D. Distance in Tree

time limit per test: 3 seconds
memory limit per test: 512 megabytes

A tree is a connected graph that doesn't contain any cycles.

The distance between two vertices of a tree is the length (in edges) of the shortest path between these vertices.

You are given a tree with n vertices and a positive number k. Find the number of distinct pairs of the vertices which have a distance of exactly k between them. Note that pairs (v, u) and (u, v) are considered to be the same pair.

Input

The first line contains two integers n and k ($1 \le n \le 50000$, $1 \le k \le 500$) — the number of vertices and the required distance between the vertices.

Next $n - 1$ lines describe the edges as "a_i b_i" (without the quotes) ($1 \le a_i, b_i \le n$, $a_i \ne b_i$), where a_i and b_i are the vertices connected by the i-th edge. All given edges are different.

Output

Print a single integer — the number of distinct pairs of the tree's vertices which have a distance of exactly k between them.

Please do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams or the `%I64d` specifier.

Examples

input Copy

```
5 2
1 2
2 3
3 4
2 5
```

output Copy

```
4
```

input Copy

```
5 3
1 2
2 3
3 4
4 5
```

output Copy

```
2
```

Note

In the first sample the pairs of vertexes at distance 2 from each other are (1, 3), (1, 5), (3, 5) and (2, 4).

You are given:

- A tree with n nodes.
- You need to count how many pairs of nodes (u, v) have distance exactly = k.

Distance means number of edges on the shortest path between them.

(1,3) → 2 edges
(1,5) → 2 edges
(3,5) → 2 edges
(2,4) → 2 edges

DP idea

We can use DP on trees to count pairs efficiently.

dp[v][d] = number of nodes in subtree of v
that are at distance d from v

dp[v][0] = 1 (the node itself)

dp[v][1] = number of children of v

dp[v][2] = nodes 2 edges away within v's subtree, and so on.

Algorithm outline

1. Choose any node as root (say, 1).

2. Use DFS to visit nodes.

3. For each node:
   - Recursively calculate dp[child][*].
   - Combine child's dp into parent's dp while counting valid pairs.

4. Return total pairs.

What are we counting again?

We want to count pairs of nodes (u, v) such that:

the distance(u, v) = k

Now, since this is a tree, there is exactly one simple path between any two nodes.

That's important — no duplicates from multiple paths.

Combining children = counting pairs

Suppose we are at node v, and we already did DFS for all its children.

We now have:

dp[child1][*], dp[child2][*], … for all children.

We want to count pairs (a, b) such that:

- a is in subtree of one child
- b is in subtree of another child (or maybe v itself)
- and distance(a, b) = k

Code

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 50005;
const int MAXK = 505;

vector<int> adj[MAXN];
int n, k;
long long ans = 0;
int dp[MAXN][MAXK];

void dfs(int v, int p) {
    dp[v][0] = 1; // itself

    for (int to : adj[v]) {
        if (to == p) continue;

        dfs(to, v);

        // count pairs going through v
        for (int d = 0; d < k; d++) {
            ans += (long long)dp[v][k - d - 1] * dp[to][d];
        }

        // merge child's dp into parent dp
        for (int d = 0; d < k; d++) {
            dp[v][d + 1] += dp[to][d];
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n >> k;
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    dfs(1, 0);
    cout << ans << "\n";
    return 0;
}
```

Complexity

Each node has at most k distances to track.
For each edge, we do O(k) merging.

So overall:
O(N * K)

which is fine since N ≤ 5e4 and K ≤ 500.