**4:The xor-longest Path**

* [查看](http://dsa.openjudge.cn/final2018test/4/)
* [提交](http://dsa.openjudge.cn/final2018test/4/submit/)
* [统计](http://dsa.openjudge.cn/final2018test/4/statistics/)
* [提问](http://dsa.openjudge.cn/final2018test/clarify/4/)

总时间限制:

2000ms

内存限制:

65536kB

**描述**

In an edge-weighted tree, the xor-length of a path *p*is defined as the xor sum of the weights of edges on *p*:

http://media.openjudge.cn/images/2975.gif

⊕ is the xor operator.

We say a path the xor-longest path if it has the largest xor-length. Given an edge-weighted tree with n nodes, can you find the xor-longest path?

**输入**

The input contains several test cases. The first line of each test case contains an integer *n*(1<=*n*<=100000), The following *n*-1 lines each contains three integers *u*(0 <= *u* < *n*),*v*(0 <= *v* < *n*),*w*(0 <= *w* < 2^31), which means there is an edge between node *u* and *v* of length *w*.

**输出**

For each test case output the xor-length of the xor-longest path.

**样例输入**

4

0 1 3

1 2 4

1 3 6

**样例输出**

7

**提示**

The xor-longest path is 0->1->2, which has length 7 (=3 ⊕ 4)

#include < algorithm >  
#include < cstdio >  
#include < cstring >  
#include < vector >  
using namespace std;  
  
const int MAXN = 100010;  
const int MAXM = 200010;  
struct Trie {  
 Trie \*children[2];  
 int value;  
} root, pool[MAXN << 5], \*ptr, emptyNode;  
// Trie树是一棵二叉树，左孩子表示当前二进制位为0，右孩子表示当前二进制位为1  
// 叶子节点额外保存value，表示当前路径对应的数，如根-左-右-右这个节点的value为(011)2=3  
  
int n;  
// 节点个数  
  
int head[MAXN], succeed[MAXM], vertex[MAXM], weight[MAXM], now;  
// head,succeed使用链表保存边表；vertex,weight存每条边的终点以及权值  
  
int rootXor[MAXN];  
// 每个点到根的路径的xor值  
  
void addEdge(int u, int v, int w) {  
 // 往链表中添加一条u->v，权值为w的边  
 \_\_\_\_\_\_\_\_(1)\_\_\_\_\_\_\_\_  
 // 新的节点的后继为u节点原来的链表头  
 vertex[now] = v;  
 // 新的边的终点为v  
 weight[now] = w;  
 // 新的边的权值为w  
 head[u] = now++;  
 // 指定u节点新的链表头为now  
}  
  
void readTree() {  
 // 读入树  
 memset(head, -1, sizeof head);  
 now = 0;  
 for (int i = 0; i < n - 1; i++) {  
 int u, v, w;  
 scanf("%d%d%d", &u, &v, &w);  
 addEdge(u, v, w);  
 addEdge(v, u, w);  
 }  
}  
  
void DFS(int x = 0, int father = -1) {  
 // DFS求出每个点到0号点的路径的xor值，并填到rootXor数组中  
 // 参数x表示当前递归到的节点，father表示当前点的父亲  
 for (int now = head[x]; now != -1; now = succeed[now]) {  
 // 遍历x节点的链表，找到所有x的出边  
 int y = vertex[now];  
 int w = weight[now];  
 // 从x到y有一条权值为w的边  
 if (\_\_\_\_\_\_\_\_(2)\_\_\_\_\_\_\_\_)  
 continue;  
 // 忽略返回父亲的边  
 rootXor[y] = \_\_\_\_\_\_\_\_(3)\_\_\_\_\_\_\_\_;  
 // 计算rootXor[y]的值，为y到0号点的路径的xor值  
 DFS(y, x);  
 }  
}  
  
int getNthBit(int value, int nBit) {  
 return value >> nBit & 1;  
}  
  
void insertTrie(Trie \*node, int value, int nBit) {  
 // 往Trie中插入value  
 // 当前在node这个节点，处理到value的第nBit个bit  
 // 如果nBit为-1，说明已经处理完毕，node为叶子节点  
 if (nBit == -1)  
 node->value = \_\_\_\_\_\_\_\_(4)\_\_\_\_\_\_\_\_;  
 else {  
 bool bit = \_\_\_\_\_\_\_\_(5)\_\_\_\_\_\_\_\_;  
 // bit为0说明当前的节点应为node的左孩子  
 // bit为1说明当前的节点应为node的右孩子  
 if (!node->children[bit]) {  
 node->children[bit] = ptr++;  
 \*node->children[bit] = emptyNode;  
 }  
 insertTrie(node->children[bit], value, nBit - 1);  
 }  
}  
  
void buildTrie() {  
 // 将所有rootXor[i]插入Trie中  
 root = emptyNode;  
 ptr = pool;  
 for (int i = 0; i < n; i++)  
 insertTrie(&root, rootXor[i], 30);  
}  
  
int queryTrie(Trie \*node, int value, int nBit) {  
 // 在Trie中查询与value的xor值最大的答案  
 // 当前在node这个节点，处理到value的第nBit个bit  
 // 如果nBit为-1，说明已经处理完毕，node为叶子节点  
 if (nBit == -1)  
 return node->value ^ value;  
 else {  
 bool bit = \_\_\_\_\_\_\_\_(5)\_\_\_\_\_\_\_\_;  
 // 同上  
 if (node->children[!bit])  
 return queryTrie(node->children[!bit], value, nBit - 1);  
 else  
 return queryTrie(node->children[bit], value, nBit - 1);  
 }  
}  
  
int getAns() {  
 // 将所有rootXor[i]在Trie中查询最大的xor答案  
 int ans = 0;  
 for (int i = 0; i < n; i++)  
 ans = max(ans, queryTrie(&root, rootXor[i], 30));  
 return ans;  
}  
  
int main() {  
 while (scanf("%d", &n) == 1) {  
 readTree();  
 DFS();  
 buildTrie();  
 printf("%d\n", getAns());  
 }  
 return 0;  
}