三元组（tree，2s，512MB）

【算法分析】

算法一：

求三个点两两距离相等三元组格式，相当于求选出三条长度相等且只有唯一公共点的路径方案数。考虑树形 DP，记状态 $f(u,i)$ 表示 u 的子树中到 u 的距离恰好为 i 的结点数，转移显然为：

$$f(u,i)=\sum_{v\in child_u} f(v,i-1)$$

记状态 $g(u,i)$ 表示在 u 的子树中选取两点，它们到其 lca 的长度均为 d，lca 到 u 的距离为 $d-i$。那么转移就只要考虑 lca 恰好为 u 的情况和 lca 不为 u 的情况：

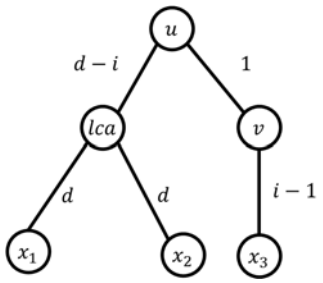
$$g(u,i)=\sum_{v\in child_u} [g(v,i+1)+f'(u,i)\times f(v,i-1)]$$

假设我们将 u 的所有儿子按一定顺序排成：$v_1,v_2,v_3,\cdots$。那么 $f'(u)$ 表示 $f(u)$ 被排在 v 之前的儿子更新后的值，下文 $g'(u)$ 的含义类似。

答案的统计需要分两种情况讨论：第一种是三个点都在路径公共点的子树内；第二种是三个点有两个在路径公共点的子树内，另一个点是公共点的祖先。

对于第一种情况，三个点都在同一个子树内，我们在公共点统计，枚举 u 的儿子 v 的时候考虑两种情况：

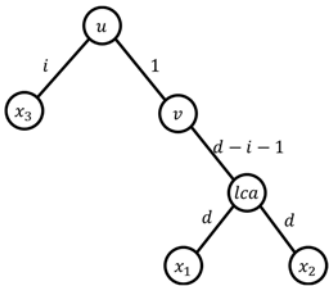
1. 有一个点在 v 的子树中，另外两个点在之前枚举的子树内，如下图：



这种情况的答案为：

$$\sum_{u=1}^{n}\sum_{v\in child_u}\sum_{i} g'(u,i)\times f(v,i-1)$$

2. 有一个点在之前枚举的子树内另外两个点在 v 的子树中，如下图：



这种情况的答案为：

$$\sum_{u=1}^{n}\sum_{v\in child_u}\sum_{i} f'(u,i)\times g(v,i+1)$$

对于第二种情况就比较简单，直接利用 g 的含义求解，我们在深度最小的点统计，这部分的答案就是 $\sum_{u=1}^{n} g(u,0)$。

所以总的答案就是

$$\sum_{u=1}^{n}\left\{g(u,0)+\sum_{v\in child_u}\sum_{i}[g'(u,i)\times f(v,i-1)+f'(u,i)\times g(v,i+1)]\right\}$$

直接计算的时间复杂度为 $O(n^2)$，期望得分 40 分。

算法二：

考虑利用长链剖分。长链剖分和重链剖分一样，也是一种树链剖分。定义一个结点的深度为它到以它为根的子树内的叶子结点的最远距离。这种方法有助于优化和深度有关的树形 DP 问题。

我们取每个点深度最大的儿子为重儿子，连接它和重儿子的边为重边。其余的儿子为轻儿子，那么除去叶子结点，其他结点有且仅有一个重儿子。

我们直接利用 u 的重儿子 $son(u)$ 的 $f(son(u))$ 右移一位后的结果作为 $f(u)$ 的初值，即 $f(u,i)=f(son(u),i-1)$。利用 $g(son(u))$ 左移一位后的结果作为 $g(u)$ 的初值，即 $g(u,i)=g(son(u),i+1)$。这两个过程可以采用将 $f(u),g(u)$ 表示成指针的方式实现，然后直接用指针的移动来 $O(1)$ 完成这个位移的过程。

接着对于 u 的其他轻儿子，我们仍然按之前的过程 DP。当我们用 v 的 DP 值来更新 u 时，枚举 i 时需要控制它在 $[0,dep(v)]$ 范围内，其中 $dep(v)$ 表示的是 v 的深度（到子树内叶子结点的最远距离）。

我们分析一下时间复杂度。定义长链是首尾相接的重边所连接的点构成的链，考虑对于每条长链，它的一整条链上的点的 DP 值都是 $O(1)$ 从儿子传递给父亲的。只有对于链顶的结点 v，它会作为 $fa(v)$ 的一个轻儿子被暴力转移 DP 值，转移的所需要的时间是 O(以 v 为链顶的长链的长度)，所以这种转移的总复杂度就是整棵树的长链的长度之和。因为每个点只会被归为一条长链，所以总时间复杂度为 $O(n)$。

类似地分析空间复杂度，长链上的点共用内存，一条长链所用内存就是 O(长链的长度)。所以总空间复杂度也是 $O(n)$ 的。

时空复杂度均为 $O(n)$，期望得分 100 分。

【参考程序】

```
//陈煜翔
#include <bits/stdc++.h>

template <class T>
inline void read(T &x)
{
    static char ch;
    while (!isdigit(ch = getchar()));
    x = ch - '0';
    while (isdigit(ch = getchar()))
        x = x * 10 + ch - '0';
}

template <class T>
```

```
inline bool relax(T &x, const T &y)
{
    return x < y ? x = y, true : false;
}

typedef long long s64;

#define trav(u) for (halfEdge *e = adj[u]; e; e = e->next)

const int MaxNV = 1e6 + 5;
const int MaxNP = 1e7 + 5;
const int MaxNE = MaxNV << 1;

struct halfEdge
{
    int v;
    halfEdge *next;
}adj_pool[MaxNE], *adj[MaxNV], *adj_tail = adj_pool;

int n;
int son[MaxNV], max_dep[MaxNV]; //max_dep[u]记录 u 到子树叶子结点的最大距离

s64 ans;
s64 pool[MaxNP], *tail = pool + MaxNV;
s64 *f[MaxNV], *g[MaxNV];

inline void addEdge(int u, int v)
{
    adj_tail->v = v;
    adj_tail->next = adj[u];
    adj[u] = adj_tail++;
}

inline void get_memory(int u) //给以 u 作为链顶的长链分配内存
{
    f[u] = tail, tail += max_dep[u] << 1 | 1; //因为一个左移一个右移
    g[u] = tail, tail += max_dep[u] << 1 | 1; //所以要分配两倍内存
}

inline void dfs1(int u, int pre)
{
    trav(u) if (e->v != pre)
    {
        dfs1(e->v, u);
```

```
        if (relax(max_dep[u], max_dep[e->v] + 1))
            son[u] = e->v; //更新重儿子
    }
}

inline void dfs2(int u, int pre)
{
    if (son[u])
    {
        f[son[u]] = f[u] + 1;
        g[son[u]] = g[u] - 1; //将重儿子和 u 用指针移动实现共用内存
        dfs2(son[u], u);      //先递归重儿子
    }
    ans += g[u][0];

    f[u][0] = 1;
    trav(u) if (e->v != pre && e->v != son[u])
    {
        get_memory(e->v);

        dfs2(e->v, u);

        for (int i = 0; i <= max_dep[e->v]; ++i)
        {
            ans += 1LL * f[e->v][i] * g[u][i + 1];
            if (i) ans += 1LL * g[e->v][i] * f[u][i - 1];
        }

        for (int i = 0; i <= max_dep[e->v]; ++i)
        {
            g[u][i + 1] += 1LL * f[u][i + 1] * f[e->v][i];
            f[u][i + 1] += f[e->v][i];
            if (i) g[u][i - 1] += g[e->v][i];
        }
    }
}

int main()
{
    freopen("tree.in", "r", stdin);
    freopen("tree.out", "w", stdout);

    read(n);
    for (int i = 1; i < n; ++i)
```

```
    {
        int u, v;
        read(u), read(v);
        addEdge(u, v), addEdge(v, u);
    }

    dfs1(1, 0);
    get_memory(1);
    dfs2(1, 0);

    std::cout << ans << std::endl;

    return 0;
}
```

字符串求值(string,2s,512MB)

【算法分析】

对于一个字符串 S 的 fail 树，树中一个结点 i 对应 S 的一个前缀 S[1:i]，结点 i 的深度相当于 S[1:i]中满足 S[1:j]=S[i-j+1:i](j<i)的 j 的数量，则 F(S)就是 S 所有前缀中满足条件的 j 的数量之和。

考虑 G(S)的含义，我们从 S 中任取两个相同的子串 S_1,S_2，它们在原串中对应的区间分别为$[l_1,r_1]$和$[l_2,r_2]$($l_1<l_2$)。对于每个子串 $S[l_1:r]$($r_2\le r\le|S|$)，S_1,S_2 都会作为它的前缀 $S[l_1:r_2]$中一个满足条件的 j（上文提到的 j）贡献 1，所以 S_1,S_2 对 G(S)的贡献就为$|S|-r_2+1$。

题目并没有要求强制在线，考虑先求出右端点为 i 的子串的 F 值之和，最后再求一遍前缀和就得到了每次添加一个字符的 G 值。

因为涉及到选相同的子串，我们用后缀自动机建出 Parent 树，考虑每次在末尾加入一个字符产生的贡献。设现在整个串 S 在 Parent 树中的点为 x，那么点 x 到根的路径上的所有点就可以代表 S 的所有后缀。这里新产生的每一个后缀都能作为我们上文所说的 S_2，与原来和它相同的子串配对并产生 1 的贡献。具体来说，我们记 Parent 树上的结点 u 原来的 Right 集合大小为 cnt[u]，所能表示字符串的最大长度为 maxl[u]，那么后缀产生的贡献就是:

$$\sum cnt[u]\cdot(\max l[u]-\max l[\,fa[u]])$$

（其中 u 是点 x 到根的路径上的点）

显然我们可以先建出整个串的 Parent 树，那么 maxl[u]-maxl[fa[u]]是已知的，只需要维护 cnt[u]，即每次把点 x 到根的路径上的所有点的 cnt[u]都加一，这很容易用树链剖分通过打区间加标记实现。对于产生的贡献，我们在线段树中预处理 maxl[u]-maxl[fa[u]]的区间和，在标记下传时维护一下区间产生的贡献，同样用树链剖分求即可。

对于 S_2 不是现在 S 的后缀的情况，实际上我们之前已经求过了，我们只要把每次后缀产生的贡献记下来，求一遍前缀和，就能得到固定右端点的子串的 F 值之和了。

时间复杂度$O(n\log^2 n)$。

这道题实际上也有在线的做法，需要动态维护 Parent 树，涉及到动态加边和删边，可以

用 LCT 实现，代码难度较大。

【参考程序】

```
//陈贤
#include <bits/stdc++.h>

template <class T>
inline void read(T &res)
{
    char ch; bool flag = false; res = 0;
    while (ch = getchar(), !isdigit(ch) && ch != '-');
    ch == '-' ? flag = true : res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
    flag ? res = -res : 0;
}

template <class T>
inline void put(T x)
{
    if (x > 9) put(x / 10);
    putchar(x % 10 + 48);
}

const int N = 4e5 + 5;
const int L = 16e5 + 5;
const int mod = 1e9 + 7;
int tpos[N], sze[N], son[N], pos[N], idx[N], top[N], fa[N], dep[N];
int sum[L], tag[L], ans[L], fans[N];
int n, T, now, E; char s[N];

inline void add(int &x, int y)
{
    x += y;
    x >= mod ? x -= mod : 0;
}

struct Edge
{
    int to; Edge *nxt;
}p[N], *lst[N], *P = p;

inline void Link(int x, int y)
{
    (++P)->nxt = lst[x]; lst[x] = P; P->to = y;
```

```
}

struct SAM
{
    int ch[26], par, maxl;

    #define par(x) tr[x].par
    #define mx(x) tr[x].maxl

    inline void Clear()
    {
        memset(ch, 0, sizeof(ch));
        par = maxl = 0;
    }
}tr[N];

inline void insertSAM(char ch, int id)
{
    int c = ch - 'a', i = now, j, p;
    tr[now = ++T].Clear();
    tpos[id] = now;
    mx(now) = mx(i) + 1;
    for (; i && !tr[i].ch[c]; i = par(i))
        tr[i].ch[c] = now;

    if (!i)
        par(now) = 1;
    else
    {
        j = tr[i].ch[c];
        if (mx(i) + 1 == mx(j))
            par(now) = j;
        else
        {
            tr[p = ++T] = tr[j];
            mx(p) = mx(i) + 1;
            par(now) = par(j) = p;
            for (; i && tr[i].ch[c] == j; i = par(i))
                tr[i].ch[c] = p;
        }
    }
}

inline void Dfs1(int x)
```

```
{
    dep[x] = dep[fa[x]] + 1;
    sze[x] = 1;
    for (Edge *e = lst[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (y == fa[x])
            continue;
        fa[y] = x;
        Dfs1(y);
        sze[x] += sze[y];
        sze[y] > sze[son[x]] ? son[x] = y : 0;
    }
}

inline void Dfs2(int x)
{
    if (son[x])
    {
        pos[son[x]] = ++E;
        idx[E] = son[x];
        top[son[x]] = top[x];
        Dfs2(son[x]);
    }

    int y;
    for (Edge *e = lst[x]; e; e = e->nxt)
        if (!top[y = e->to])
        {
            pos[y] = ++E;
            idx[E] = y;
            top[y] = y;
            Dfs2(y);
        }
}

inline void Init()
{
    Dfs1(1);
    pos[1] = top[1] = idx[1] = E = 1;
    Dfs2(1);
}

#define sL s << 1
```

```
#define sR s << 1 | 1

inline void addTag(int s, int v)
{
    add(tag[s], v);
    ans[s] = (ans[s] + 1ll * sum[s] * v) % mod;
}

inline void Update(int s)
{
    ans[s] = ans[sL];
    add(ans[s], ans[sR]);
}

inline void downDate(int s)
{
    if (tag[s])
    {
        addTag(sL, tag[s]);
        addTag(sR, tag[s]);
        tag[s] = 0;
    }
}

inline void Build(int s, int l, int r)
{
    if (l == r)
        return (void)(sum[s] = mx(idx[l]) - mx(par(idx[l])));
    int mid = l + r >> 1;
    Build(sL, l, mid); Build(sR, mid + 1, r);
    sum[s] = sum[sL];
    add(sum[s], sum[sR]);
}

inline int Query(int s, int l, int r, int x, int y)
{
    if (l == x && r == y)
        return ans[s];
    downDate(s);

    int mid = l + r >> 1;
    if (y <= mid)
        return Query(sL, l, mid, x, y);
    else if (x > mid)
```

```
        return Query(sR, mid + 1, r, x, y);
    else
    {
        int res = Query(sL, l, mid, x, mid);
        add(res, Query(sR, mid + 1, r, mid + 1, y));
        return res;
    }
}

inline void Modify(int s, int l, int r, int x, int y)
{
    if (l == x && r == y)
        return addTag(s, 1);
    downDate(s);

    int mid = l + r >> 1;
    if (y <= mid)
        Modify(sL, l, mid, x, y);
    else if (x > mid)
        Modify(sR, mid + 1, r, x, y);
    else
    {
        Modify(sL, l, mid, x, mid);
        Modify(sR, mid + 1, r, mid + 1, y);
    }
    Update(s);
}

inline void pathModify(int x)
{ //维护 x 到根路径上的 right 集合大小
    while (top[x] != 1)
        Modify(1, 1, E, pos[top[x]], pos[x]), x = fa[top[x]];
    Modify(1, 1, E, 1, pos[x]);
}

inline int pathQuery(int x)
{ //计算后缀产生的贡献
    int res = 0;
    while (top[x] != 1)
        add(res, Query(1, 1, E, pos[top[x]], pos[x])), x = fa[top[x]];
    add(res, Query(1, 1, E, 1, pos[x]));
    return res;
}
```

```
int main()
{
    freopen("string.in", "r", stdin);
    freopen("string.out", "w", stdout);

    scanf("%d%s", &n, s + 1);
    tr[now = T = 1].Clear();
    for (int i = 1; i <= n; ++i)
        insertSAM(s[i], i);

    for (int i = 2; i <= T; ++i)
        Link(par(i), i);
    Init();
    Build(1, 1, E);

    for (int i = 1; i <= n; ++i)
    {
        fans[i] = fans[i - 1];
        add(fans[i], pathQuery(tpos[i]));
        pathModify(tpos[i]);
    }
    for (int i = 1; i <= n; ++i)
        add(fans[i], fans[i - 1]);
    for (int i = 1; i <= n; ++i)
        put(fans[i]), putchar('\n');

    fclose(stdin); fclose(stdout);
    return 0;
}
```

仙人题(cac,2s,512MB)

【算法分析】

看到「将任意可能出现在这两点的简单路径上的点的值加上v」，我们很容易想到圆方树。

圆方树就是把原图中的点看作圆点，对于图的每个点双连通分量建一个方点，每个方点向其点双连通分量包含的所有圆点分别连一条边，形成一棵树，即圆方树。

圆方树有一个性质：如果原图中w可以出现在u到v的简单路径上，那么一定存在一个方点x使得x与w间有边。

也就是说，修改相当于对于圆方树u到v的路径上：

（1） 所有的圆点权值加v。

（2） 如果圆点w不在圆方树u到v的路径上，但u到v的路径上的点与w的距离最小值为1，则圆点w的权值加v。

发现这样的修改无法直接维护。我们不妨引入一个新的数组a_u，初始全0，且满足圆点u的权值为$a_u + \sum_{(u,v)} a_v$。这样我们就能把对圆点的修改转化为对方点的修改了。而对方点

的修改就只需考虑圆方树上u到v的路径。

如果把圆方树u到v的路径上所有的方点的a加上v，那么所有不在圆方树u到v的路径上的点权都满足了等式，u和v也满足了等式。但如果圆点w位于u到v的路径上且不为u和v，那么「圆方树u到v的路径上所有的方点的a加上v」这个操作会使得w的权值加上2v。故我们还要把所有这样的w权值减掉v。这样就完成了修改。

上面提到的修改都可以用树剖+线段树来实现，但查询单点权值时需要计算$a_u + \sum_{(u,v)} a_v$的值，这好像不能直接实现。考虑如何求$\sum_{(u,v)} a_v$。

我们可以把上式中的v分为三种：v是u的父亲、重儿子、轻儿子。

前两者可以直接查询。对于轻儿子，我们维护b_u表示u所有轻儿子的a之和。由于树剖进行路径操作时经过的轻边不超过$O(\log n)$条，故可以在修改的过程中，若遇到一条边(u,v)是轻边（u是v的父亲），则处理b_u的变化。复杂度$O(n + q\log^2 n)$。

【参考程序】

```
// 陈栉旷
#include <algorithm>
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cstdio>
#include <cctype>
#include <vector>
#include <ctime>
#include <cmath>
#include <map>

template <class T>
inline void read(T &res)
{
    char ch; bool flag = false; res = 0;
    while (ch = getchar(), !isdigit(ch) &&ch != '-');
    ch == '-' ? flag = true : res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
    flag ? res = -res : 0;
}

template <class T>
inline void put(T x)
{
    if (x > 9) put(x / 10);
    putchar(x % 10 + 48);
}

using std::vector;
const int mod = 998244353;
```

```
const int N = 1e6 + 5;
const int M = 4e6 + 5;
int dfn[N], low[N], stk[N];
int dep[N], pos[N], fa[N], son[N];
int sze[N], top[N], idx[N];
int val[N], sum1[M], sum2[M], tag1[M], tag2[M];
int n, m, Q, E, C, tis, top_s; bool col[M];
vector<int> v[N];

template <class T>
inline void CkMin(T &x, T y) {x > y ? x = y : 0;}

struct Edge
{
    int to; Edge *nxt;
}p[M], *lst[N], *P = p;

inline void Link(int x, int y)
{
    (++P)->nxt = lst[x]; lst[x] = P; P->to = y;
    (++P)->nxt = lst[y]; lst[y] = P; P->to = x;
}

inline void Tarjan(int x, int Fa)
{
    dfn[x] = low[x] = ++tis;
    stk[++top_s] = x; int y, w;
    for (int i = 0, im = v[x].size(); i<im; ++i)
    {
        y = v[x][i];
        if (y == Fa)
            continue;
        if (!dfn[y])
        {
            Tarjan(y, x);
            CkMin(low[x], low[y]);

            if (dfn[x] <= low[y])
            {
                ++C;
                Link(x, C);
                while (w = stk[top_s], stk[top_s + 1] != y)
                    Link(w, C), --top_s;
            }
```

```
        }
        else
            CkMin(low[x], dfn[y]);
    }
}

#define sL s << 1
#define sR s << 1 | 1

inline void add(int &x, int y)
{
    x += y;
    x >= mod ? x -= mod : 0;
}

inline void addTag1(int s, int v)
{
    if (col[s]) add(sum1[s], v);
    add(tag1[s], v);
}

inline void addTag2(int s, int v)
{
    if (!col[s]) add(sum2[s], v);
    add(tag2[s], v);
}

inline void downDate(int s)
{
    if (tag1[s])
    {
        addTag1(sL, tag1[s]);
        addTag1(sR, tag1[s]);
        tag1[s] = 0;
    }
    if (tag2[s])
    {
        addTag2(sL, tag2[s]);
        addTag2(sR, tag2[s]);
        tag2[s] = 0;
    }
}

inline void Build(int s, int l, int r)
```

```
{
    if (l == r)
        return (void)(col[s] = idx[l] > n);

    int mid = l + r >> 1;
    Build(sL, l, mid); Build(sR, mid + 1, r);
}

inline void Modify(int s, int l, int r, int x, int y, int v)
{
    if (l == x && r == y)
        return addTag1(s, v), addTag2(s, mod - v);
    downDate(s);

    int mid = l + r >> 1;
    if (y <= mid)
        Modify(sL, l, mid, x, y, v);
    else if (x > mid)
        Modify(sR, mid + 1, r, x, y, v);
    else
    {
        Modify(sL, l, mid, x, mid, v);
        Modify(sR, mid + 1, r, mid + 1, y, v);
    }
}

inline void Modify2(int s, int l, int r, int x, int v)
{
    if (l == r)
        return add(sum2[s], v);
    downDate(s);

    int mid = l + r >> 1;
    x <= mid ? Modify2(sL, l, mid, x, v) : Modify2(sR, mid + 1, r, x, v);
}

inline int Query1(int s, int l, int r, int x)
{
    if (l == r)
        return sum1[s];
    downDate(s);

    int mid = l + r >> 1;
    return x <= mid ? Query1(sL, l, mid, x) : Query1(sR, mid + 1, r, x);
```

```
}

inline int Query2(int s, int l, int r, int x)
{
    if (l == r)
        return sum2[s];
    downDate(s);

    int mid = l + r >> 1;
    return x <= mid ? Query2(sL, l, mid, x) : Query2(sR, mid + 1, r, x);
}

inline void Dfs1(int x)
{
    sze[x] = 1;
    dep[x] = dep[fa[x]] + 1;
    for (Edge *e = lst[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (y == fa[x])
            continue;
        fa[y] = x;
        Dfs1(y);
        sze[x] += sze[y];
        sze[y] >sze[son[x]] ? son[x] = y : 0;
    }
}

inline void Dfs2(int x)
{
    if (son[x])
    {
        pos[son[x]] = ++E;
        idx[E] = son[x];
        top[son[x]] = top[x];
        Dfs2(son[x]);
    }

    int y;
    for (Edge *e = lst[x]; e; e = e->nxt)
        if (!top[y = e->to])
        {
            pos[y] = ++E;
            idx[E] = y;
```

```
            top[y] = y;
            Dfs2(y);
        }
}

inline void Init()
{
    Dfs1(1);
    idx[1] = top[1] = pos[1] = E = 1;
    Dfs2(1);
}

inline void pathModify(int x, int y, int v)
{
    while (top[x] != top[y])
    {
        if (dep[top[x]] < dep[top[y]])
            std::swap(x, y);
        if (top[x] > n)
            add(val[fa[top[x]]], v);
        Modify(1, 1, C, pos[top[x]], pos[x], v);
        x = fa[top[x]];
    }
    if (dep[x] < dep[y])
        std::swap(x, y);
    if (y == top[y] && top[y] > n && fa[top[y]])
        add(val[fa[top[y]]], v);
    Modify(1, 1, C, pos[y], pos[x], v);
}

int main()
{
    freopen("cac.in", "r", stdin);
    freopen("cac.out", "w", stdout);

    read(n); read(m); read(Q);
    for (int i = 1, x, y; i<= m; ++i)
    {
        read(x); read(y);
        v[x].push_back(y);
        v[y].push_back(x);
    }

    C = n;
```

```
    Tarjan(1, 0);
    Init();
    Build(1, 1, C);

    int k, x, y, v;
    for (int i = 1; i<= Q; ++i)
    {
        read(k);
        if (!k)
        {
            read(x); read(y); read(v);
            pathModify(x, y, v);
            Modify2(1, 1, C, pos[x], v);
            Modify2(1, 1, C, pos[y], v);
        }
        else
        {
            read(x);
            int res = val[x];
            add(res, Query2(1, 1, C, pos[x]));
            if (fa[x])
                add(res, Query1(1, 1, C, pos[fa[x]]));
            if (son[x])
                add(res, Query1(1, 1, C, pos[son[x]]));
            put(res), putchar('\n');
        }
    }

    fclose(stdin); fclose(stdout);
    return 0;
}
```