

「杂题记录」「YuGu P6018」「Ynoi」Fusion tree

Jiayi Su (ShuYuMo)

2020-07-10 17:34:30

给你一棵 n 个结点的树，每个结点有权值。 m 次操作。需要支持以下操作。

- 将树上与一个节点 x 距离为 1 的节点上的权值 $+1$ 。这里树上两点间的距离定义为从一点出发到另外一点的最短路径上边的条数。
- 在一个节点 x 上的权值 $-v$ 。
- 询问树上与一个节点 x 距离为 1 的所有节点上的权值的异或和。对于 100% 的数据，满足 $1 \leq n \leq 5 \times 10^5$ ， $1 \leq m \leq 5 \times 10^5$ ， $0 \leq a_i \leq 10^5$ ， $1 \leq x \leq n$ ， $opt \in \{1, 2, 3\}$ 。保证任意时刻每个节点的权值非负。

每个结点建立一棵 trie 维护其儿子的权值，trie 应该支持全局加一。可以使用在每一个结点上设置懒标记来标记儿子的权值的增加量

全局加一（维护异或和）

这个不太好表述啊~

01-trie 数是指字符集为 $\{0, 1\}$ 的 Trie 树。

01-trie 树可以用来维护一堆数字的异或和，支持修改（删除 + 重新插入），和全部维护值加一。

如果要维护异或和，需要按值从低位到高位建立 trie。

一个约定：文中说当前节点往上指当前节点到根这条路径，当前节点往下指当前结点的子树。

插入 & 删除

如果要维护异或和，我们只需要知道某一位上 0 和 1 个数的奇偶性即可，也就是对于数字 1 来说，当且仅当这一位上数字 1 的个数为奇数时，这一位上的数字才是 1。

对于每一个节点，我们需要记录以下三个量：- $ch[o][0/1]$ 指节点 o 的两个儿子， $ch[o][0]$ 指下一位是 0，同理 $ch[o][1]$ 指下一位是 1。- $w[o]$ 指节点 o 到其父亲节点这条边上数值的数量（权值）。每插入一个数字 x ， x 二进制拆分后在 trie 树上路径的权值都会 $+1$ 。- $xorv[o]$ 指以 o 为根的子树维护的异或和。

具体维护结点的代码如下所示。

```
void maintain(int o){
    w[o] = xorv[o] = 0;
    if(ch[o][0]){ w[o] += w[ch[o][0]]; xorv[o] ^= xorv[ch[o][0]] << 1; }
    if(ch[o][1]){ w[o] += w[ch[o][1]]; xorv[o] ^= (xorv[ch[o][1]] << 1) | (w[ch[o][1]] & 1); }
    //w[o] = w[o] & 1;
    //只需知道奇偶性即可，不需要具体的值。当然这句话删掉也可以，因为上文就只利用了他的奇偶性。
}
```

插入和删除的代码非常相似。

需要注意的地方就是：

- 这里的 MAXH 指 trie 的深度，也就是强制让每一个叶子节点到根的距离为 MAXH。对于一些比较小的值，可能有时候不需要建立这么深（例如：如果插入数字 4，分解成二进制后为 100，从根开始插入 001 这三位即可），但是我们强制插入 MAXH 位。这样做的目的是为了便于全局 +1 时处理进位。例如：如果原数字是 3 (11)，++ 之后变成 4 (100)，如果当初插入 3 时只插入了 2 位，那这里的进位就没了。
- 插入和删除，只需要修改叶子节点的 w[] 即可，在回溯的过程中一路维护即可。

```
namespace trie{
    const int MAXH = 21;
    int ch[_ * (MAXH + 1)][2], w[_ * (MAXH + 1)], xorv[_ * (MAXH + 1)];
    int tot = 0;
    int mknode(){ ++tot; ch[tot][1] = ch[tot][0] = w[tot] = xorv[tot] = 0; return tot;}
    void maintain(int o){
        w[o] = xorv[o] = 0;
        if(ch[o][0]){ w[o] += w[ch[o][0]]; xorv[o] ^= xorv[ch[o][0]] << 1; }
        if(ch[o][1]){ w[o] += w[ch[o][1]]; xorv[o] ^= (xorv[ch[o][1]] << 1) | (w[ch[o][1]] & 1); }
        w[o] = w[o] & 1;
    }
    void insert(int &o, int x, int dp){
        if(!o) o = mknode();
        if(dp > MAXH) return (void)(w[o] ++);
        insert(ch[o][ x&1 ], x >> 1, dp + 1);
        maintain(o);
    }
}
```

全局加一

```
void addall(int o){
    swap(ch[o][0], ch[o][1]);
    if(ch[o][0]) addall(ch[o][0]);
    maintain(o);
}
```

不知道你能不能直接看懂呢？

我们思考一下二进制意义下 +1 是如何操作的。

我们只需要从低位到高位开始找第一个出现的 0，把它变成 1，然后这个位置后面的 1 都变成 0 即可。

下面给出几个例子感受一下。（括号内的数字表示其对应的十进制数字）

```
1000 (10) + 1 = 1001 (11)
10011 (19) + 1 = 10100 (20)
11111 (31) + 1 = 100000 (32)
10101 (21) + 1 = 10110 (22)
10000000111111(16447) + 1 = 10000001000000(16448)
```

回顾一下 w[o] 的定义：w[o] 指节点 o 到其父亲节点这条边上数值的数量（权值）。

有没有感觉这个定义有点怪呢？如果在父亲结点存储到两个儿子的这条边的边权也许会更接近于习惯。但是在这里，在交换左右儿子的时候，在儿子结点存储到父亲这条边的距离，显然更加方便。

01-trie 合并

指的是将上述的两个 01-trie 进行合并，同时合并维护的信息。

我来编写此文字的初衷就是，合并 01-trie 的文字好像比较少？其实合并 01-trie 和合并线段树的思路非常相似。可以搜索合并线段树来学习如何合并 01-trie。

其实合并 trie 非常简单，就是考虑一下我们有一个 `int marge(int a, int b)` 函数，这个函数传入两个 trie 树位于同一相对位置的结点编号，然后合并完成后返回合并完成的结点编号。

考虑怎么实现？

```
int marge(int a, int b){
    if(!a) return b; // 如果 a 没有这个位置上的结点，返回 b
    if(!b) return a; // 如果 b 没有这个位置上的结点，返回 a
    // 如果 a, b 都健在，那就把 b 的信息合并到 a 上，然后递归操作。
    // 如果需要的合并是将 a, b 合并到一棵新树上，这里可以新建结点，然后进行合并。这里的代码实现仅仅是将 b 的信息
    w[a] = w[a] + w[b];
    xorv[a] ^= xorv[b];
    //不要使用 maintain, maintain 是根据 a 的两个儿子的数值进行信息合并，而这里需要 a b 两个节点进行信息合并
    ch[a][0] = marge(ch[a][0], ch[b][0]);
    ch[a][1] = marge(ch[a][1], ch[b][1]);
    return a;
}
```

顺便说一句，其实 trie 都可以合并，换句话说，trie 合并不仅仅限于 01-trie。

Code

```
namespace trie{
    const int _n = _ * 25;
    int rt[_];
    int ch[_n][2];
    int w[_n];
    int xorv[_n];
    // w[i] is in order to save the weight of edge which is connect `i` and its `parent`.
    int tot = 0;
    void maintain(int o){
        w[o] = xorv[o] = 0;
        if(ch[o][0]){ w[o] += w[ch[o][0]]; xorv[o] ^= xorv[ch[o][0]] << 1; }
        if(ch[o][1]){ w[o] += w[ch[o][1]]; xorv[o] ^= (xorv[ch[o][1]] << 1) | (w[ch[o][1]] & 1); }
    }
    inline int mknode(){ ++tot; ch[tot][0] = ch[tot][1] = 0; w[tot] = 0; return tot; }
    void insert(int &o, int x, int dp){
        if(!o) o = mknode();
        if(dp > 20) return (void)(w[o] ++);
        //     if(dp == 0)cerr << "New start" << endl;
        //     cerr << "inserted " << (x&1) << endl;
    }
}
```

```

        insert(ch[o][ x&1 ], x >> 1, dp + 1);
        maintain(o);
//      cerr << w[o] << endl;
    }
    void erase(int o, int x, int dp){
        if(dp > 20) return (void )(w[o]--);
        erase(ch[o][ x&1 ], x >> 1, dp + 1);
        maintain(o);
    }
    void addall(int o){
        swap(ch[o][1], ch[o][0]);
        if(ch[o][0]) addall(ch[o][0]);
        maintain(o);
    }
}

int head[_];
struct edges{
    int node;
    int nxt;
}edge[_ << 1];
int tot = 0;
void add(int u, int v){
    edge[++tot].nxt = head[u];
    head[u] = tot;
    edge[tot].node = v;
}

int n, m;
int rt;
int lztar[_];
int fa[_];
void dfs0(int o, int f){
    fa[o] = f;
    for(int i = head[o]; i; i = edge[i].nxt){
        int node = edge[i].node;
        if(node == f) continue;
        dfs0(node, o);
    }
}

int V[_];
inline int get(int x){ return (fa[x] == -1 ? 0 : lztar[fa[x]]) + V[x]; }
int main()
{
#ifdef LOCAL_JUDGE
//    freopen("in.in", "r", stdin);
//    freopen("out.txt", "w", stdout);
#endif
}

```

```

#endif
// freopen("in.in", "r", stdin);
clock_t c1 = clock();

n = read(), m = read();
for(int i = 1; i < n; i++){
    int u = read(), v = read(); //read();
    add(u, v); add(rt = v, u);
}
// show(rt);
dfs0(rt, -1);
for(int i = 1; i <= n; i++) { V[i] = read(); if(fa[i] != -1)trie::insert(trie::rt[fa[i]], V[i], 0);
// puts("OK");
while(m--){
    int opt = read(), x = read(); //cerr << "opt = " << opt << endl;
// if(get(x) < 0) puts("data error");
    if(opt == 1){
        lztar[x] ++; //cerr << "add lztar[ " << x << " ]" << endl;
        if(x != rt) {
            if(fa[fa[x]])trie::erase(trie::rt[fa[fa[x]]], get(fa[x]), 0);
            V[fa[x]] ++;
            if(fa[fa[x]])trie::insert(trie::rt[fa[fa[x]]], get(fa[x]), 0);
        }
        trie::addall(trie::rt[x]);
    } else if(opt == 2){
        int v = read();
        if(x != rt) trie::erase(trie::rt[fa[x]], get(x), 0);
        V[x] -= v;
        if(x != rt) trie::insert(trie::rt[fa[x]], get(x), 0);
    } else {
        int res = 0;
        res = trie::xorv[trie::rt[x]];
        res ^= get(fa[x]);
        printf("%d\n", res);
    }
}
std::cerr << "\n\nTime: " << clock() - c1 << " ms" << std::endl;
return 0;
}

```