

慕课网《玩转算法面试》

# 玩儿转算法面试

讲师：liuyubobobo

版权所有，侵权必究

liuyubobobo

慕课网《玩转算法面试》

# 栈和队列的使用

讲师：lilyubobobo

版权所有，侵权必究

慕课网《玩转算法面试》

# 栈 Stack

讲师：lilyubobobo

版权所有，侵权必究

慕课网《玩转算法面试》

# 栈的基础使用

讲师：lucyqobobo

版权所有，侵权必究

## 20. Valid Parentheses



facebook



Bloomberg



ZENEfits

## 20. Valid Parentheses

给定一个字符串，只包含 (, [, {, ), ], }，判定字符串中的括号匹配是否合法。

- 如 "()", "()[]{}" 是合法的
- 如 "[()", "([)]" 是非合法的

## 20. Valid Parentheses

{ [ ( ) ] }

**Stack**





# 20. Valid Parentheses

{ [ ( ) ] }



**Stack**





## 20. Valid Parentheses

{ [ ( ) ] }



**Stack**



# 20. Valid Parentheses

{ [ ( ) ] }



**Stack**



# 20. Valid Parentheses

{ [ ( ) ] }



**Stack**



# 20. Valid Parentheses

{ [ ( ) ] }



**Stack**



# 20. Valid Parentheses

{ [ ( ) ] }



Stack



## 20. Valid Parentheses

{ [ } ]

**Stack**



## 20. Valid Parentheses

{ [ } ]



**Stack**





# 20. Valid Parentheses

{ [ } ]



**Stack**



## 20. Valid Parentheses

{ [ } ]



Stack



栈顶元素反映了在嵌套的层次关系中，**最近的**需要匹配的元素

慕课网《玩转算法面试》

# 实践：解决20

讲师：liuyubobobo

版权所有，侵权必究

# 150. Evaluate Reverse Polish Notation



逆波兰表达式求值。给定一个数组，表示一个逆波兰表达式。求其值。

- 如：[ "2", "1", "+", "3", "\*" ], 表示  $(2+1)*3 = 9$
- 如：[ "4", "13", "5", "/", "+" ], 表示  $4+(13/5) = 6$

# 150. Evaluate Reverse Polish Notation



逆波兰表达式求值。给定一个数组，表示一个逆波兰表达式。求其值。

- 运算的种类 ( $+$ ,  $-$ ,  $*$ ,  $/$ )
- 字符串表达的数字种类 (整数)

# 71. Simplify Path

The Facebook logo, consisting of the word "facebook" in white lowercase letters on a blue rectangular background.

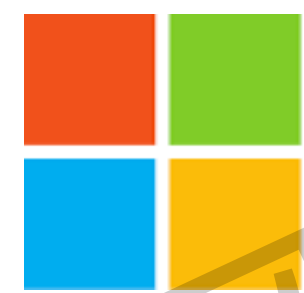
Microsoft

给定一个Unix系统下的路径，简化这个路径。

- 如 `/home/`，简化后为 `/home`
- 如 `/a/./b/../../c/`，简化后为 `/c`



# 71. Simplify Path



Microsoft

给定一个Unix系统下的路径，简化这个路径。

- 这个路径是否一定合法？
- 不能回退的情况？（如 `/../`，返回 `/`）
- 多余的`/`？（如 `/home//hello/`，返回 `/home/hello`）



慕课网《玩转算法面试》

# 栈和递归的紧密关系

讲师：huayupobobo

版权所有，侵权必究

# 递归算法

慕课网《玩转算法面试》

讲师：liuyubobobo

二叉树中的算法

版权所有，侵权必究

144. Binary Tree Preorder Traversal

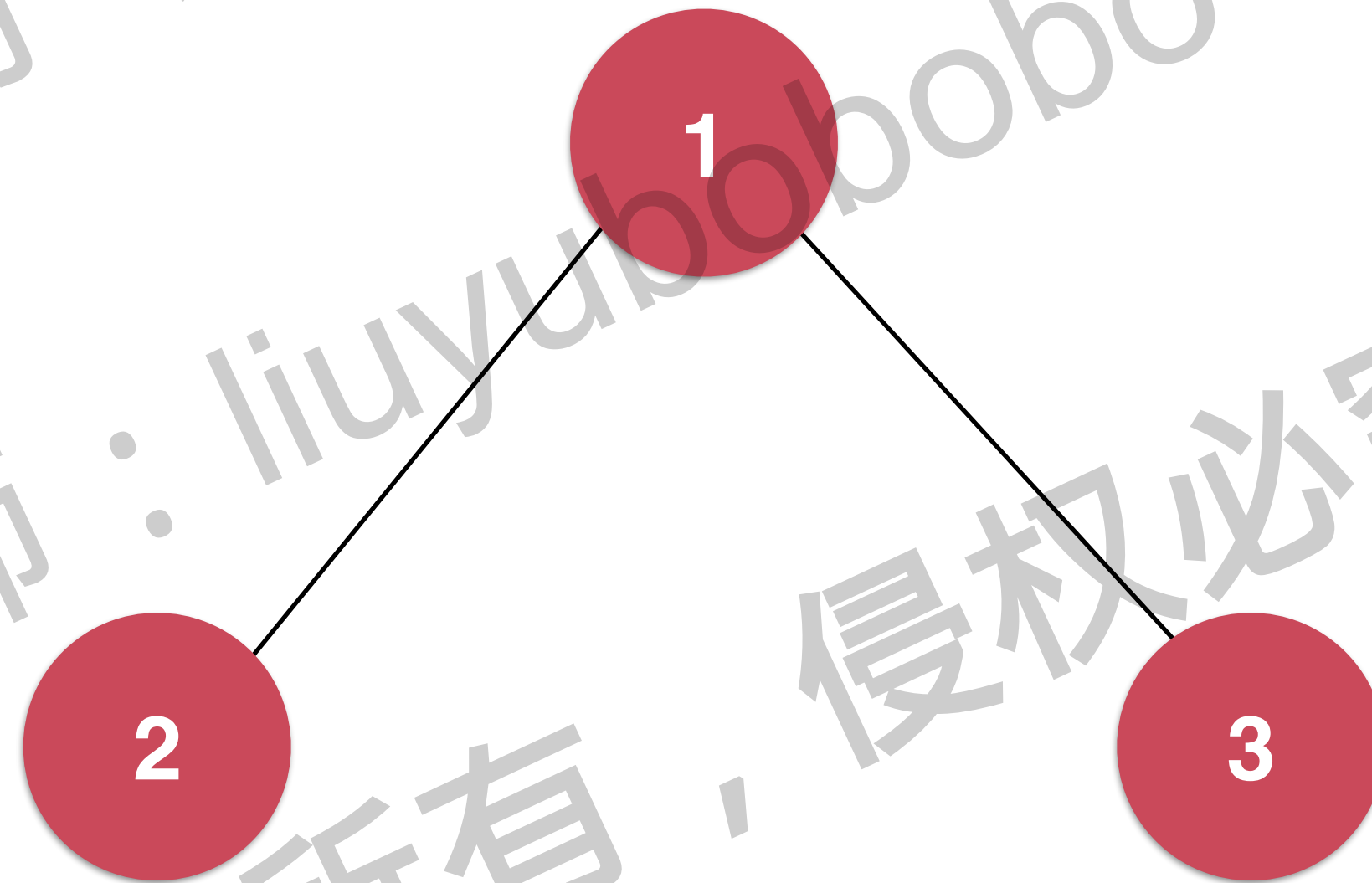
94. Binary Tree Inorder Traversal

145. Binary Tree Postorder Traversal

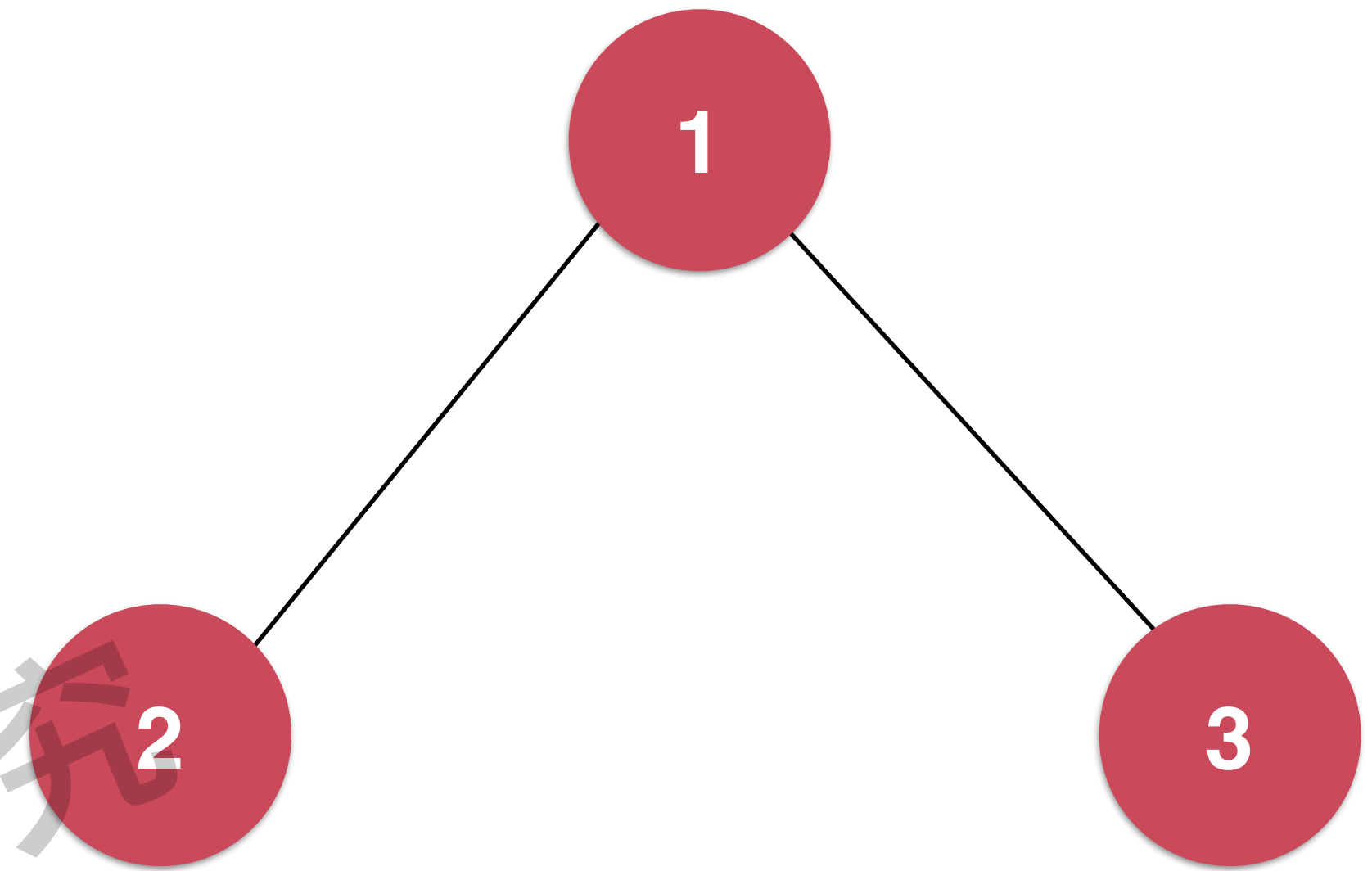
# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

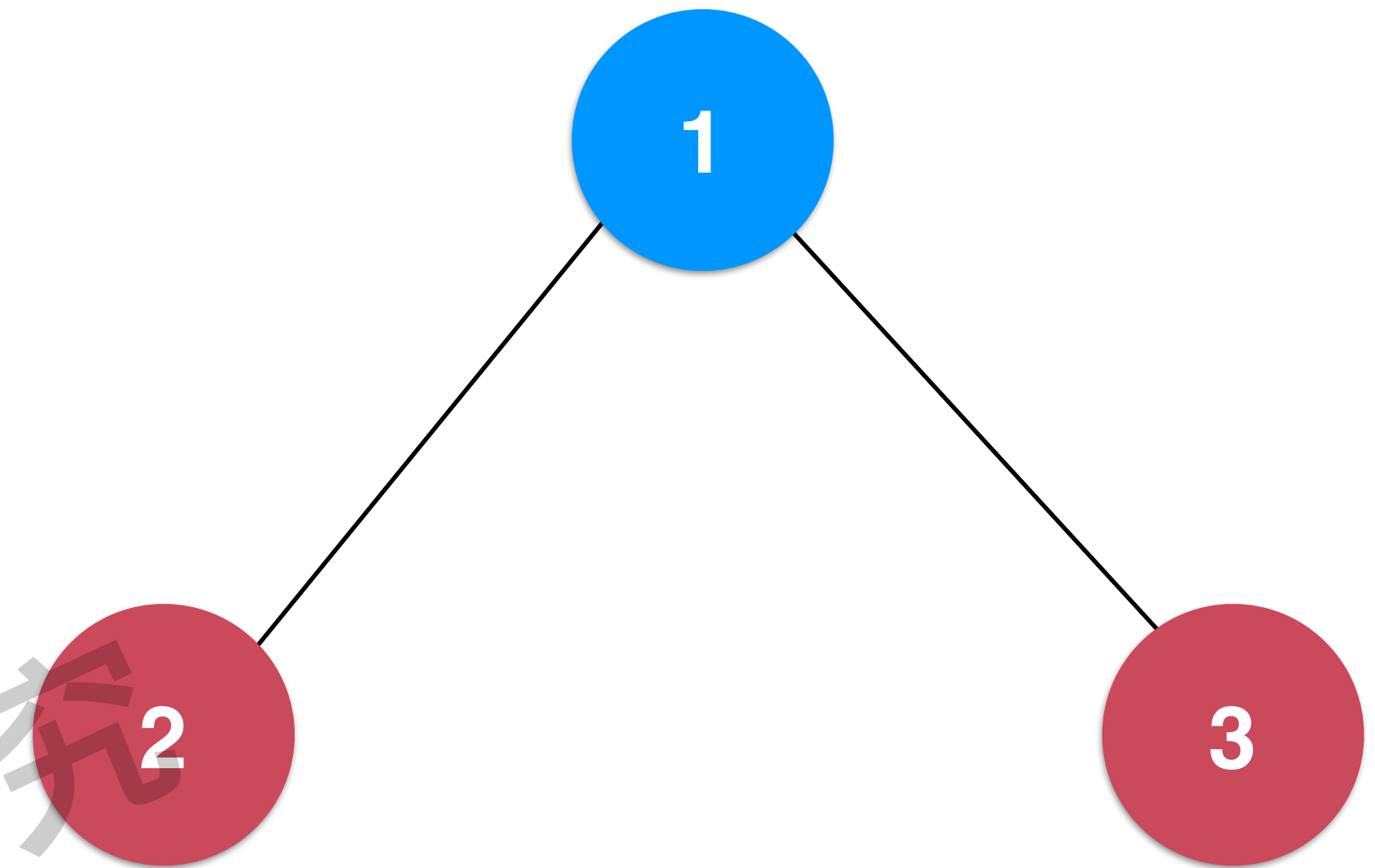


# 144. Binary Tree Preorder Traversal



# 144. Binary Tree Preorder Traversal

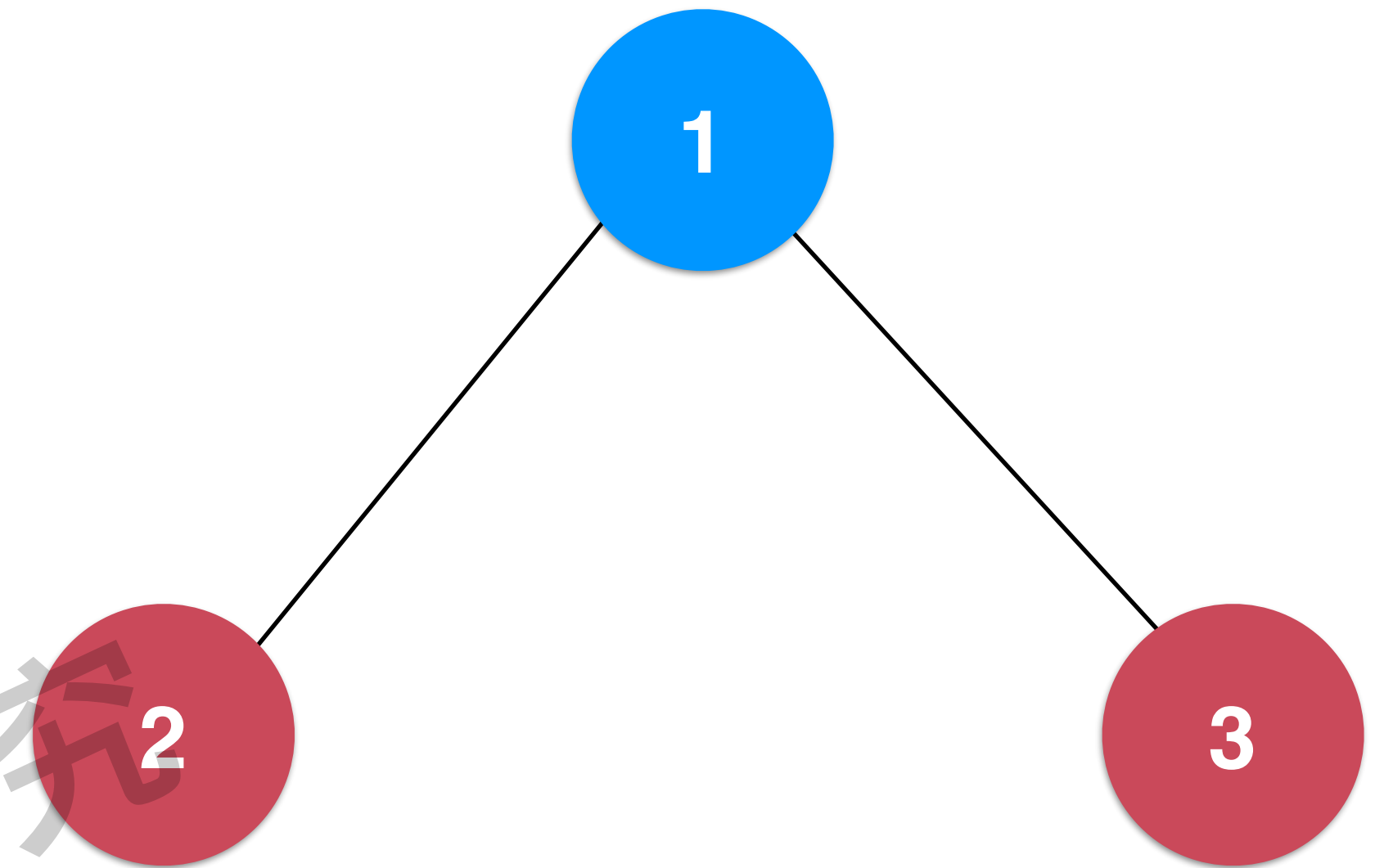
```
▶ void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```





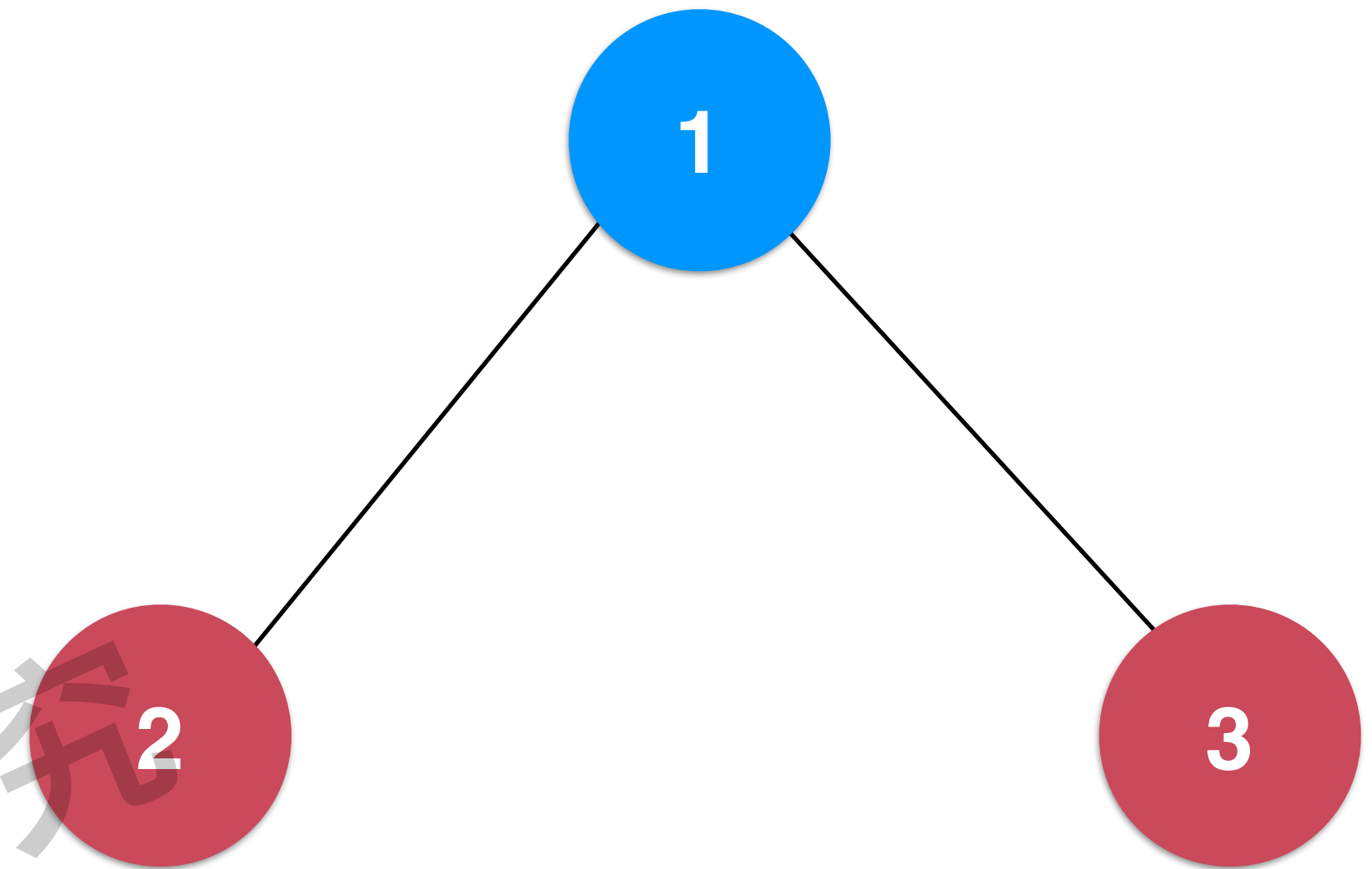
# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



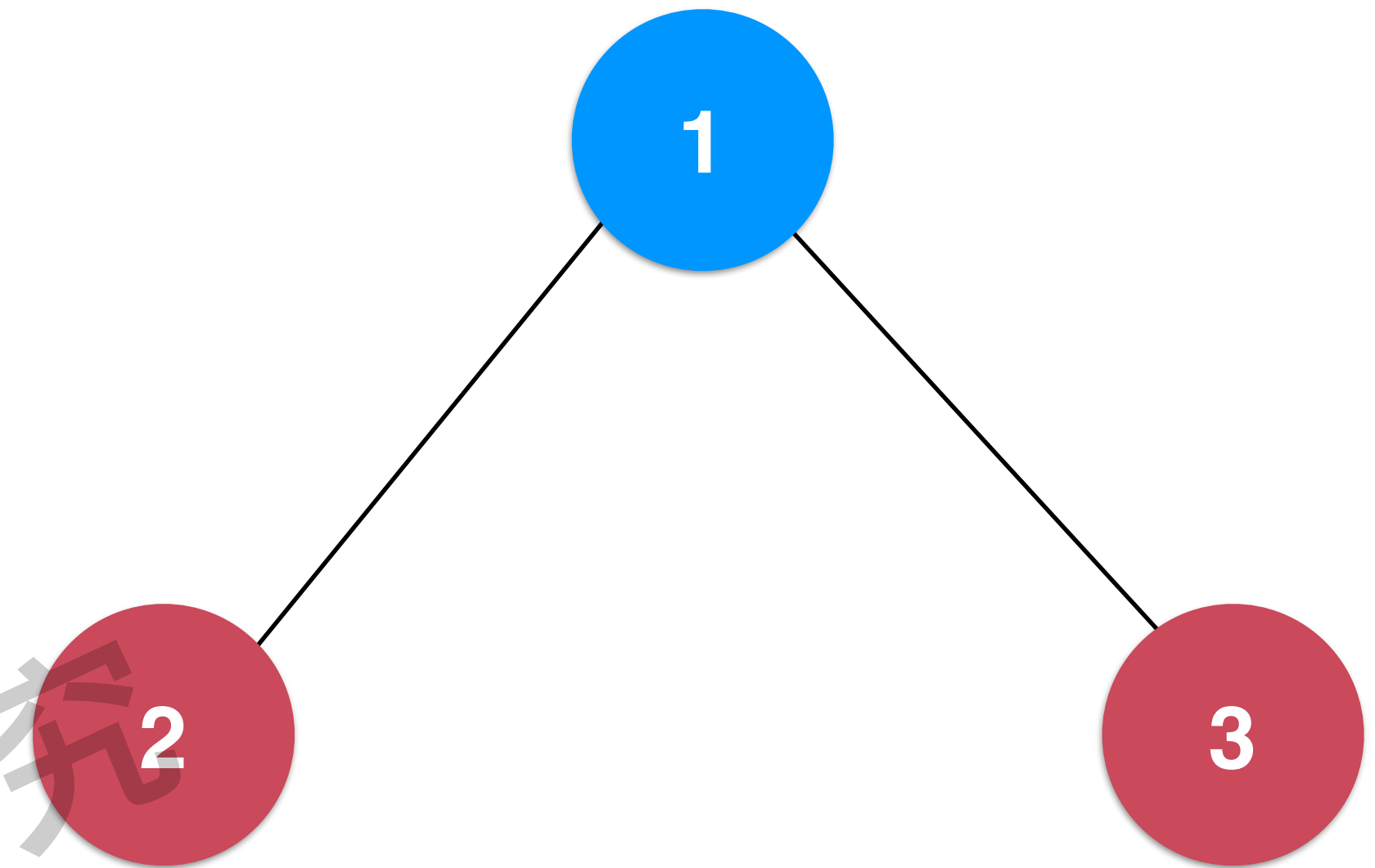
# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

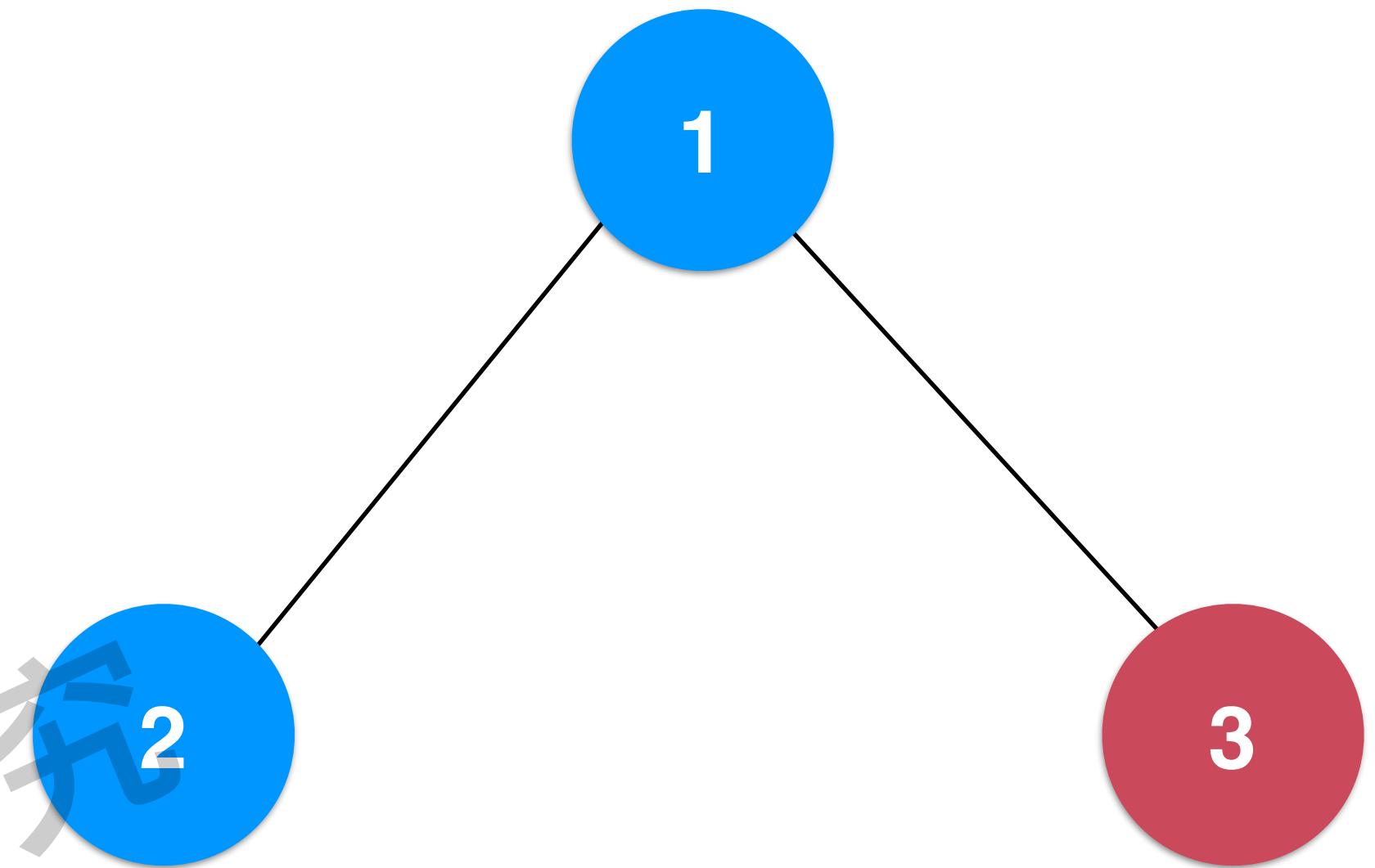
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

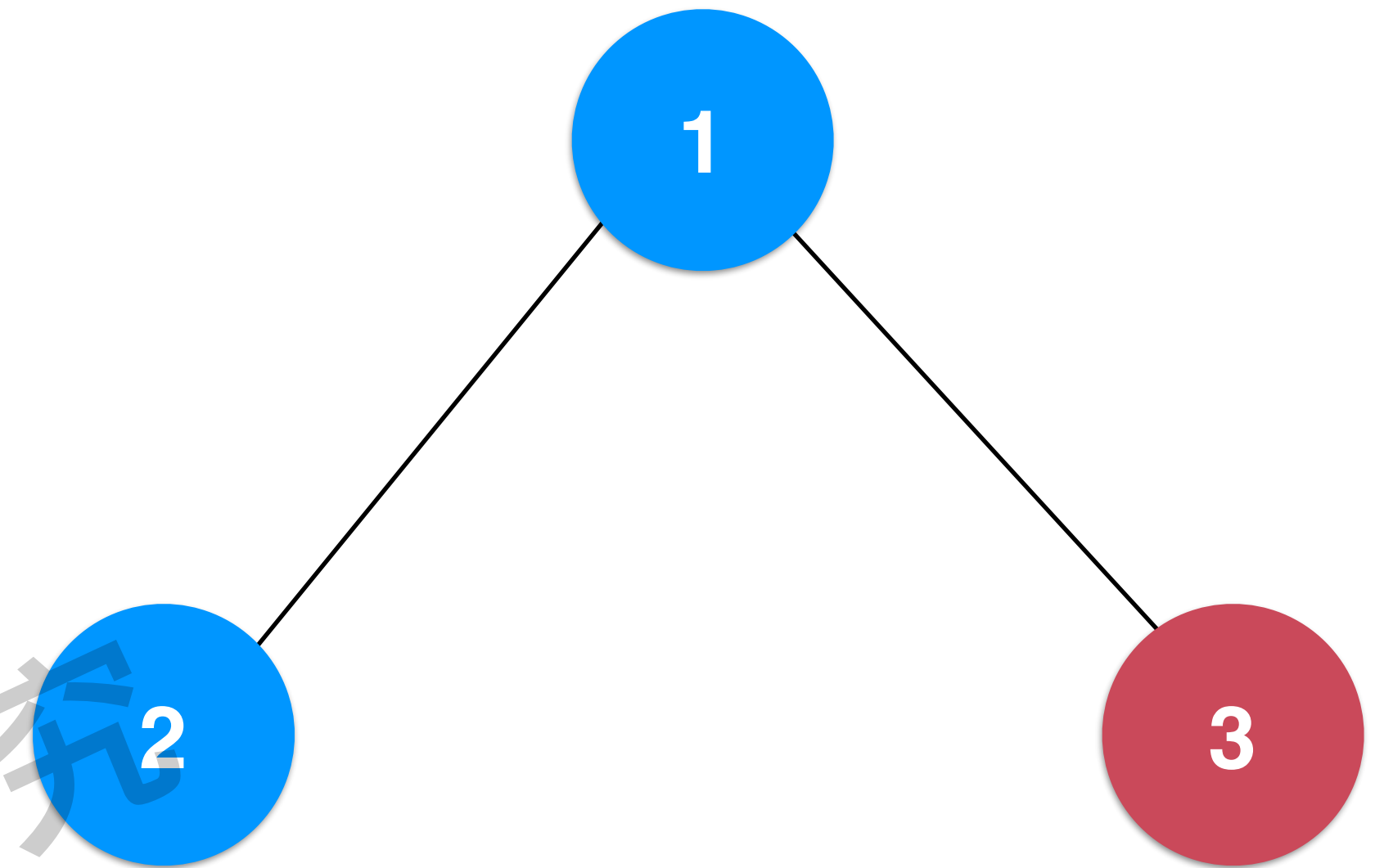


```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

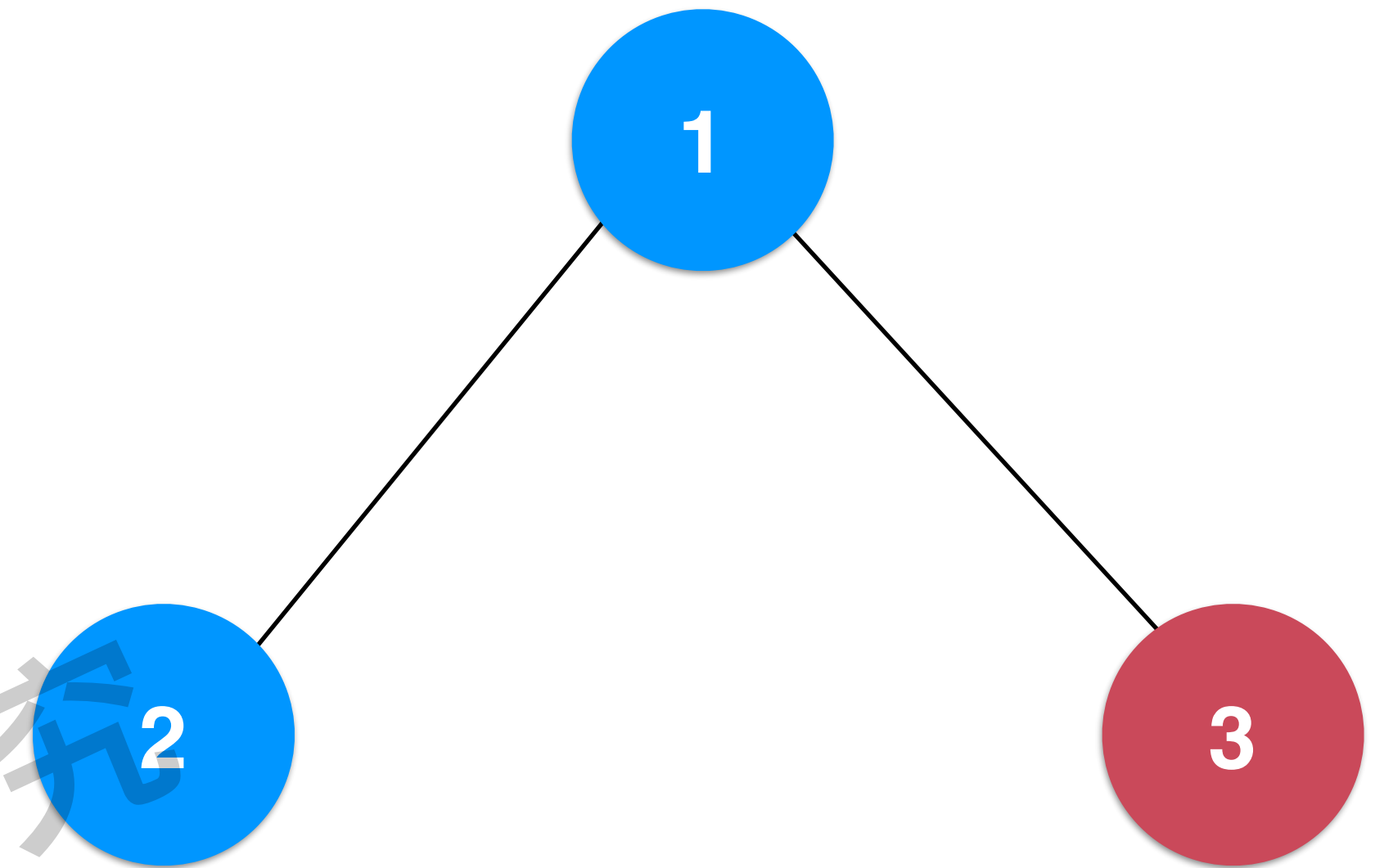
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

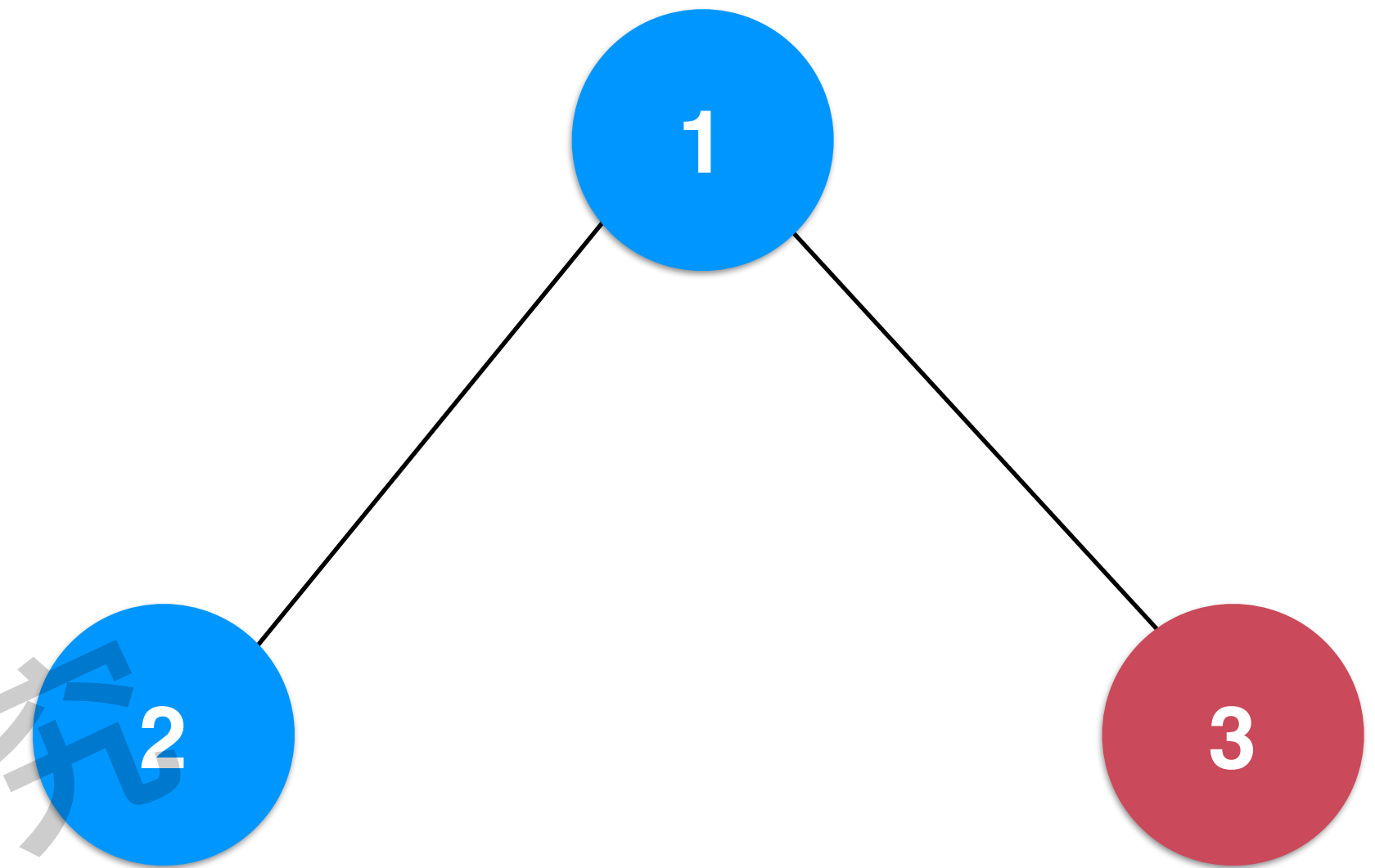




# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

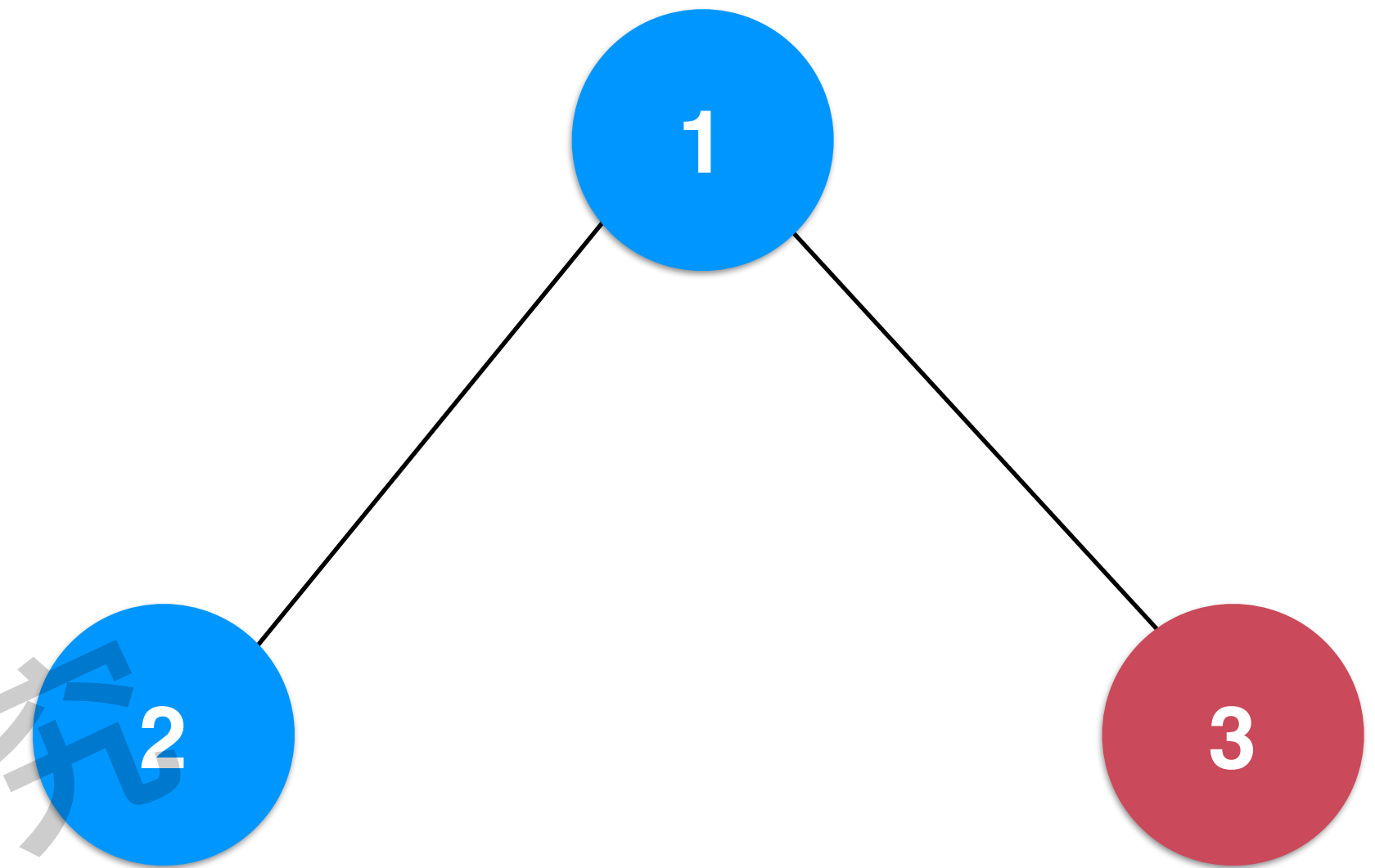




# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

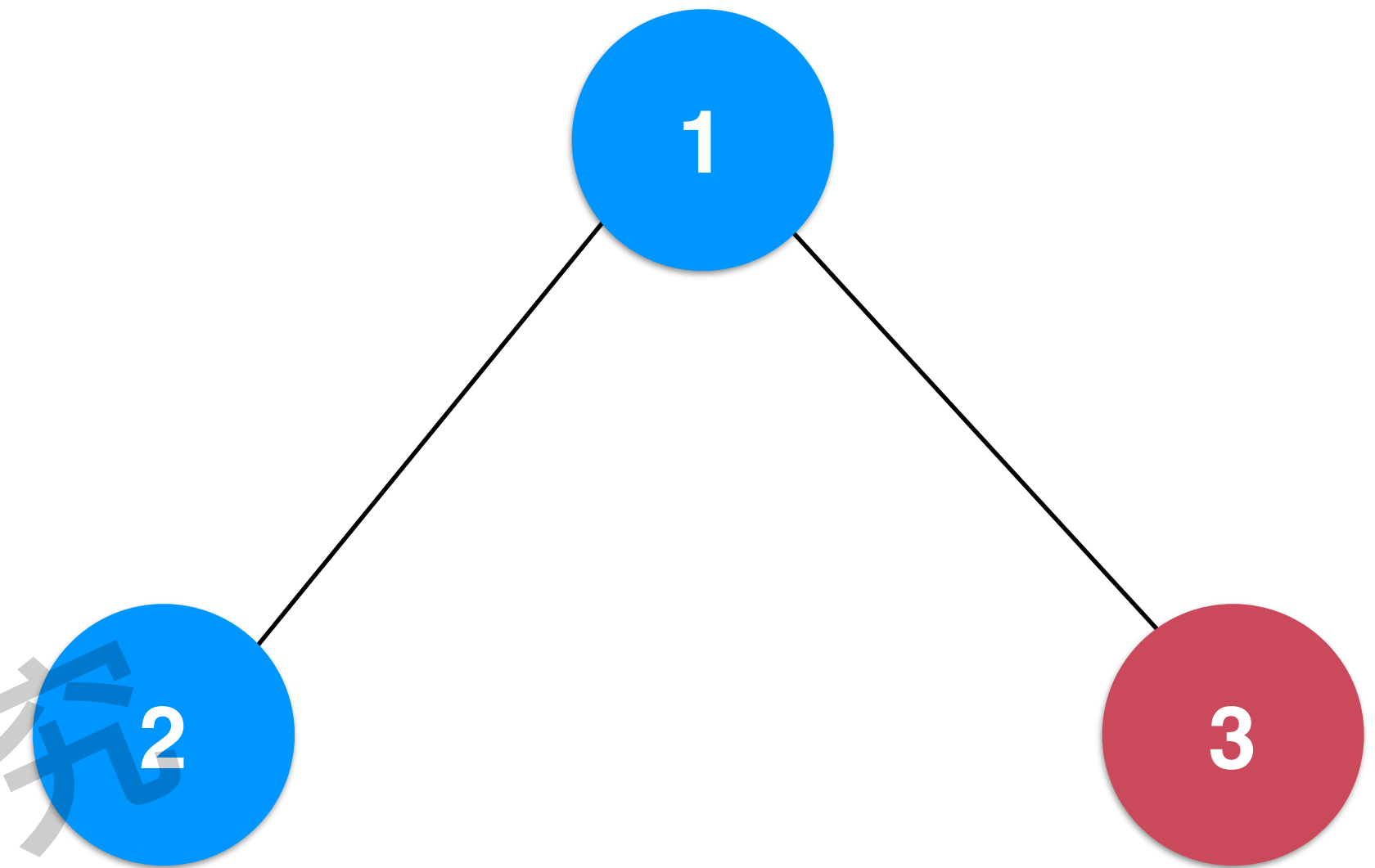
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

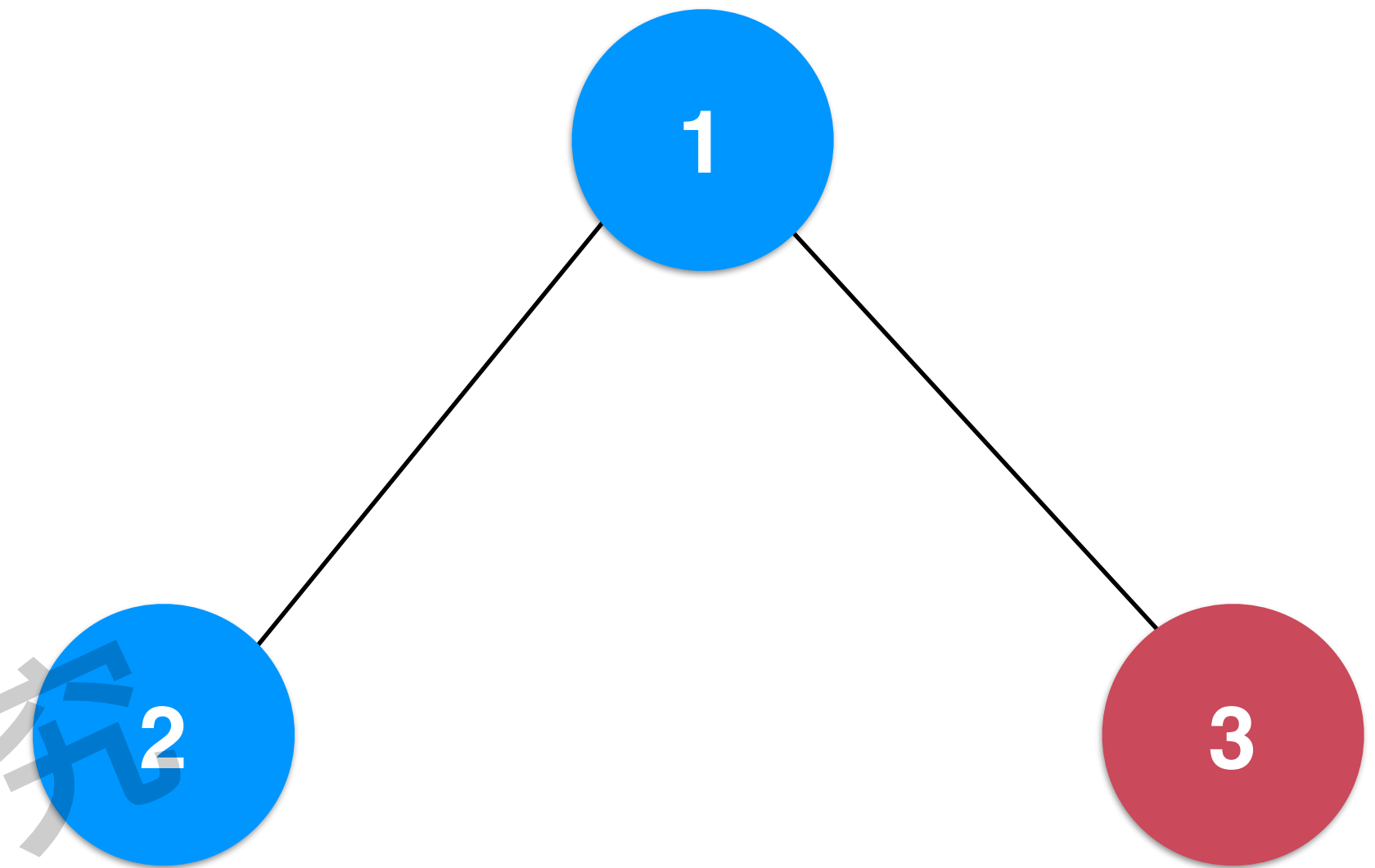
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



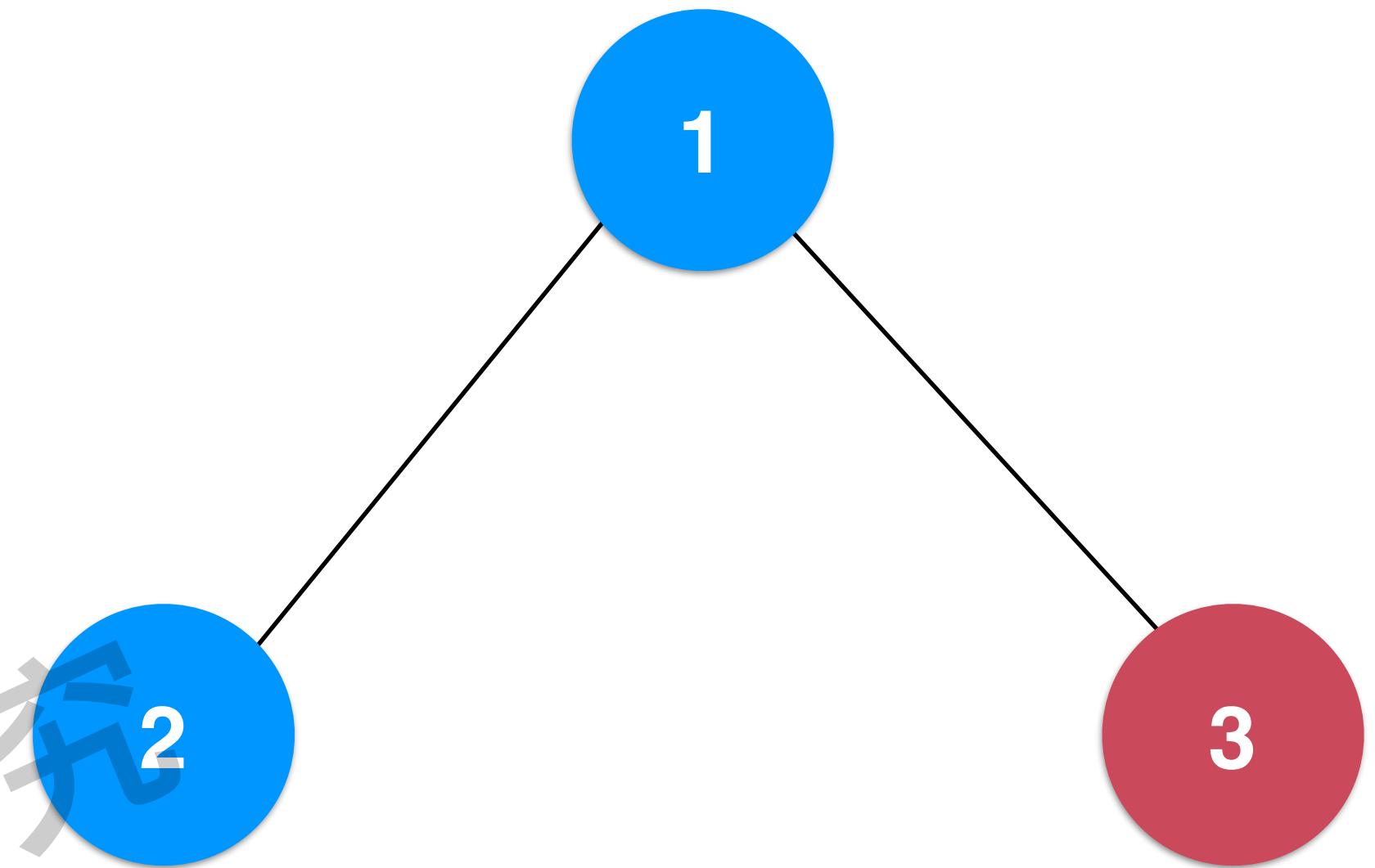
# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

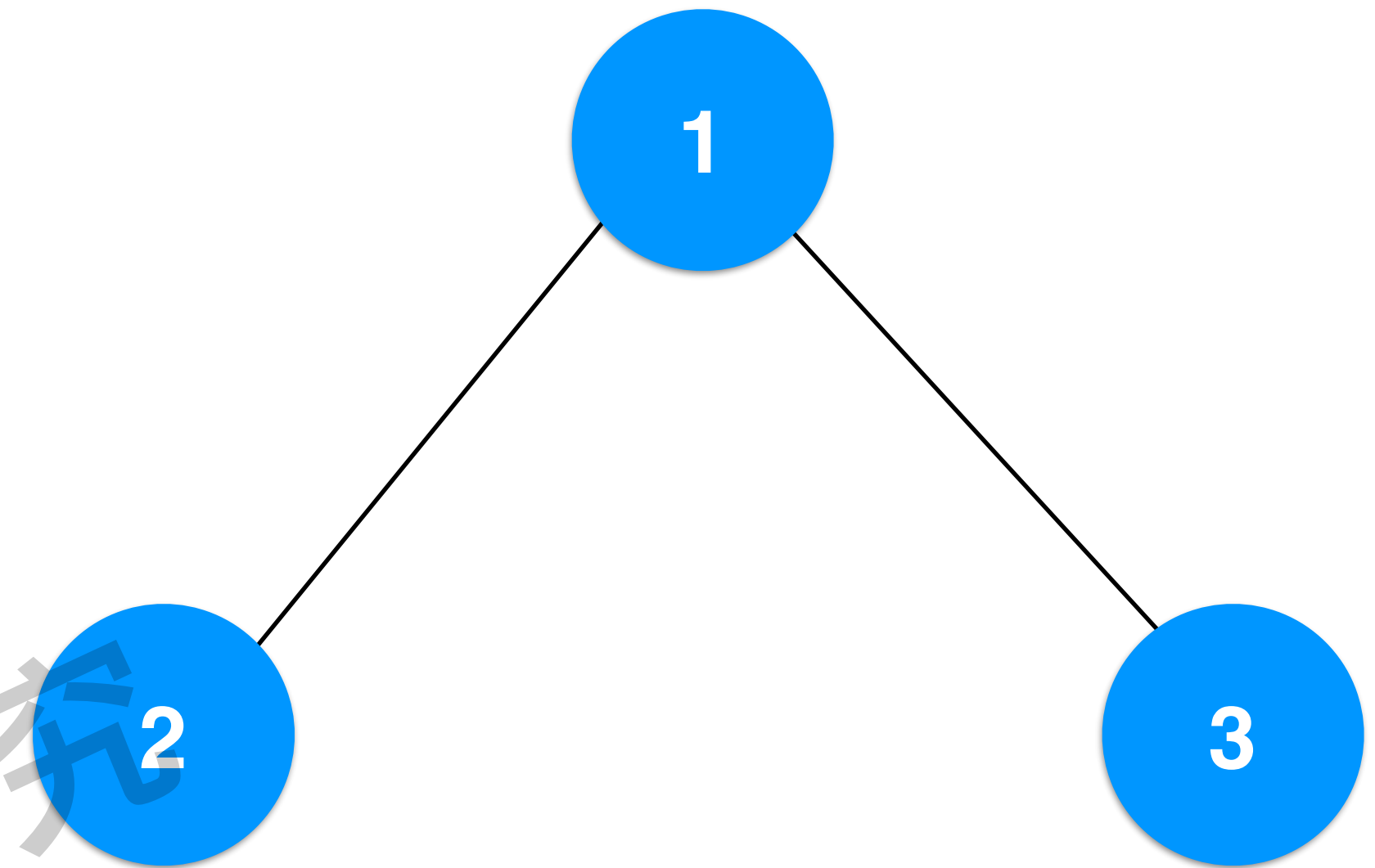
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

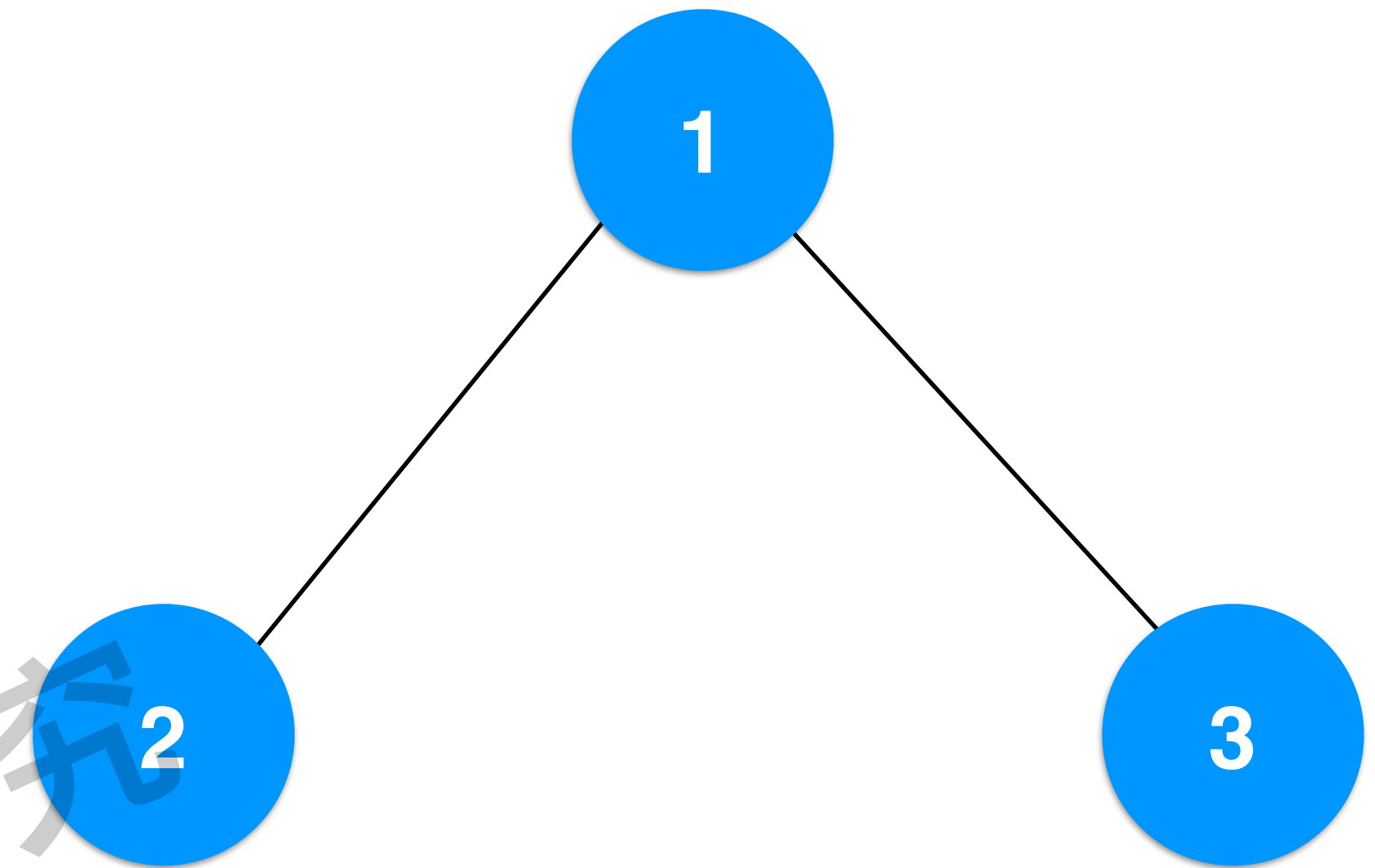
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

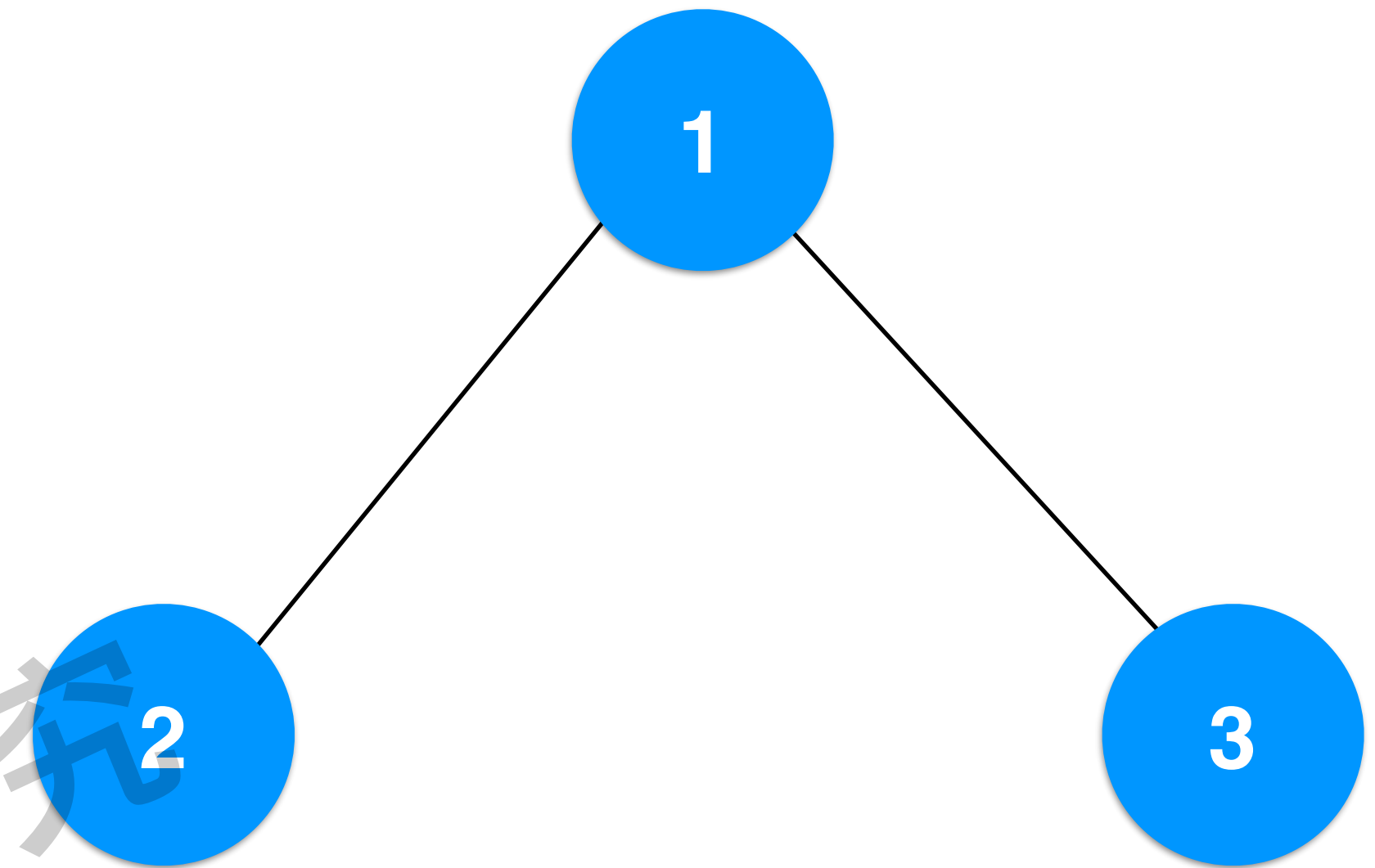


```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

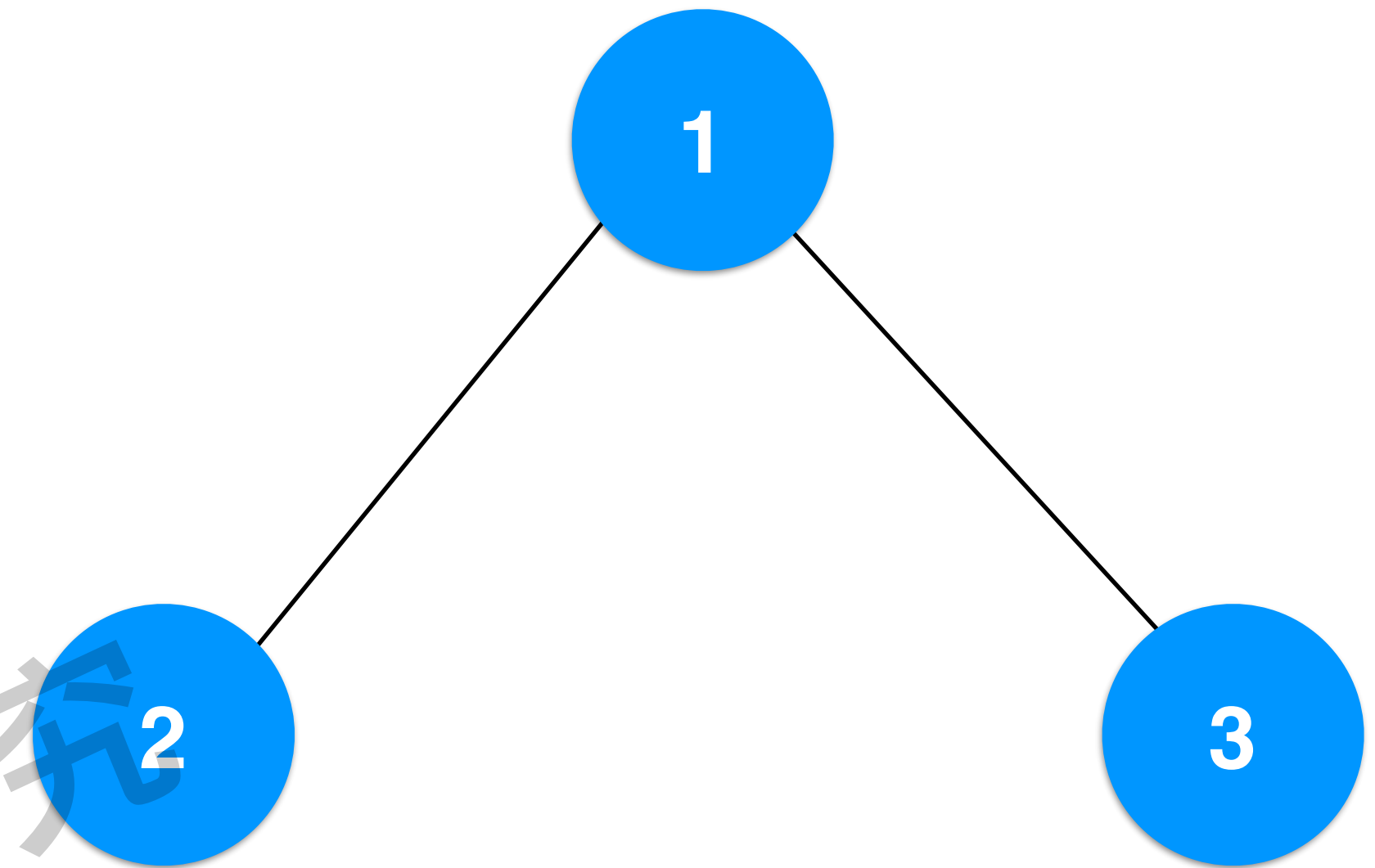


```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

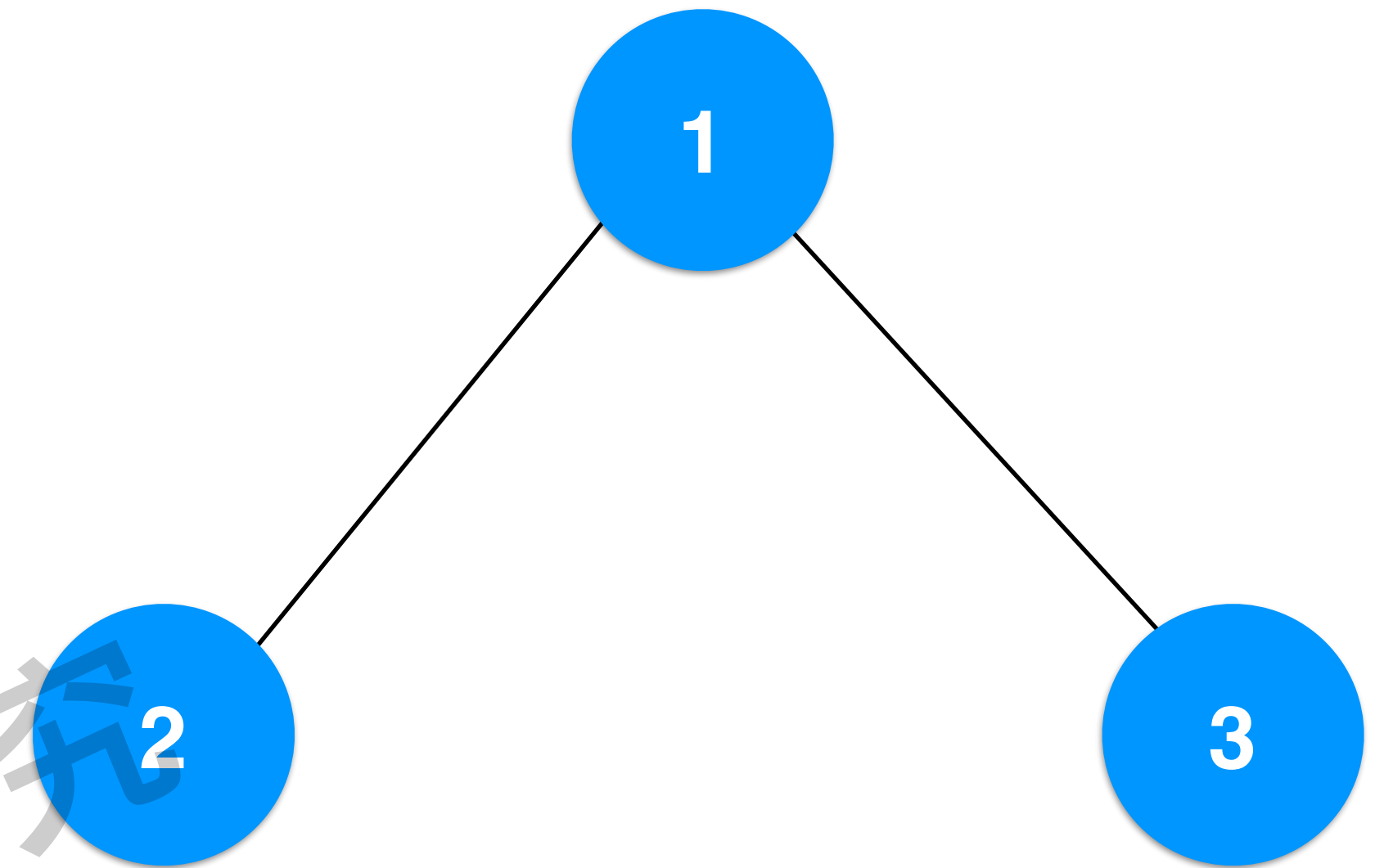
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

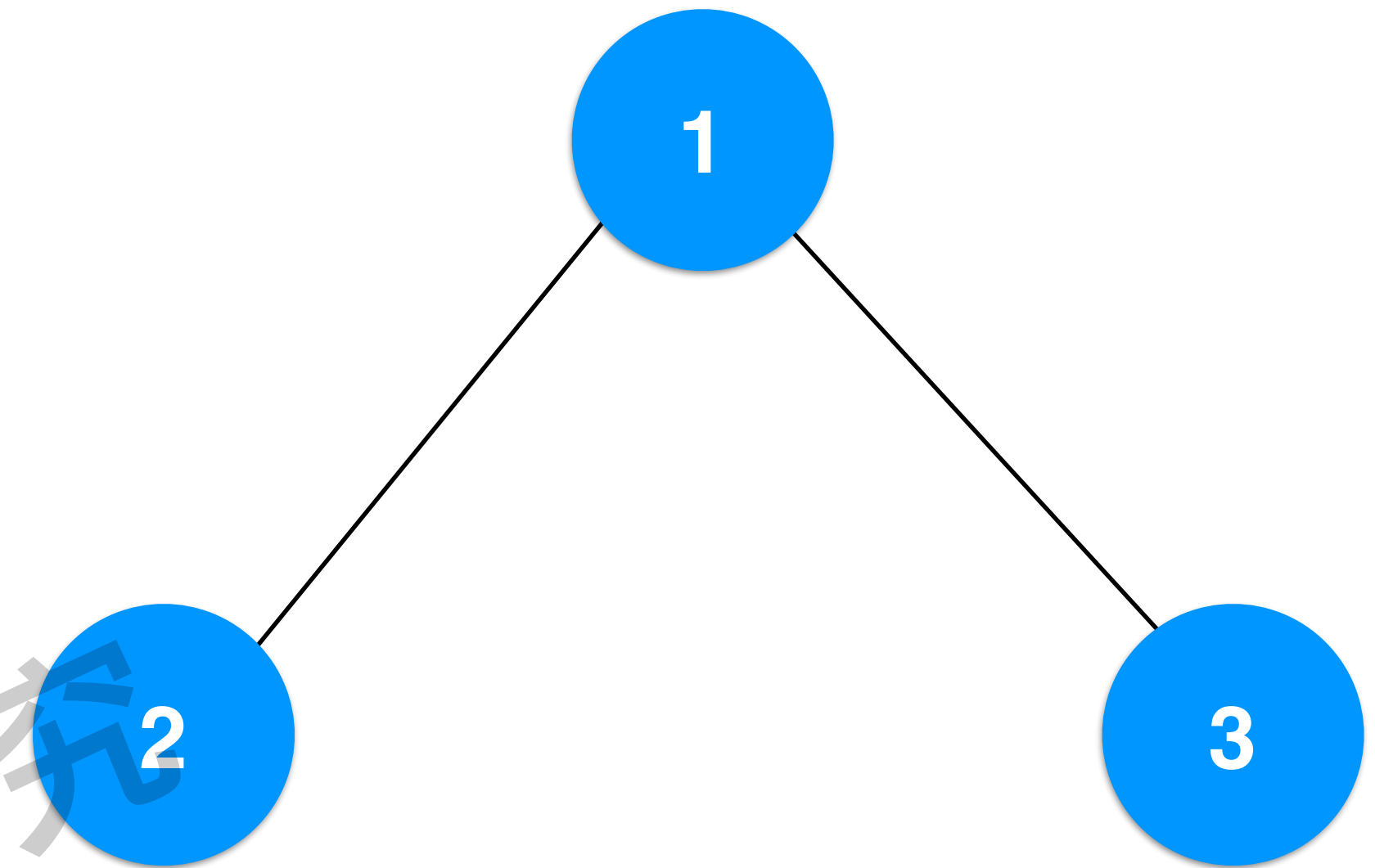
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

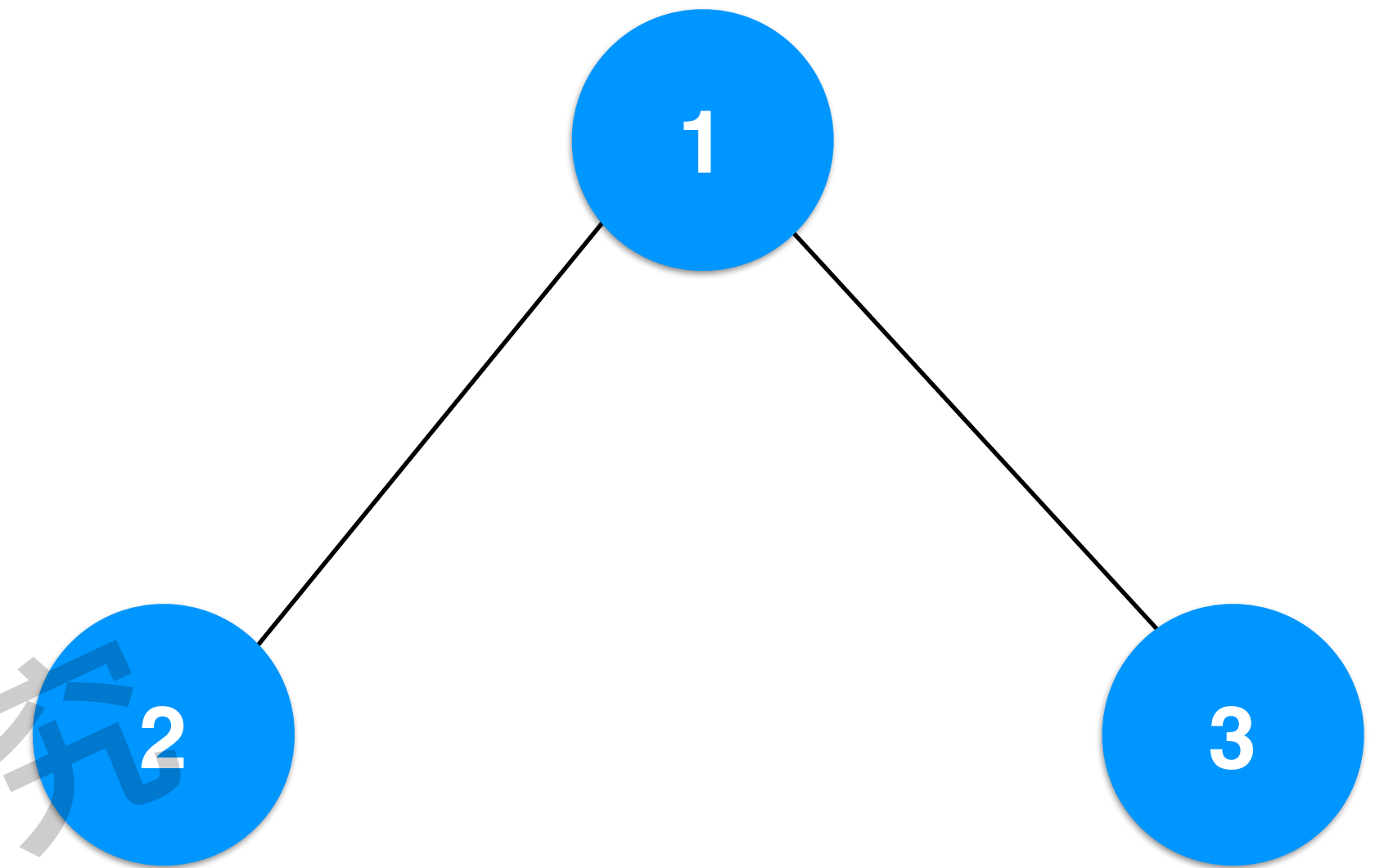
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

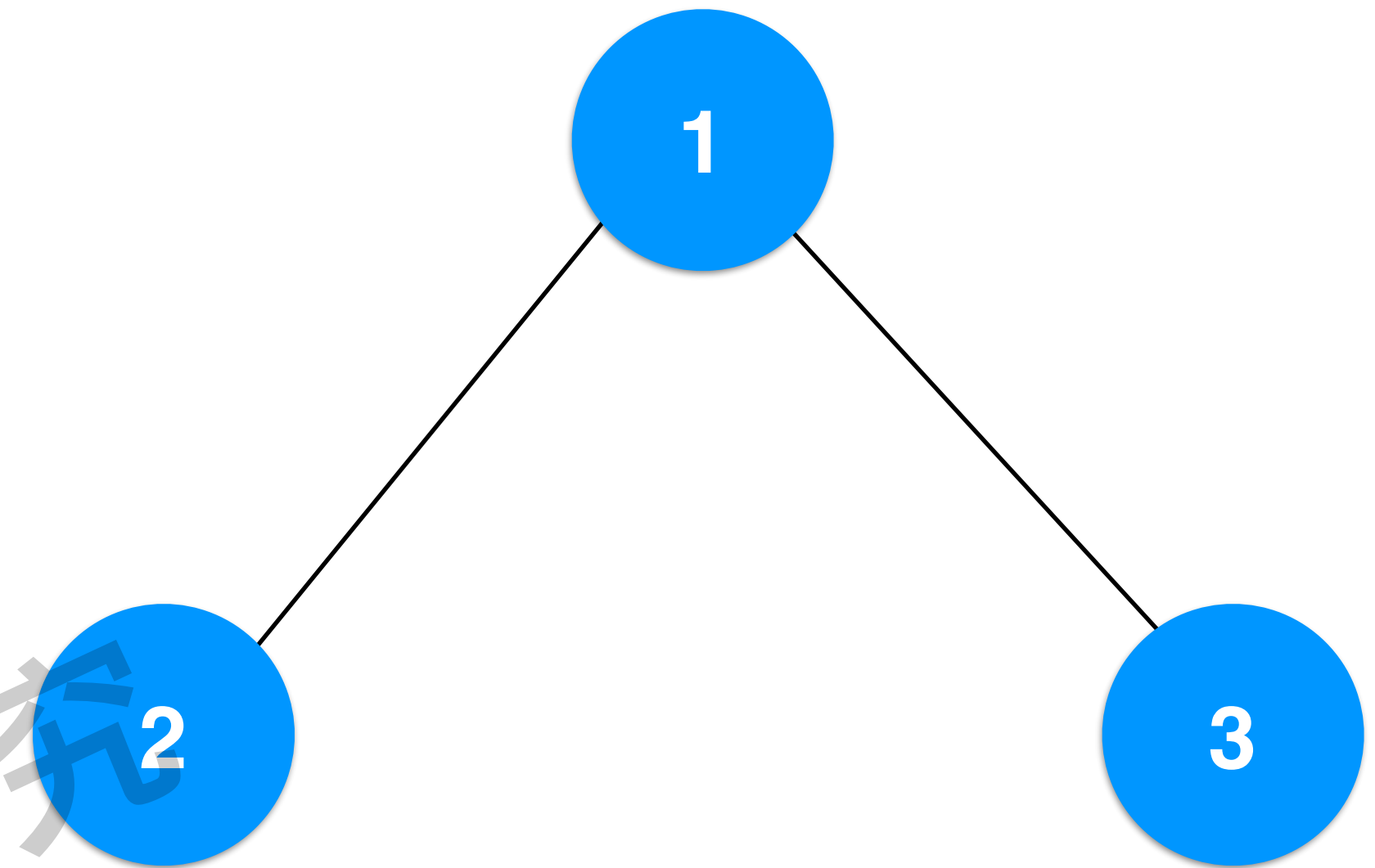
# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

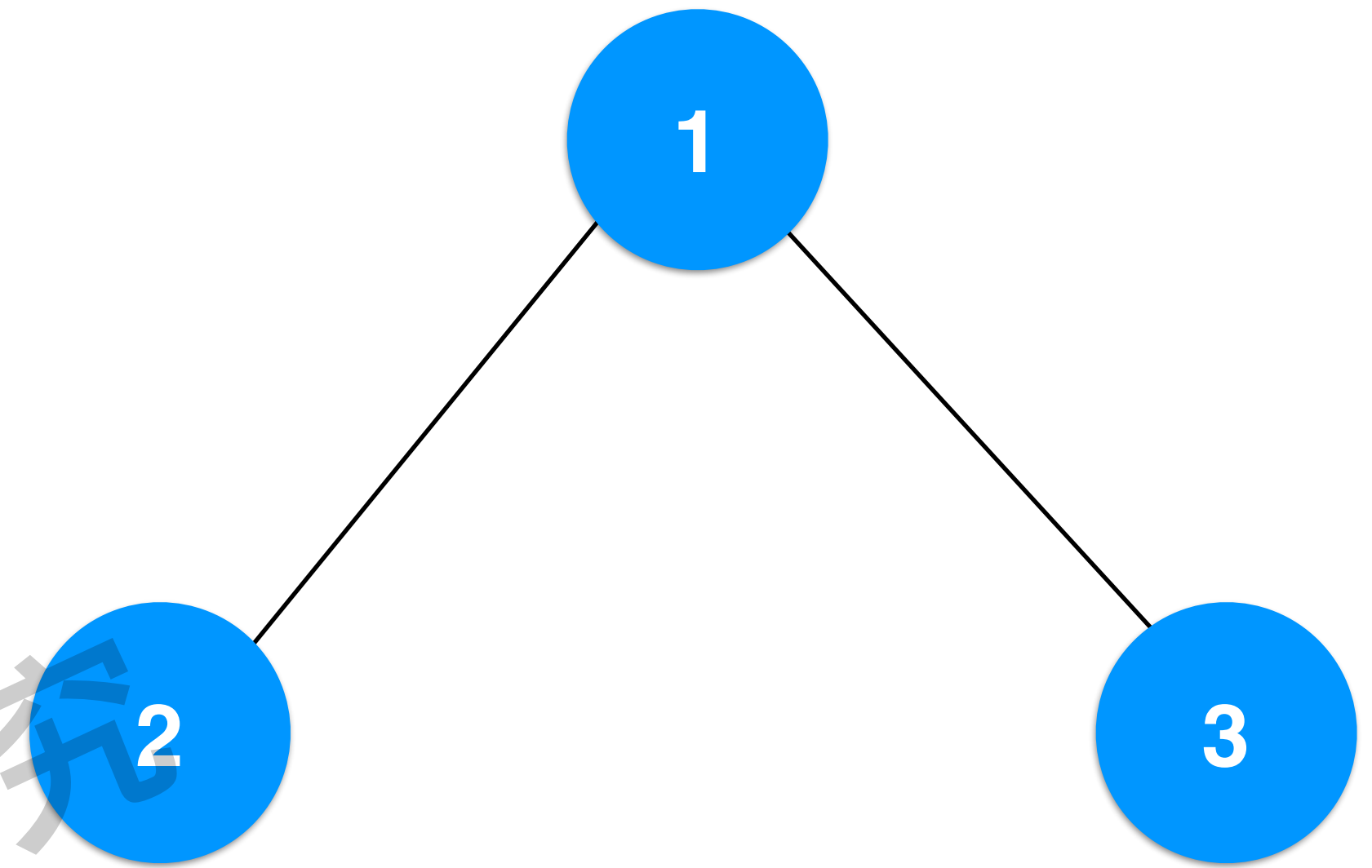


# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal





144. Binary Tree Preorder Traversal

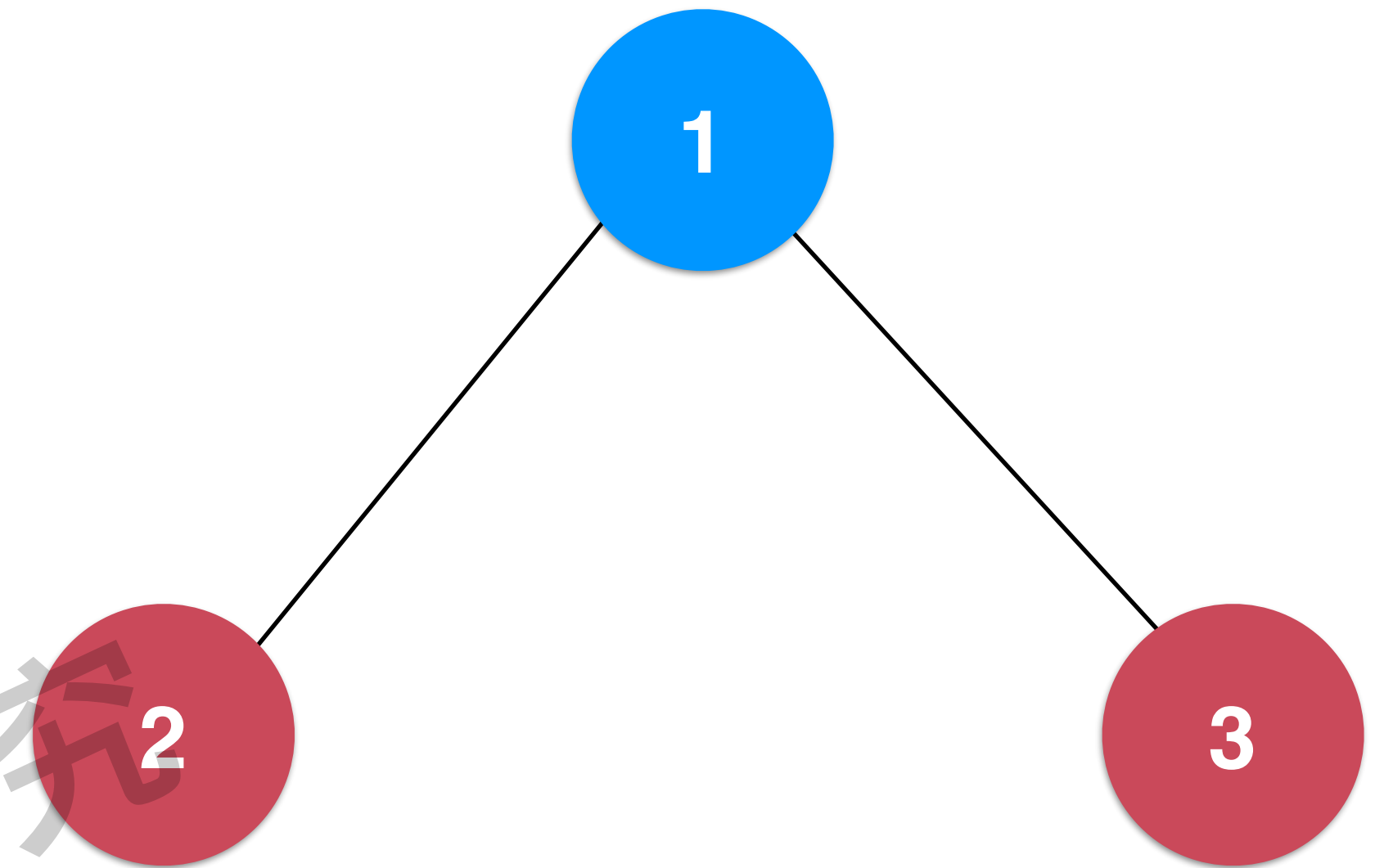
94. Binary Tree Inorder Traversal

145. Binary Tree Postorder Traversal



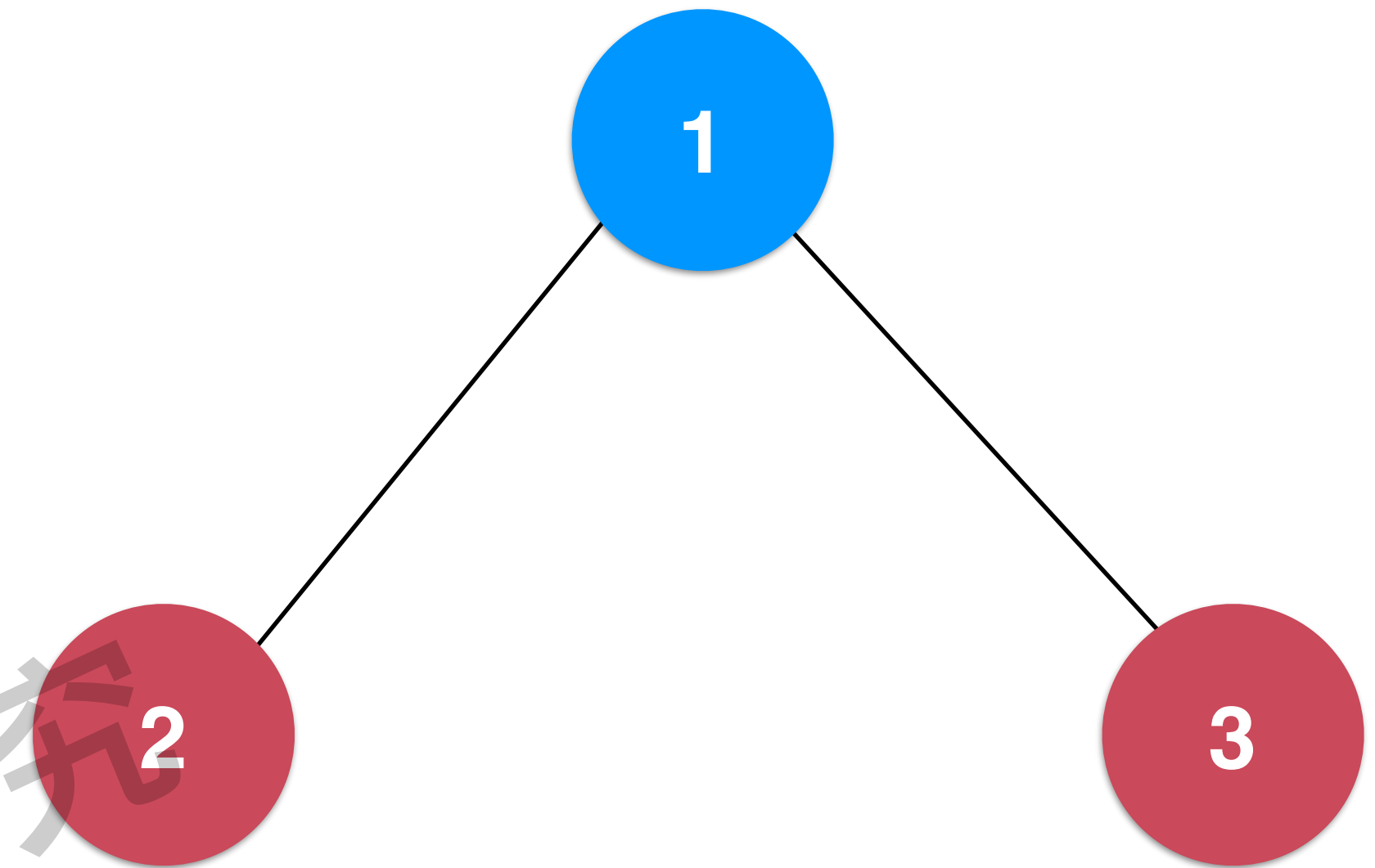
# 144. Binary Tree Preorder Traversal

```
▶ void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



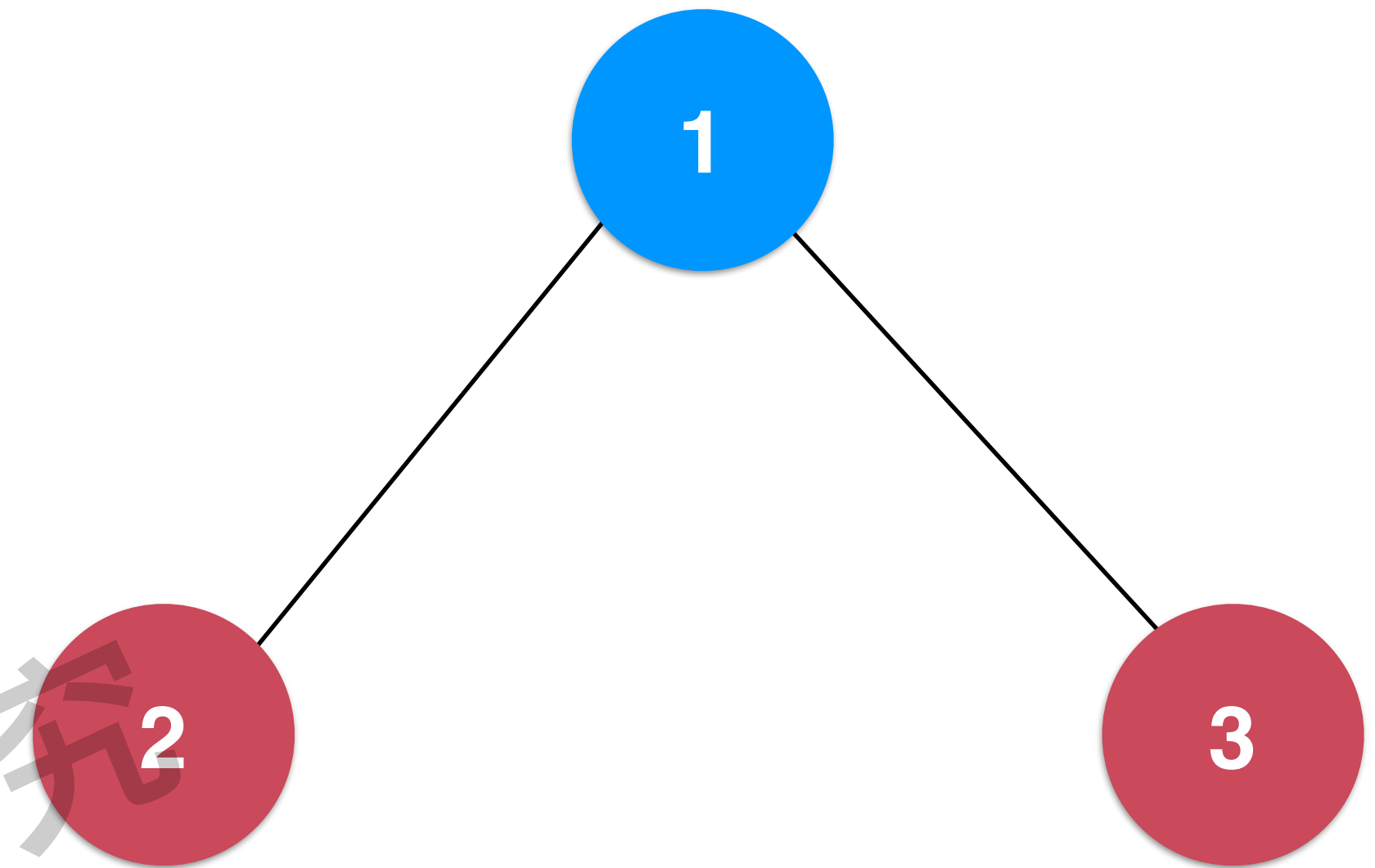
# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

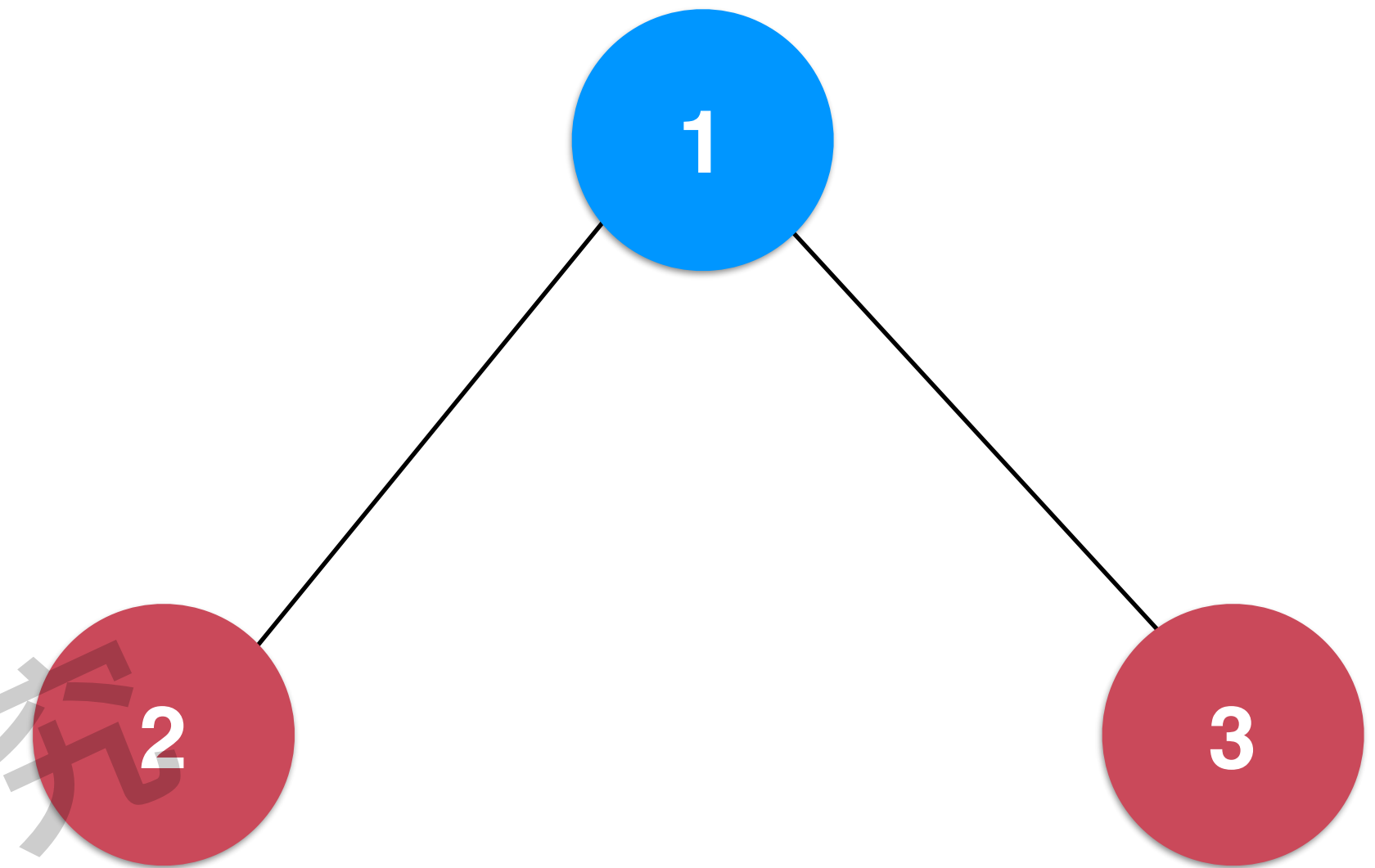
cout 1

go 1-L

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

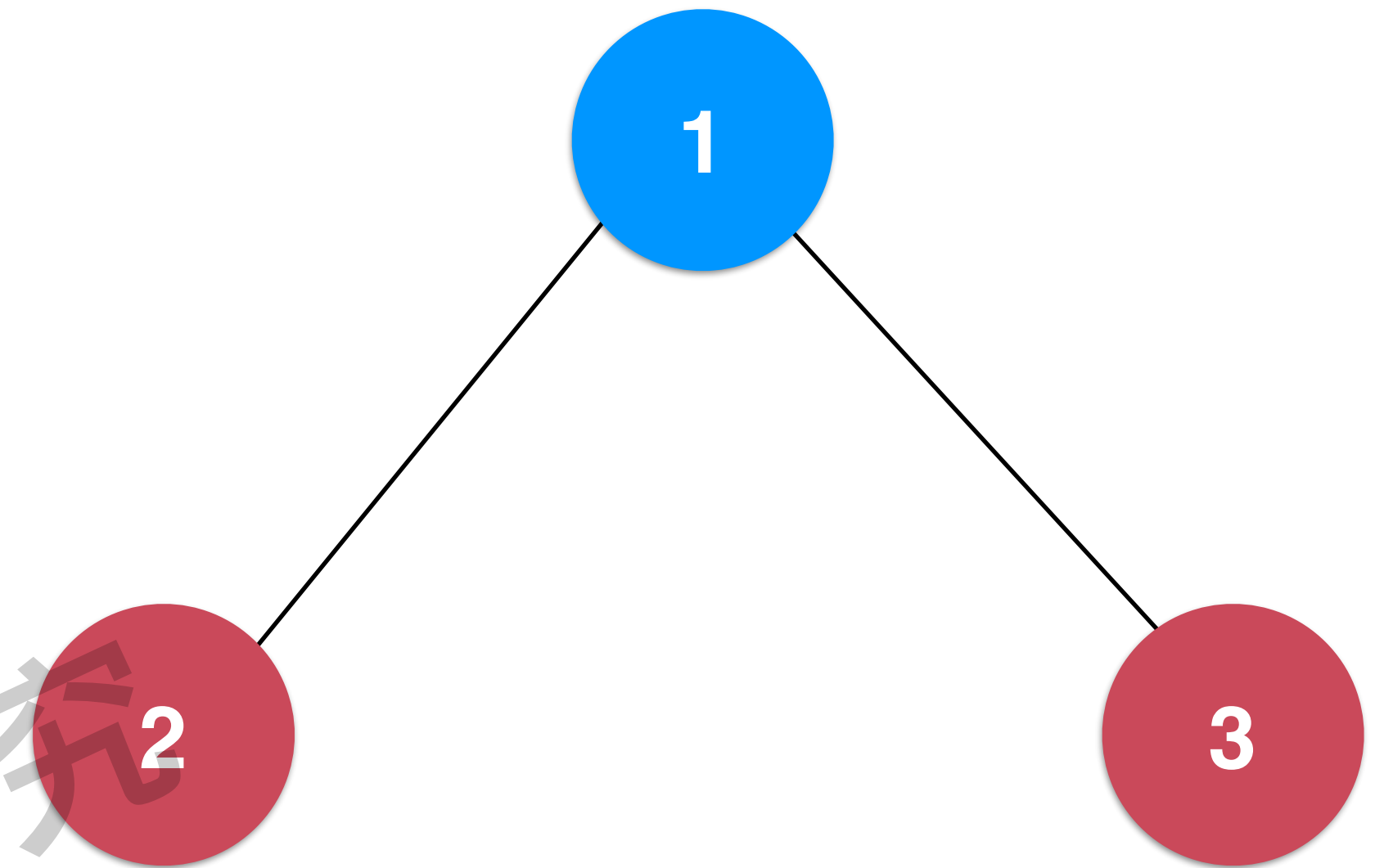
cout 1

go 1-L

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

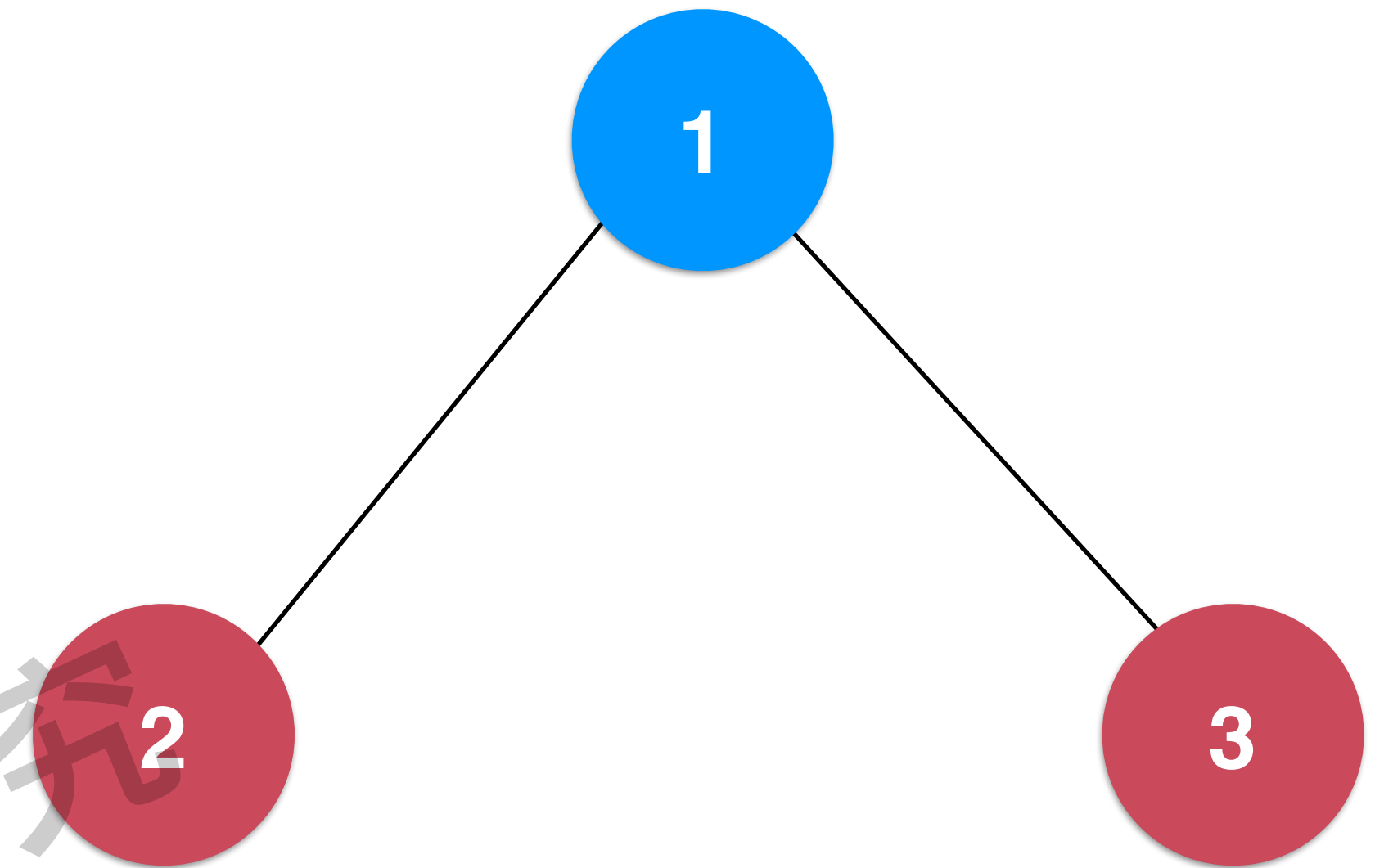


# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 1   go 1-L   go 1-R  
**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



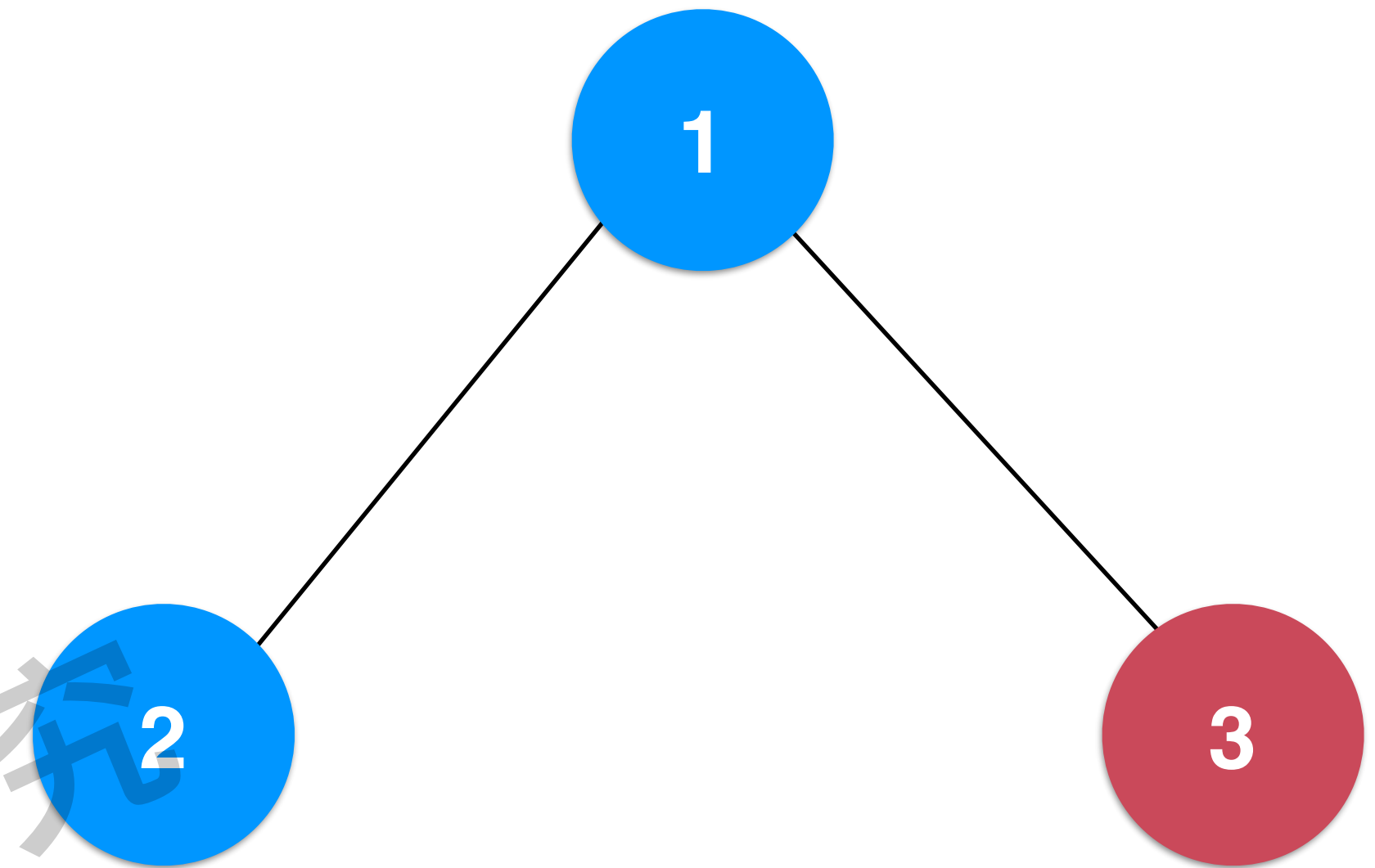


# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 1   go 1-L   go 1-R  
**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



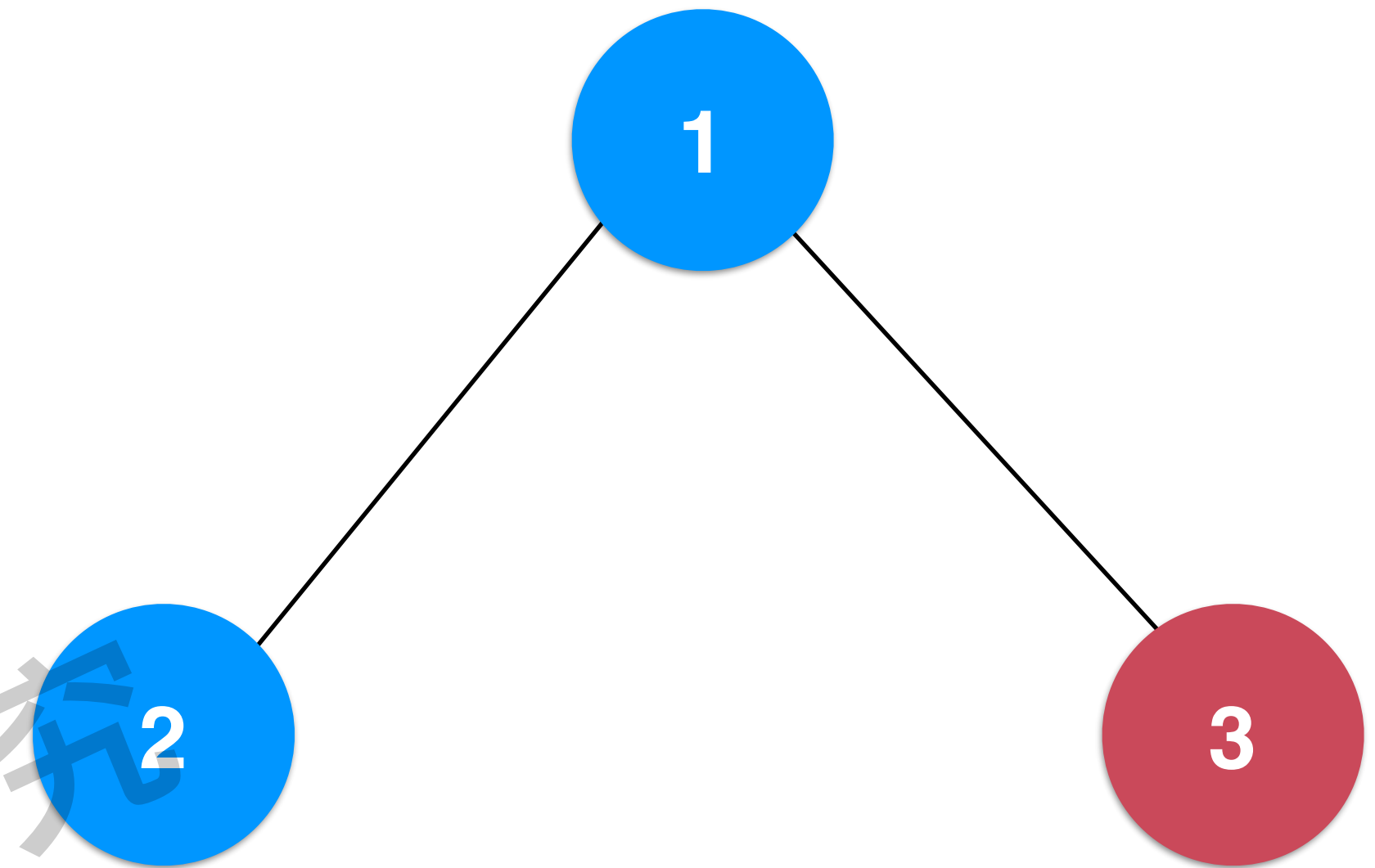


# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 1   go 1-L   go 1-R  
**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

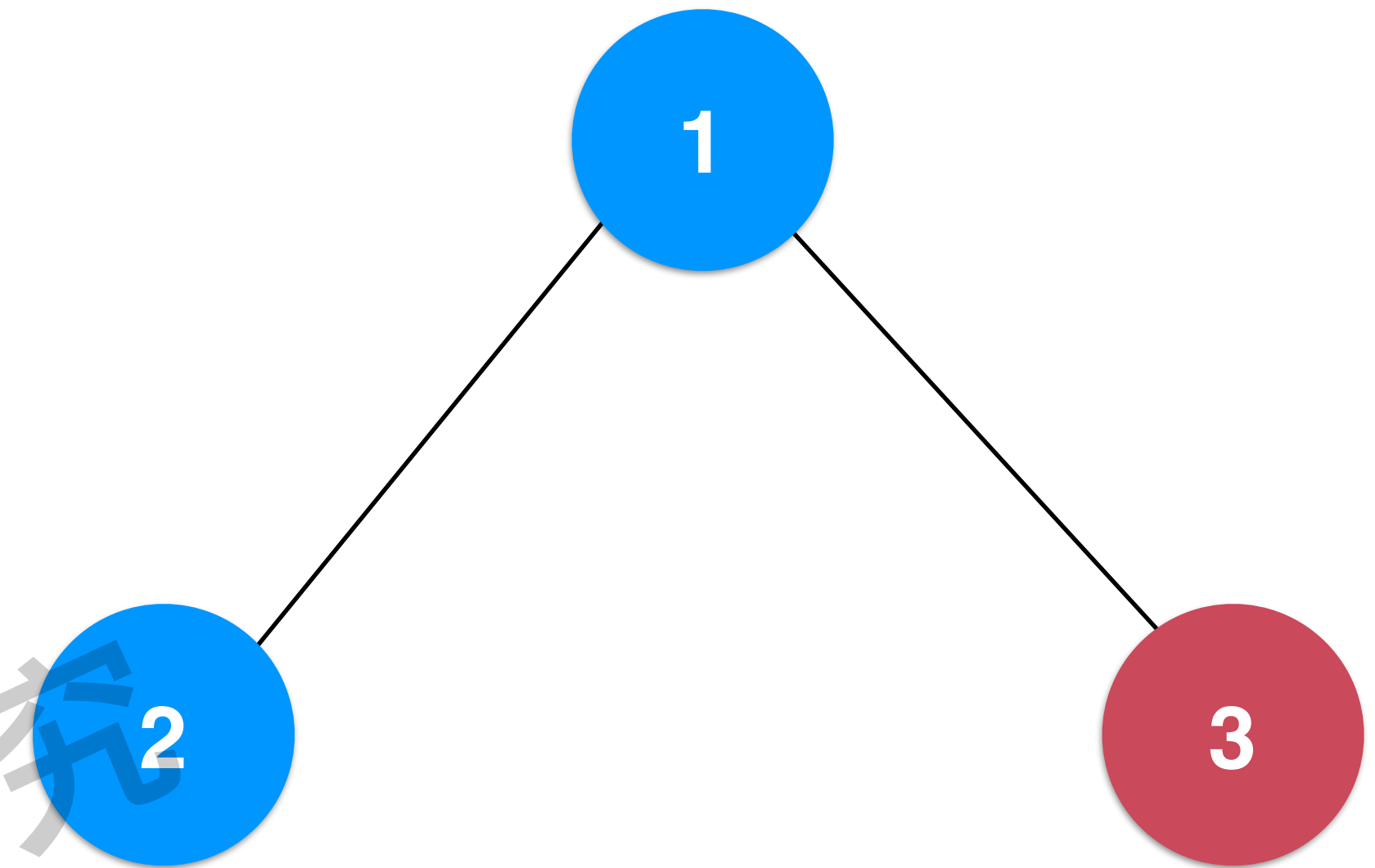
cout 1

go 1-L

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

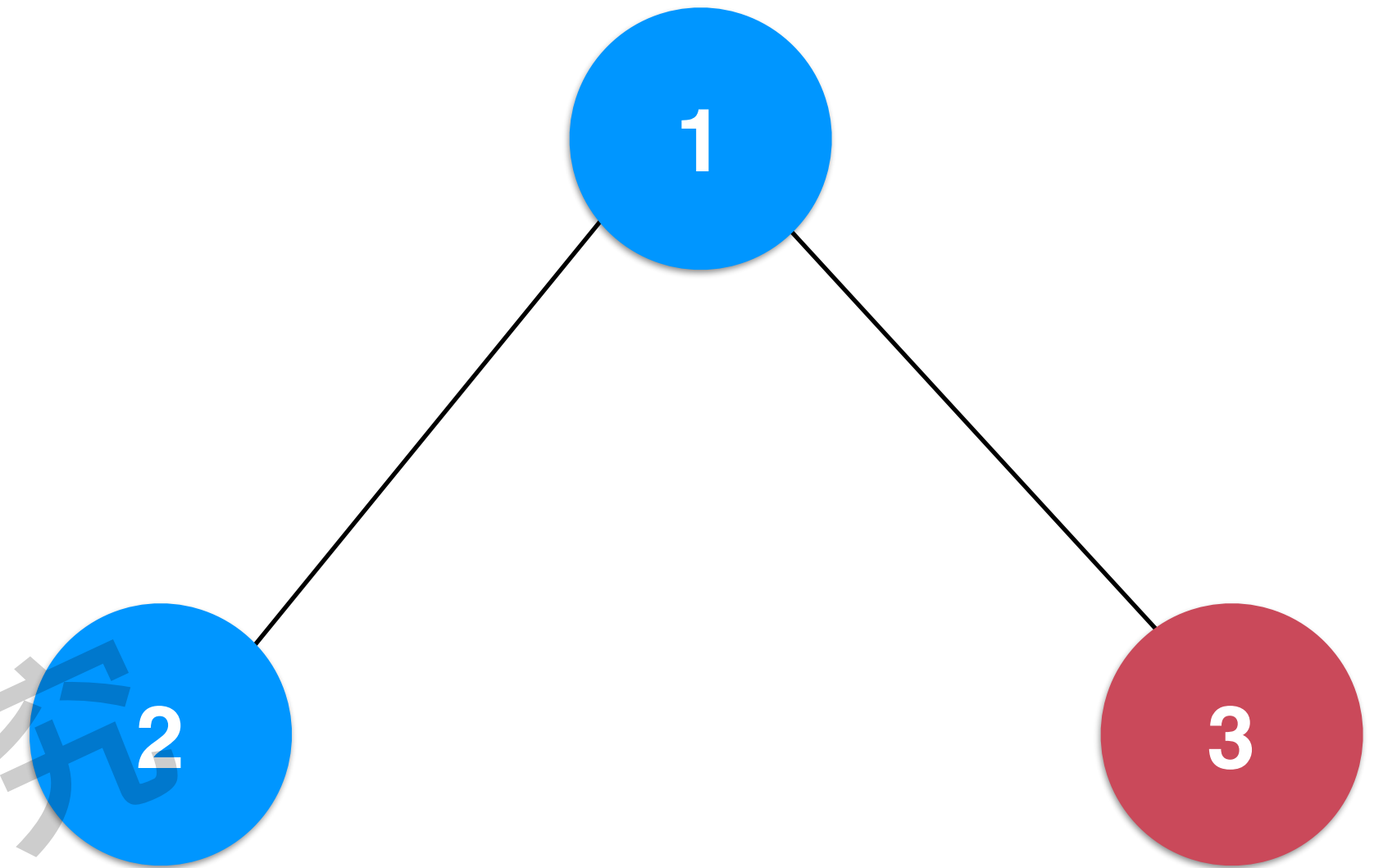
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 2   go 2-L   go 2-R

cout 1   go 1-L   go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

cout 2

go 2-L

go 2-R

```
void preorder( TreeNode* node ){
```

```
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }
```

```
}
```

cout 1

go 1-L

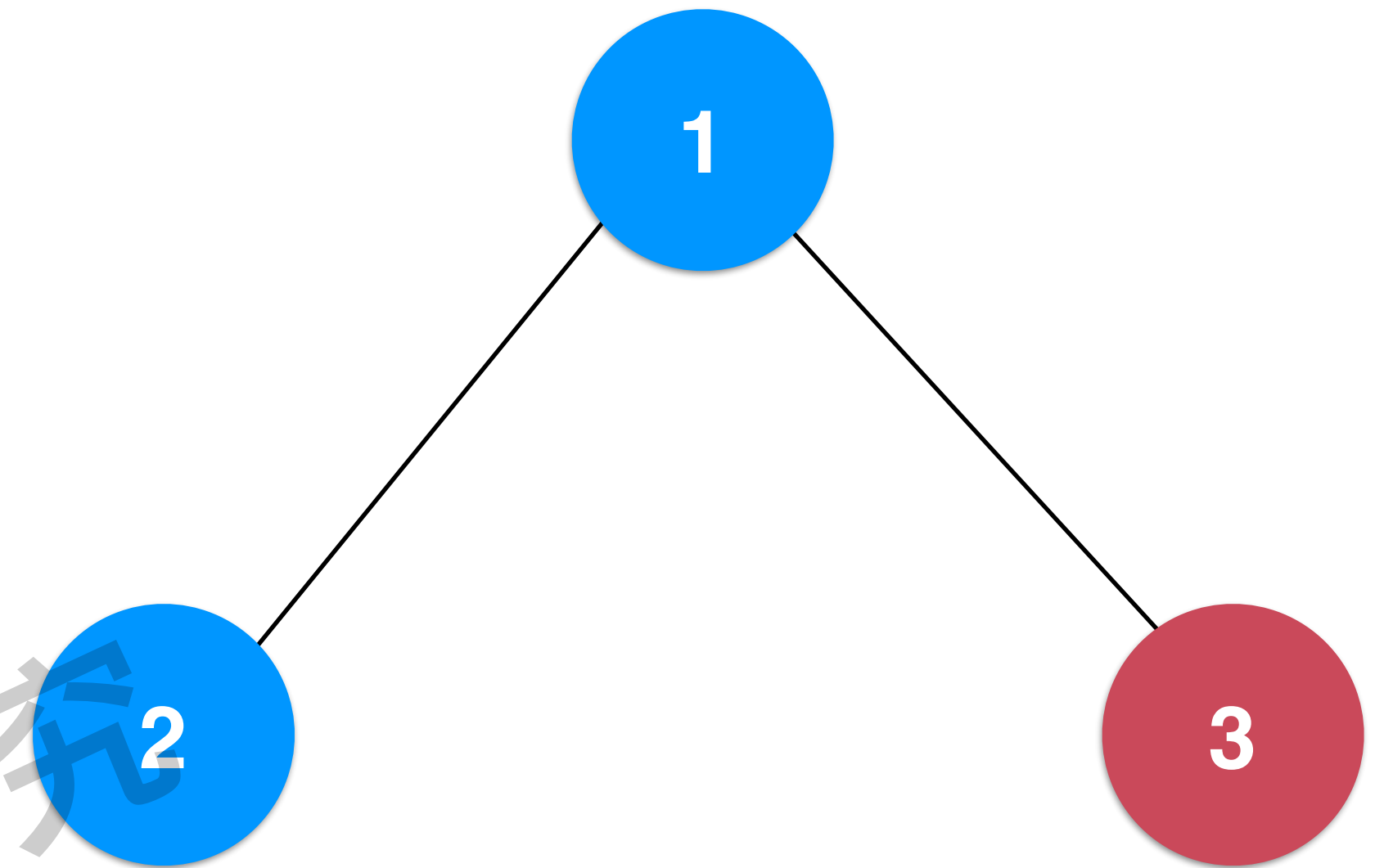
go 1-R

**Stack**

```
void preorder( TreeNode* node ){
```

```
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }
```

```
}
```

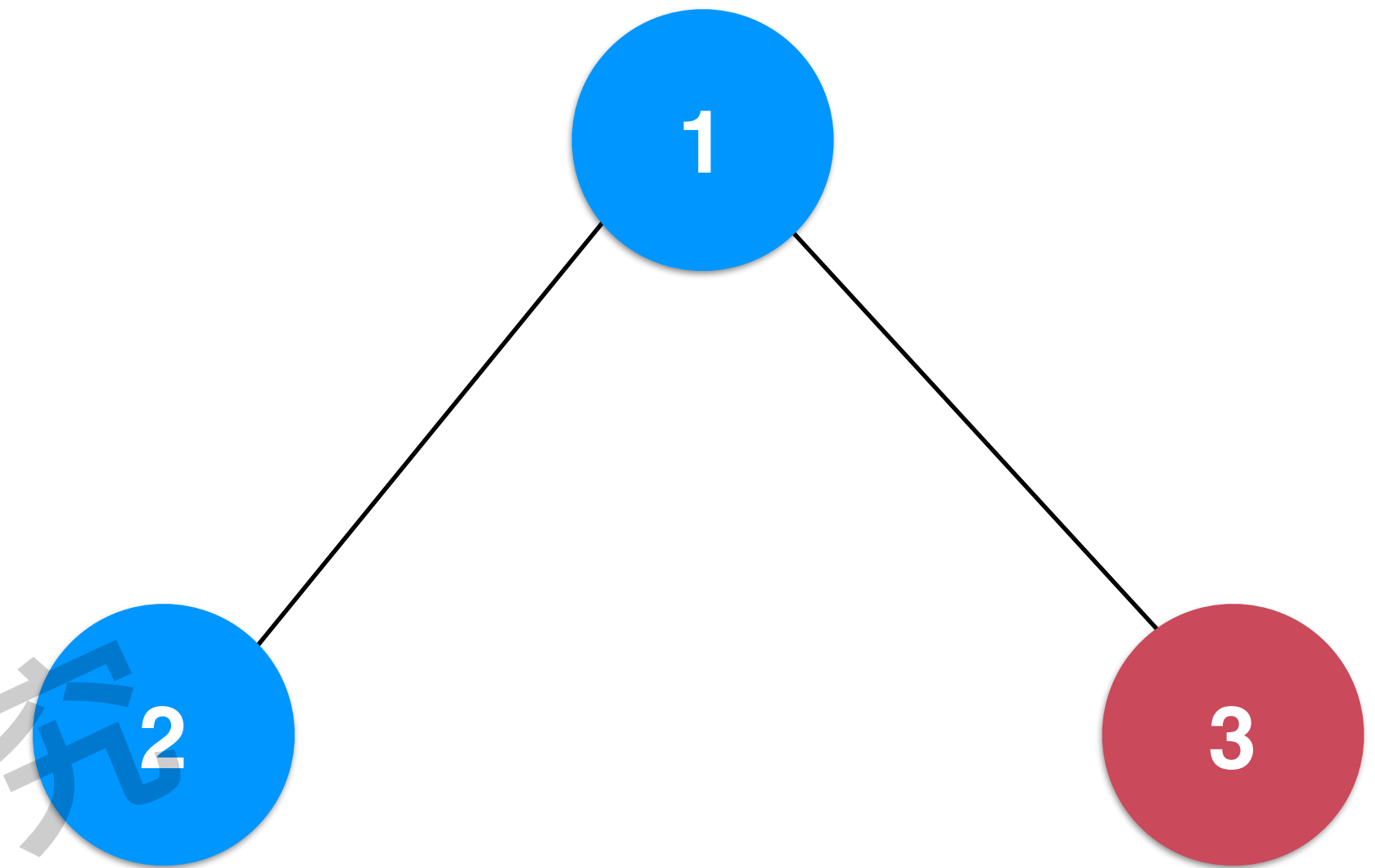


# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 1   go 1-L   go 1-R  
**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```





# 144. Binary Tree Preorder Traversal

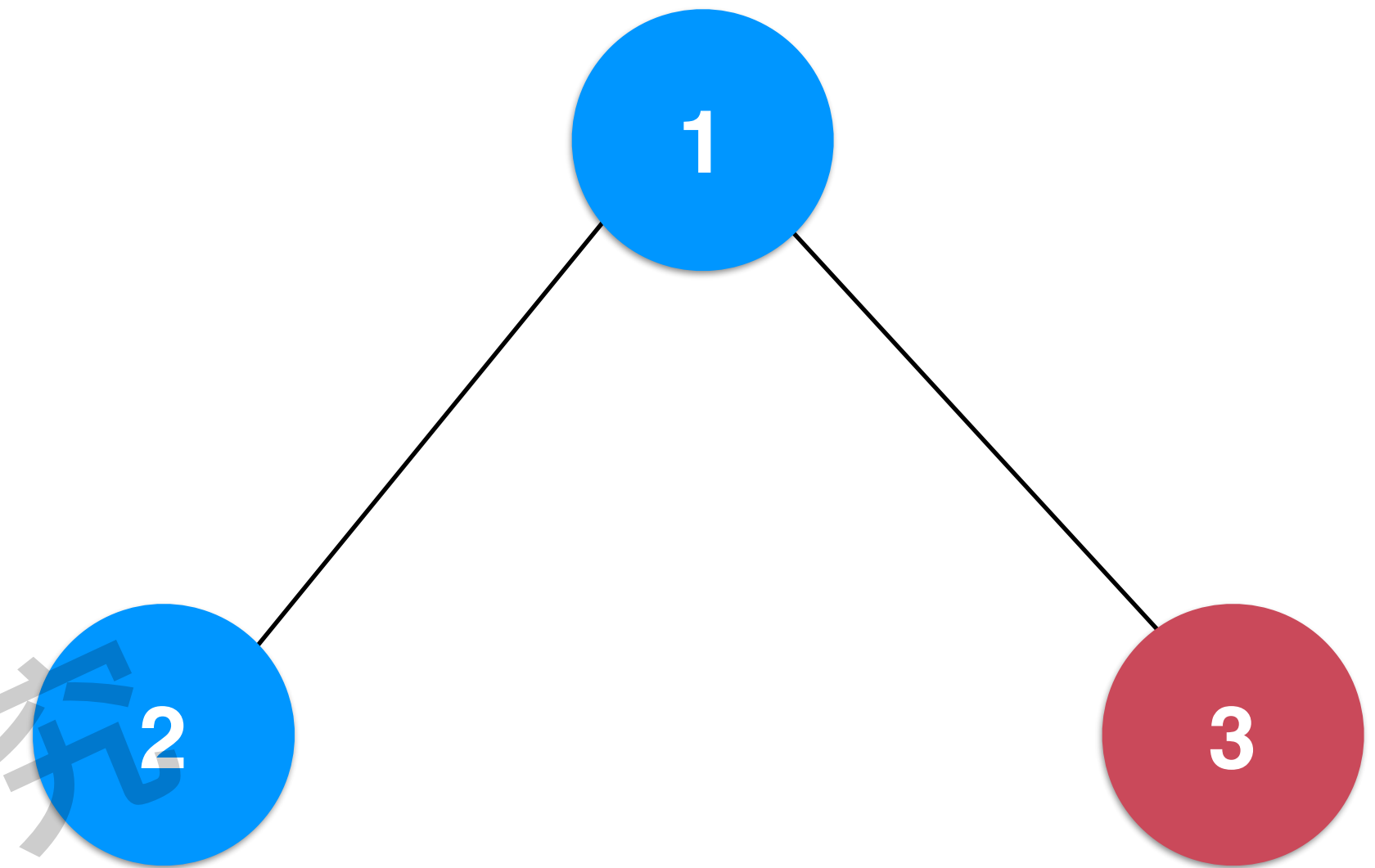
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 2   go 2-L   go 2-R

cout 1   go 1-L   go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

cout 2

go 2-L

go 2-R

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

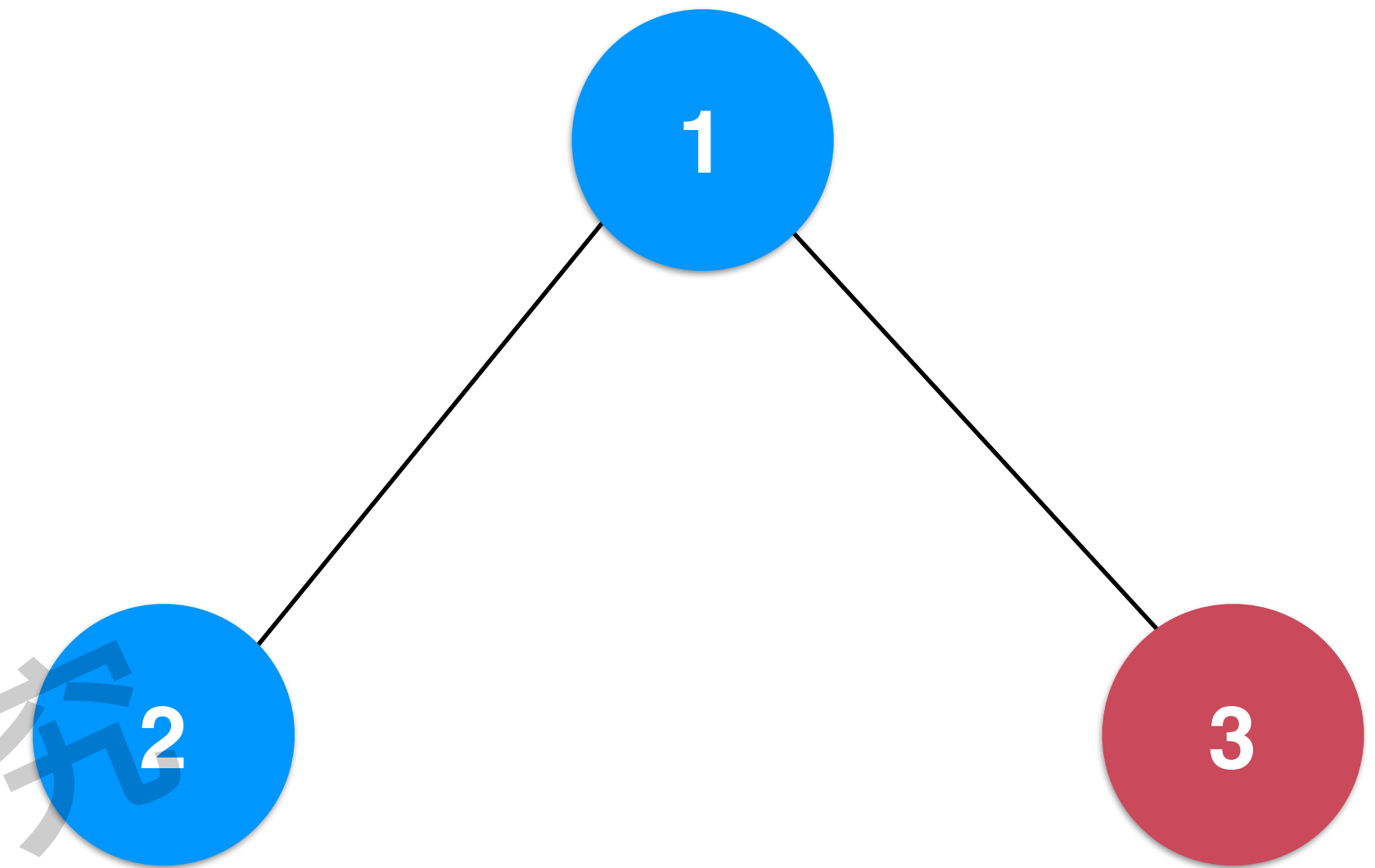
cout 1

go 1-L

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



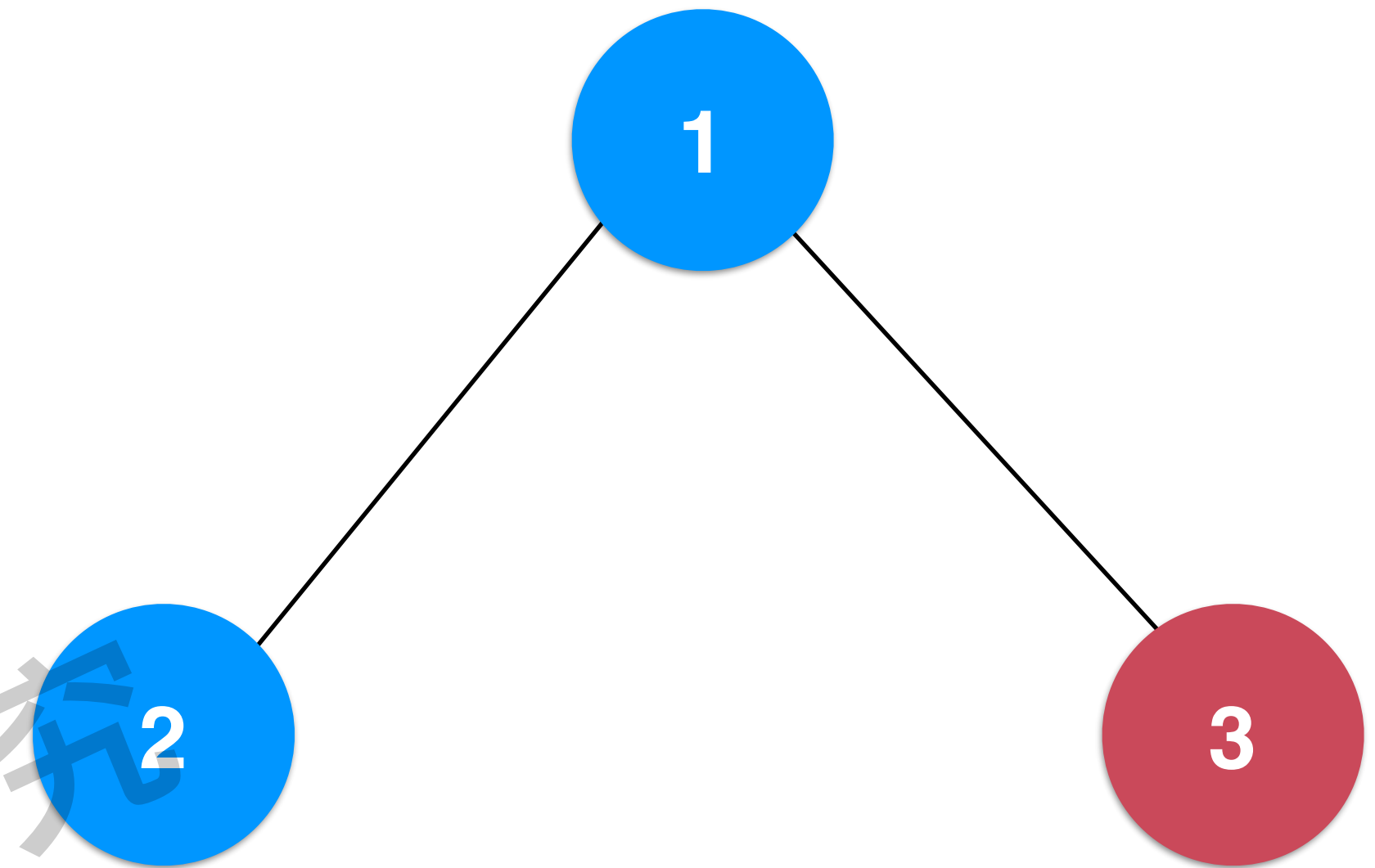


# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 1   go 1-L   go 1-R  
**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

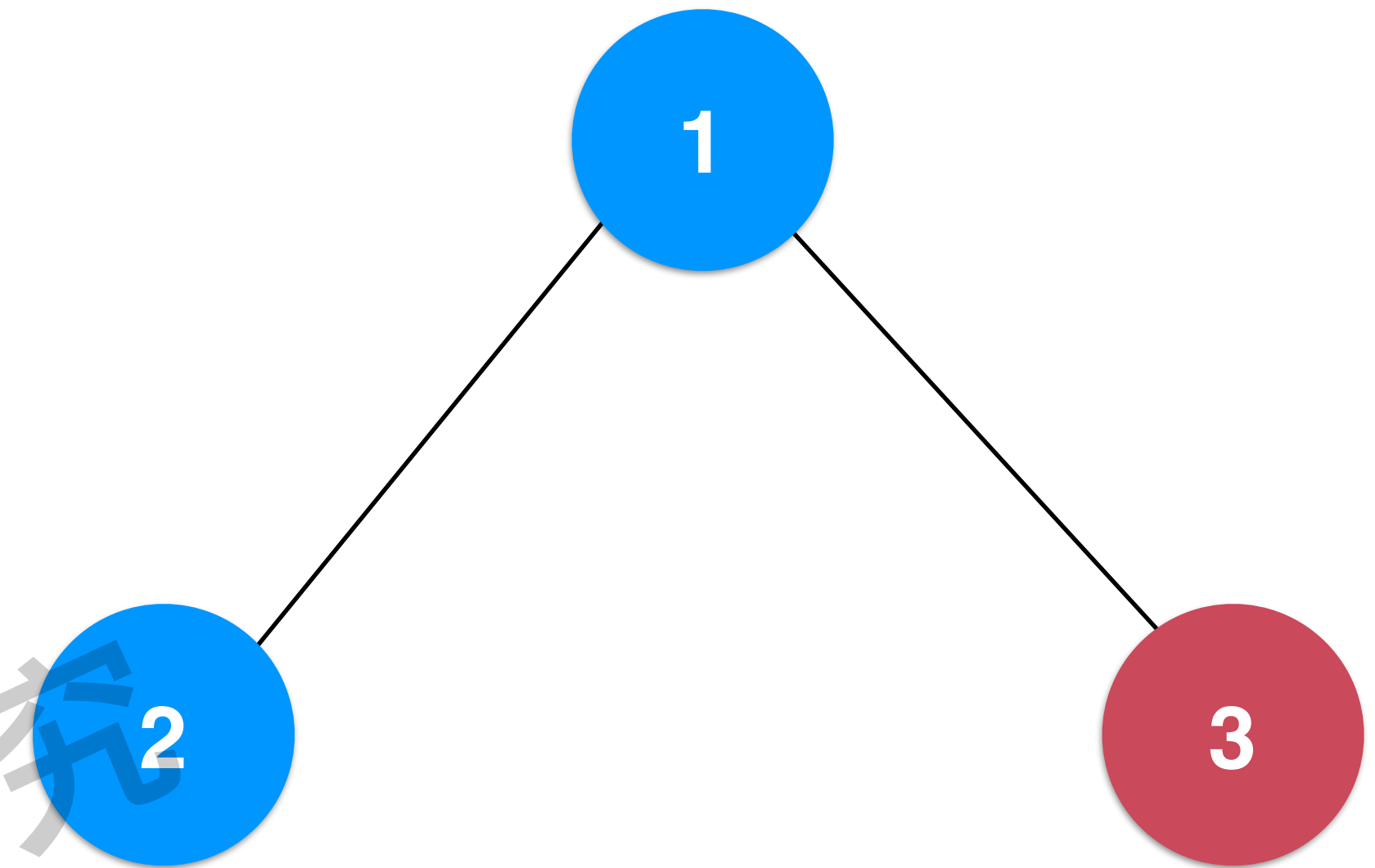
cout 1

go 1-L

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

cout 1

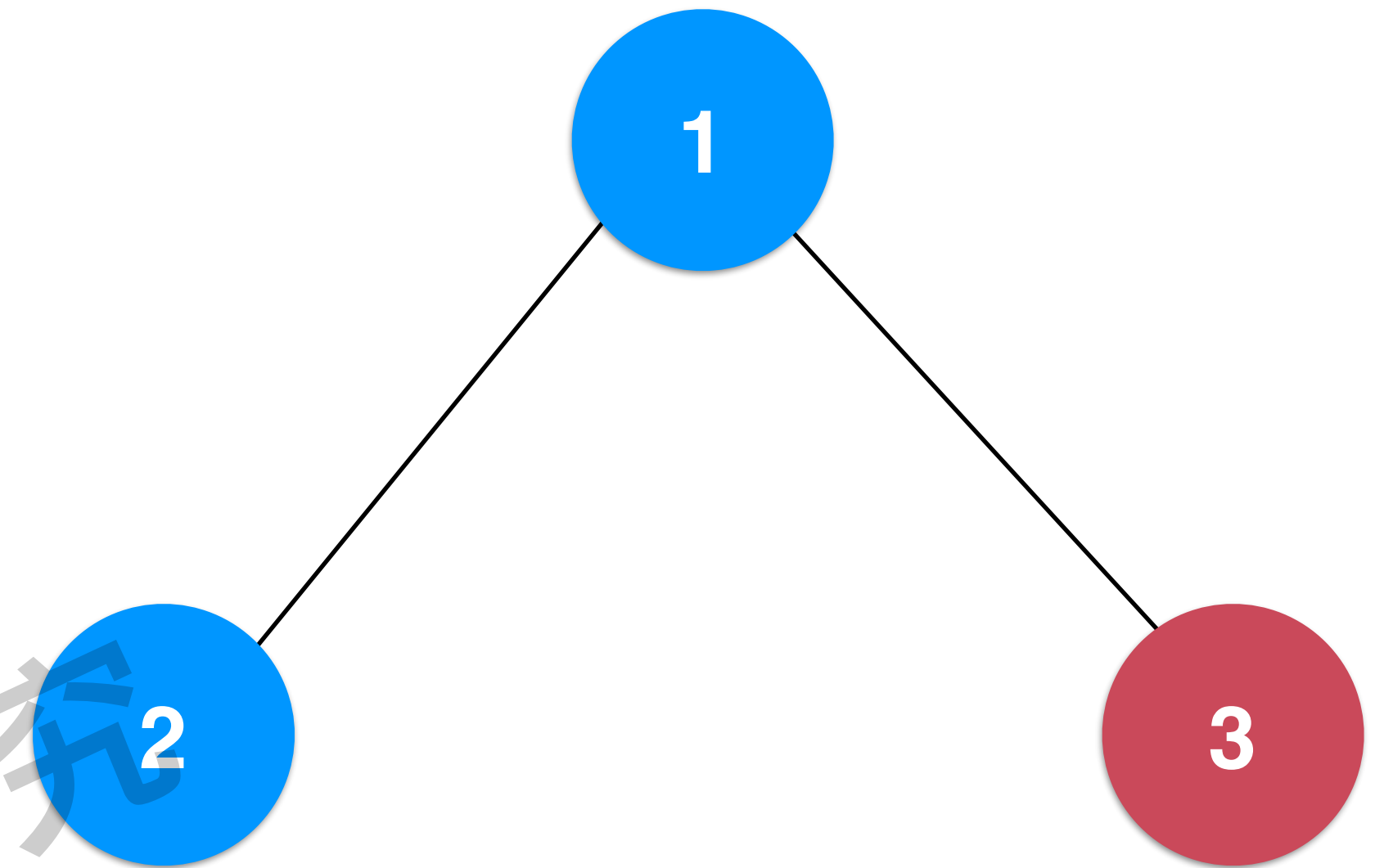
go 1-L

go 1-R

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

Stack

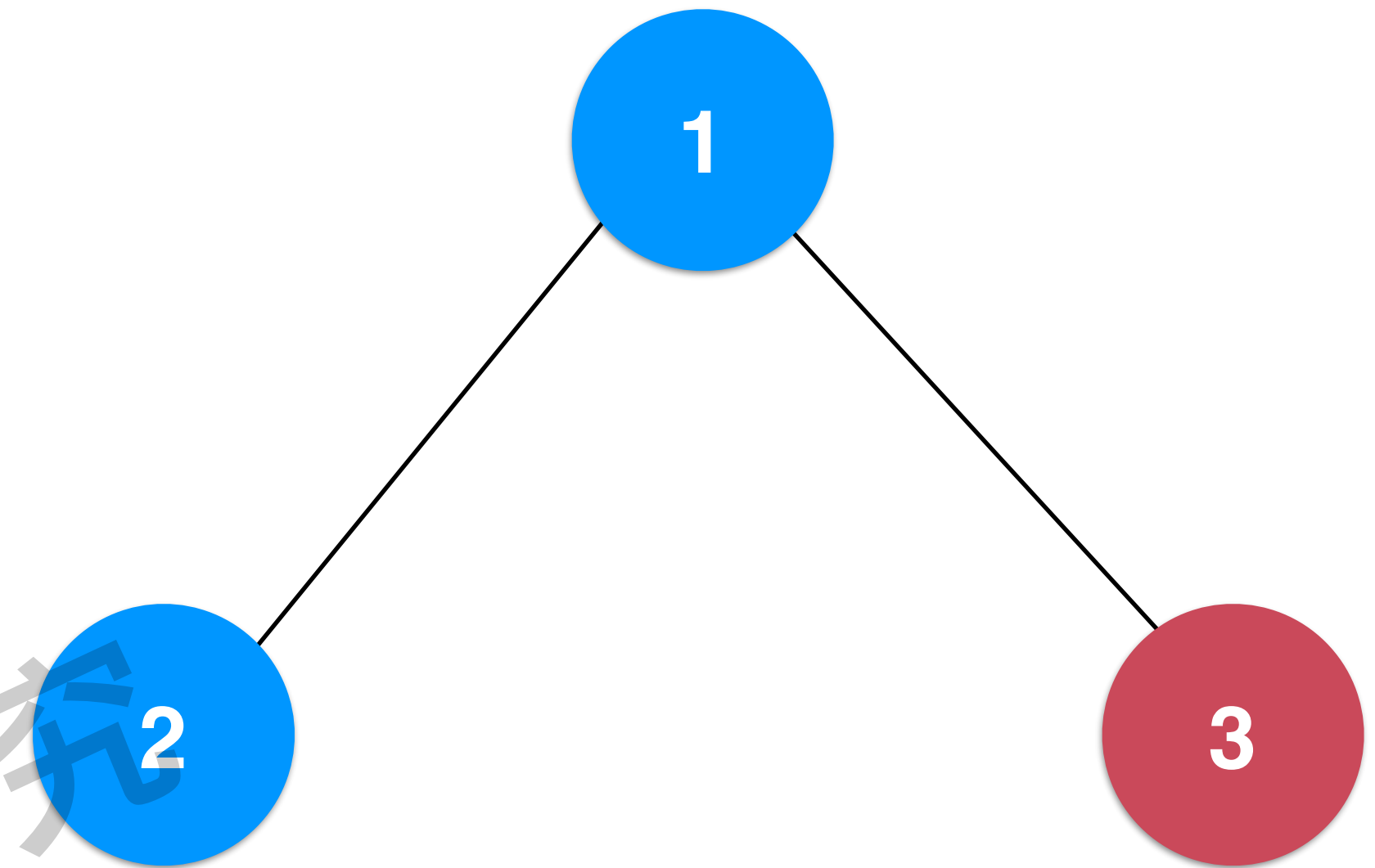
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

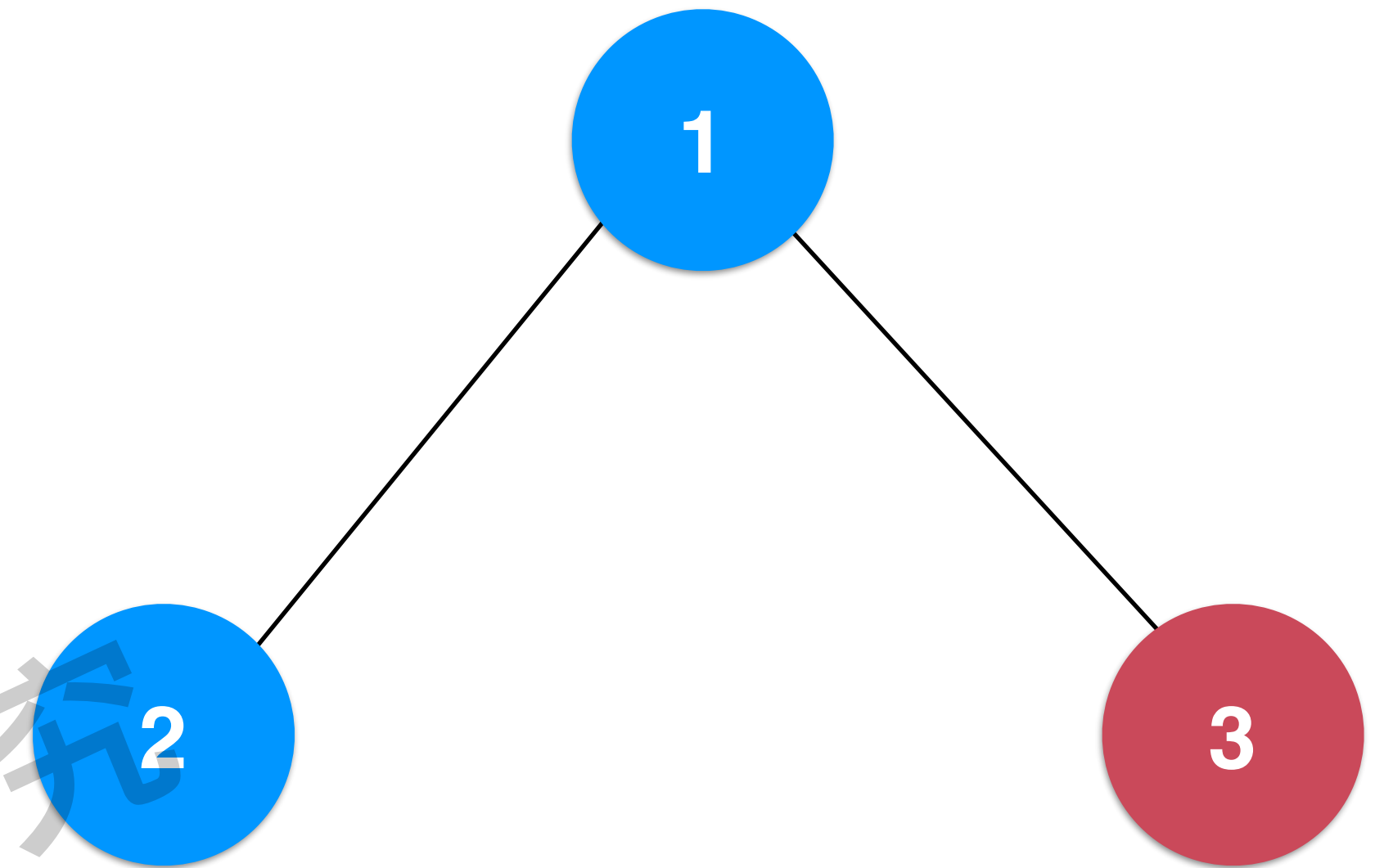
Stack



# 144. Binary Tree Preorder Traversal

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

Stack



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

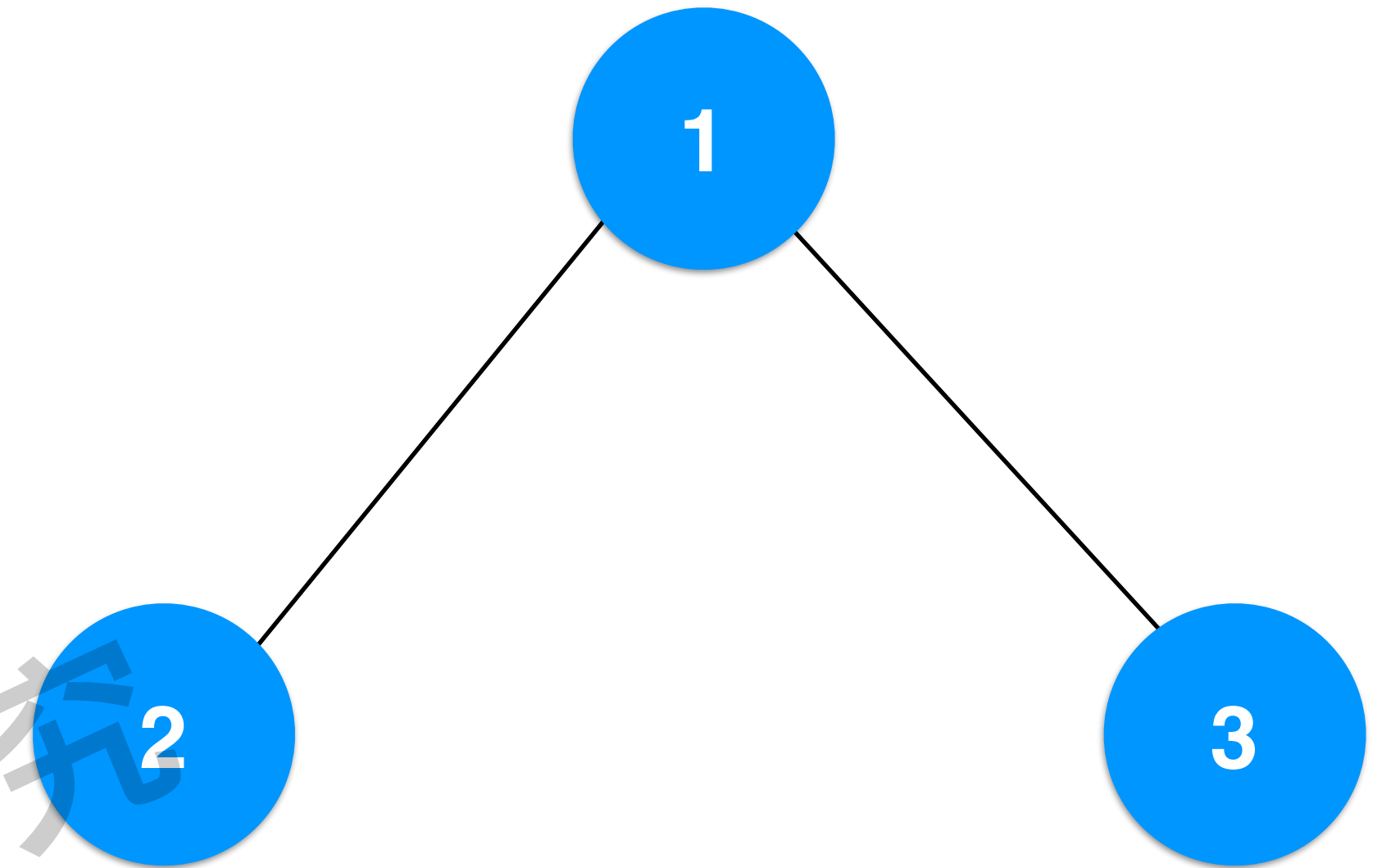
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 1

go 1-L

go 1-R

**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



希望大家自己完成后续模拟

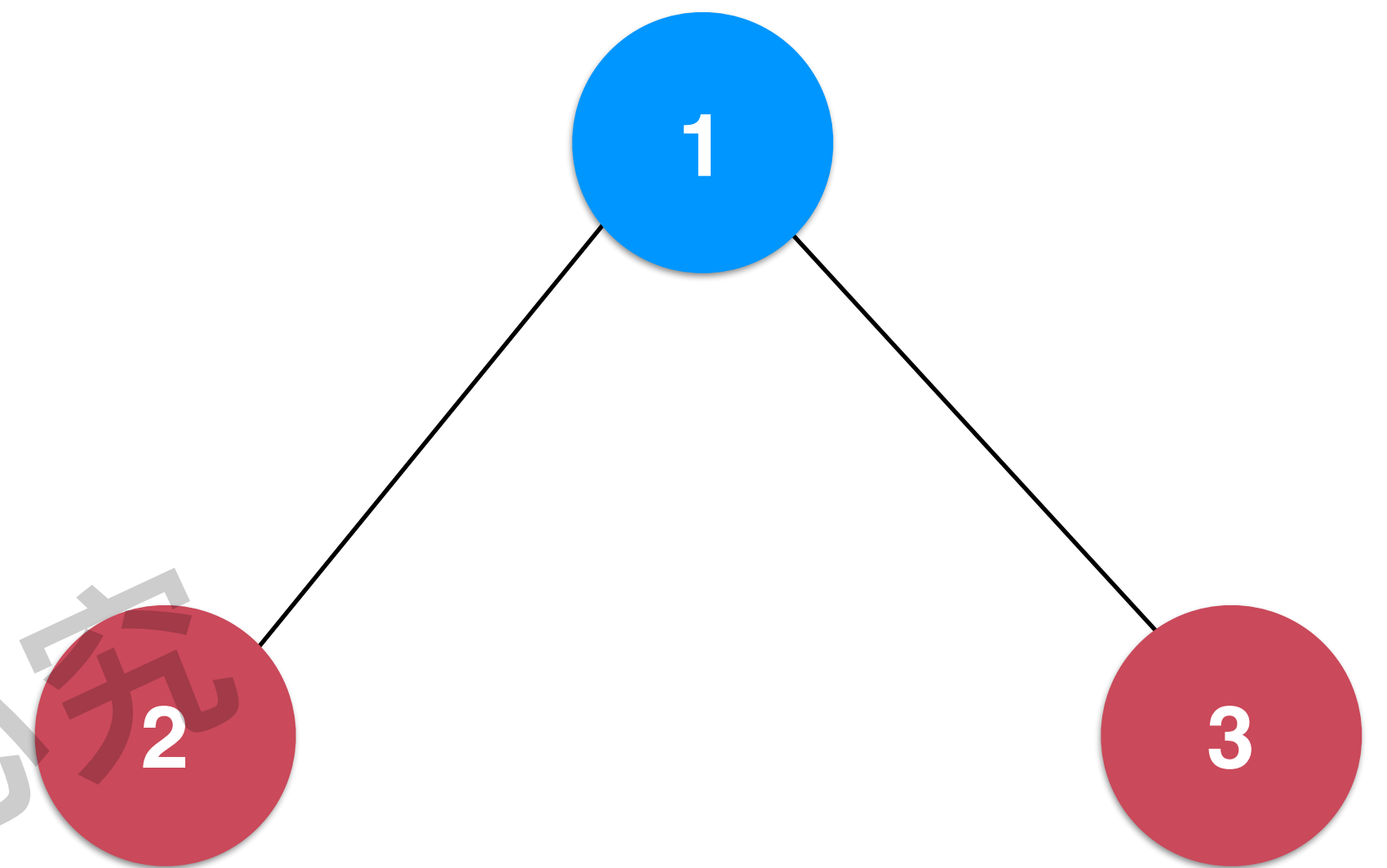


希望大家对前，中，后序遍历，都使用类似的模拟

慕课网《玩转算法面试》  
讲师：liuyubobobo  
版权所有，侵权必究

# 使用栈模拟系统栈，写出非递归程序

# 144. Binary Tree Preorder Traversal



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

cout 1

go 1-L

go 1-R

**Stack**

# 144. Binary Tree Preorder Traversal



cout 1

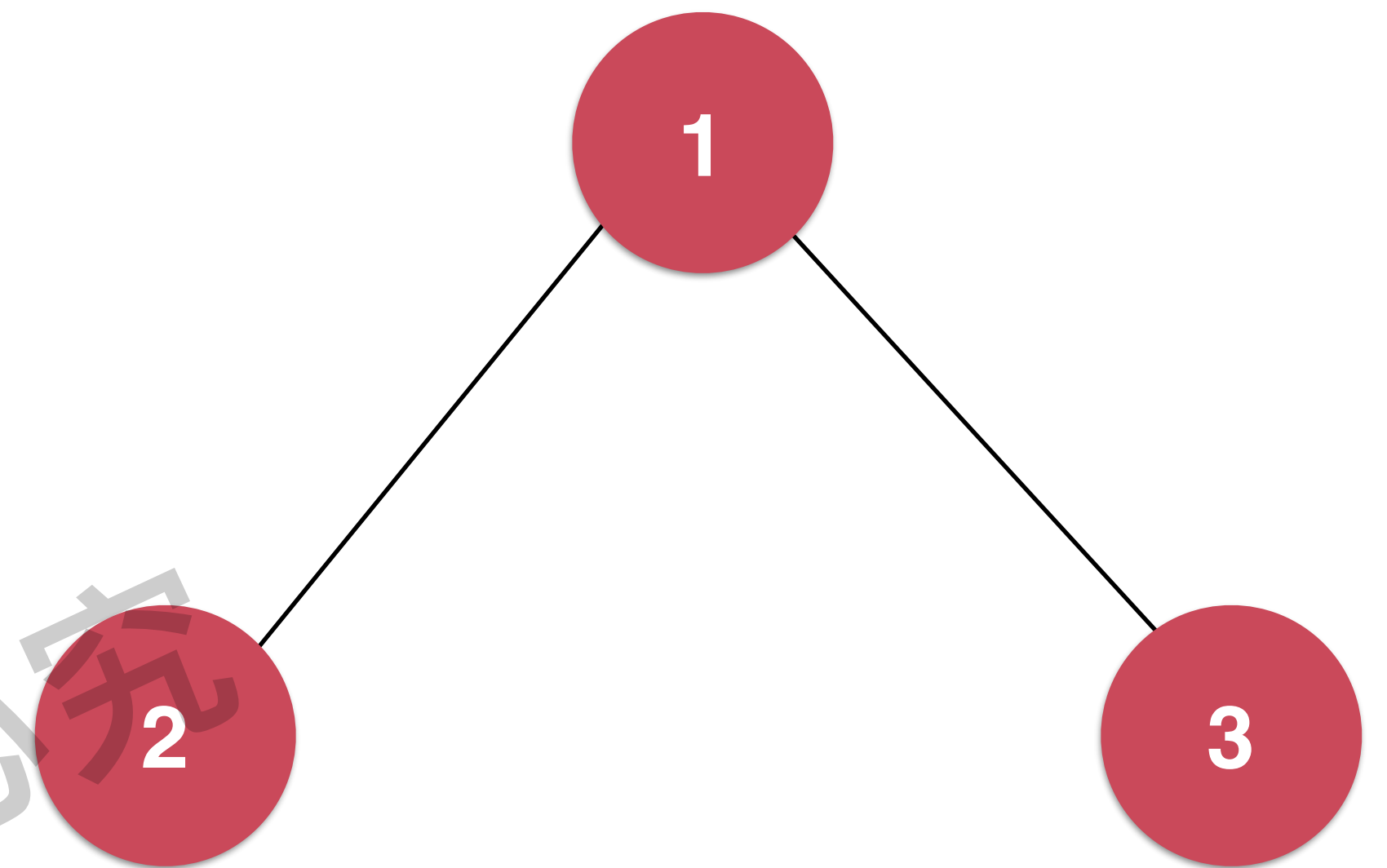
go 1-L

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal



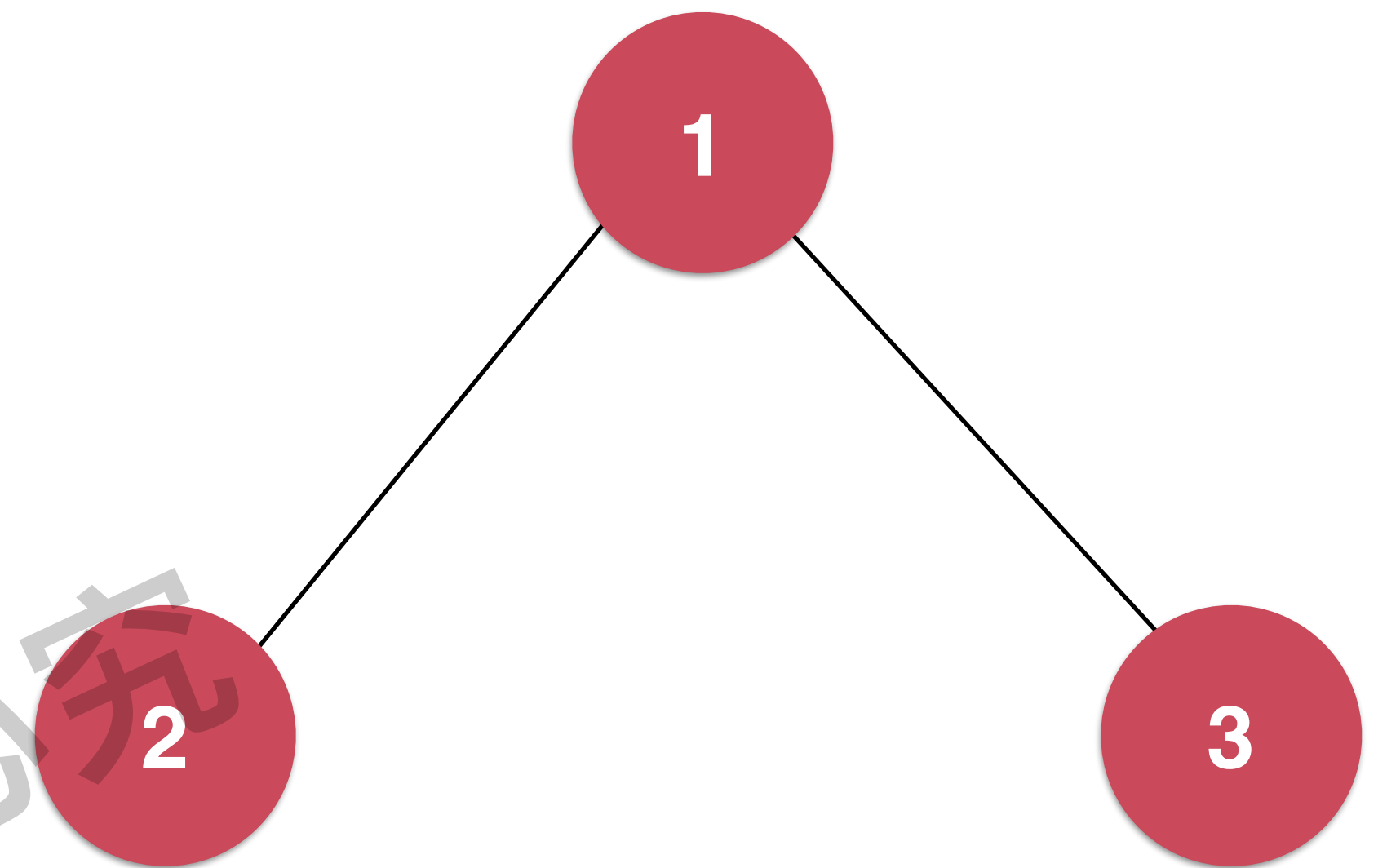
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

go 1

**Stack**

# 144. Binary Tree Preorder Traversal

go 1



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

**Stack**

# 144. Binary Tree Preorder Traversal



cout 1

go 1-L

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



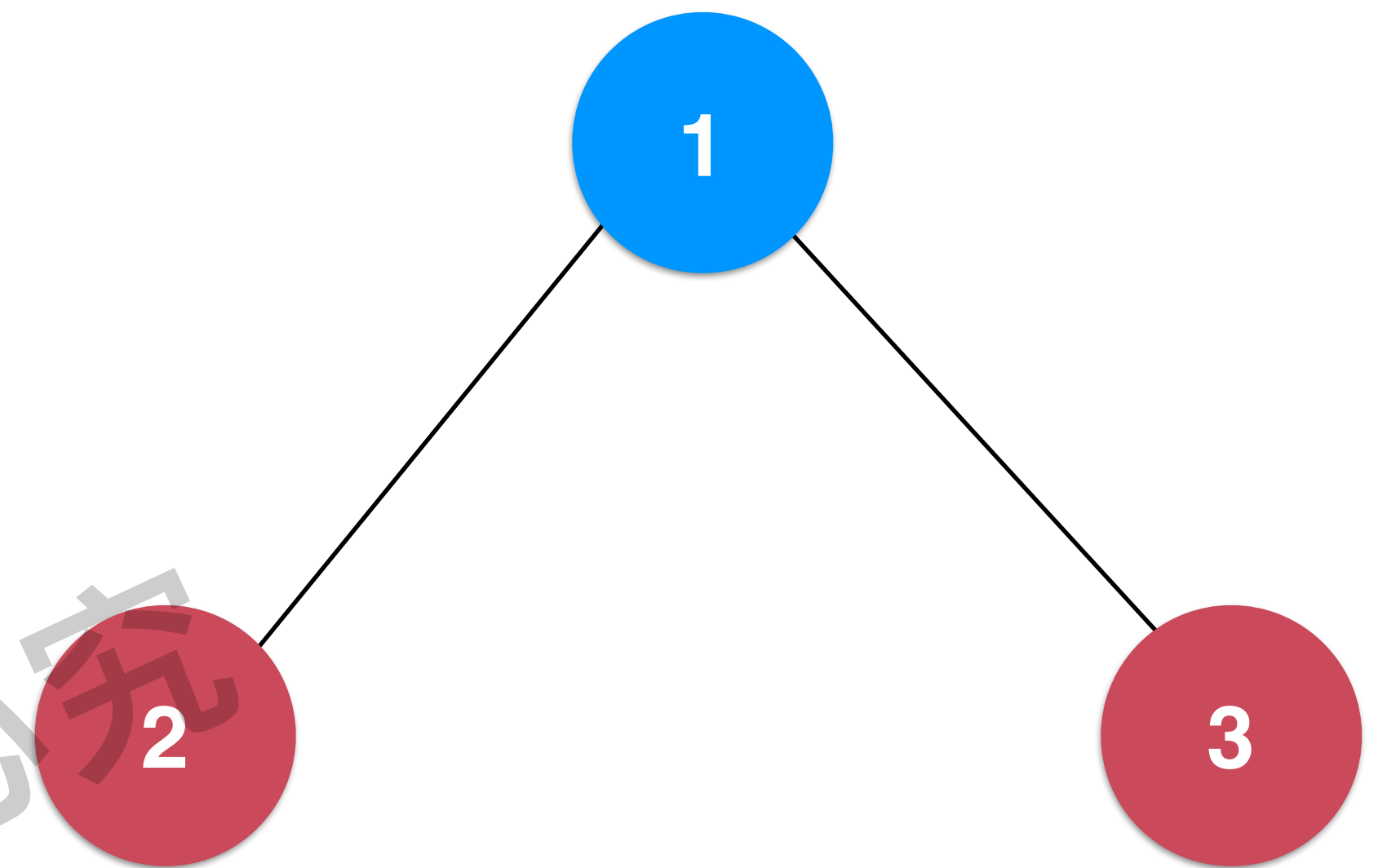
# 144. Binary Tree Preorder Traversal

cout 1

go 1-L

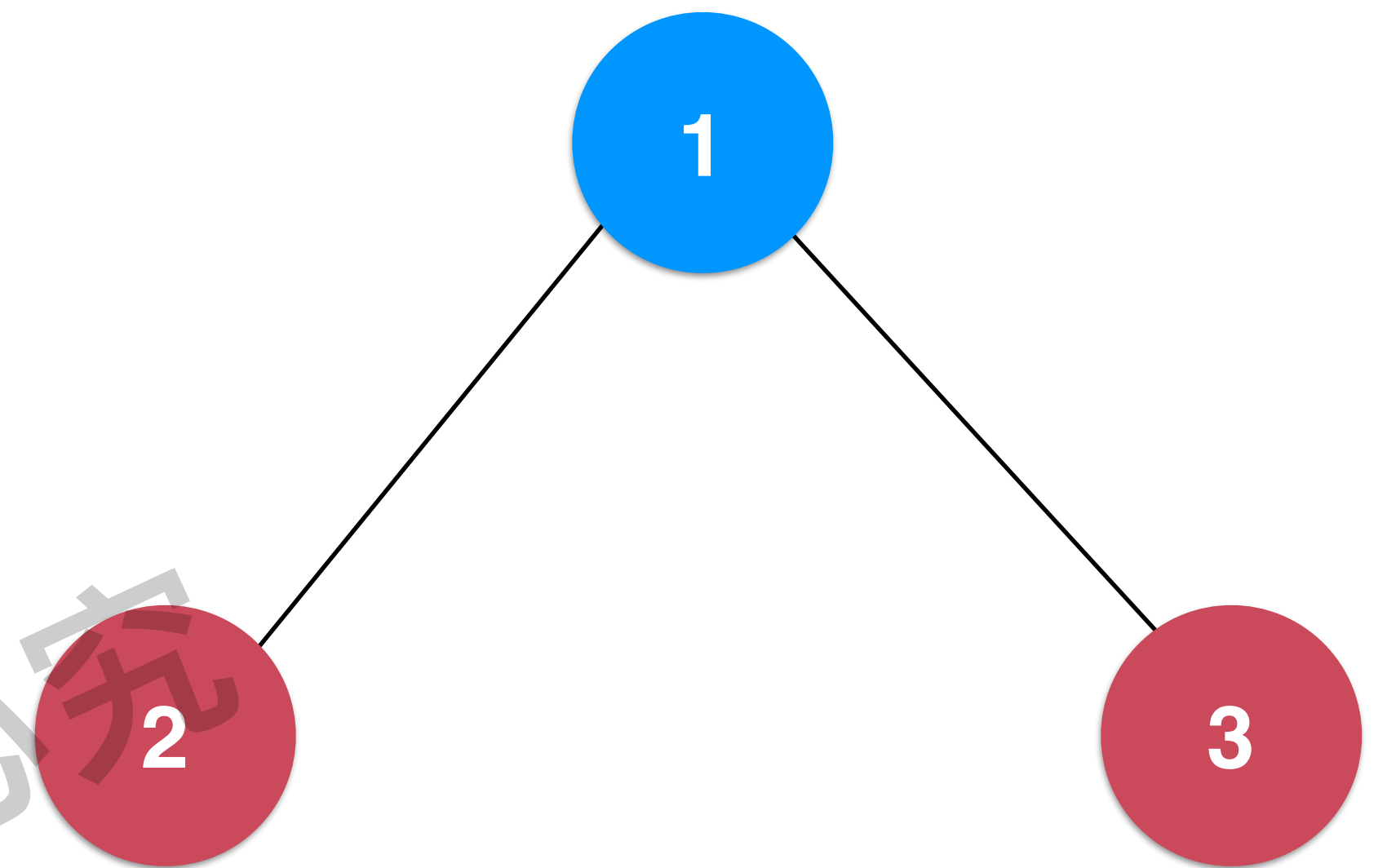
go 1-R

**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal



go 1-L

go 1-R

**Stack**

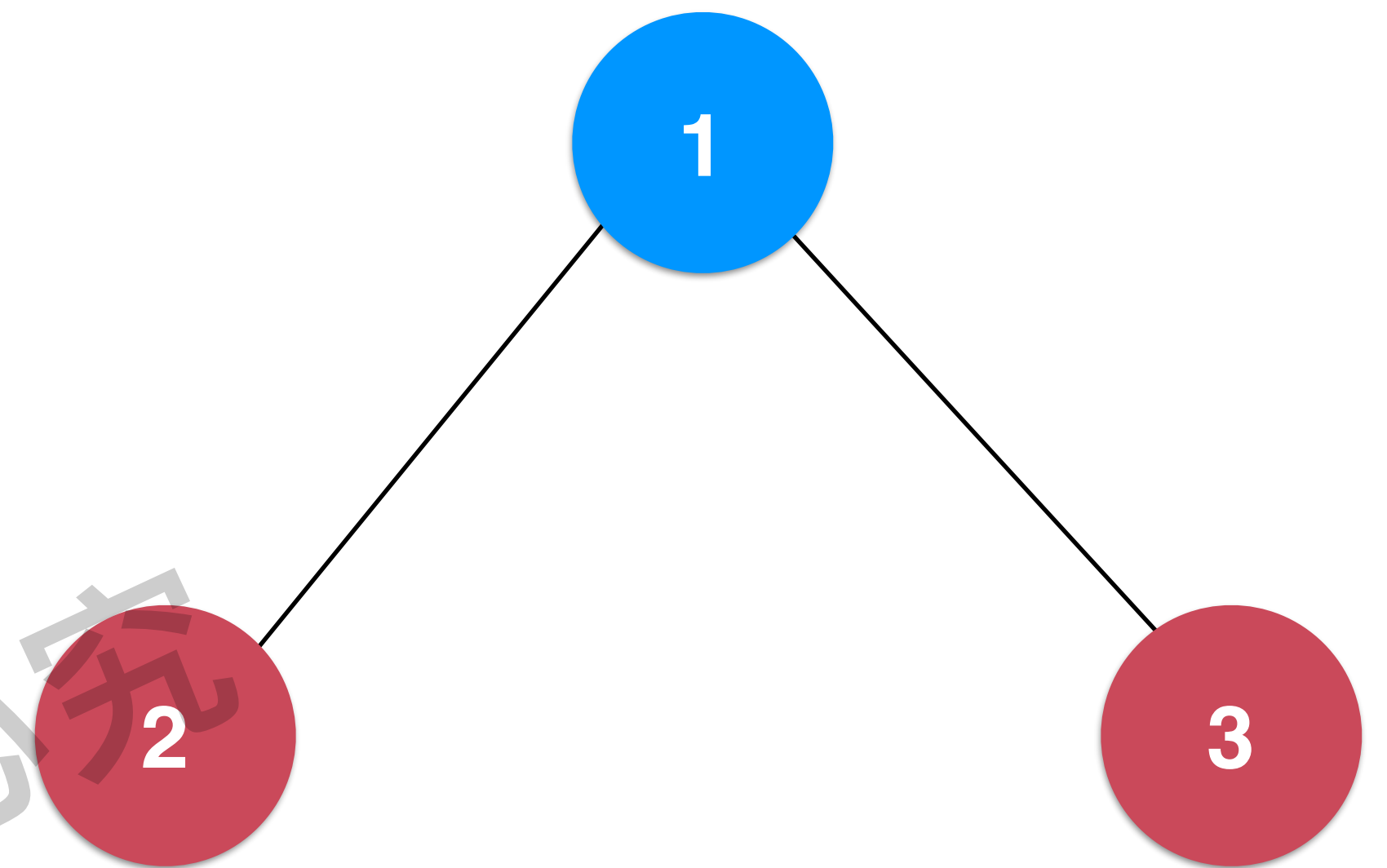
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

go 1-L

go 1-R

**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal



cout 2

go 2-L

go 2-R

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

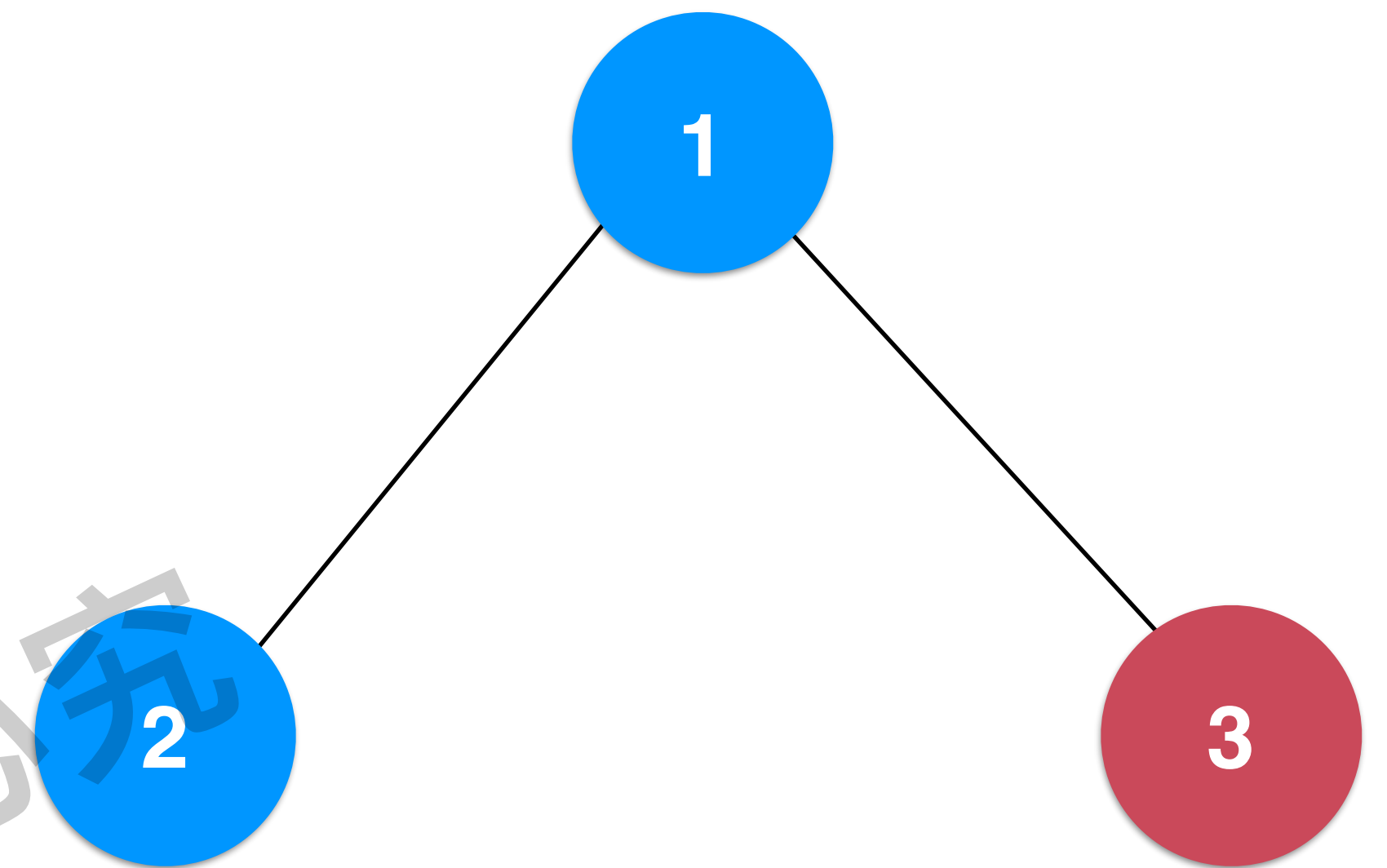
cout 2

go 2-L

go 2-R

go 1-R

**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

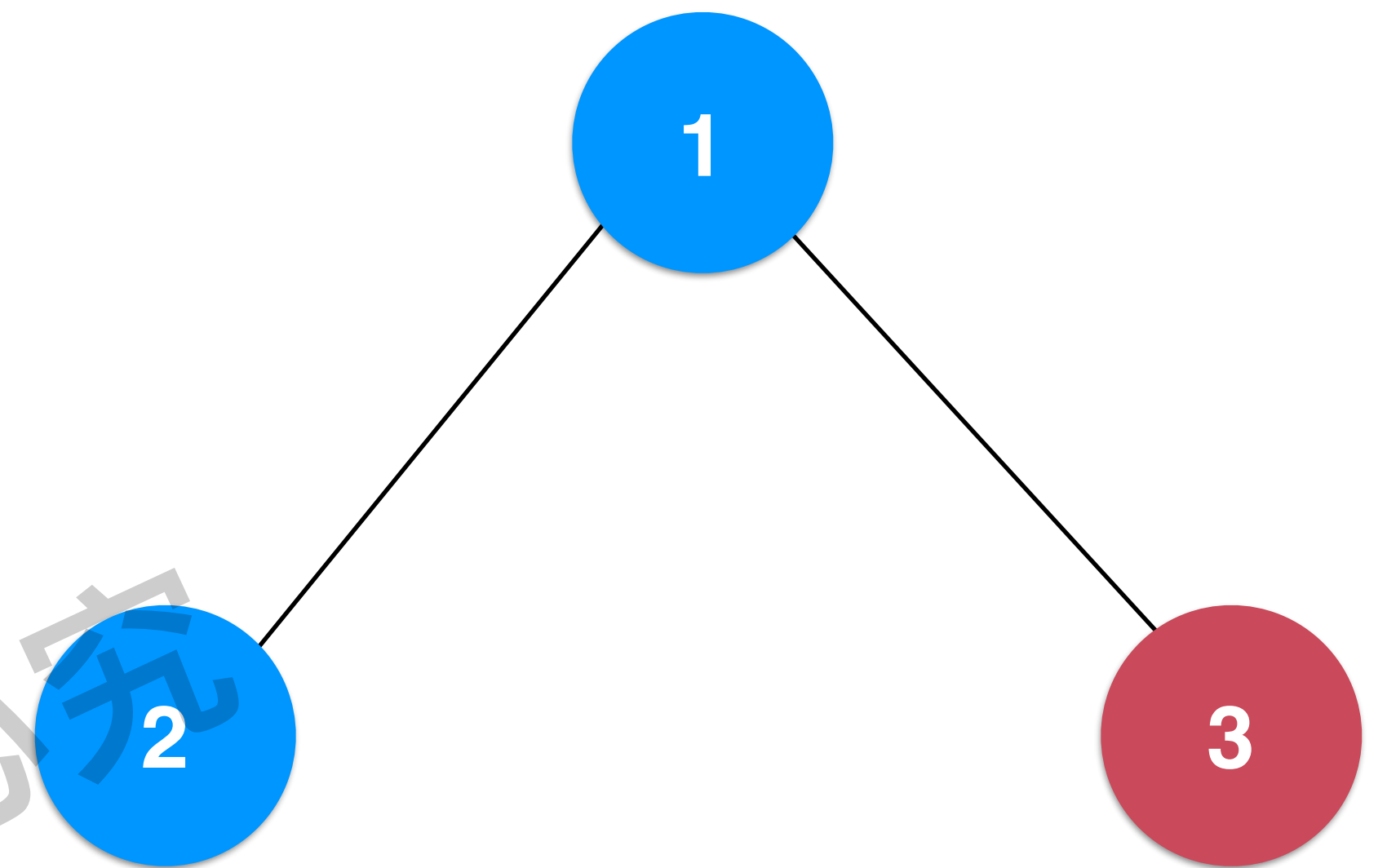
cout 2

go 2-L

go 2-R

go 1-R

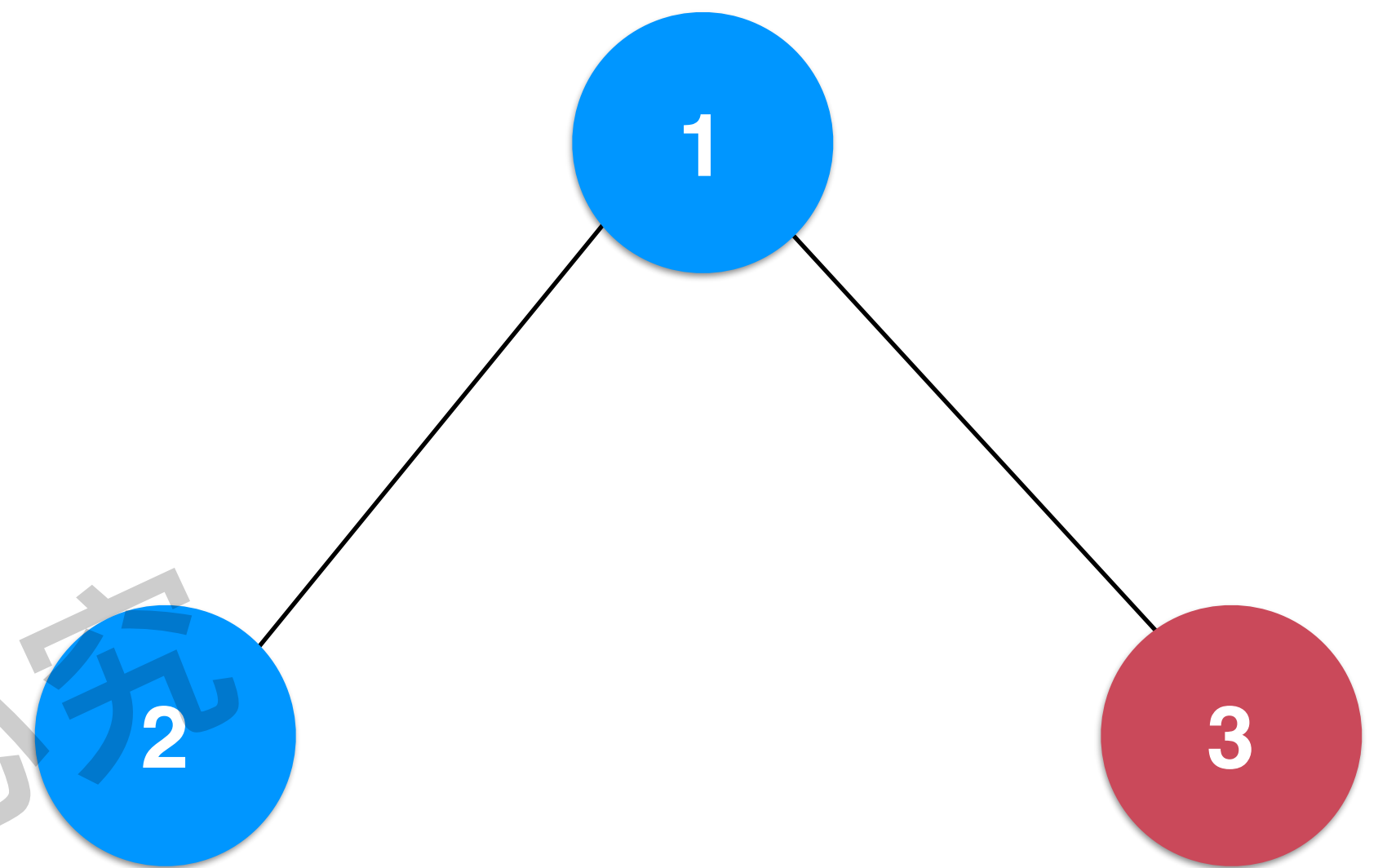
**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```



# 144. Binary Tree Preorder Traversal



go 2-L

go 2-R

go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

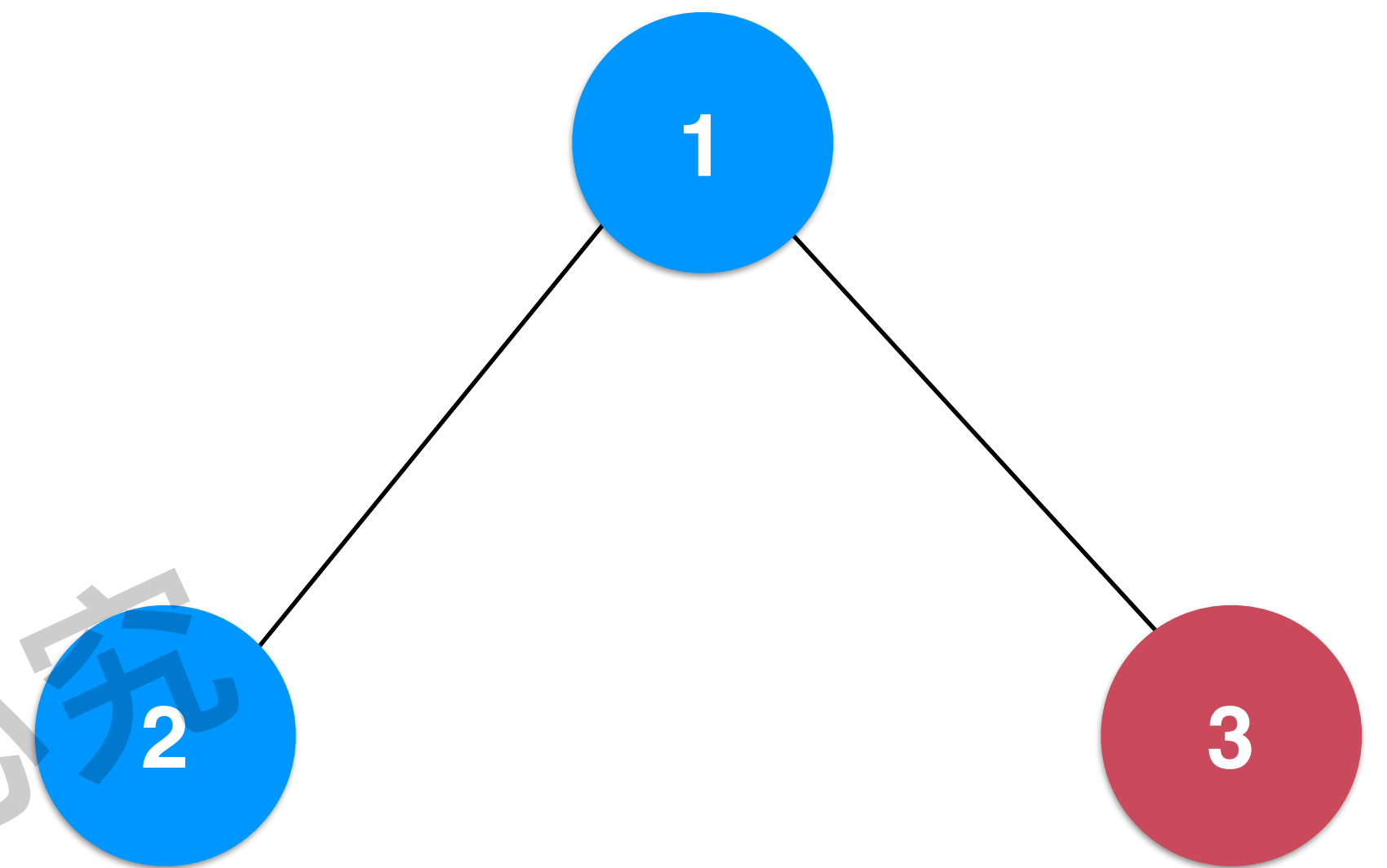
# 144. Binary Tree Preorder Traversal

go 2-L

go 2-R

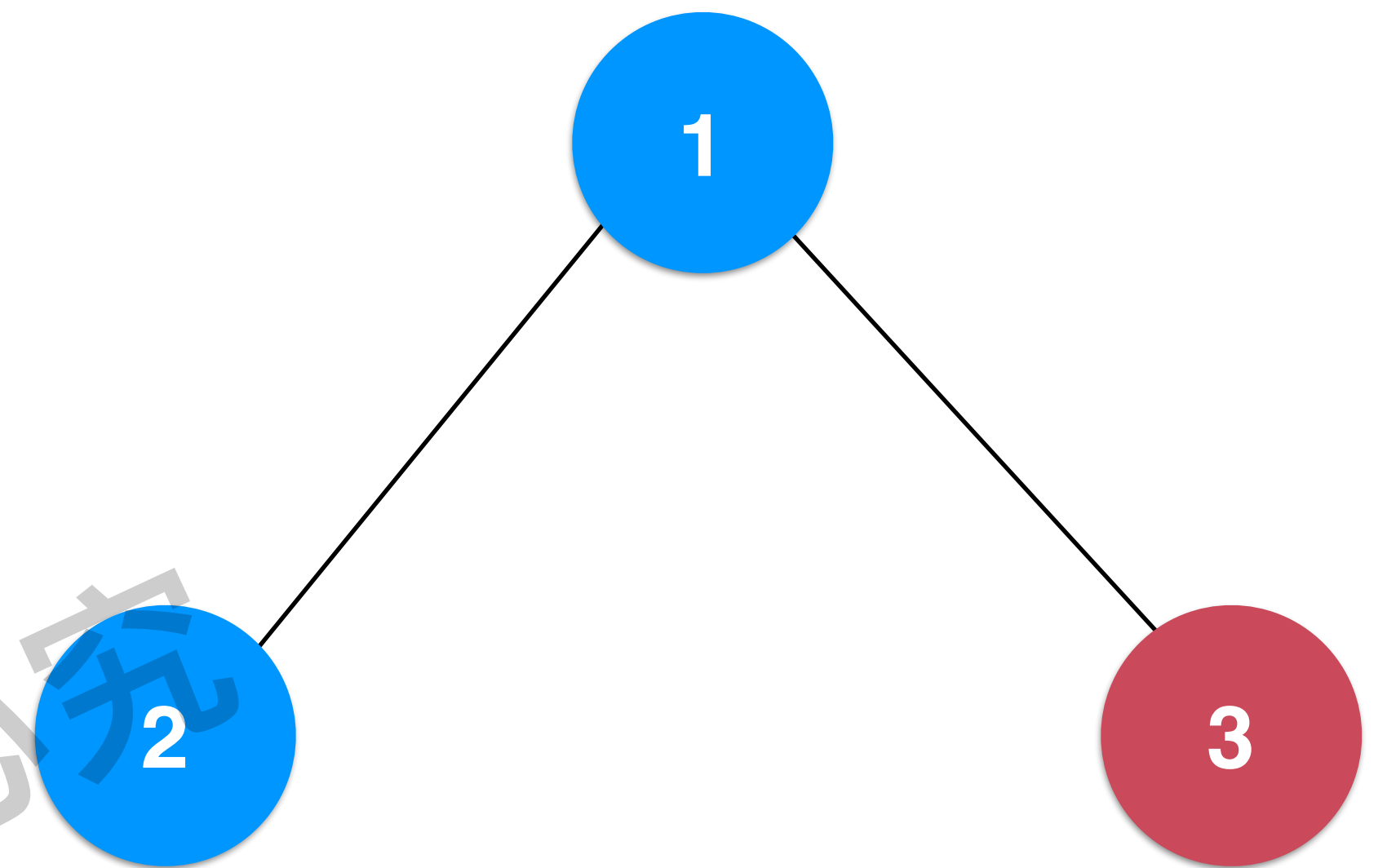
go 1-R

**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal



go 2-R

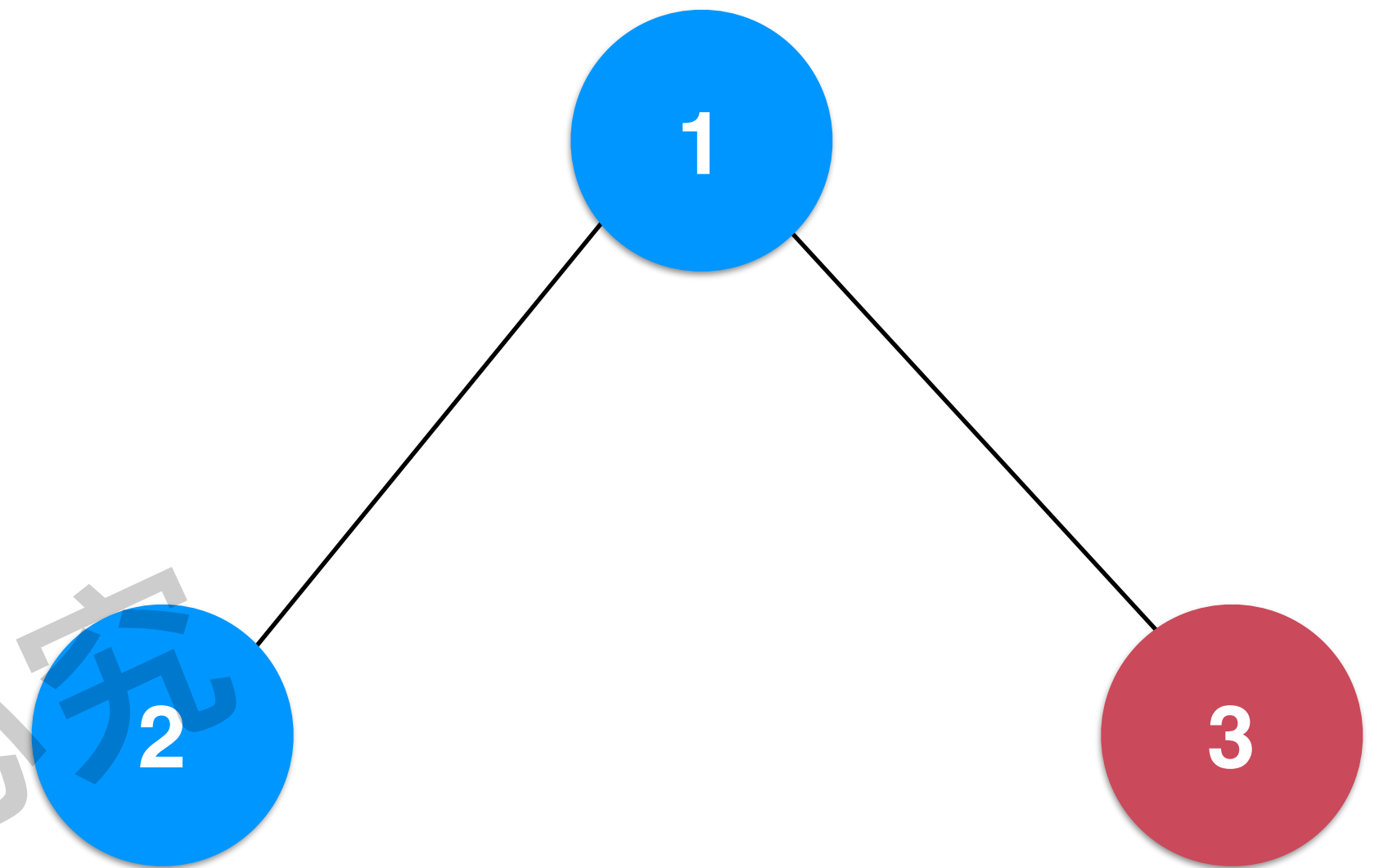
go 1-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

go 2-R

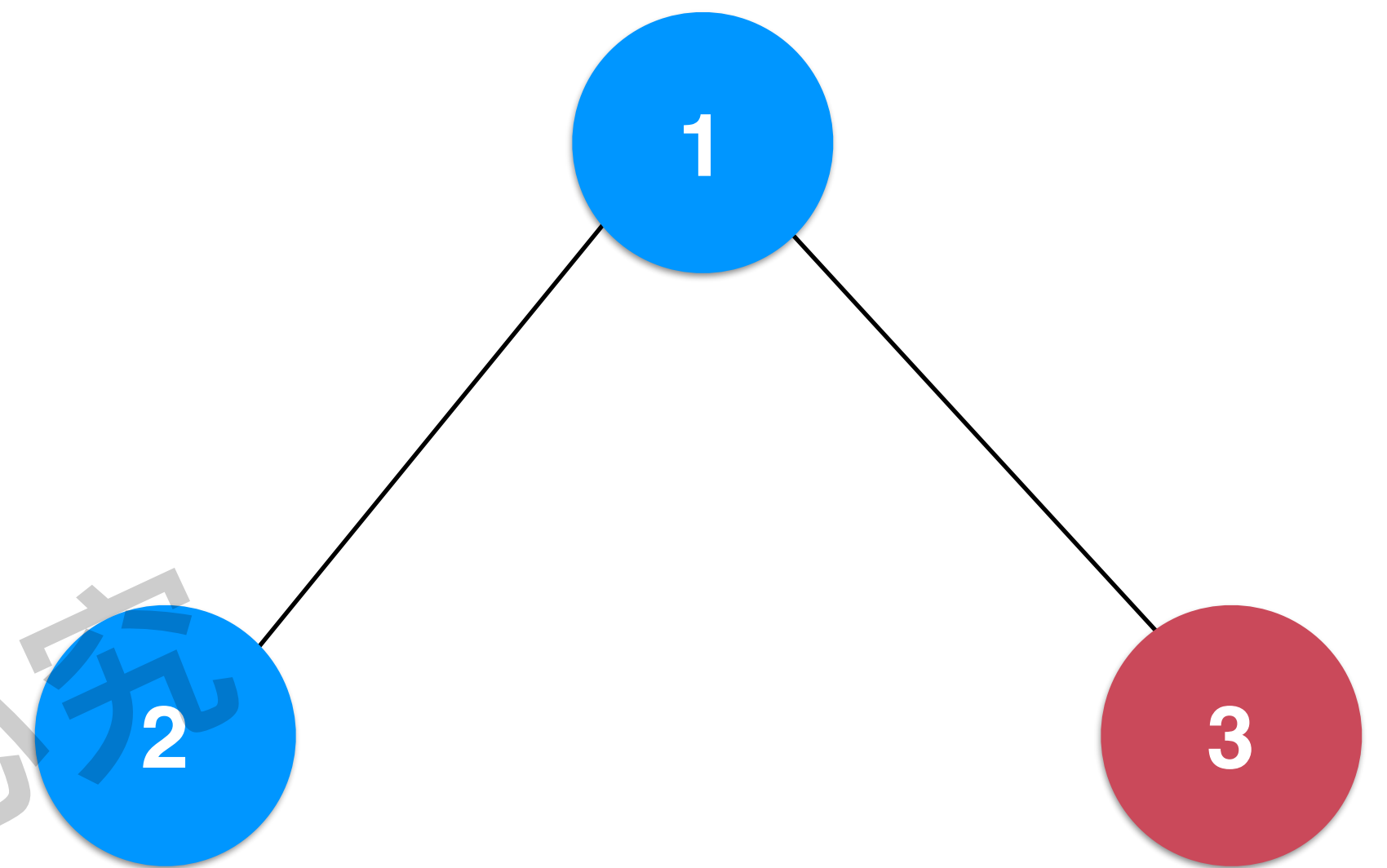


```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

go 1-R

**Stack**

# 144. Binary Tree Preorder Traversal



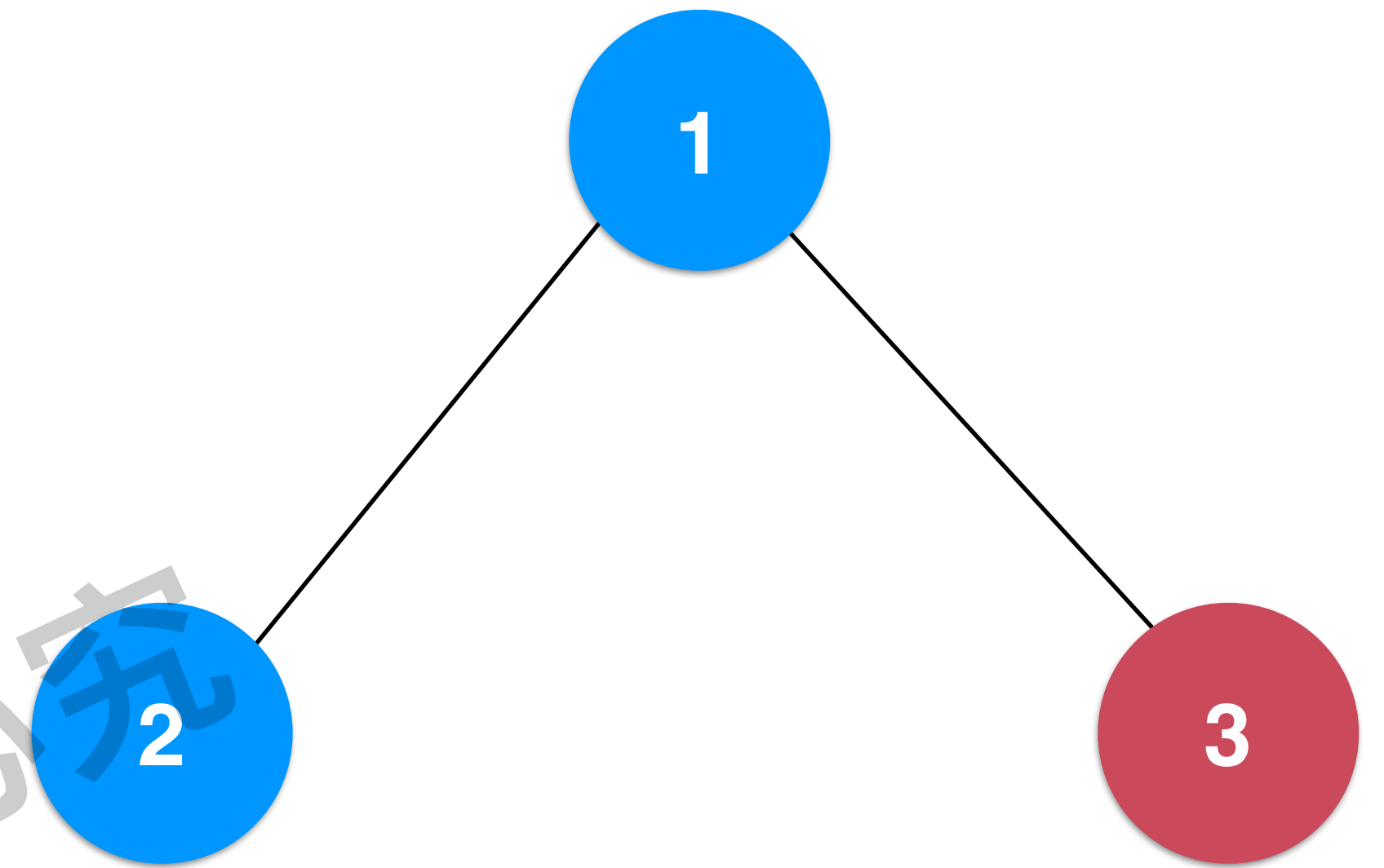
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

go 1-R

**Stack**

# 144. Binary Tree Preorder Traversal

go 1-R



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

**Stack**



# 144. Binary Tree Preorder Traversal



cout 3

go 3-L

go 3-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal



cout 3

go 3-L

go 3-R

**Stack**

```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

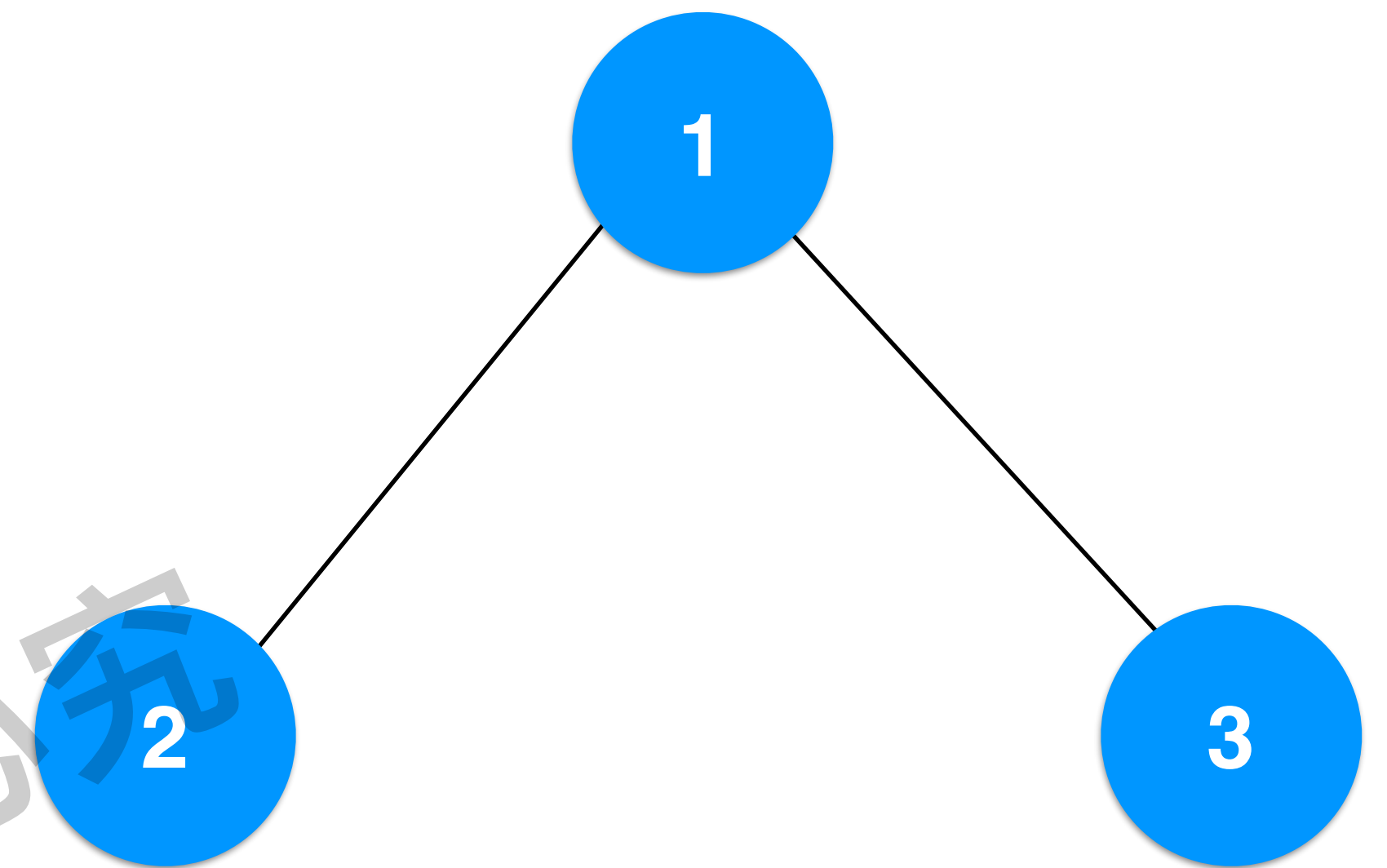
# 144. Binary Tree Preorder Traversal

cout 3

go 3-L

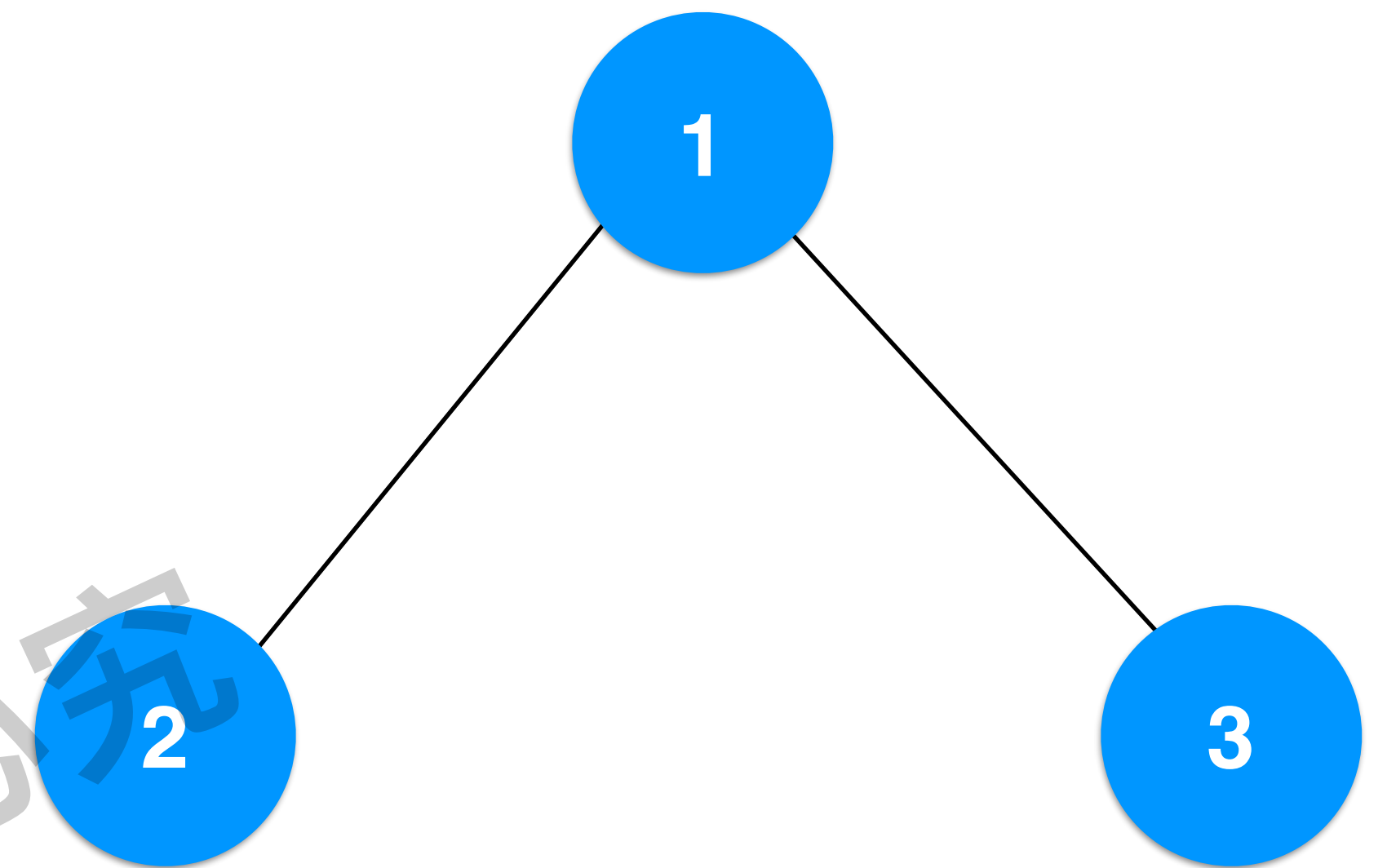
go 3-R

**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal



go 3-L

go 3-R

**Stack**

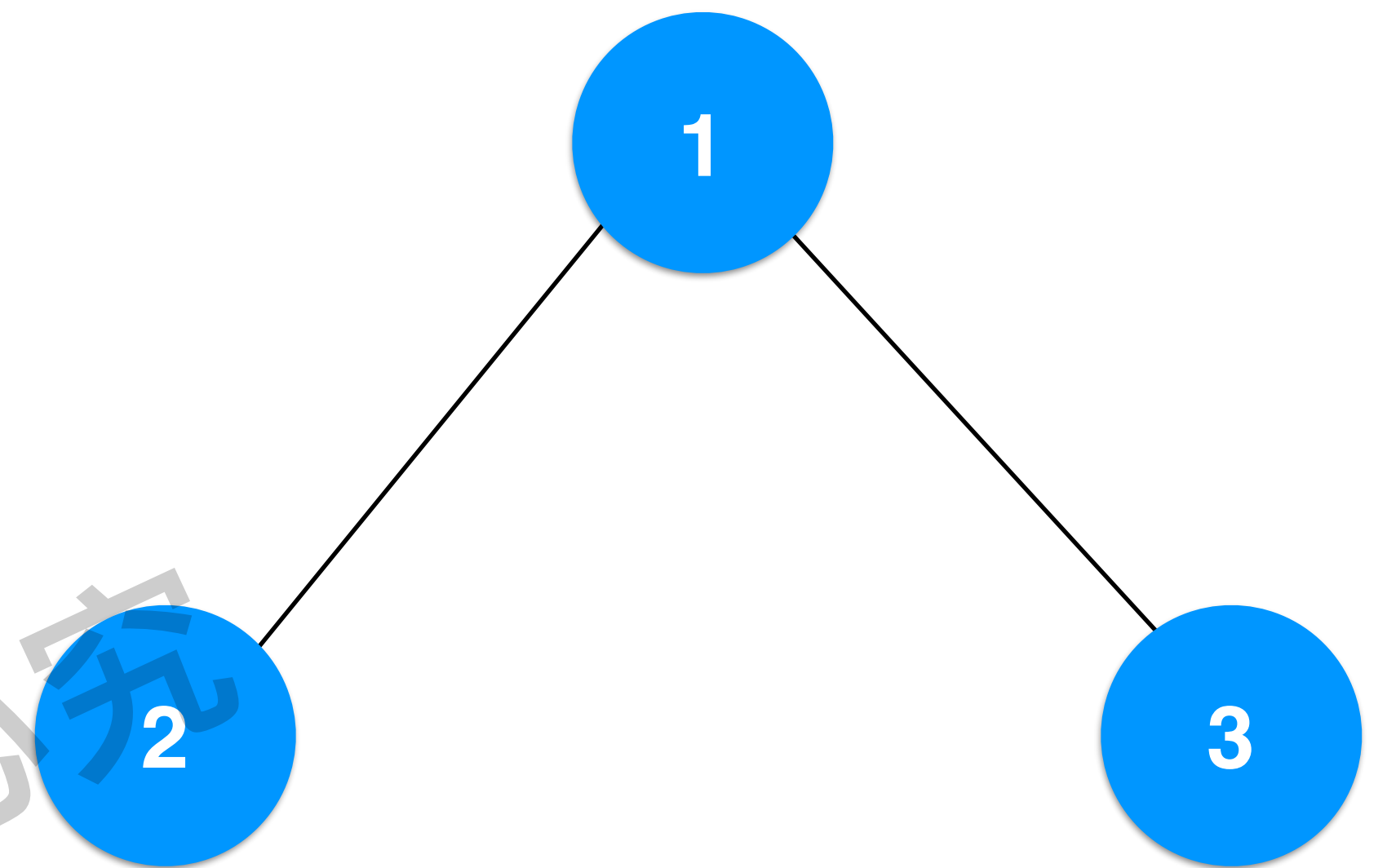
```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal

go 3-L

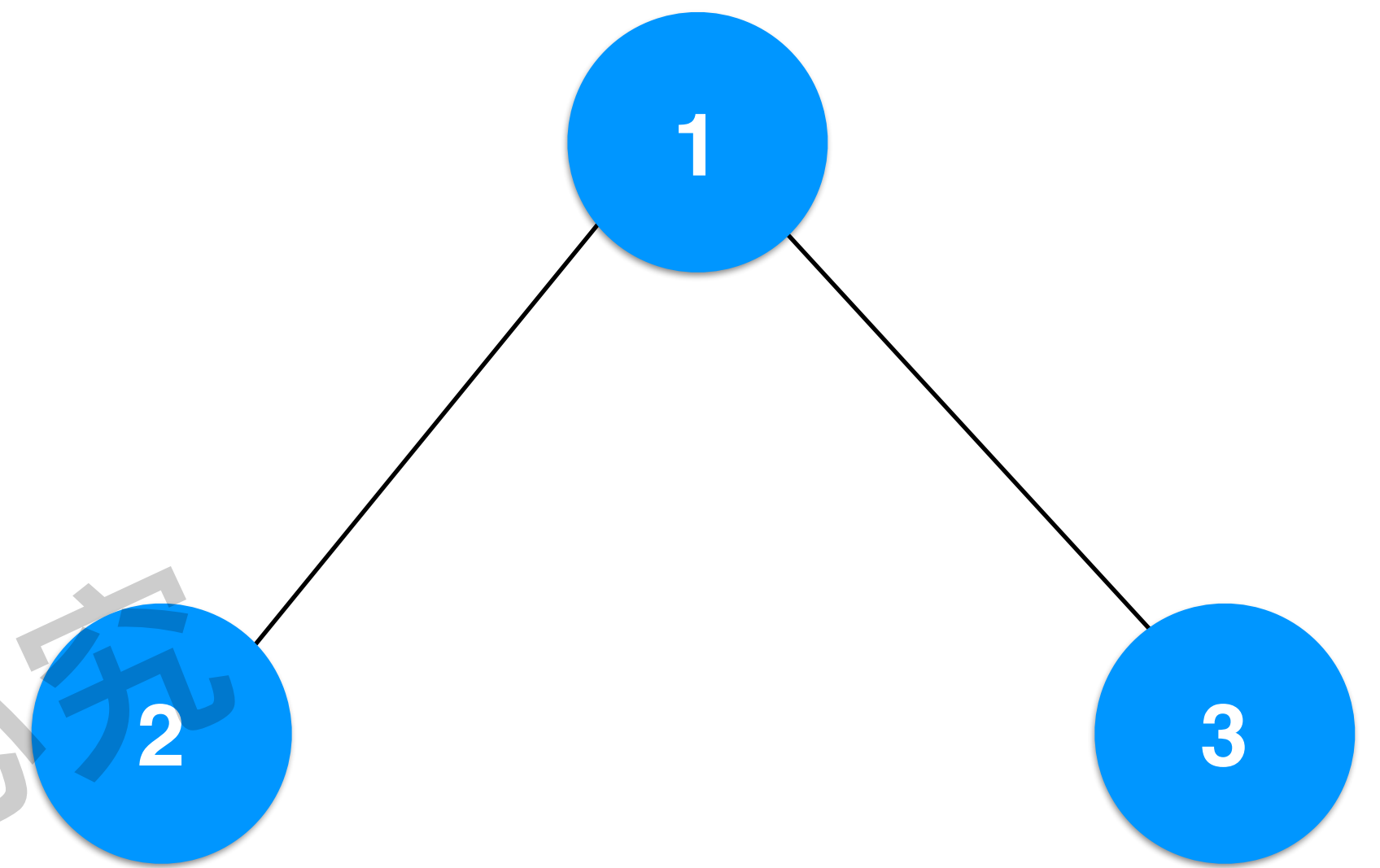
go 3-R

**Stack**



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

# 144. Binary Tree Preorder Traversal



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

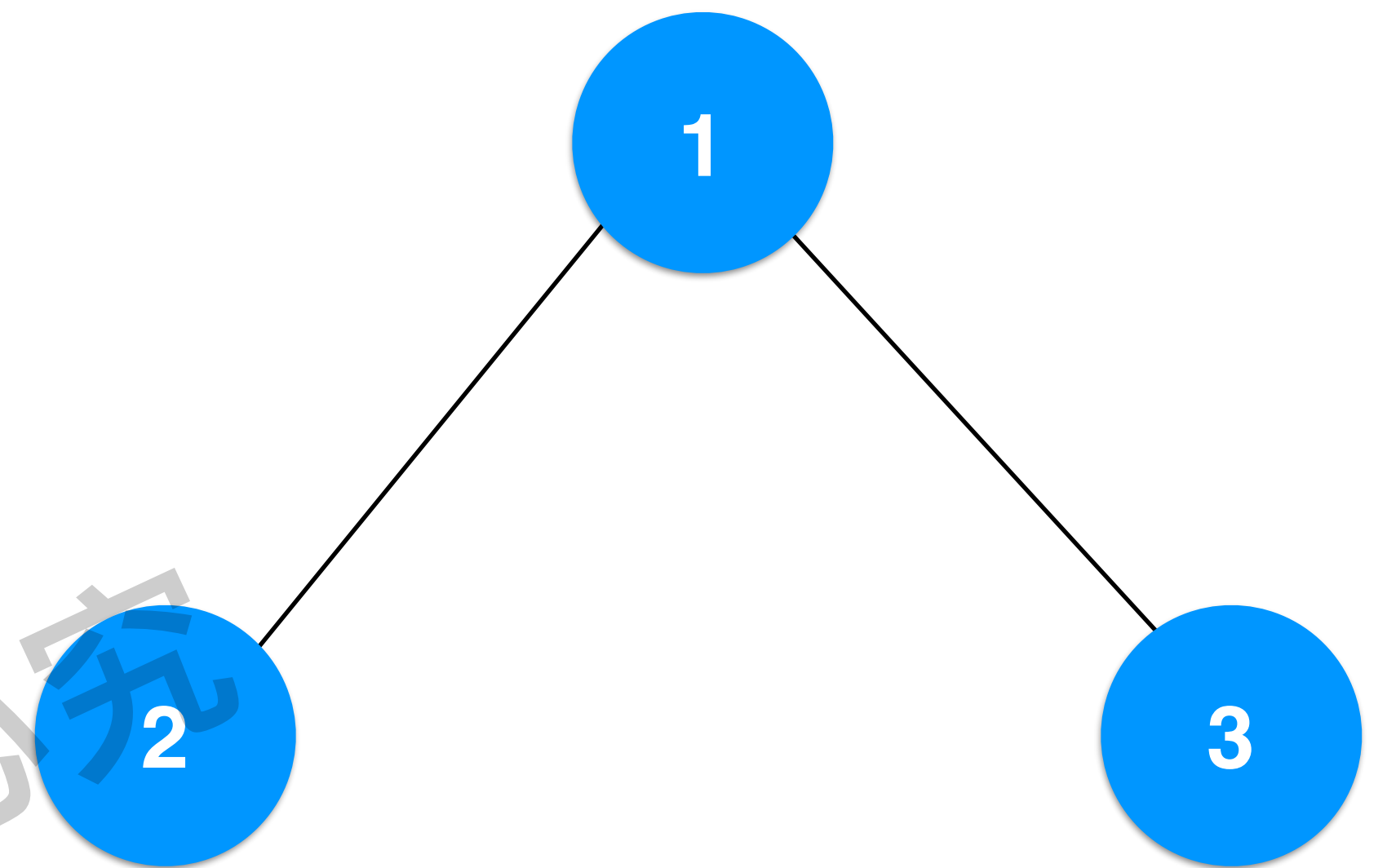
go 3-R

**Stack**



# 144. Binary Tree Preorder Traversal

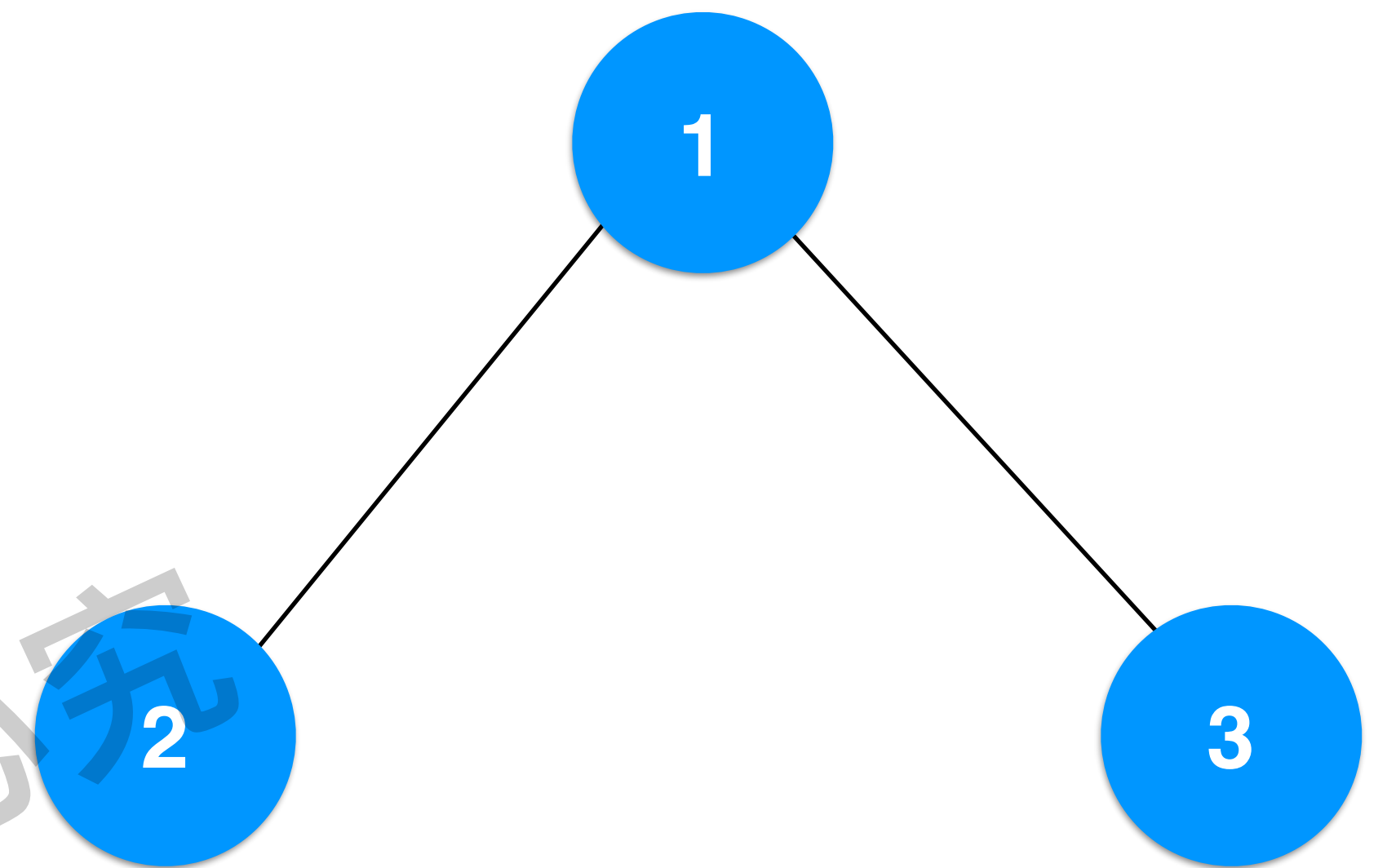
go 3-R



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

**Stack**

# 144. Binary Tree Preorder Traversal



```
void preorder( TreeNode* node ){  
    if( node ){  
        cout << node->val;  
        preorder( node->left );  
        preorder( node->right );  
    }  
}
```

**Stack**

# 实践：非递归完成前序遍历

慕课网《玩转算法面试》  
讲师：lilyubobobo

版权所有，侵权必究

# 实践：修改非递归完成中序遍历

慕课网《玩转算法面试》

讲师：liuyubobobo

版权所有，侵权必究

# 实践：修改非递归完成后序遍历

慕课网《玩转算法面试》

讲师：liuyubobobo

版权所有，侵权必究

# 教科书上的经典非递归方法

慕课网《玩转算法面试》  
讲师：lucybobobo

版权所有，侵权必究



# 341. Flatten Nested List Iterator



给出一个嵌套的整型列表。列表中的项或者为一个整数，或者是另一个列表。设计一个迭代器，遍历这个整型列表中的所有整数。

- 如 `[ [1, 1], 2, [1, 1] ]`
- 如 `[ 1, [ 4, [6] ] ]`

# 341. Flatten Nested List Iterator

```
class NestedInteger {  
public:  
    bool isInteger() const;  
    int getInteger() const;  
    const vector<NestedInteger> &getList() const;  
};  
  
class NestedIterator {  
public:  
    NestedIterator(vector<NestedInteger> &nestedList) { }  
  
    int next() { }  
  
    bool hasNext() { }  
};
```

# 341. Flatten Nested List Iterator

```
class NestedIterator {  
public:  
    NestedIterator(vector<NestedInteger> &nestedList) {}  
  
    int next() {}  
  
    bool hasNext() {}  
};
```

对于  $[[1, 1], 2, [1, 1]]$ ，在hasNext()为true的情况下  
不断调用next()，依次获得 1 1 2 1 1

# 341. Flatten Nested List Iterator

```
class NestedIterator {  
public:  
    NestedIterator(vector<NestedInteger> &nestedList) {}  
  
    int next() {}  
  
    bool hasNext() {}  
};
```

对于 [ 1 , [ 4 , [ 6 ] ] ]，在hasNext()为true的情况下  
不断调用next()，依次获得 1 4 6

慕课网《玩转算法面试》

# 队列 Queue

讲师：lilyubobobo

版权所有，侵权必究

# 队列 Queue

队列的基本应用 - 广度优先遍历

- 树；层序遍历
- 图；无权图的最短路径



# 102. Binary Tree Level Order Traversal



对一个二叉树进行层序遍历

# 102. Binary Tree Level Order Traversal

对一个二叉树进行层序遍历



慕课网《玩转算法面试》

# 实践：实现102

讲师：liuyubobobo

版权所有，侵权必究

# 107. Binary Tree Level Order Traversal II

对一个二叉树进行层序遍历，返回从底层到上层每层的节点。



# 103. Binary Tree Zigzag Level Order Traversal



Bloomberg



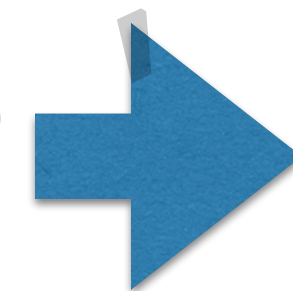
对一个二叉树进行层序遍历，按照“之”字形的顺序返回所有节点

3  
/  
9

20

/  
15

7



[

[3],

[20,9],

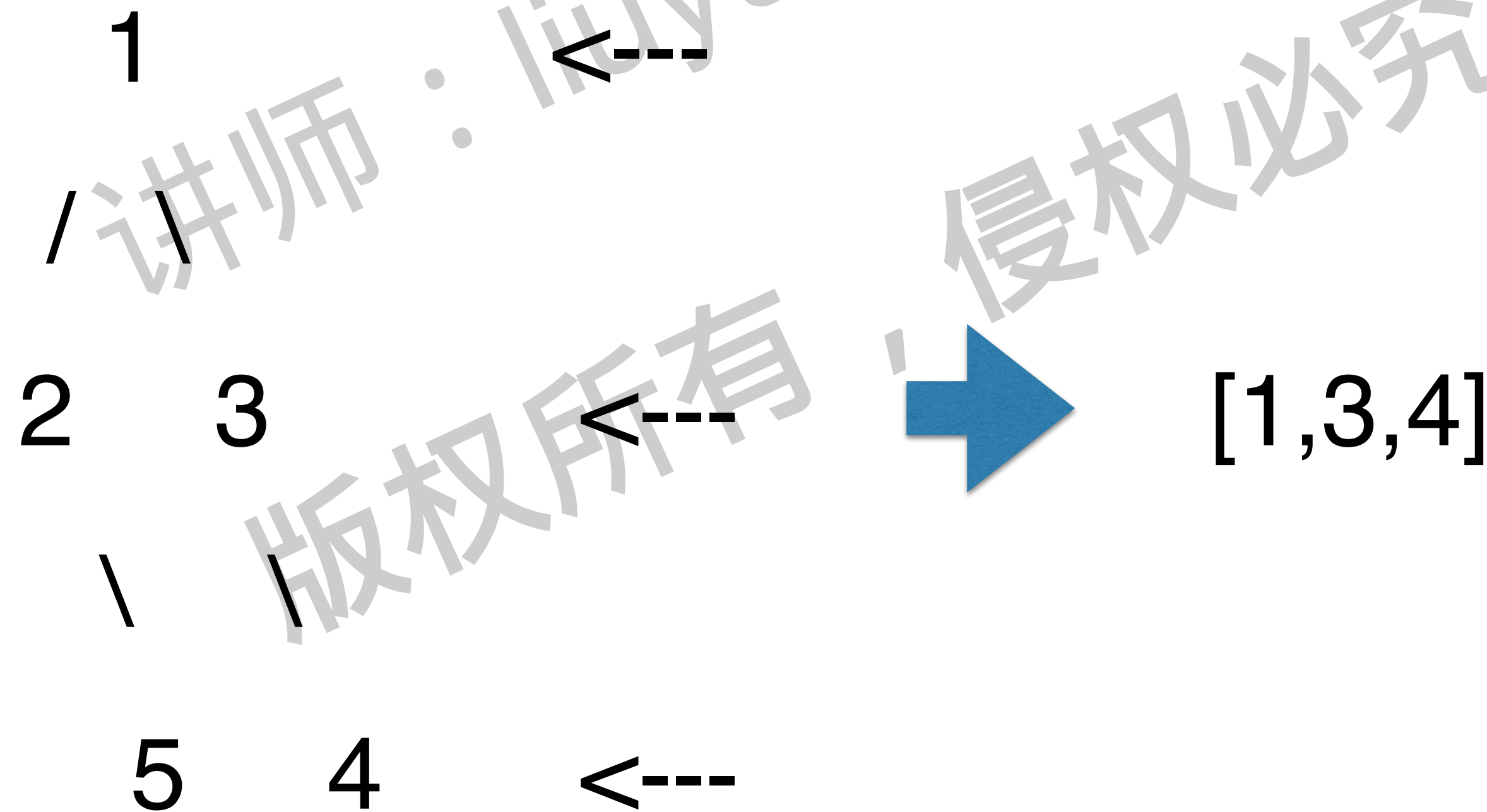
[15,7]

]

# 199. Binary Tree Right Side View

**amazon**

想象你站在一棵二叉树的右侧，返回所有你能看见的节点。





慕课网《玩转算法面试》

# BFS和图的最短路径

讲师：liuyubobobo

版权所有，侵权必究

# 279. Perfect Squares



给出一个正整数 $n$ ，寻找最少的完全平方数，使他们的和为 $n$ 。

- 完全平方数：1, 4, 9, 16...

- $12 = 4 + 4 + 4$

- $13 = 4 + 9$

# 279. Perfect Squares



给出一个正整数 $n$ ，寻找最少的完全平方数，使他们的和为 $n$ 。

- 没有解怎么办？
- 是否可能没有解？

# 279. Perfect Squares

直觉解法？贪心？

$$12 = 9 + 1 + 1 + 1$$

$$12 = 4 + 4 + 4$$

# 279. Perfect Squares

对问题建模：

整个问题转化为一个图论问题。

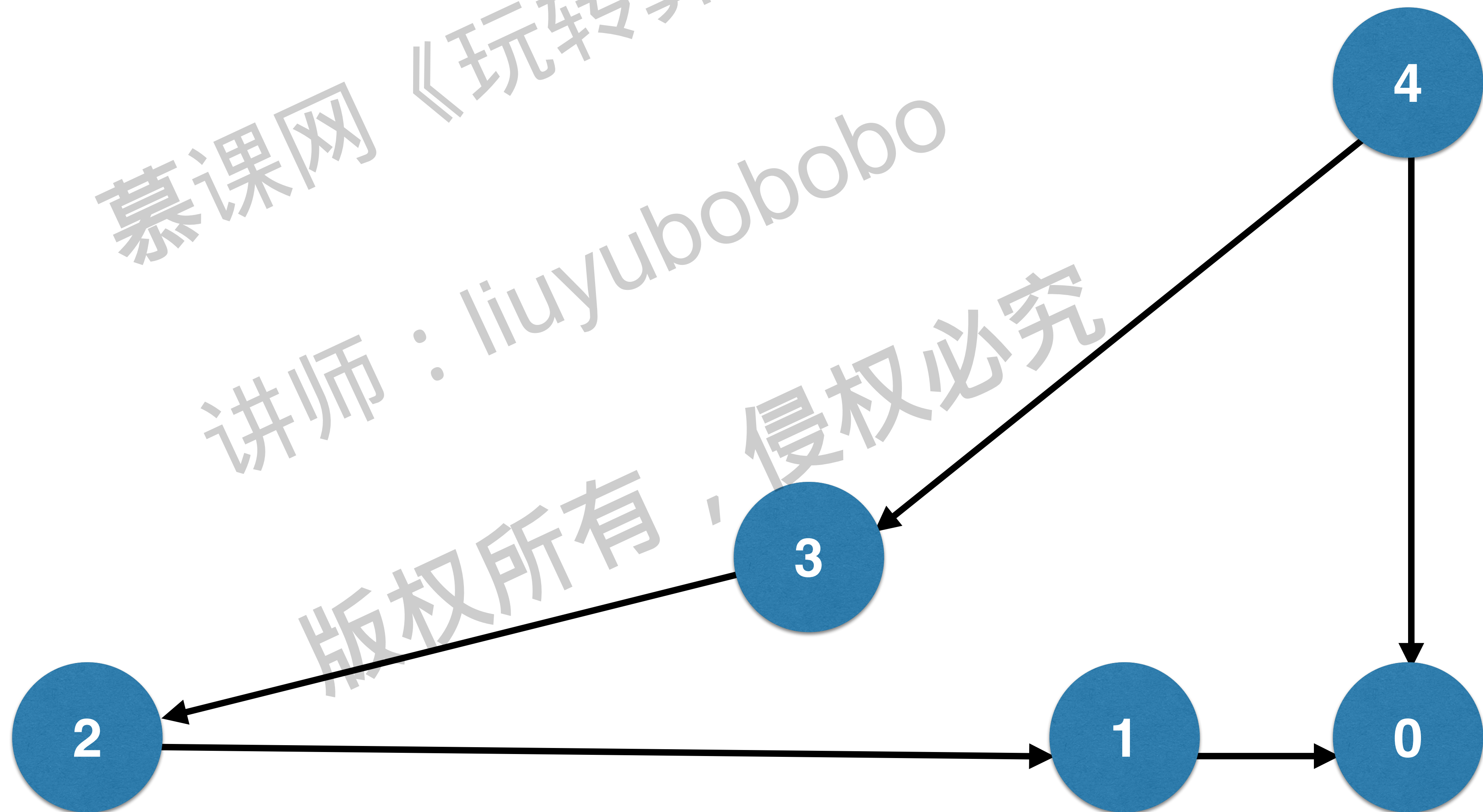
从 $n$ 到 $0$ ，每个数字表示一个节点；

如果两个数字 $x$ 到 $y$ 相差一个完全平方数，则连接一条边。

我们得到了一个无权图。

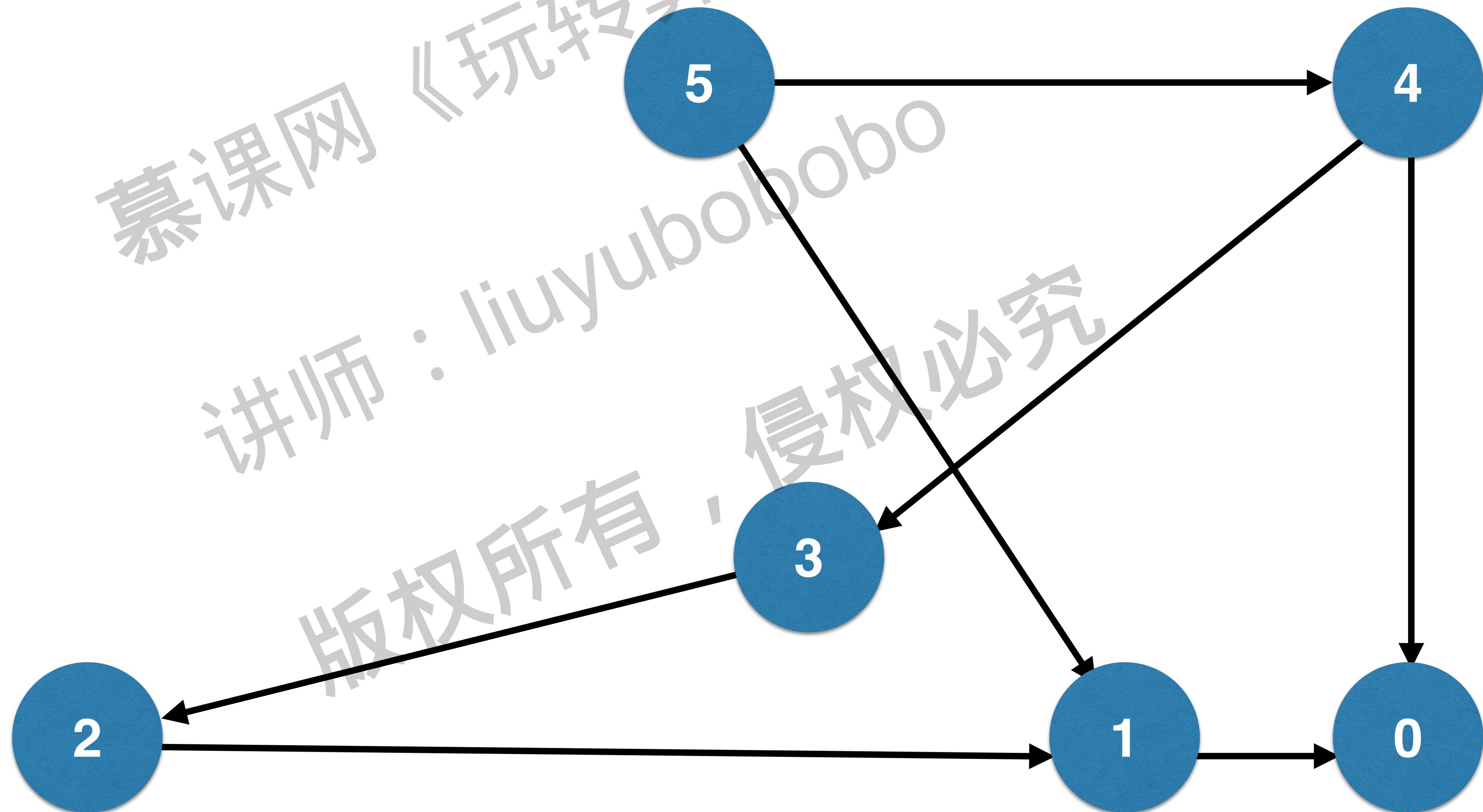
原问题转化成，求这个无权图中从 $n$ 到 $0$ 的最短路径。

# 279. Perfect Squares

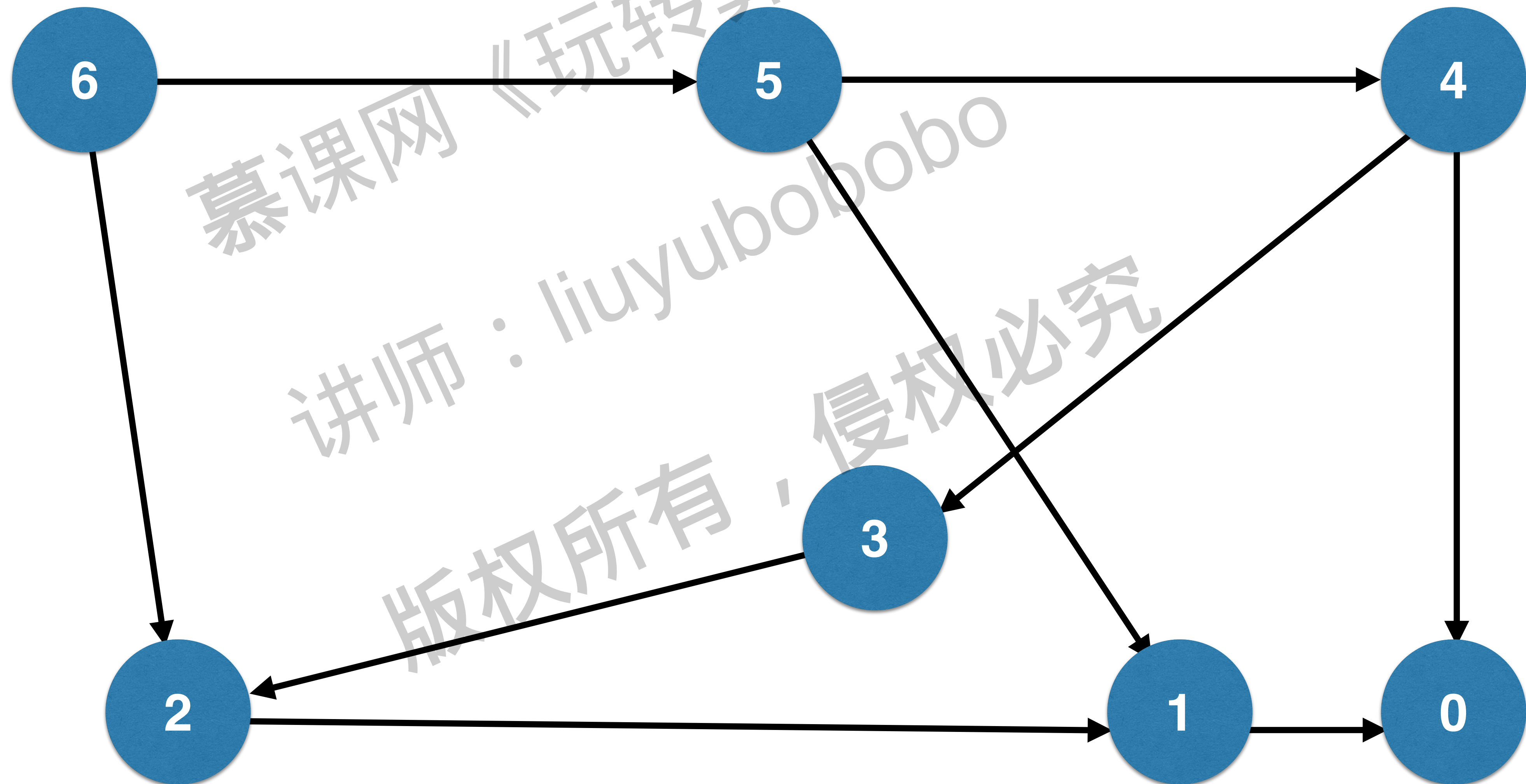




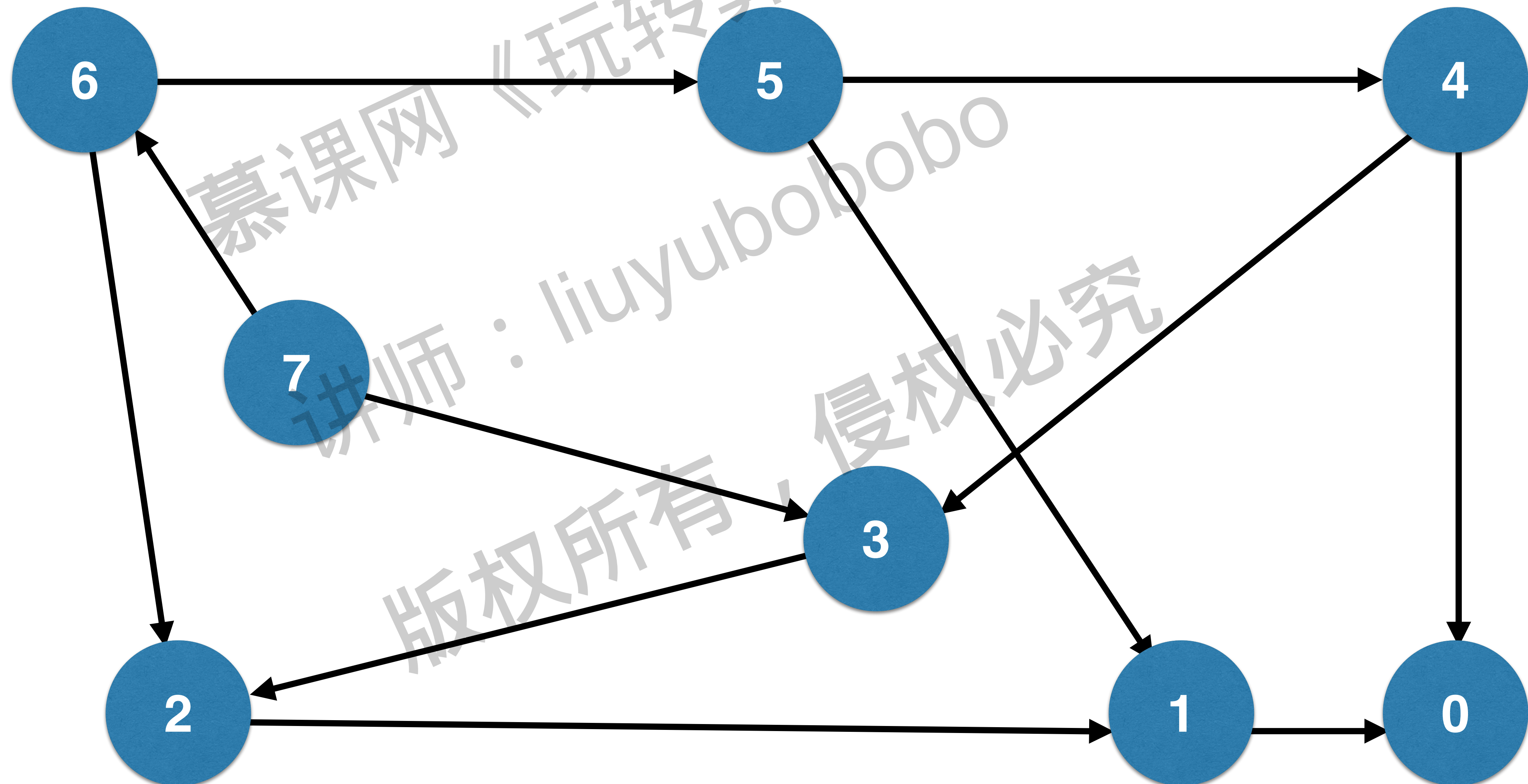
# 279. Perfect Squares



# 279. Perfect Squares

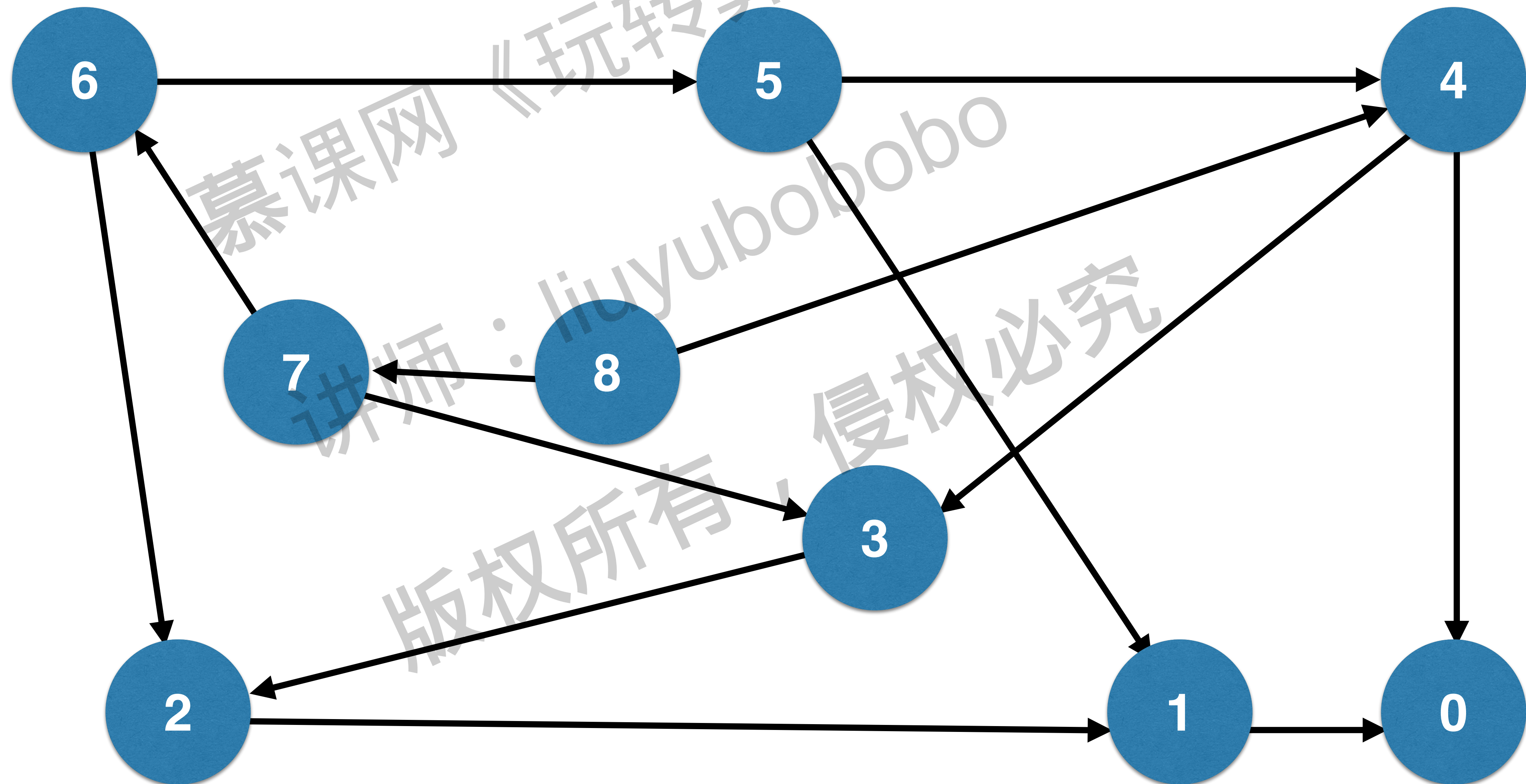


# 279. Perfect Squares

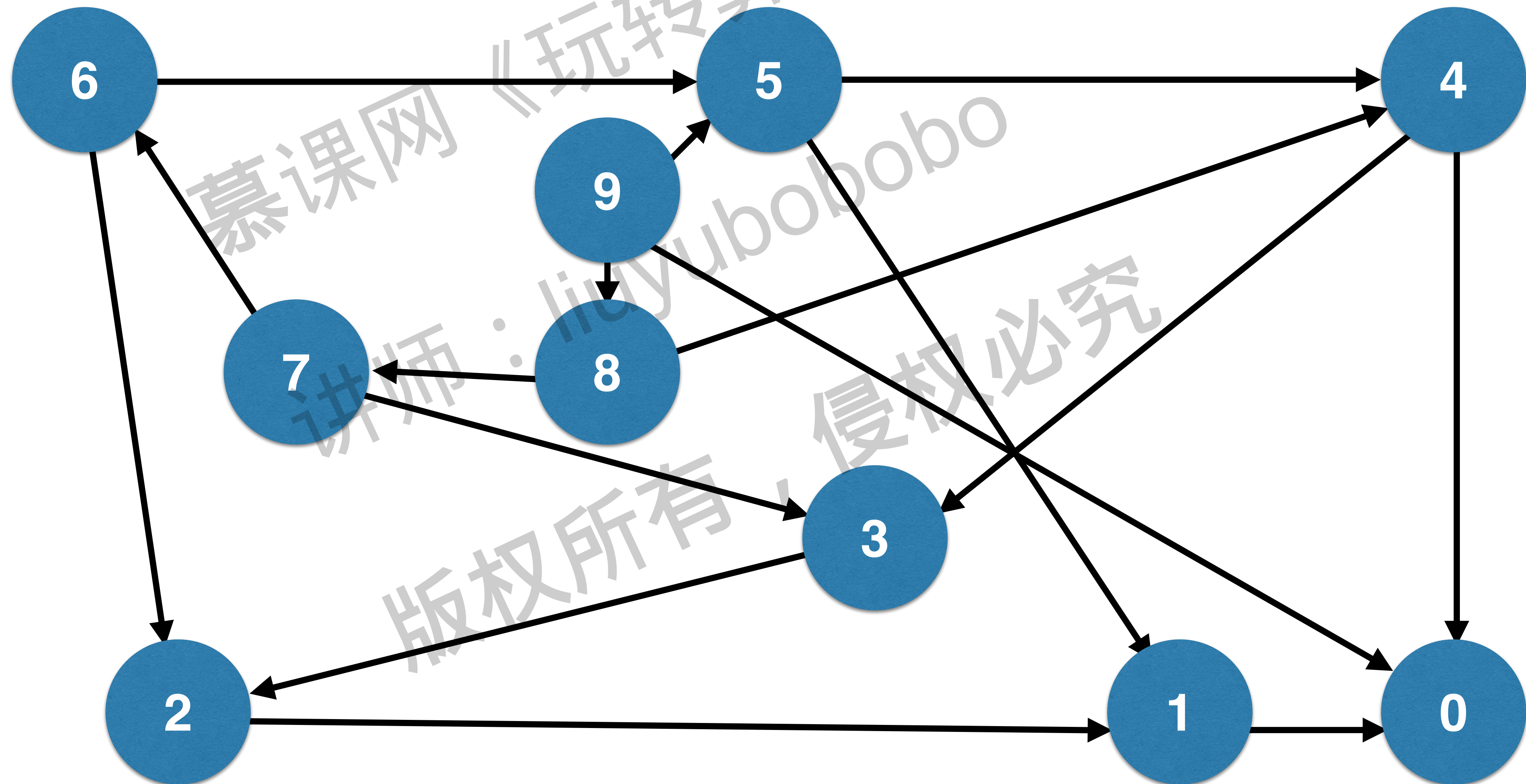




# 279. Perfect Squares



# 279. Perfect Squares



慕课网《玩转算法面试》

# 实践：解决279

讲师：liuyubobobo

版权所有，侵权必究



# 127. Word Ladder



snapchat



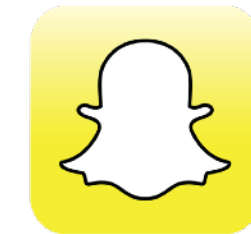
给出两个单词 (beginWord 和 endWord), 以及一个单词列表, 寻找一条从 beginWord 到 endWord 的最短变换路径。每次变换只能修改单词的一个字母。

# 127. Word Ladder

facebook

amazon

LinkedIn



snapchat

yelp

- beginWord = "hit", endWord = "cog"
- 单词列表是 ["hot", "dot", "dog", "lot", "log", "cog"]
- 我们可以找到的最短的变换路径为:  
"hit" -> "hot" -> "dot" -> "dog" -> "cog",
- 结果为5

# 126. Word Ladder II



给出两个单词 (beginWord 和 endWord), 以及一个单词列表, 寻找**所有**从 beginWord 到 endWord 的最短变换路径。每次变换只能修改单词的一个字母。

# 126. Word Ladder II

- beginWord = "hit", endWord = "cog"
- 单词列表是 ["hot", "dot", "dog", "lot", "log", "cog"]
- [  
    ["hit", "hot", "dot", "dog", "cog"],  
    ["hit", "hot", "lot", "log", "cog"]  
]

慕课网《玩转算法面试》

# 优先队列

讲师：liuyubobobo

版权所有，侵权必究

慕课网《玩转算法面试》

# 优先队列也是队列

讲师：liuyubobobo

版权所有，侵权必究



# 优先队列的底层实现：堆

对于堆的底层实现，白板编程

# 使用优先队列解决算法问题

学习使用语言中的优先队列容器

C++语言: `priority_queue`

# 实践：使用C++中的优先队列

慕课网《玩转算法面试》  
讲师：liuyubobobo

版权所有，侵权必究

# 使用优先队列解决算法问题

慕课网《玩转算法面试》

讲师：liuyubobobo

版权所有，侵权必究

# 347. Top K Frequent Elements



给定一个非空数组，返回前k个出现频率最高的元素。

- 如给定  $[1, 1, 1, 2, 2, 3]$ ,  $k = 2$
- 返回  $[1, 2]$
- 注意k的合法性问题

# 347. Top K Frequent Elements



给定一个非空数组，返回前k个出现频率最高的元素。

最简单的思路：扫描一遍统计频率；排序找到前k个出现频率最高的元素。 $O(n \log n)$



# 347. Top K Frequent Elements



给定一个非空数组，返回前k个出现频率最高的元素。

维护一个含有k个元素的优先队列。如果遍历到的元素比队列中的最小频率元素的频率高，则取出队列中最小频率的元素，将新元素入队。最终，队列中剩下的，就是前k个出现频率最高的元素。

# 347. Top K Frequent Elements



给定一个非空数组，返回前k个出现频率最高的元素。

思路2：维护优先队列，时间复杂度： $O(n \log k)$

慕课网《玩转算法面试》

# 实践：解决347

讲师：liuyubobobo

版权所有，侵权必究

# 347. Top K Frequent Elements



给定一个非空数组，返回前k个出现频率最高的元素。

思路3：维护优先队列，时间复杂度：  $O(n \log(n-k))$

## 23. Merge k Sorted Lists

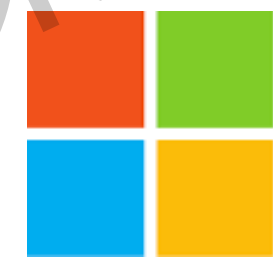
Google

facebook

amazon

LinkedIn

UBER



Microsoft

airbnb

twitter

有k个有序数组，将他们归并为一个有序数组



慕课网《玩转算法面试》

# 玩儿转算法面试

讲师：liuyubobobo

版权所有，侵权必究

liuyubobobo