
17 }
18 inline void setr(int i) {
19     if (!i) return ;
20     T[i].rev ^= 1; swap(lc, rc);
21 }
22 inline void push(int i) {
23     bool &rev = T[i].rev;
24     if (rev) setr(lc), setr(rc);
25     rev = 0;
26 }
27 inline void rotate(int x) {
28     int fa = f[x], ffa = f[f[x]], wx = get(x);
29     if (~get(fa)) T[ffa].ch[T[ffa].ch[1] == fa] = x;
30     f[x] = ffa; f[fa] = x; f[T[x].ch[wx ^ 1]] = fa;
31     T[fa].ch[wx] = T[x].ch[wx ^ 1]; T[x].ch[wx ^ 1] = fa;
32     maintain(fa); maintain(x);
33 }
34 void clt(int i) {
35     static int st[maxn], top;
36     st[top = 1] = i;
37     while (~get(i)) st[++top] = i = f[i];
38     while (top) push(st[top--]);
39 }
40 void Splay(int i) {
41     clt(i);
42     for (int fa = f[i]; ~get(i); rotate(i), fa = f[i])
43         if (~get(fa)) rotate(get(i) == get(fa) ? fa : i);
44 }
45 void access(int i) {
46     for (int p = 0; i; i = f[p = i]) {
47         Splay(i);
48         T[i].sv += T[rc].v;
49         T[i].sv -= T[p].v;
50         rc = p; maintain(i);
51     }
52 }
53 inline void make_rt(int i) { access(i); Splay(i); setr(i); }
54 inline void split(int x, int y) { make_rt(x); access(y); Splay(y); }

```

```

55  int find_rt(int i) { access(i); Splay(i); while (lc) i = lc; return Splay(i), i; }
56  inline void link(int x, int y) {
57      if (find_rt(x) == find_rt(y)) return ;
58      split(x, y); f[x] = y; // 这里必须 split(x, y)
59      T[y].sv += T[x].v; maintain(y);
60  }

```

## 例题

1. 简要题意：给定  $n$  个点  $m$  条边的无向图，求边权最大值和最小值差值最小的生成树，图有自环

$$n \leq 5 \times 10^4, m \leq 2 \times 10^5$$

简要题解：我们考虑钦定最大边的长度，然后求这种情况下的最小差值生成树，最终求所有情况的最小值即是答案

那么这就相当于是一个不断向最小生成树中加边，然后删掉产生的环上的最小边的过程，这个过程可以用 *LCT* 实现，时间复杂度  $O(n \log n)$

Luogu P4234 最小差值生成树

2. 简要题意：给定  $n$  个点，现在有  $m$  个操作，操作有两种，第一种操作是给定  $x, y$ ，加入一条  $(x, y)$  的无向边，保证任意时刻都为森林；第二种操作是给定  $x, y$ ，保证  $(x, y)$  这条边存在，求有多少对点的最短路径经过这条边

简要题解：容易发现操作就是连边同时维护子树 *size*，*LCT* 可以解决这些问题，时间复杂度  $O(n \log n)$

Luogu P4219 [BJOI2014]大融合

3. 简要题意：给定一个  $n$  个点  $m$  条边的无向图，现在有  $q$  次询问，每次询问给定一个区间  $[l, r]$ ，求只包含编号在  $[l, r]$  内的边的连通块的数量，强制在线

$$n, q \leq 10^5, m \leq 2 \times 10^5$$

简要题解：不妨先考虑不强制在线的情况，我们使用扫描线同时用线段树维护左端点的答案，那么如果我们加入编号为  $r$  的  $(u, v)$  这条边，它会使左端点在  $[l, r]$  的答案减一，这个  $l$  是从  $r - 1$  向前扫不断加边直至  $u$  和  $v$  连通

我们发现这个东西是可以用  $LCT$  维护的，我们只需要用  $LCT$  维护最大生成树即可求这个东西，每次不断替换环上编号最小的边

强制在线的话，我们只需要将扫描线的过程可持久化一下即可，时间复杂度  $O(n \log n)$

[Luogu P5385 \[Cnoi2019\]须臾幻境](#)

-----

本文结束  感谢阅读

-----

[# Tech](#) [# LCT](#)

[< bzoj 4154 \[lpsc2015\]Generating Synergy](#)

[Luogu P4148 简单题 >](#)

© 2020 – 2022  DDOSvoid

 1.8m |  27:07

9084 |  17815

由 [Hexo](#) & [NexT.Gemini](#) 强力驱动