```cpp
#include <algorithm>
#include <bitset>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <deque>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <stack>
#include <string>
#include <utility>
#include <vector>
#include <list>
#define For(i,a,b) for(int i=(a); i<=(b) ; i++)
#define _For(i,a,b) for(int i=(a); i>=(b) ; i--)
#define Memset(a,b); memset((a),(b),sizeof((a)));
using namespace std;
const int INF = 0x3f3f3f3f;
typedef  long long LL;typedef  unsigned long long ULL;typedef  long double LDB;
inline LL CinLL(){LL x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return x*f;}
inline int Cin(){int x=0,f=1;char ch=getchar();while(!isdigit(ch)){if(ch=='-')f=-1;ch=getchar();}while(isdigit(ch))x=x*10+ch-'0',ch=getchar();return f*x;}
const int N = 1e4+5;
int n,k,ans;
struct Edge{
    int to,val;
};
vector<Edge>edge[N];
int subtree_size[N];
int book[N];
int ds[N],n_ds;
int tds[N],n_tds;
void init()
{
    int x,y,v;Edge tmp;
    For(i,1,n-1)
    {
        x = Cin();y = Cin();v = Cin();
        tmp.to = y;tmp.val = v;
        edge[x].push_back(tmp);
        tmp.to = x;
        edge[y].push_back(tmp);
    }
}
void csh()
{
    Memset(book,0);
    Memset(subtree_size,0);
    For(i,1,n) edge[i].clear();
    ans = 0;
}
int count_subtree_size(int now,int pre)
{
    subtree_size[now] = 1;
    For(i,0,edge[now].size()-1)
    {
        int _to = edge[now][i].to;
        if(_to == pre || book[_to]) continue;
        subtree_size[now] += count_subtree_size(_to,now);
    }
    return subtree_size[now];
}
int found_center(int now,int pre,int half)
 {
    For(i,0,edge[now].size()-1)
    {
        int _to = edge[now][i].to;
        if(_to == pre || book[_to]) continue;
        if(subtree_size[_to] > half) return found_center(_to,now,half);
    }
    return now;
}
void count_subtree_path(int now,int pre,int path)
{
    tds[++n_tds] = path;
    For(i,0,edge[now].size()-1)
    {
        int _to = edge[now][i].to;
        if(_to == pre || book[_to]) continue;
        count_subtree_path(_to,now,path + edge[now][i].val);
    }
}
int count_pairs(int d[],int siz)
{
    sort(d+1,d+siz+1);
    int res = 0;
    For(i,1,siz)
    {
        int c = (k - d[i]);
        int j = upper_bound(d+1,d+siz+1,c) - d-1;
        if(j > siz) j = siz;
        if(j > i)
        res+=(j - i);
    }

    return res;
}
void solve_subtree(int now)
{
    count_subtree_size(now,-1);
    now = found_center(now,-1,subtree_size[now]/2);
    book[now] = 1;
    For(i,0,edge[now].size()-1)
    {
        int _to = edge[now][i].to;
        if(book[_to]) continue;
        solve_subtree(_to);
    }
    n_ds = 0;
    ds[++n_ds] = 0;
    For(i,0,edge[now].size()-1)
    {
        int _to = edge[now][i].to;
        if(book[_to]) continue;
        n_tds = 0;
        count_subtree_path(_to,now,edge[now][i].val);
        int t = count_pairs (tds,n_tds);
        ans -= t;
        For(i,1,n_tds)
        ds[++n_ds] = tds[i];
    }
    int t = count_pairs(ds,n_ds);
    ans+=t;
    book[now] = 0;
}
int main()
{
    while(scanf("%d%d",&n,&k)!=EOF && n && k)
    {
        init();
        solve_subtree(1);
        printf("%d\n",ans);
        csh();
    }
}
```