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

```cpp
#define lc T[i].ch[0]
#define rc T[i].ch[1]
struct LinkCutTree {
    int v, val, ch[2];
    bool rev;
} T[maxn]; int f[maxn];
inline int get(int i) {
    if (T[f[i]].ch[0] == i) return 0;
    if (T[f[i]].ch[1] == i) return 1;
    return -1;
}
inline void maintain(int i) { // maintain 操作需要保证 lc 或 rc 等于 0 无影响
    T[i].v = T[i].val ^ T[lc].v ^ T[rc].v;
}
inline void setr(int i) { // 所有的标记下放操作都需要判断 i 是否为 0
    if (!i) return ;
    T[i].rev ^= 1; swap(lc, rc);
}
inline void push(int i) {
    bool &rev = T[i].rev;
    if (rev) setr(lc), setr(rc);
    rev = 0;
}
inline void rotate(int x) {
    int fa = f[x], ffa = f[f[x]], wx = get(x);
    if (~get(fa)) T[ffa].ch[T[ffa].ch[1] == fa] = x;
    f[x] = ffa; f[fa] = x; f[T[x].ch[wx ^ 1]] = fa;
    T[fa].ch[wx] = T[x].ch[wx ^ 1]; T[x].ch[wx ^ 1] = fa;
    maintain(fa); maintain(x);
}
void clt(int i) {
    static int st[maxn], top;
    st[top = 1] = i;
    while (~get(i)) st[++top] = i = f[i];
    while (top) push(st[top--]);
}
void Splay(int i) {
    clt(i);
    for (int fa = f[i]; ~get(i); rotate(i), fa = f[i])
        if (~get(fa)) rotate(get(i) == get(fa) ? fa : i);
}
void access(int i) { for (int p = 0; i; i = f[p = i]) Splay(i), rc = p, maintain(i
inline void make_rt(int i) { access(i); Splay(i); setr(i); }
inline void split(int x, int y) { make_rt(x); access(y); Splay(y); }
int find_rt(int i) { access(i); Splay(i); while (lc) i = lc; return Splay(i), i; }
inline void link(int x, int y) {
```

```
    if (find_rt(x) == find_rt(y)) return ;
    split(x, y); f[x] = y;
}
inline void cut(int x, int y) {
    if (find_rt(x) != find_rt(y)) return ;
    split(x, y);
    if (T[y].ch[0] == x && !T[x].ch[1])
        T[y].ch[0] = f[x] = 0, maintain(y);
}
```

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 $n + m$

```
struct Edge {
    int u, v, w;
} e[maxm];

#define lc T[i].ch[0]
#define rc T[i].ch[1]
struct LinkCutTree {
    int v, val, ch[2];
    bool rev;
} T[maxn + maxm]; int f[maxn + maxm];
inline int get(int i) {
    if (T[f[i]].ch[0] == i) return 0;
    if (T[f[i]].ch[1] == i) return 1;
    return -1;
}
inline void maintain(int i) {
    T[i].v = T[i].val;
    if (e[T[lc].v].w > e[T[i].v].w) T[i].v = T[lc].v;
    if (e[T[rc].v].w > e[T[i].v].w) T[i].v = T[rc].v;
}
inline void setr(int i) {
    if (!i) return ;
    T[i].rev ^= 1; swap(lc, rc);
}
inline void push(int i) {
    bool &rev = T[i].rev;
    if (rev) setr(lc), setr(rc);
    rev = 0;
}
inline void rotate(int x) {
```

```
    int fa = f[x], ffa = f[f[x]], wx = get(x);
    if (~get(fa)) T[ffa].ch[T[ffa].ch[1] == fa] = x;
    f[x] = ffa; f[fa] = x; f[T[x].ch[wx ^ 1]] = fa;
    T[fa].ch[wx] = T[x].ch[wx ^ 1]; T[x].ch[wx ^ 1] = fa;
    maintain(fa); maintain(x);
}
void clt(int i) {
    static int st[maxn + maxm], top;
    st[top = 1] = i;
    while (~get(i)) st[++top] = i = f[i];
    while (top) push(st[top--]);
}
void Splay(int i) {
    clt(i);
    for (int fa = f[i]; ~get(i); rotate(i), fa = f[i])
        if (~get(fa)) rotate(get(i) == get(fa) ? fa : i);
}
void access(int i) { for (int p = 0; i; i = f[p = i]) Splay(i), rc = p, maintain(i
inline void make_rt(int i) { access(i); Splay(i); setr(i); }
inline void split(int x, int y) { make_rt(x); access(y); Splay(y); }
int find_rt(int i) { access(i); Splay(i); while (lc) i = lc; return Splay(i), i; }
inline void link(int x, int y) {
    if (find_rt(x) == find_rt(y)) return ;
    split(x, y); f[x] = y;
}
inline void cut(int x, int y) {
    if (find_rt(x) != find_rt(y)) return ;
    split(x, y);
    if (T[y].ch[0] == x && !T[x].ch[1])
        T[y].ch[0] = f[x] = 0, maintain(y);
}
```

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

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

```
}
inline int get(int i) {
    if (T[f[i]].ch[0] == i) return 0;
    if (T[f[i]].ch[1] == i) return 1;
    return -1;
}
inline void maintain(int i) {
    T[i].v = T[i].val + T[i].sv + T[lc].v + T[rc].v;
}
inline void setr(int i) {
    if (!i) return ;
    T[i].rev ^= 1; swap(lc, rc);
}
inline void push(int i) {
    bool &rev = T[i].rev;
    if (rev) setr(lc), setr(rc);
    rev = 0;
}
inline void rotate(int x) {
    int fa = f[x], ffa = f[f[x]], wx = get(x);
    if (~get(fa)) T[ffa].ch[T[ffa].ch[1] == fa] = x;
    f[x] = ffa; f[fa] = x; f[T[x].ch[wx ^ 1]] = fa;
    T[fa].ch[wx] = T[x].ch[wx ^ 1]; T[x].ch[wx ^ 1] = fa;
    maintain(fa); maintain(x);
}
void clt(int i) {
    static int st[maxn], top;
    st[top = 1] = i;
    while (~get(i)) st[++top] = i = f[i];
    while (top) push(st[top--]);
}
void Splay(int i) {
    clt(i);
    for (int fa = f[i]; ~get(i); rotate(i), fa = f[i])
        if (~get(fa)) rotate(get(i) == get(fa) ? fa : i);
}
void access(int i) {
    for (int p = 0; i; i = f[p = i]) {
        Splay(i);
        T[i].sv += T[rc].v;
        T[i].sv -= T[p].v;
        rc = p; maintain(i);
    }
}
inline void make_rt(int i) { access(i); Splay(i); setr(i); }
inline void split(int x, int y) { make_rt(x); access(y); Splay(y); }
```

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

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

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

   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

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

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 [Ipsc2015]Generating Synergy

Luogu P4148 简单题 ›