```cpp
//
//   There is a tree with N vertices, numbered 1,2, ... N.
//   For each i (1 ⩽ i ⩽ N-1), the i-th edge connects Vertex x[i] and y[i].
//   Taro has decided to paint each vertex in white or black,
//   so that any black vertex can be reached from any other black vertex
//   by passing through only black vertices.
//   You are given a positive integer M. For each v (1 ⩽ v ⩽ N), answer the following question:
//     - Assuming that Vertex v has to be black,
//       find the number of ways in which the vertices can be painted, modulo M.
//
//     Time Complexity: O(N+M)
//

#include <bits/stdc++.h>
#define ll long long
using namespace std;

int n;
ll MOD;

vector<ll> in, out, excepts;
vector<vector<int>> adj;

// ans[node] = in[node] * out[node]
int dfs_in(int node, int parent = -1) {
    int size = adj[node].size();
    vector<ll> ans_child(size, 1);
    vector<ll> dp_pref(size, 1), dp_suf(size, 1);
    for (int i = 0; i < size; i++) {
        int next_node = adj[node][i];
        if (next_node == parent) {
            dp_pref[i] = (i-1 >= 0 ? dp_pref[i-1] : 1);
            continue;
        }

        int black_ways = dfs_in(next_node, node);
        ans_child[i] = (black_ways + 1) % MOD;
        in[node] = (in[node] * (black_ways + 1) % MOD) % MOD;
        dp_pref[i] = ((i-1 >= 0 ? dp_pref[i-1] : 1) * (black_ways + 1) % MOD) % MOD;
    }

    if (size > 1) {
        // calculate the suffixes
        dp_suf[size-1] = ans_child[size-1];
        for (int i = size - 2; i >= 0; i--) {
            dp_suf[i] = (dp_suf[i+1] * ans_child[i]) % MOD;
        }

        // except the i-th child
        for (int i = 0; i < size; i++) {
            int child = adj[node][i];
            if (child == parent) continue;

            ll except =
                ((i-1 >= 0 ? dp_pref[i-1]: 1) *
                (i+1 < size ? dp_suf[i+1] : 1)) % MOD;

            excepts[child] = except;
        }
    }

    return in[node];
}

void dfs_out(int node, int parent = -1) {
    int size = adj[node].size();
    for (int i = 0; i < size; i++) {
        int child = adj[node][i];
        if (child == parent) continue;

        if (parent == -1) out[child] = (excepts[child] + 1) % MOD;
        else out[child] = ((excepts[child] * out[node] % MOD) + 1) % MOD;

        dfs_out(child, node);
    }
}

int main() {
    cin >> n >> MOD;
    in.resize(n, 1);
    out.resize(n, 1);
    adj.resize(n);
    excepts.resize(n, 1);
    for (int i = 0; i < n-1; i++) {
        int x, y;
        cin >> x >> y;
        x--; y--;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }

    dfs_in(0); dfs_out(0);
    // root node doesn't have out[root]
    cout << in[0] % MOD << "\n";
    for (int i = 1; i < n; i++) {
        cout << (in[i] * out[i]) % MOD << "\n";
    }
    return 0;
}
```