二叉树遍历

题目描述：

根据给定的二叉树结构描述字符串，输出该二叉树按照中序遍历结果字符串。中序遍历顺序为：左子树，根结点，右子树。

输入描述：

由大小写字母、左右大括号、逗号组成的字符串：

1、字母代表一个节点值，左右括号内包含该节点的子节点。

2、左右子节点使用逗号分隔，逗号前为空则表示左子节点为空，没有逗号则表示右子节点为空。

3、二叉树节点数最大不超过 100。

注：输入字符串格式是正确的，无需考虑格式错误的情况。

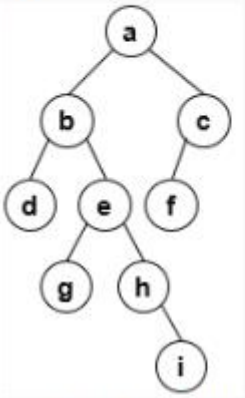
输出描述：

输出一个字符串，为二叉树中序遍历各节点值的拼接结果。

补充说明：
中序遍历是二叉树遍历的一种，遍历方式是首先遍历左子树，然后访问根结点，最后遍历右子树。

示例

示例1
输入：a{b{d,e{g,h{,i}}},c{f}}
输出：dbgehiafc
说明：



中序遍历，首先遍历左子树，再访问根节点，最后遍历右子树，比如：
a有左子树，访问其左子树
b有左子树，访问其左子树
d没有左子树，读取值"d"
b的左子树已经访问，读取值"b"，再访问其右子树
e有左子树，访问其左子树
g没有左子树，读取其值"g"
e的左子树已经访问，读取值"e"，再访问其右子树
依次类推.....

```
const r1 = require("readline").createInterface({ input: process.stdin });
var iter = r1[Symbol.asyncIterator]();
const readline = async () => (await iter.next()).value;

void (async function () {
    function TreeNode(val, left, right) {
        this.val = val === undefined ? 0 : val;
        this.left = left === undefined ? null : left;
        this.right = right === undefined ? null : right;
    }
    line = await readline();
    const dfs = (str) => {
        if (!str) {
            return null;
        }
        const root = new TreeNode(str[0]);
        if (str.length === 1) {
            return root;
        }
        let leaf = str.slice(2, -1);
        const stack = [];
        let i = 0;
```

```
        for (; i < leaf.length; i++) {
            if (stack.length === 0 && leaf[i] === ",") {
                break;
            }
            if (leaf[i] === "{") {
                stack.push("{");
            }
            if (leaf[i] === "}") {
                stack.pop();
            }
        }
        root.left = dfs(leaf.slice(0, i));
        root.right = dfs(leaf.slice(i + 1));
        return root;
    };
    const head = dfs(line);
    const res = [];
    const inorder = (root) => {
        if (!root) {
            return;
        }
        inorder(root.left);
        res.push(root.val);
        inorder(root.right);
    };
    inorder(head);
    console.log(res.join(""));
})();
```