字符串化繁为简

题目描述：

给定一个输入字符串，字符串只可能由英文字母（'a'～'z'、'A'~'Z'）和左右小括号（'('、')'）组成。当字符串里存在小括号时，小括号是成对的，可以有一个或多个小括号对，小括号对不会嵌套，小括号对内可以包含 1 个或多个英文字母，也可以不包含英文字母。当小括号对内包含多个英文字母时，这些字母之间是相互等效的关系，而且等效关系可以在不同的小括号对之间传递，即当存在'a'和'b'等效和存在'b'和'c'等效时，'a'和'c'也等效，另外，同一个英文字母的大写字母和小写字母也相互等效（即使它们分布在不同的括号对里）

需要对这个输入字符串做简化，输出一个新的字符串，输出字符串里只需保留输入字符串里的没有被小括号对包含的字符 （按照输入字符串里的字符顺序），并将每个字符替换为在小括号对里包含的且字典序最小的等效字符。

如果简化后的字符串为空，请输出为"0"。

示例：

输入字符串为"never(dont)give(run)up(f)()"，初始等效字符集合为('d', 'o', 'n', 't')、('r', 'u', 'n')，由于等效关系可以传递，因此最终等效字符集合为('d', 'o', 'n', 't', 'r', 'u')，将输入字符串里的剩余部分按字典序最小的等效字符替换后得到"devedgivedp"

输入描述：

input_string

输入为 1 行，代表输入字符串

输出描述：

output_string

输出为 1 行，代表输出字符串

补充说明：

输入字符串的长度在 1~100000 之间

示例 1

输入：

```
()abd
```

输出：

```
abd
```

说明：

输入字符串里没有被小括号包含的子字符串为"abd"，其中每个字符没有等效字符，输出为"abd"

示例 2

输入：

```
(abd)demand(fb)()for
```

输出：

```
aemanaaor
```

说明：

等效字符集为('a', 'b', 'd', 'f')，输入字符串里没有被小括号包含的子字符串集合为"demandfor"，将其中字符替换为字典序最小的等效字符后输出为："aemanaaor"

示例 3

输入：

```
()happy(xyz)new(wxy)year(t)
```

输出：

```
happwnewwear
```

说明：

等效字符集为('x', 'y', 'z', 'w'),输入字符串里没有被小括号包含的子字符串集合为"happynewyear"，将其中字符替换为字典序最小的等效字符后输出为："happwnewwear"

示例 4

输入：

```
()abcdefgAC(a)(Ab)(C)
```

输出：

```
AAcdefgAC
```

说明：

等效字符集为('a', 'A', 'b')，输入字符串里没有被小括号包含的子字符串集合为"abcdefgAC"，将其中字符替换为字典序最小的等效字符后输出为："AAcdefgAC"

```
import java.util.Scanner;
```

```
import java.util.LinkedList;
import java.util.TreeSet;


public class Main {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		String s = sc.nextLine();

		StringBuilder sb = new StringBuilder();

		LinkedList<TreeSet<Character>> list = new LinkedList<>();



		Boolean flag = false;
		for(int i = 0 ; i < s.length(); i++){
			char c = s.charAt(i);

			if(c == '('){
				flag = true;
				list.add(new TreeSet<>());
			}else if(c == ')'){
				flag = false;
				if(list.getLast().size() == 0){
					list.removeLast();
				}
			}else{
				if(!flag){
					sb.append(c);
				}else{
					list.getLast().add(c);
				}
			}
		}

		outer:
		while(true){
			for(int i = 0 ; i < list.size(); i++){
				for(int j = i+1 ; j < list.size(); j++){
					if(mode(list.get(i),list.get(j))){
						list.get(i).addAll(list.get(j));
						list.remove(j);
						continue outer;
					}
```

```
                }
            }
            break;
        }

        char[] charm = sb.toString().toCharArray();

        for(TreeSet<Character> a : list){
            Character first = a.first();
            for(int i = 0 ; i < charm.length; i++){
                if(a.contains(charm[i])){
                    charm[i] = first;
                }
            }
        }

        String str = new String(charm);

        System.out.println(str.length() == 0 ? "0" : str);
    }

    public static boolean mode(TreeSet<Character> a, TreeSet<Character> b){
        for(char c = 'a' ; c <= 'z'; c++){
            char x = (char)(c-32);
            boolean m = a.contains(c) || a.contains(x);
            boolean n = b.contains(c) || b.contains(x);
            if(m && n){
                return true;
            }
        }
        return false;
    }
}
```