递归与栈：解决表达式求值

胡船长
初航我带你，远航靠自己

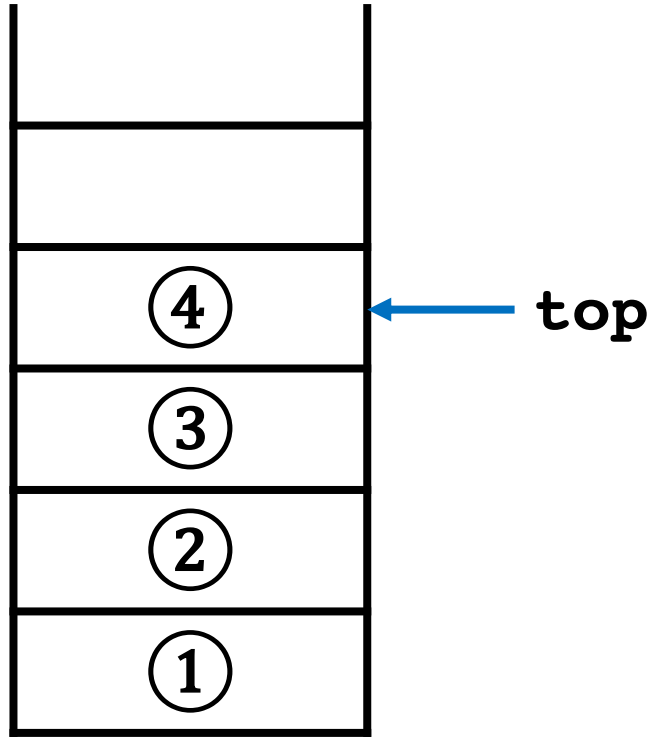
栈基础知识

大约用时：（ 40 mins ）

下一部分：栈的典型应用场景

栈

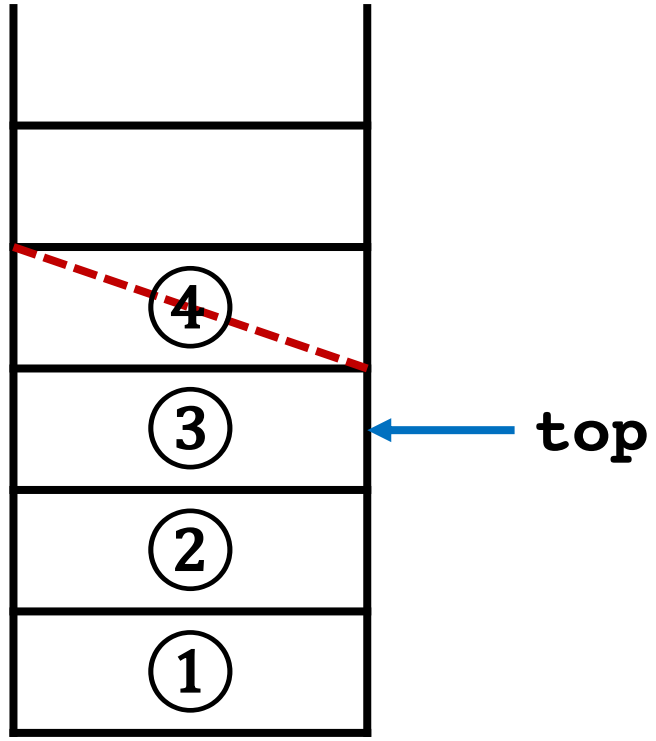
1、size = 5
2、top = 3
3、data_type = xxx



栈-出栈

1、size = 5
2、top = 3
3、data_type = xxx

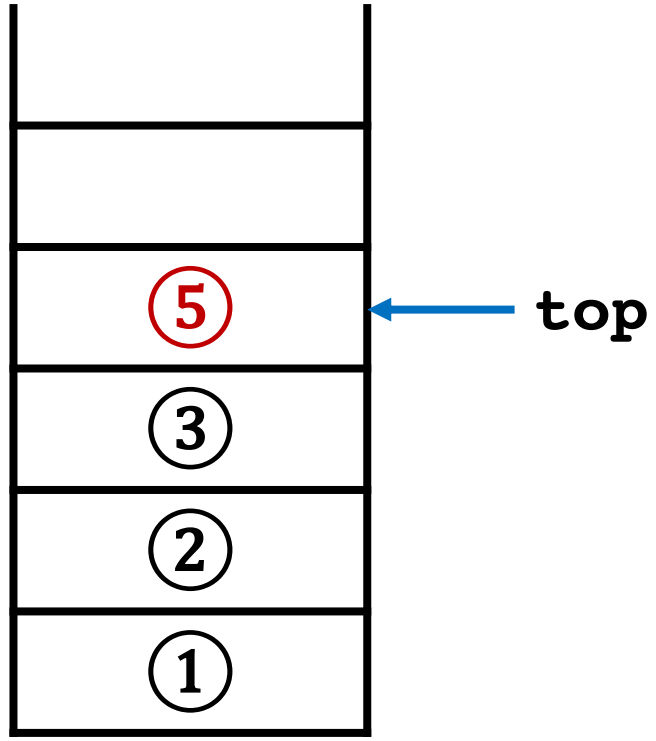
1、size = 5
2、top = 2
3、data_type = xxx



栈-入栈

1、size = 5
2、top = 2
3、data_type = xxx

1、size = 5
2、top = 3
3、data_type = xxx



栈适合解决什么问题？

LEETCODE 20 ： 括号匹配

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

LEETCODE 20 ： 括号匹配

思考：

问题简化成只有一种括号，怎么做？

想到用栈的同学，在简化问题中，能不能不用栈？

LEETCODE 20 ： 括号匹配

1、(()()())()
2、(()))()
3、()()

LEETCODE 20 ： 括号匹配

1、(()()())()　　TRUE

2、(()))()　　FALSE

3、(()()　　FALSE

LEETCODE 20 ： 括号匹配

结论：

1、在任意一个位置上，左括号数量 >= 右括号数量

2、在最后一个位置上，左括号数量 == 右括号数量

3、程序中只需要记录左括号数量和右括号数量即可

LEETCODE 20 ： 括号匹配


```
1  bool isValid(char *s) {
2      int32_t lnum = 0, rnum = 0;
3      int32_t len = strlen(s);
4      for (int32_t i = 0; i < len; i++) {
5          switch (s[i]) {
6              case '(' : ++lnum; break;
7              case ')' : ++rnum; break;
8              default : return false;
9          }
10         if (lnum >= rnum) continue;
11         return false;
12     }
13     return lnum == rnum;
14 }
```

存在问题：

程序能不能更优化？

思考：

rnum 变量一定是需要的么？

LEETCODE 20 ： 括号匹配


```
1  bool isValid(char *s) {
2      int32_t lnum = 0;
3      int32_t len = strlen(s);
4      for (int32_t i = 0; i < len; i++) {
5          switch (s[i]) {
6              case '(' : ++lnum; break;
7              case ')' : --lnum; break;
8              default : return false;
9          }
10         if (lnum >= 0) continue;
11         return false;
12     }
13     return lnum == 0;
14 }
```

LEETCODE 20 ： 括号匹配

思考：

1、我们获得了怎样新的思维方式?
2、+1 可以等价为『进』，-1可以等价为『出』
3、一对( )可以等价为一个完整的事件
4、(( ))可以看做事件与事件之间的完全包含关系
5、由括号的等价变换，得到了一个新的数据结构

LEETCODE 20 ： 括号匹配

可以处理具有完全包含关系的问题

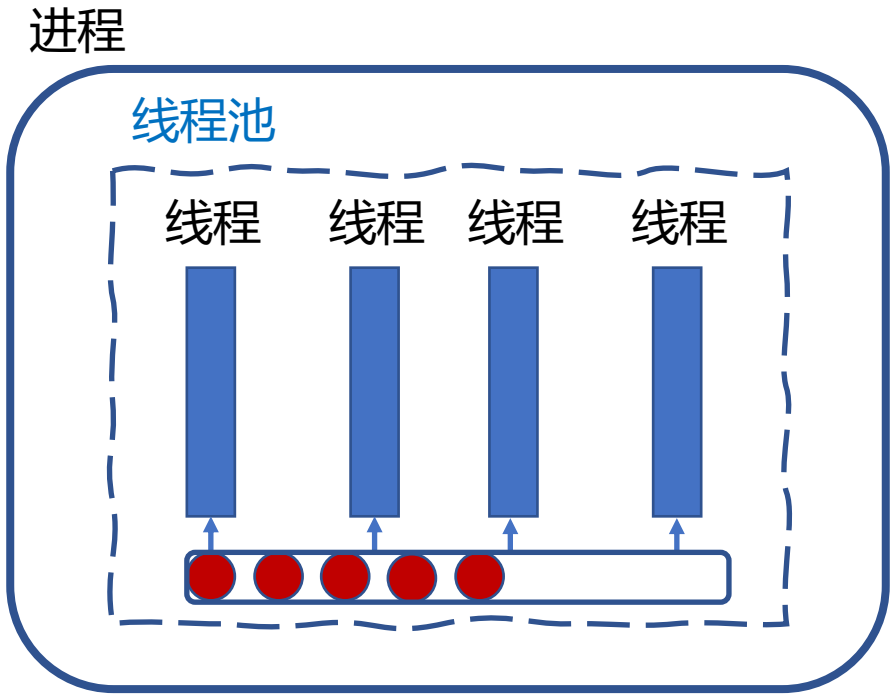
经典的栈实现方法

栈的典型应用场景

大约用时：（ 20 mins ）

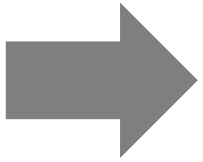
下一部分：经典面试题-栈的基本操作

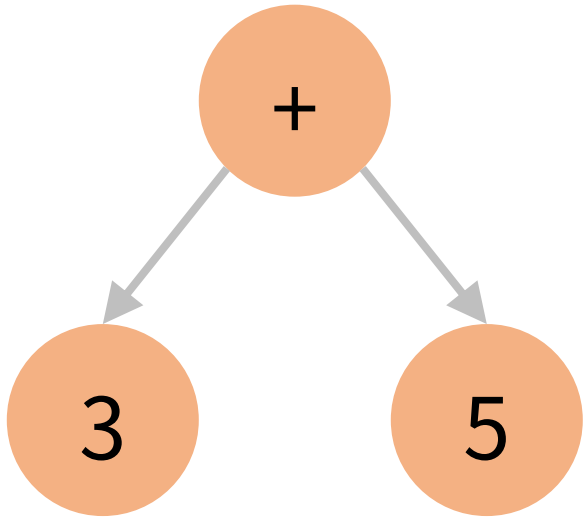
场景一：操作系统中的线程栈



场景二：表达式求值

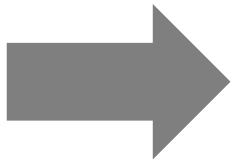
3 + 5

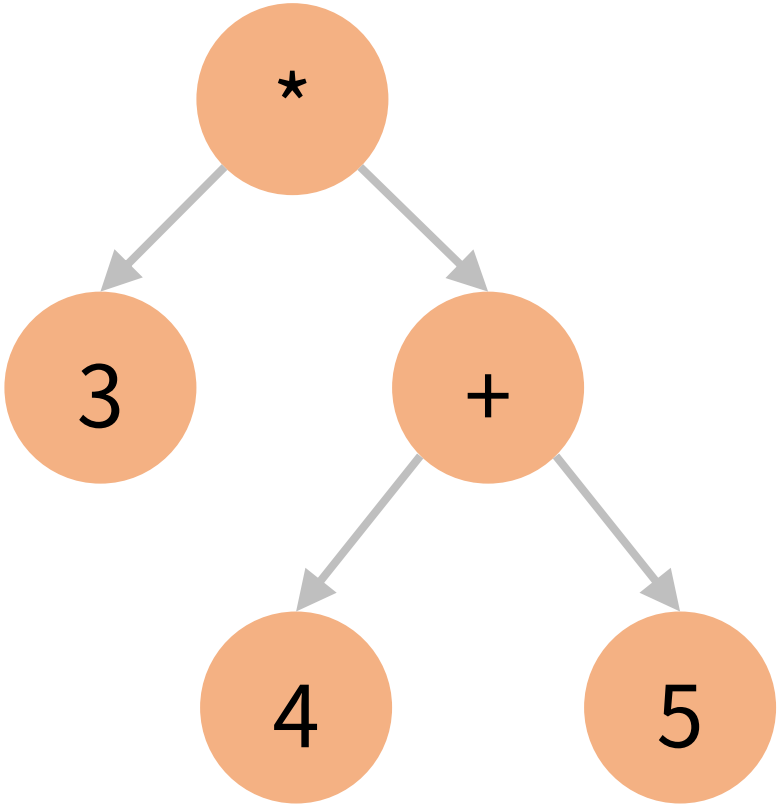




场景二：表达式求值

3 * (4 + 5)





场景二：表达式求值（板书）

经典面试题-栈的基本操作

大约用时：（ 40 mins ）

下一部分：经典面试题-栈结构扩展应用

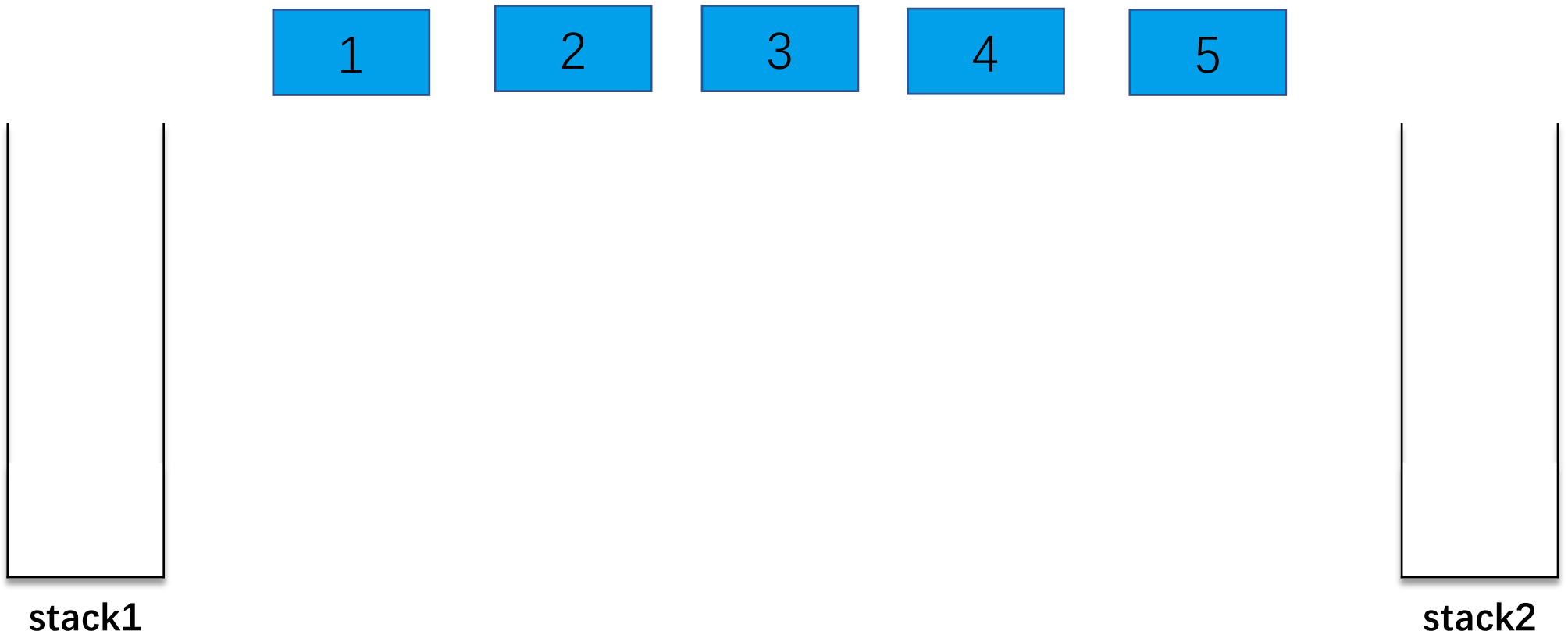
面试题03.04.化栈为队

门徒计划，带你开启算法精进之路

LeetCode-03.04 化栈为队

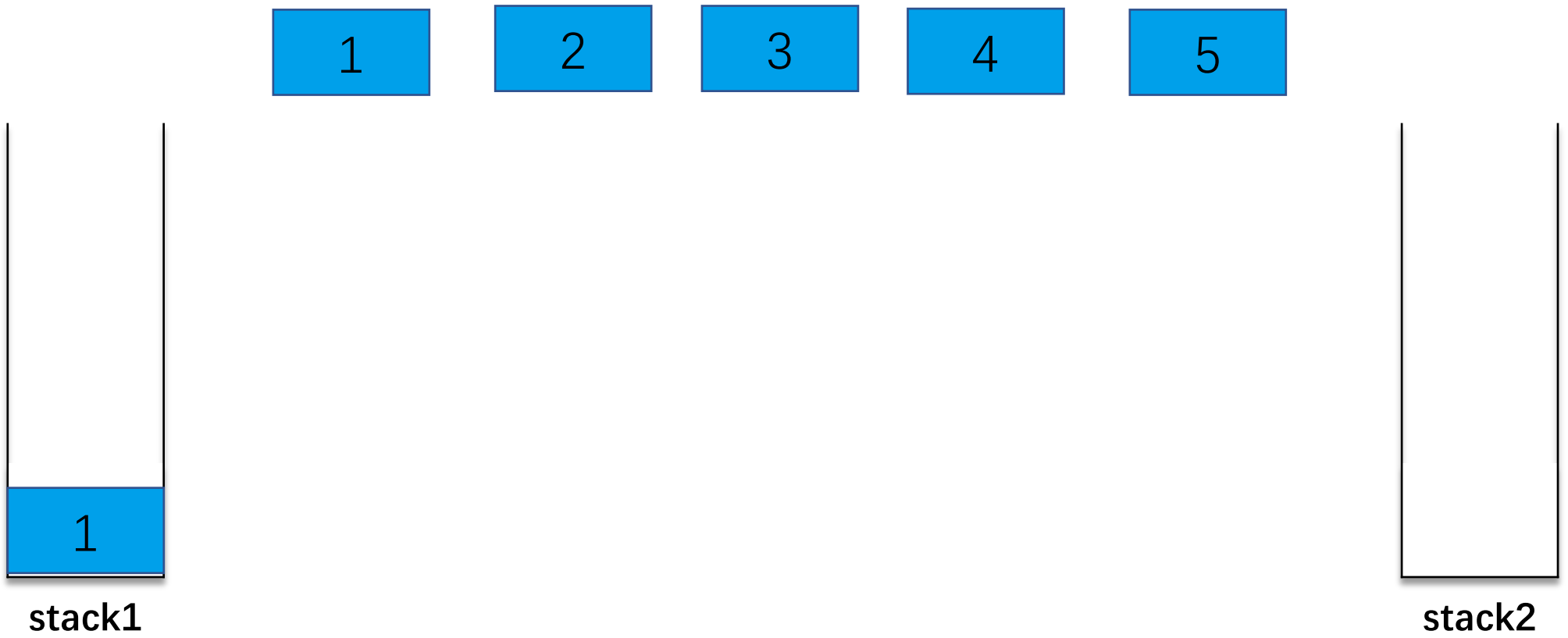
首先举例，我们有上面这样的一个队列，然后对它进行push/pop/peek/empty四个操作

LeetCode-03.04 化栈为队



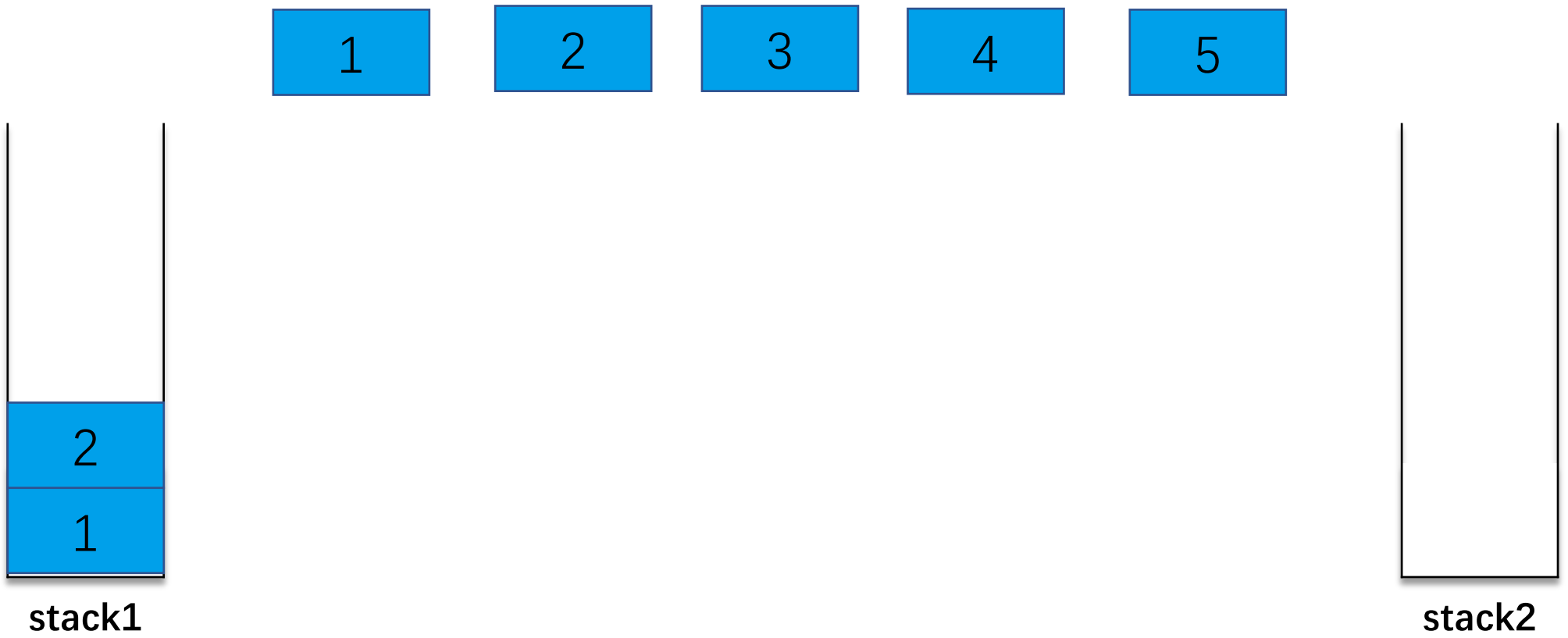
首先创建两个栈：栈1是stack1 ，栈2 是stack2;首先一开始，两个栈都是空的。栈1是stack1 ，栈2 是stack2

LeetCode-03.04 化栈为队



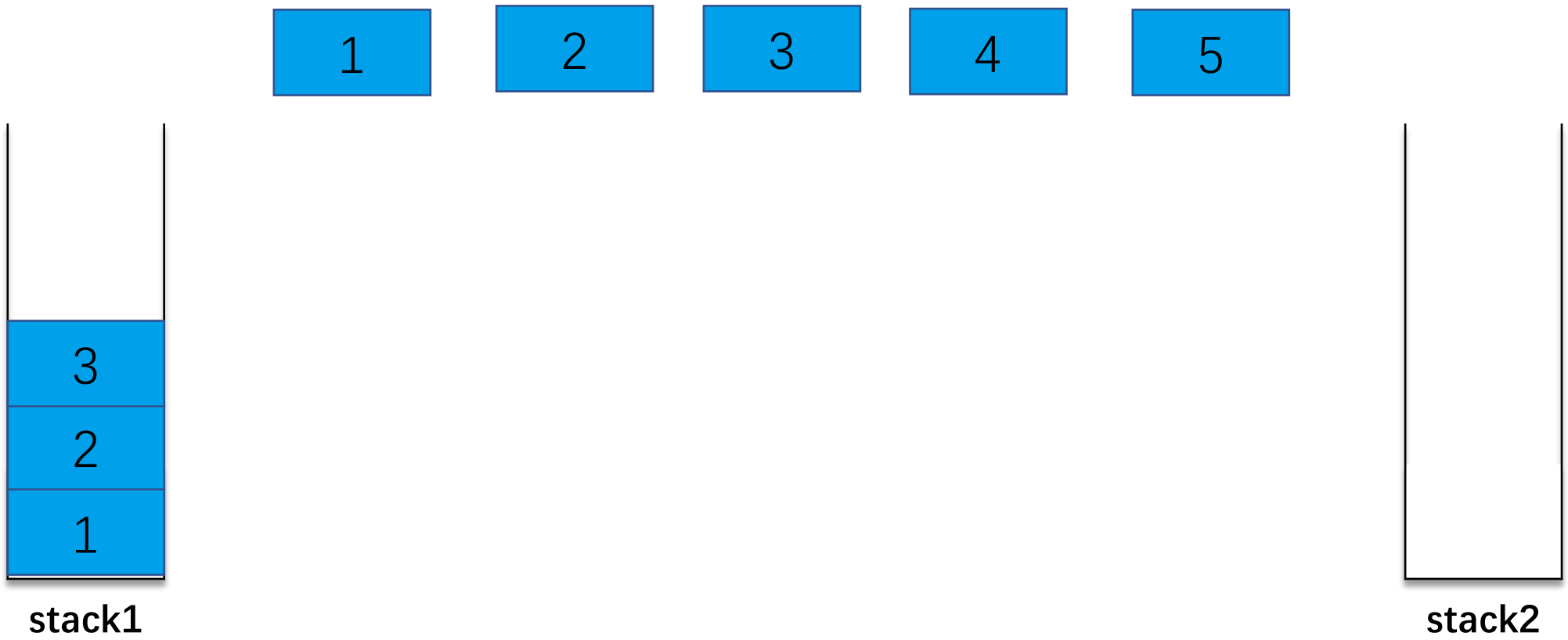
接着，我们执行第一个操作： push to top；然后我们的push操作就是，往 stack1 里面push一个操作

LeetCode-03.04 化栈为队



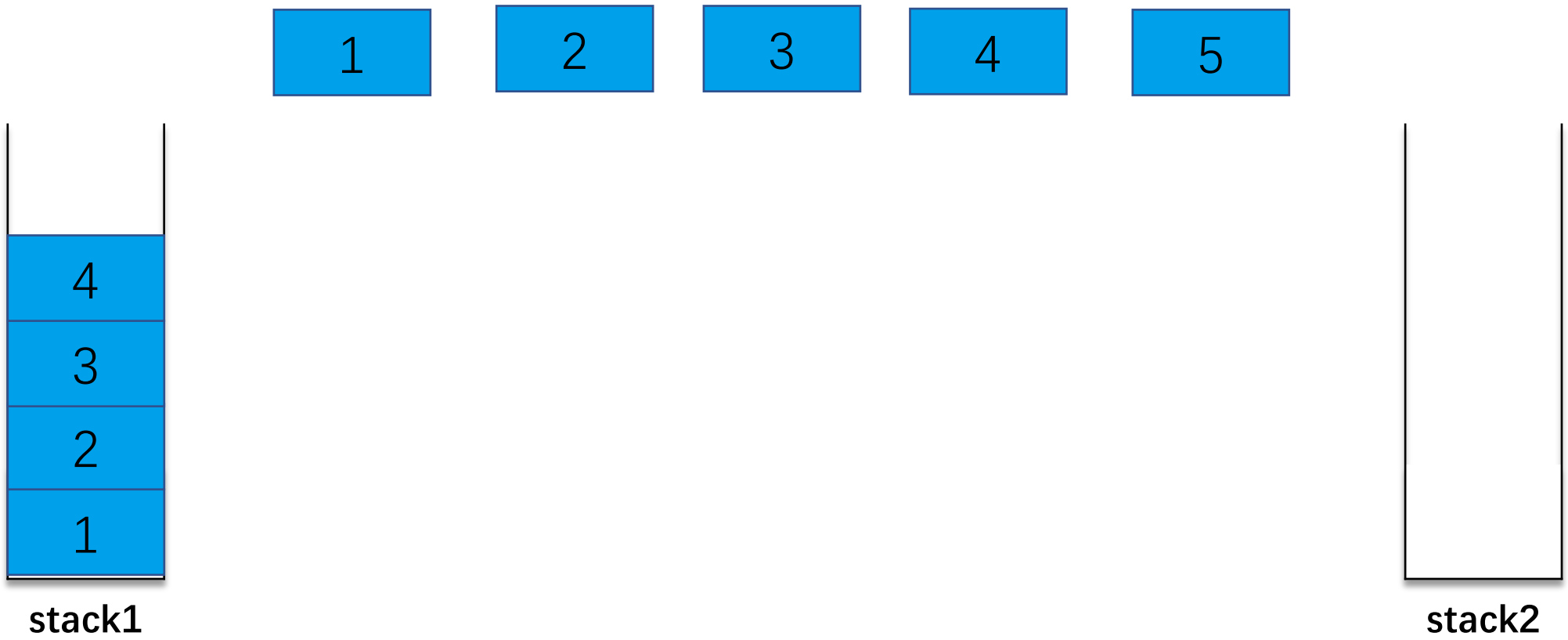
接着，依次往 stack1 里面push

LeetCode-03.04 化栈为队



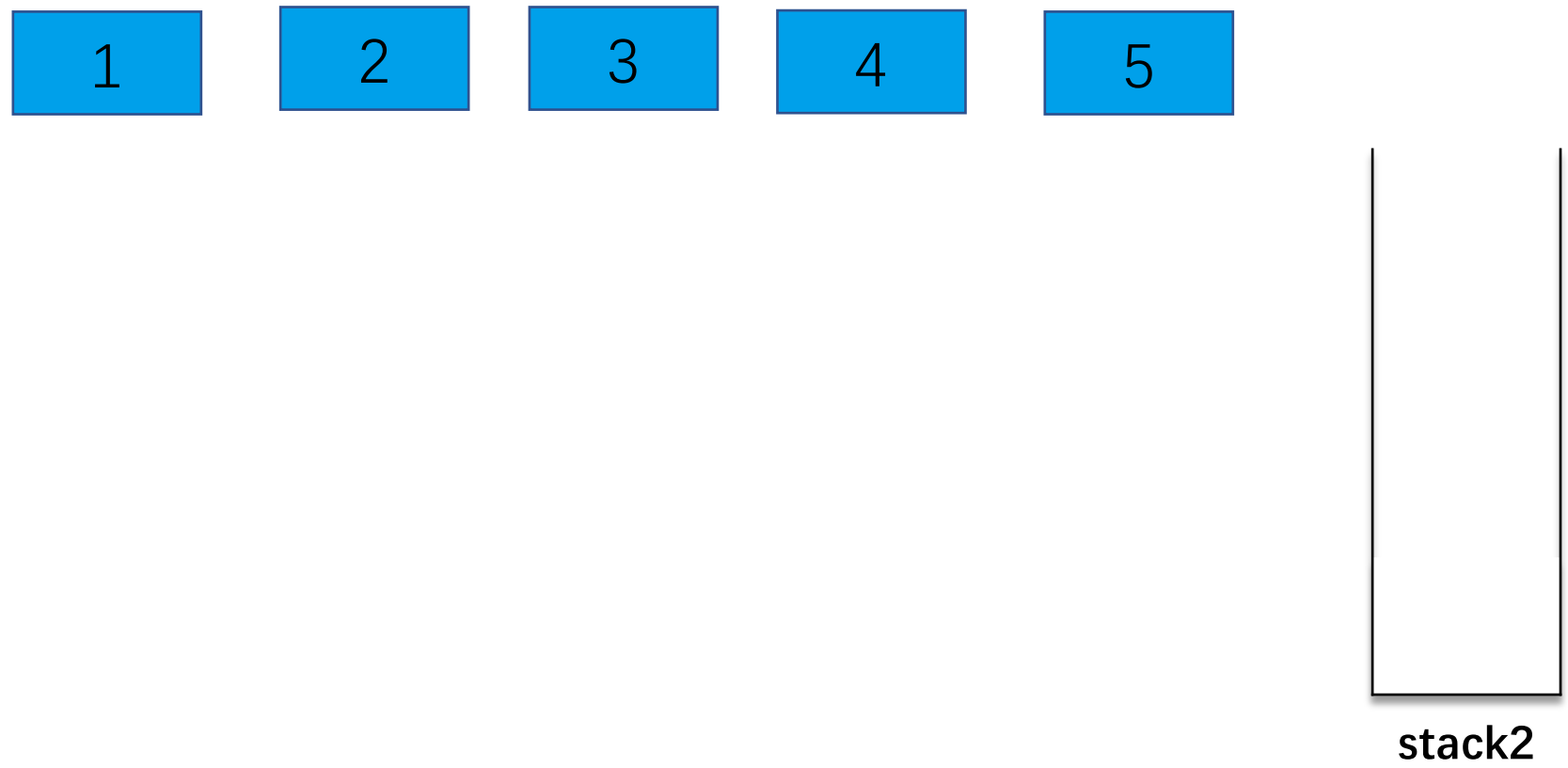
接着，依次往 stack1 里面push

LeetCode-03.04 化栈为队



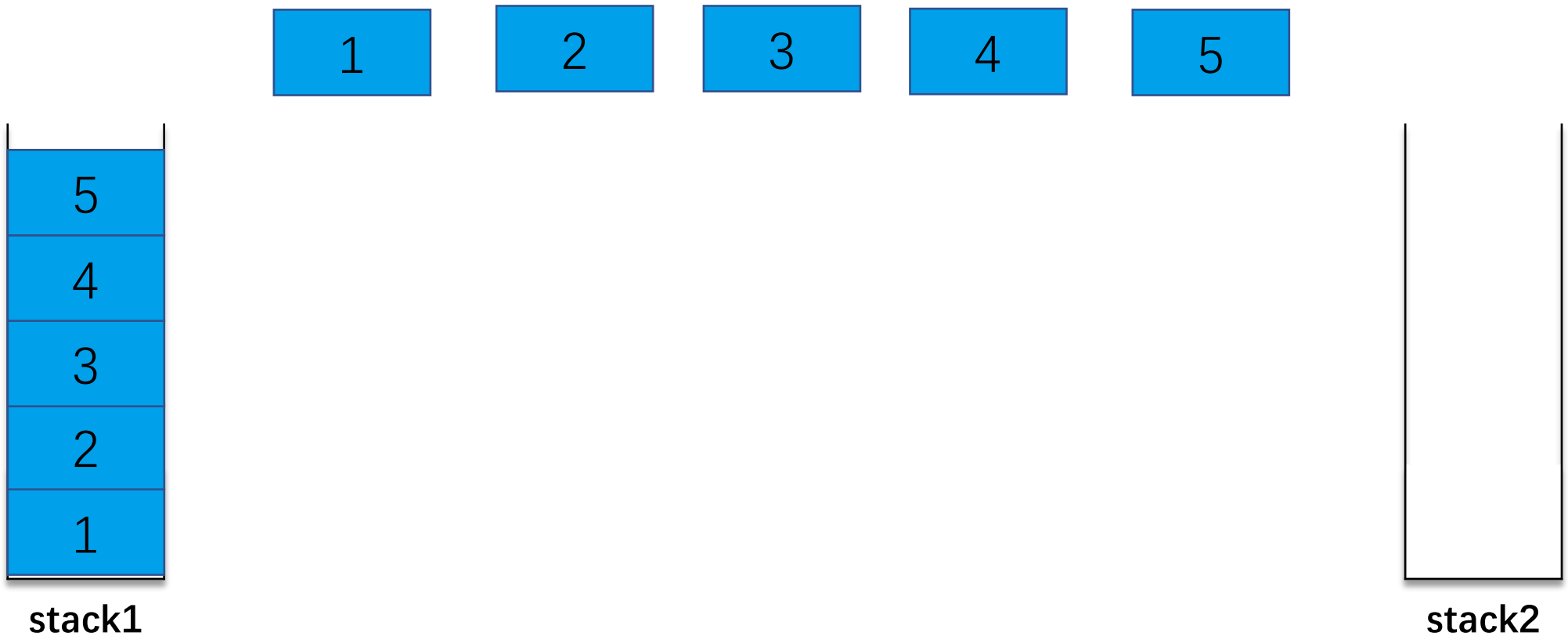
接着，依次往 stack1 里面push

LeetCode-03.04 化栈为队



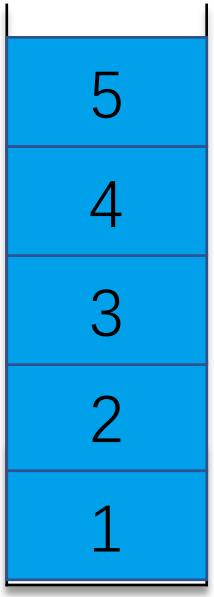
接着，依次往 stack1 里面push

LeetCode-03.04 化栈为队

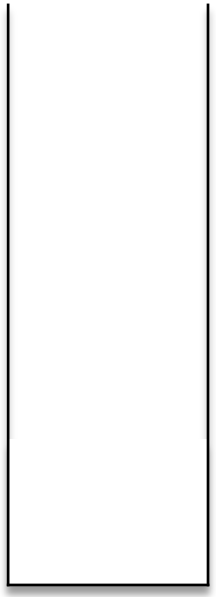


此时，队列里面的元素，依次加入到了 stack1 里

LeetCode-03.04 化栈为队



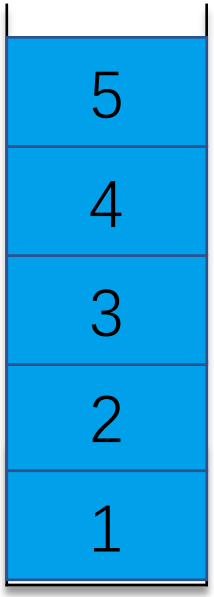
stack1



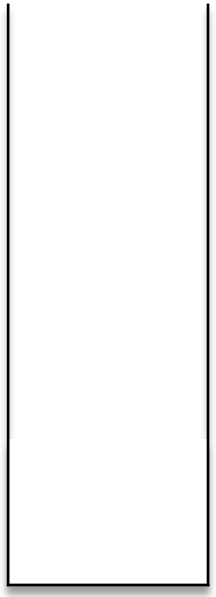
stack2

此时，执行第二个操作: pop from top

LeetCode-03.04 化栈为队



stack1



stack2

首先 要判断： stack2 长度 是否等于0

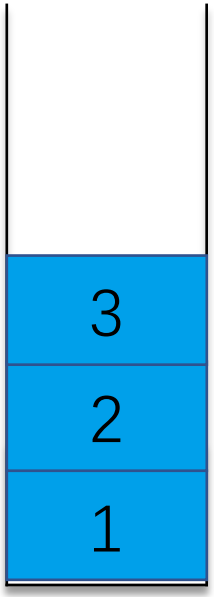
LeetCode-03.04 化栈为队

如果stack2的长度等于0，就在stack1的长度不为0 的时候，对stack1进行pop删除操作

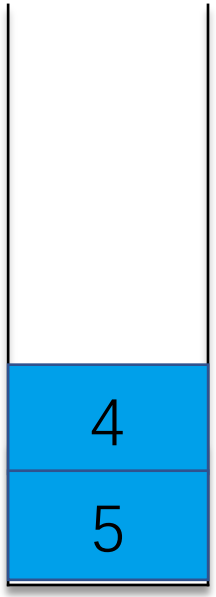
LeetCode-03.04 化栈为队

因为栈的特性是，先进后出，后进先出；第一个删除的便是后进去的元素5

LeetCode-03.04 化栈为队



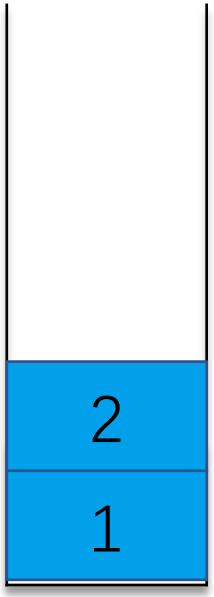
stack1



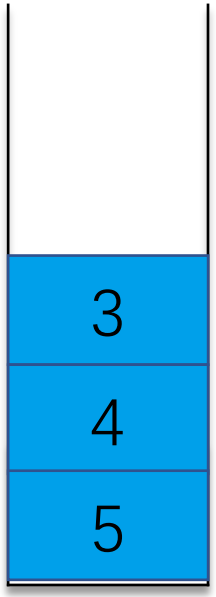
stack2

依次将stack1里面删除的元素，添加到stack2的里面

LeetCode-03.04 化栈为队



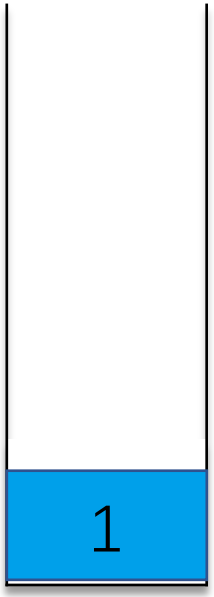
stack1



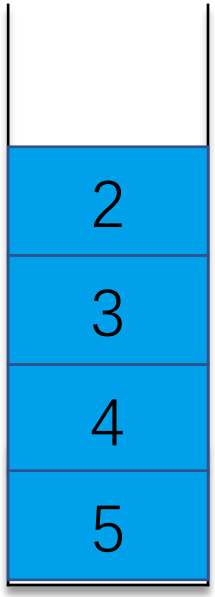
stack2

接着，重复上面的操作

LeetCode-03.04 化栈为队



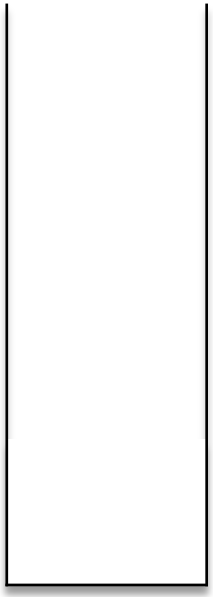
stack1



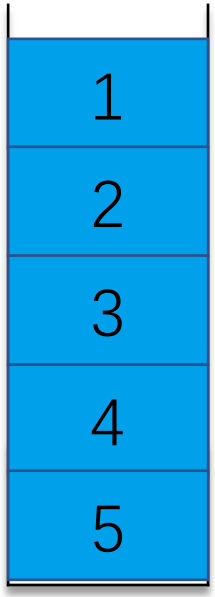
stack2

接着，重复上面的操作

LeetCode-03.04 化栈为队



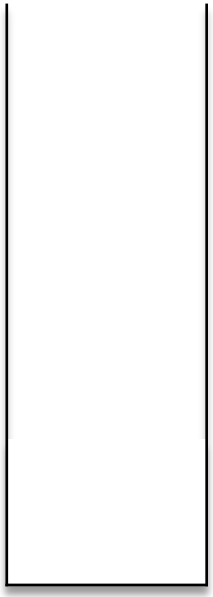
stack1



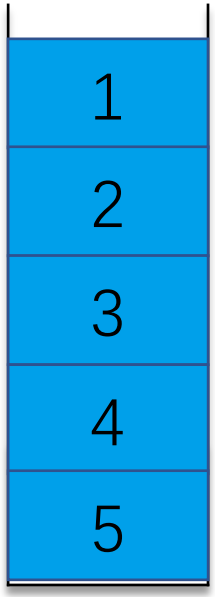
stack2

接着，重复上面的操作

LeetCode-03.04 化栈为队



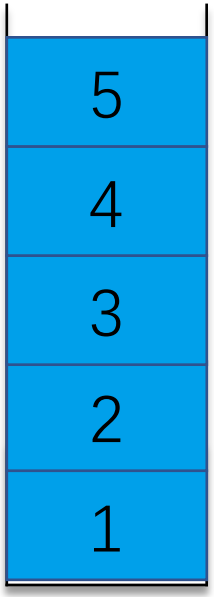
stack1



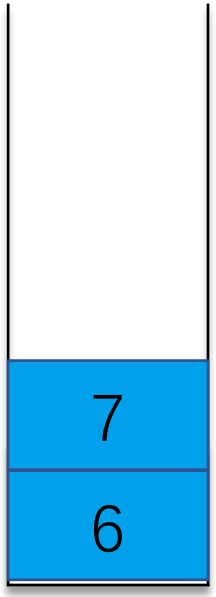
stack2

直到，stack1的长度为空的时候停止

LeetCode-03.04 化栈为队



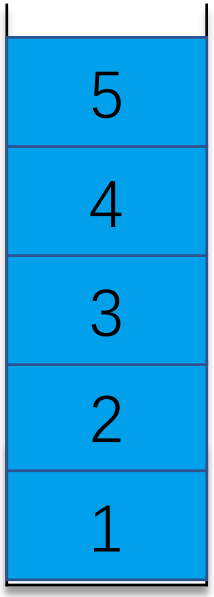
stack1



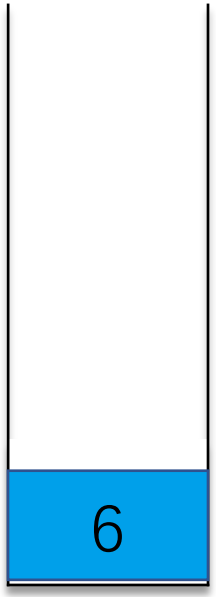
stack2

如果stack2的长度不等于0

LeetCode-03.04 化栈为队



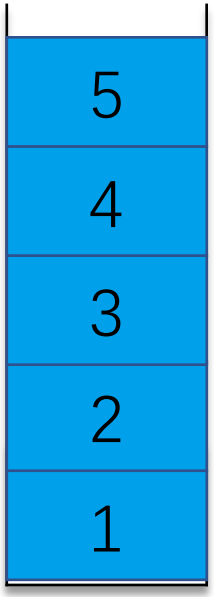
stack1



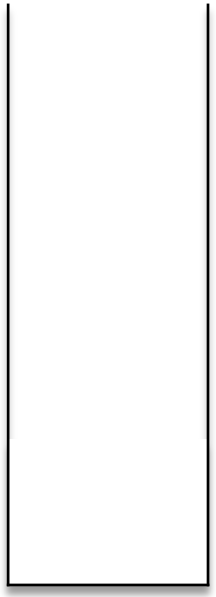
stack2

我们就先将stack2里面的进行删除

LeetCode-03.04 化栈为队



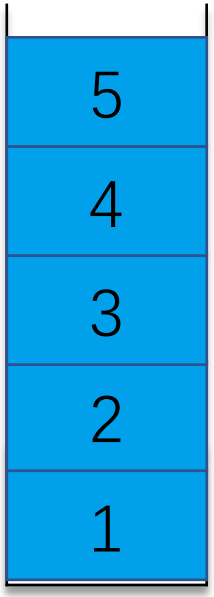
stack1



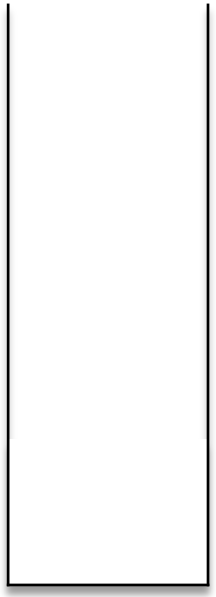
stack2

我们就先将stack2里面的进行删除

LeetCode-03.04 化栈为队



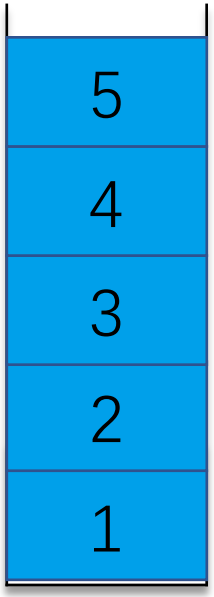
stack1



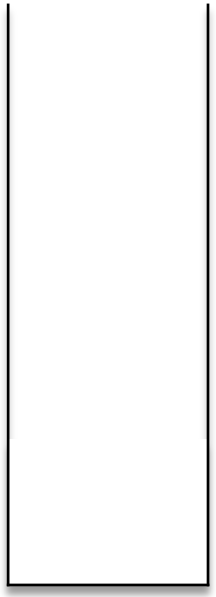
stack2

一步一步删除后，直到 stack2为空

LeetCode-03.04 化栈为队



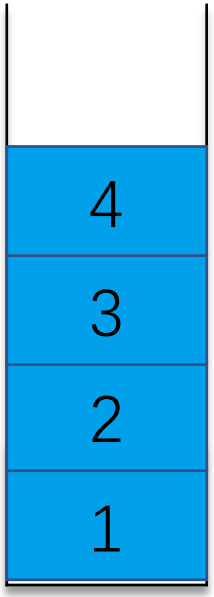
stack1



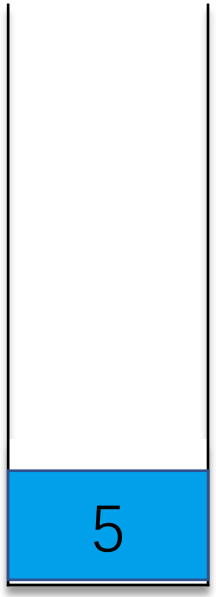
stack2

然后再重复stack2 的长度等于0时的操作

LeetCode-03.04 化栈为队



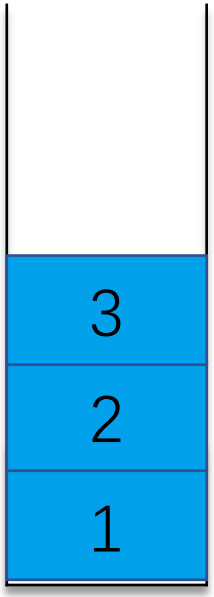
stack1



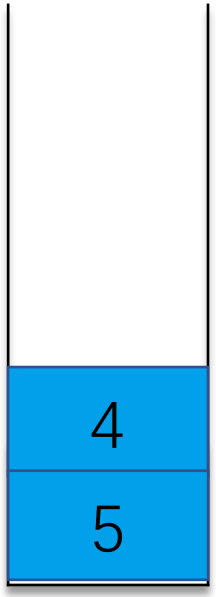
stack2

就是再将stack1里面的元素删除，添加到是stack2里面，直到，stack1的长度为空

LeetCode-03.04 化栈为队



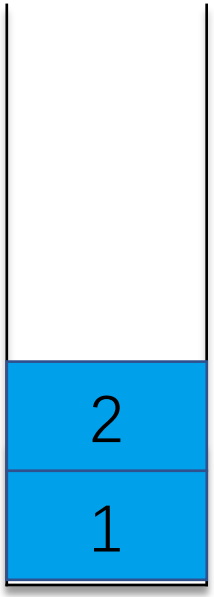
stack1



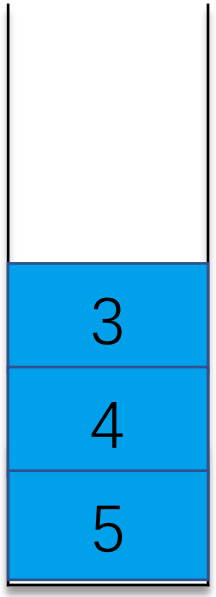
stack2

不断重复此操作，直到，stack1的长度为空

LeetCode-03.04 化栈为队



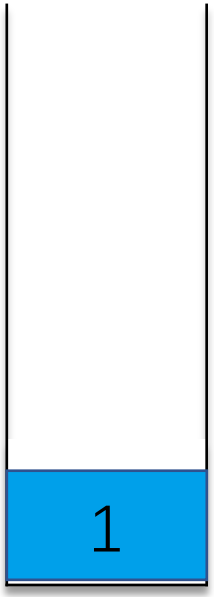
stack1



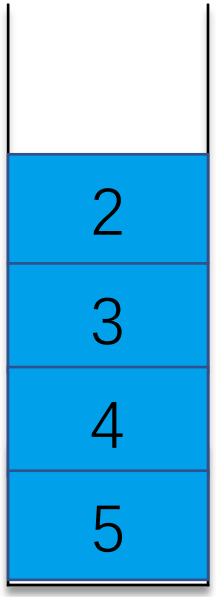
stack2

不断重复此操作，直到，stack1的长度为空

LeetCode-03.04 化栈为队



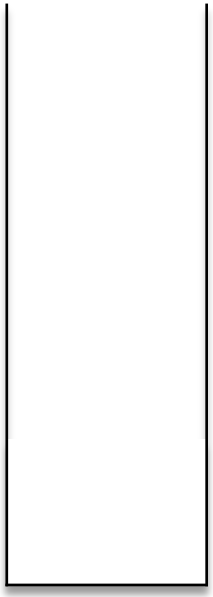
stack1



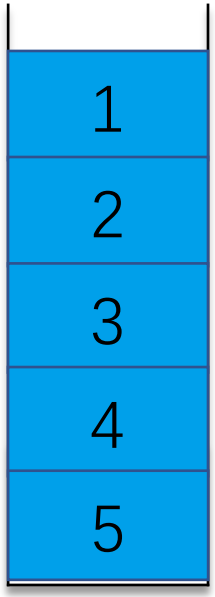
stack2

不断重复此操作，直到，stack1的长度为空

LeetCode-03.04 化栈为队



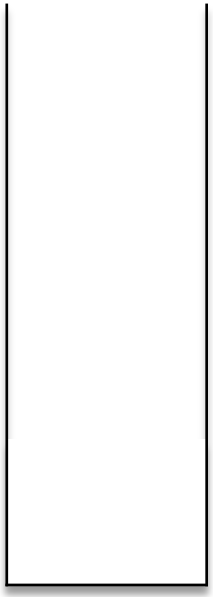
stack1



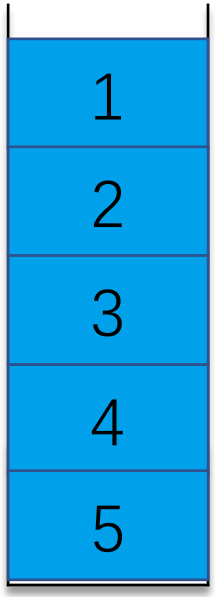
stack2

不断重复此操作，直到，stack1的长度为空

LeetCode-03.04 化栈为队



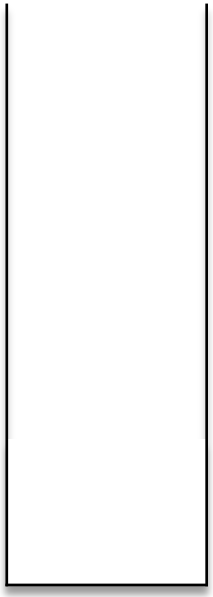
stack1



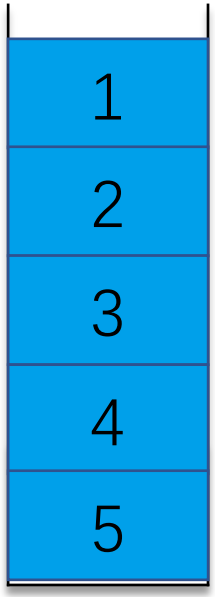
stack2

最后，stack1的长度为空

LeetCode-03.04 化栈为队



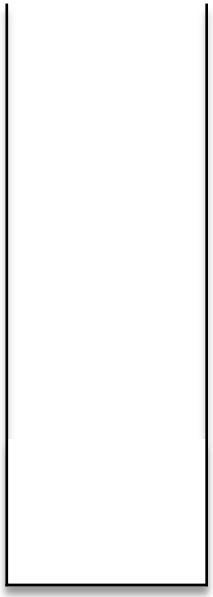
stack1



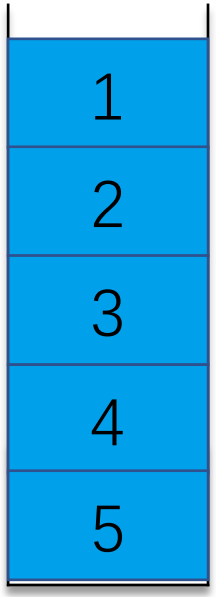
stack2

接着，执行第三个操作: peek from top

LeetCode-03.04 化栈为队



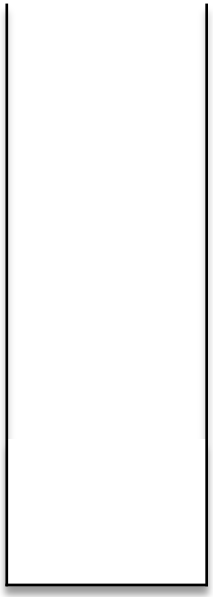
stack1



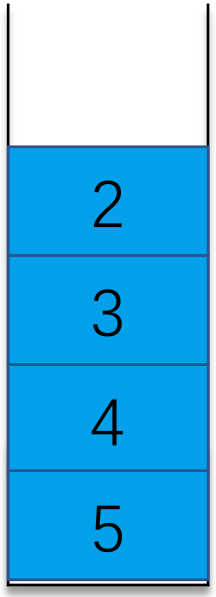
stack2

还是判断stack2的长度是否为0，不为0，就自我删除

LeetCode-03.04 化栈为队



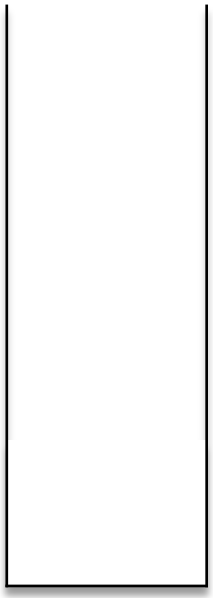
stack1



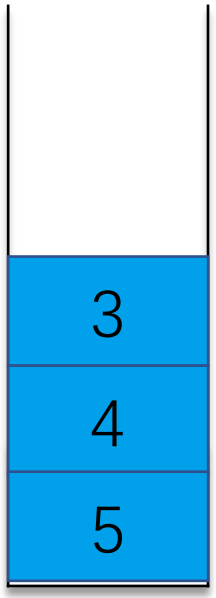
stack2

不断地尾部删除

LeetCode-03.04 化栈为队



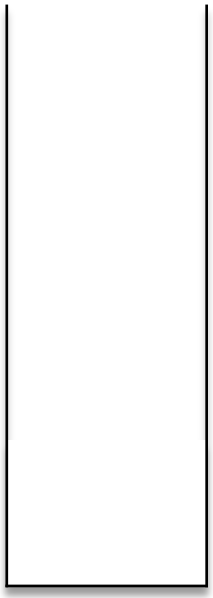
stack1



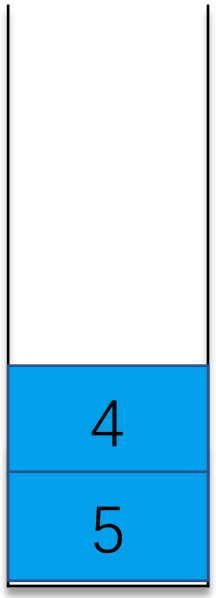
stack2

不断地尾部删除

LeetCode-03.04 化栈为队



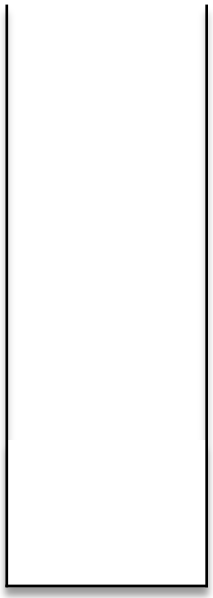
stack1



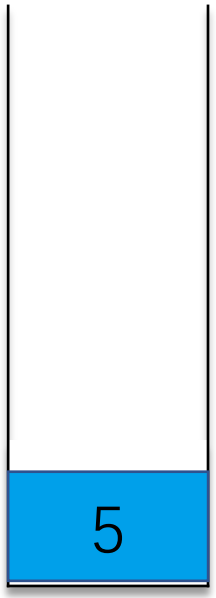
stack2

不断地尾部删除

LeetCode-03.04 化栈为队



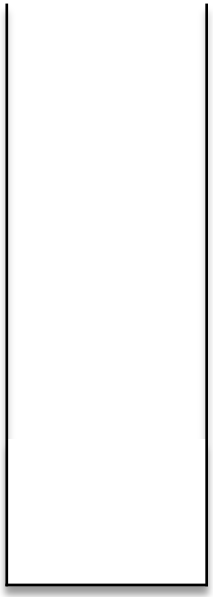
stack1



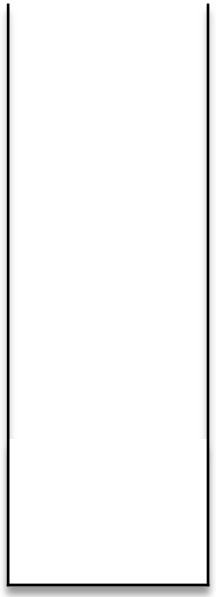
stack2

不断地尾部删除

LeetCode-03.04 化栈为队



stack1



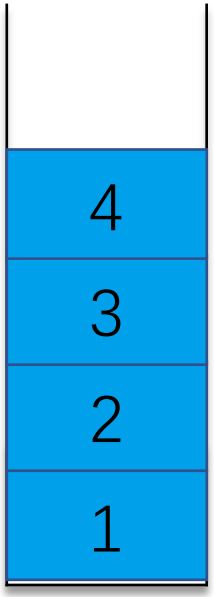
stack2

直到， stack2.length = 0;

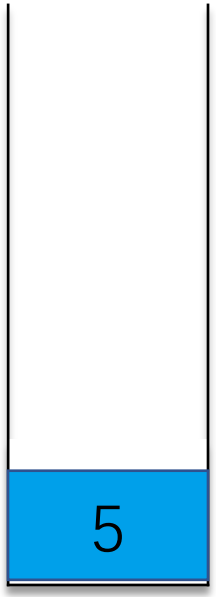
LeetCode-03.04 化栈为队

第二种情况，如果 stack2的长度为0，还是将stack1从尾部删除，添加到 stack2

LeetCode-03.04 化栈为队



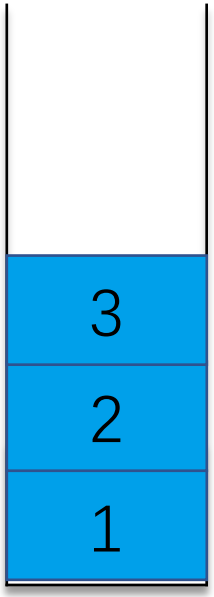
stack1



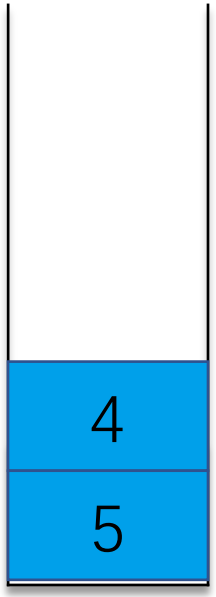
stack2

第二种情况，如果 stack2的长度为0，还是将stack1从尾部删除，添加到 stack2

LeetCode-03.04 化栈为队



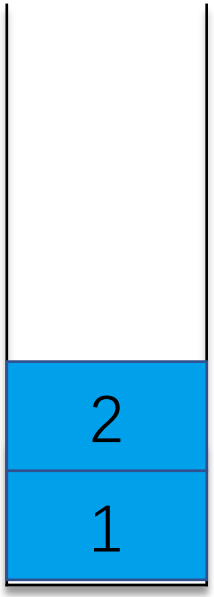
stack1



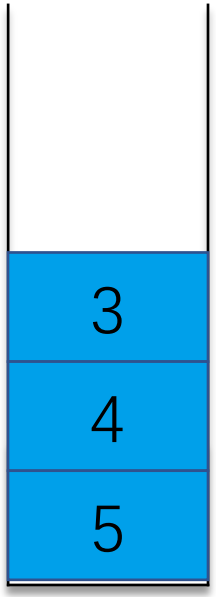
stack2

第二种情况，如果 stack2的长度为0，还是将stack1从尾部删除，添加到 stack2

LeetCode-03.04 化栈为队



stack1



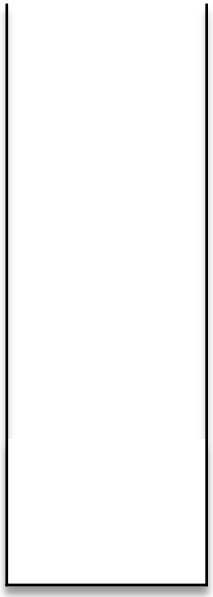
stack2

第二种情况，如果 stack2的长度为0，还是将stack1从尾部删除，添加到 stack2

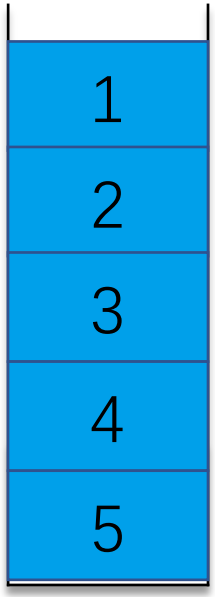
LeetCode-03.04 化栈为队

第二种情况，如果 stack2的长度为0，还是将stack1从尾部删除，添加到 stack2

LeetCode-03.04 化栈为队



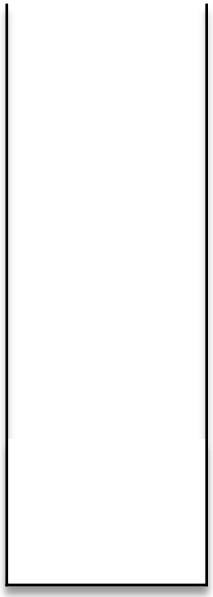
stack1



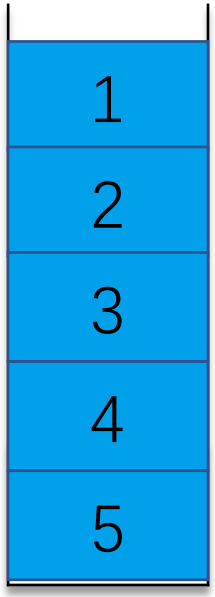
stack2

第二种情况，如果 stack2的长度为0，还是将stack1从尾部删除，添加到 stack2

LeetCode-03.04 化栈为队



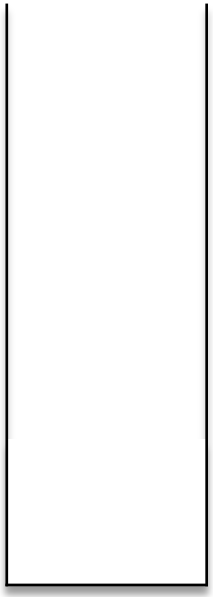
stack1



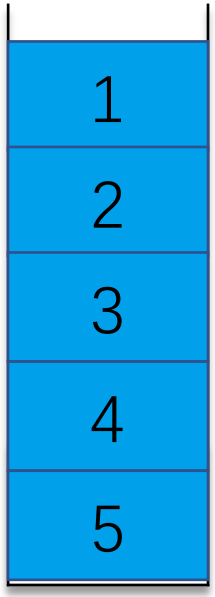
stack2

直到， stack1为空，说明全部元素添加到 stack2

LeetCode-03.04 化栈为队



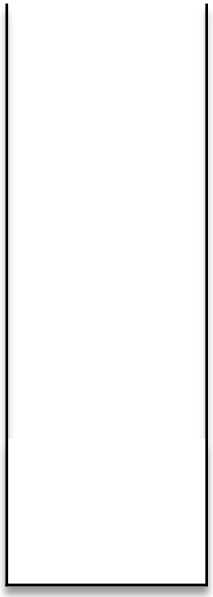
stack1



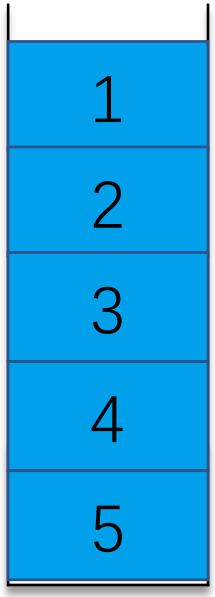
stack2

这里的peek方法是查看，在前端里面，这里的操作和刚刚的pop的操作是一样的

LeetCode-03.04 化栈为队



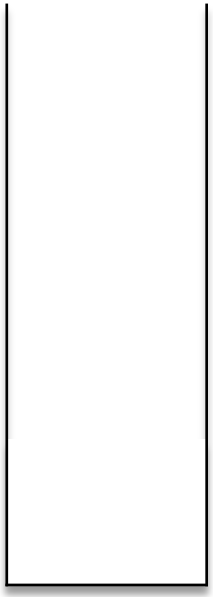
stack1



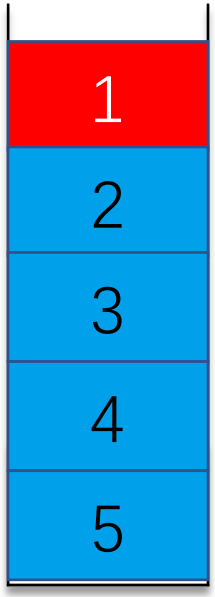
stack2

到了这里，需要注意 peek 和 pop 的返回值不同

LeetCode-03.04 化栈为队



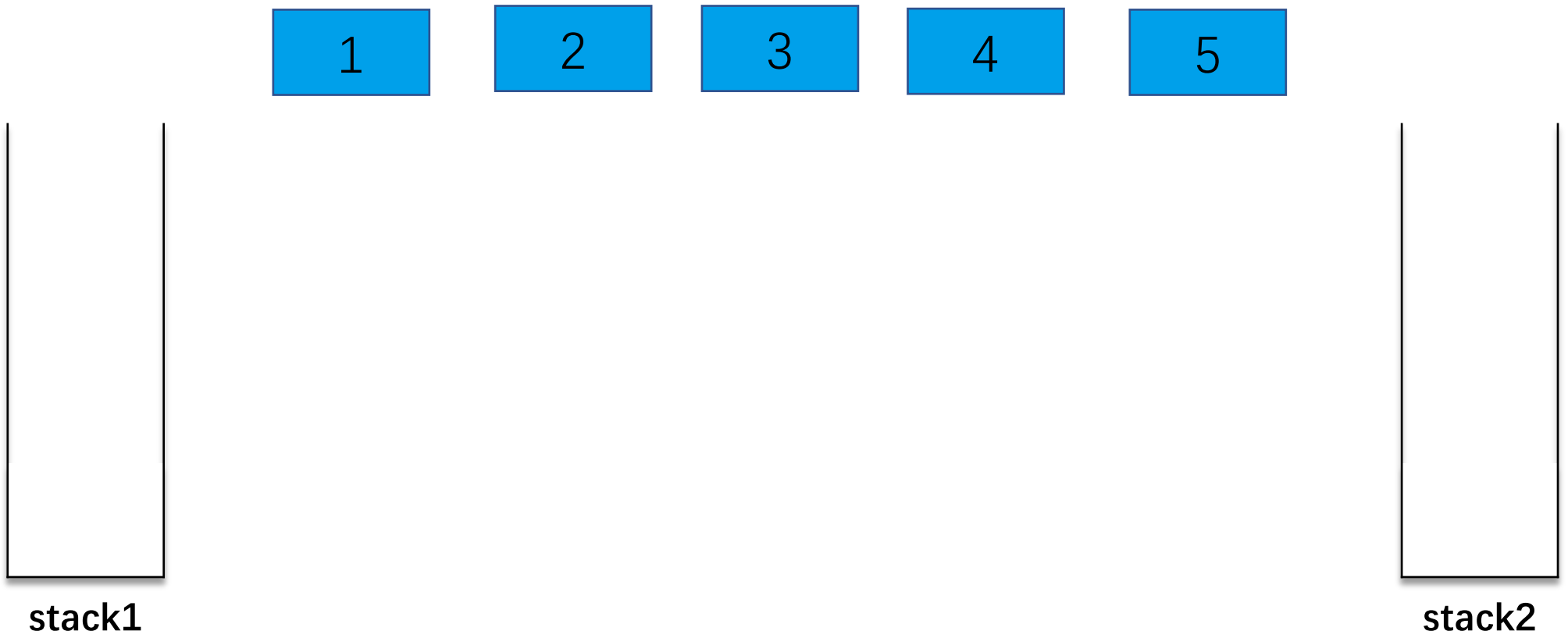
stack1



stack2

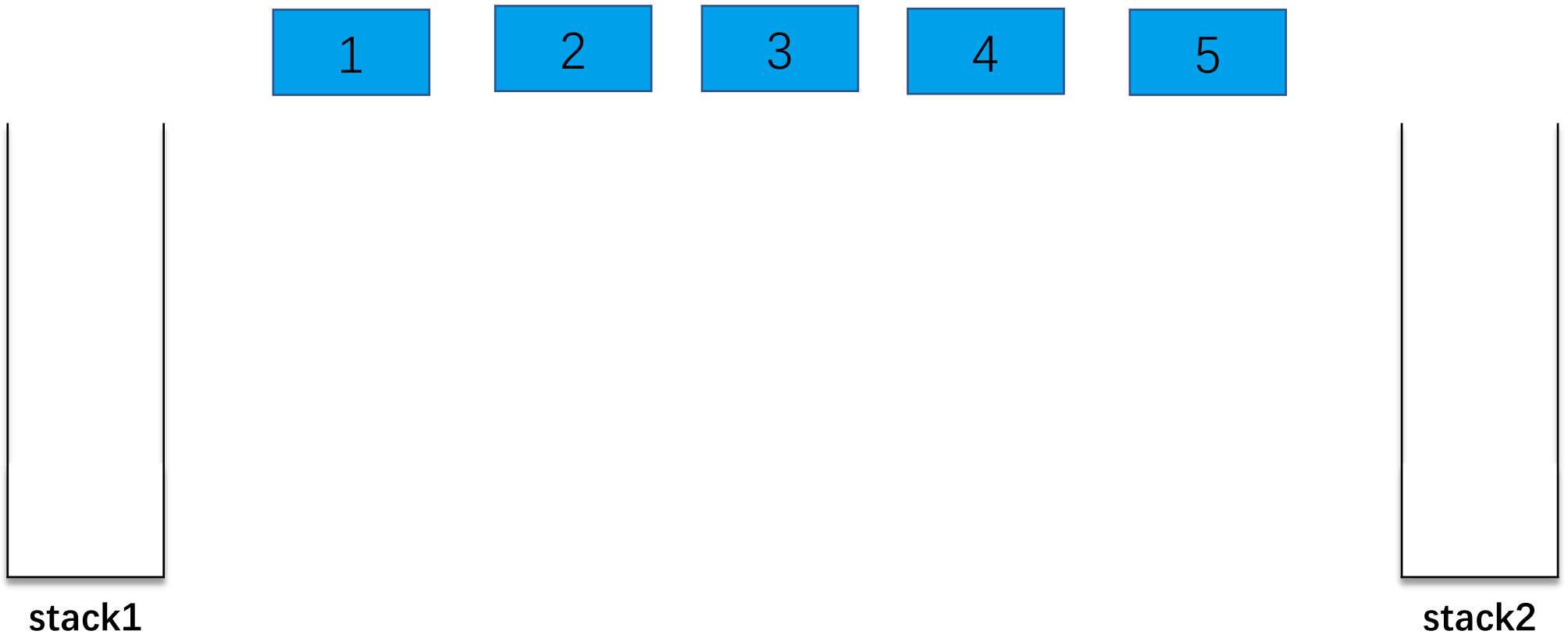
peek 返回 stack2回栈顶的元素，就是length-1 个元素

LeetCode-03.04 化栈为队



最后一个操作是： empty ,这个方法是最后进行判断队列为空

LeetCode-03.04 化栈为队



empty 只是最后返回， stack1和 stack2的长度都为0时候

682.棒球比赛

门徒计划，带你开启算法精进之路

LeetCode-682 棒球比赛

844.比较含退格的字符串

门徒计划，带你开启算法精进之路

LeetCode-844 比较含退格的字符串

946.验证栈序列

门徒计划，带你开启算法精进之路

LeetCode-946 验证栈序列

pushed序列

popped序列

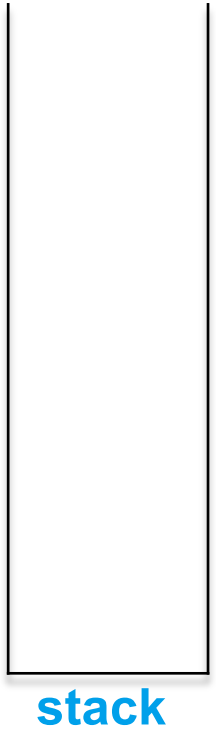
LeetCode-946 验证栈序列

pushed序列

1	2	3	4	5

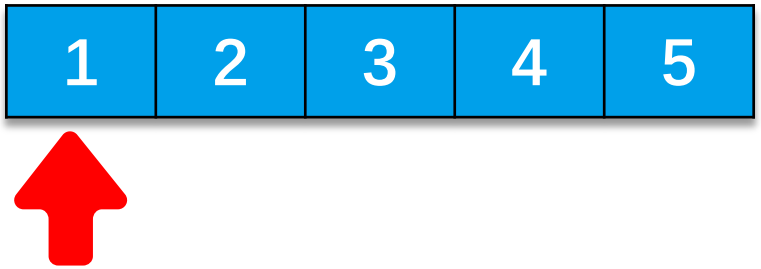
popped序列

4	5	3	2	1



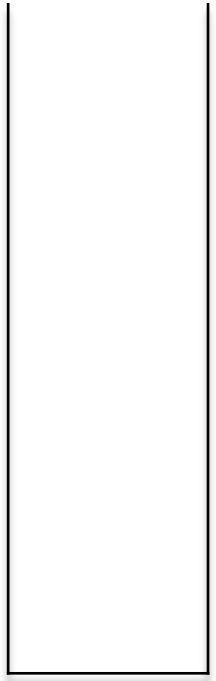
LeetCode-946 验证栈序列

pushed序列



popped序列

遍历一次入一次stack



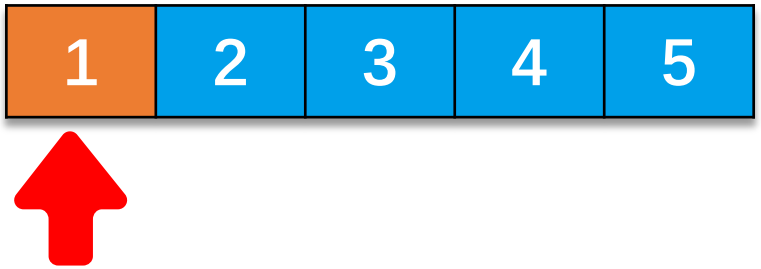
stack

左边操作中

右边操作中

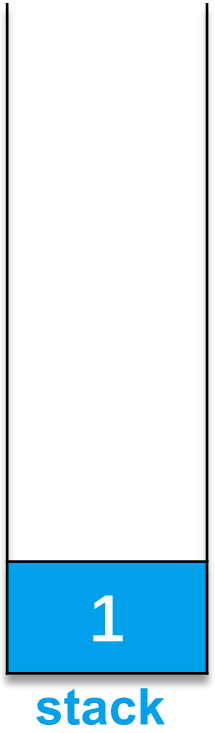
LeetCode-946 验证栈序列

pushed序列



popped序列

遍历一次入一次stack

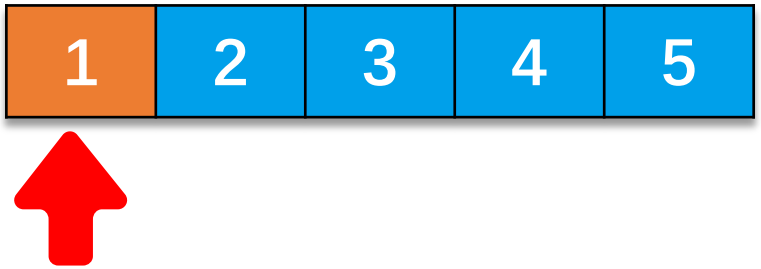


左边操作中

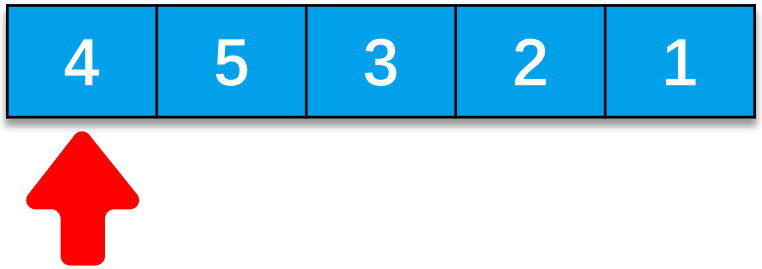
右边操作中

LeetCode-946 验证栈序列

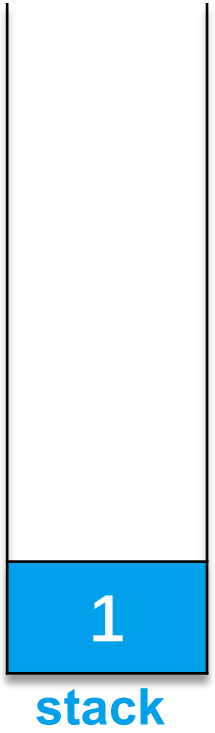
pushed序列



popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束遍历



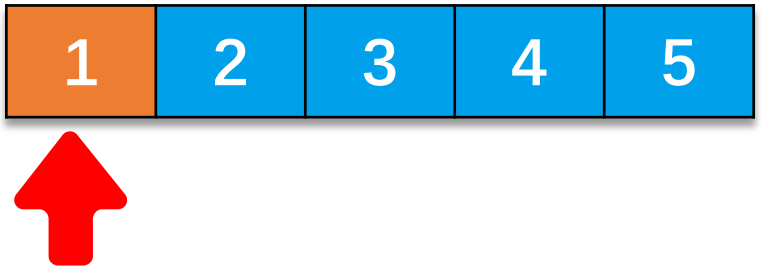
stack

左边操作中

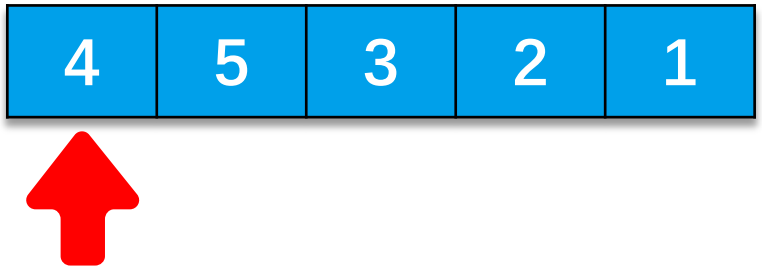
右边操作中

LeetCode-946 验证栈序列

pushed序列

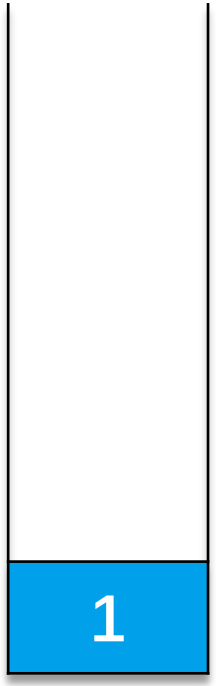


popped序列



遍历一次入一次stack

左边操作中

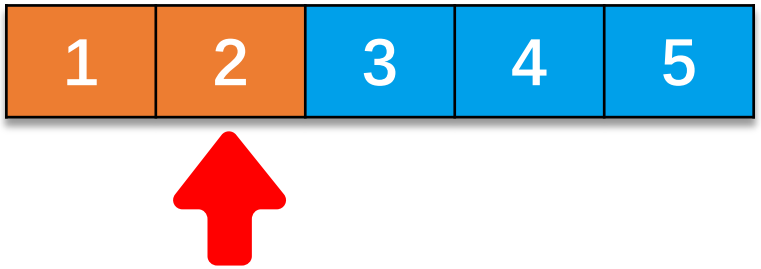


stack

右边操作中

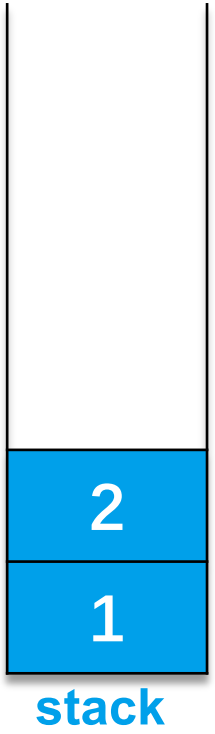
LeetCode-946 验证栈序列

pushed序列



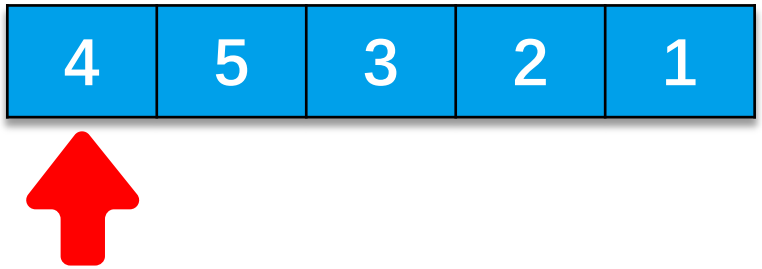
遍历一次入一次stack

左边操作中



stack

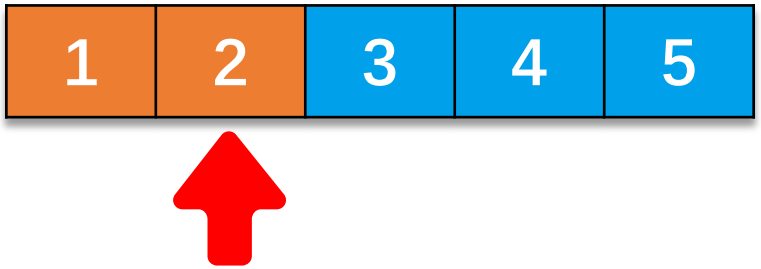
popped序列



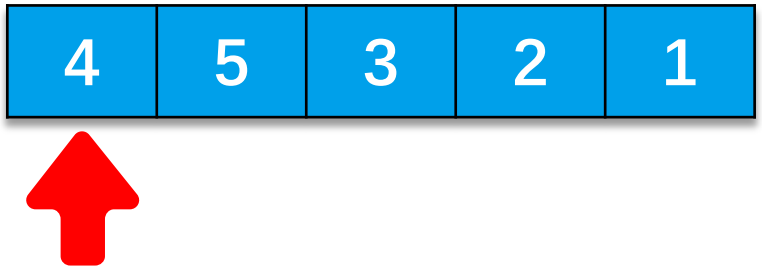
右边操作中

LeetCode-946 验证栈序列

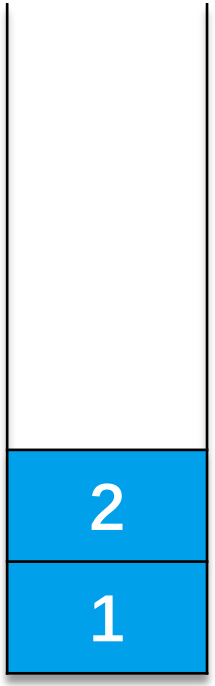
pushed序列



popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束遍历



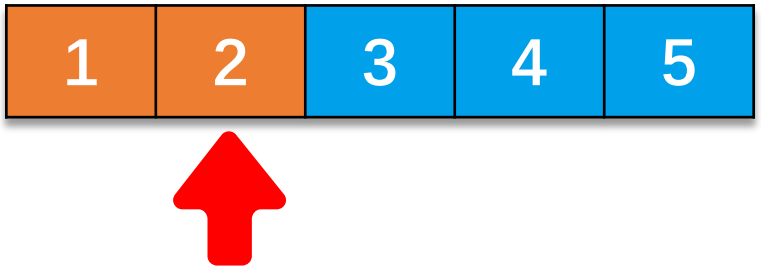
stack

左边操作中

右边操作中

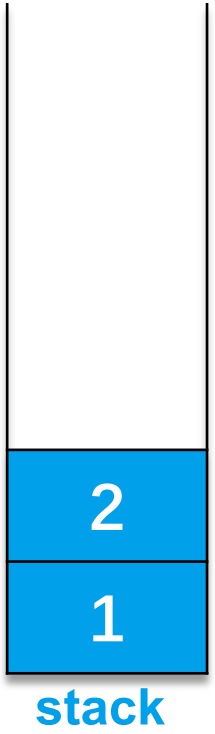
LeetCode-946 验证栈序列

pushed序列



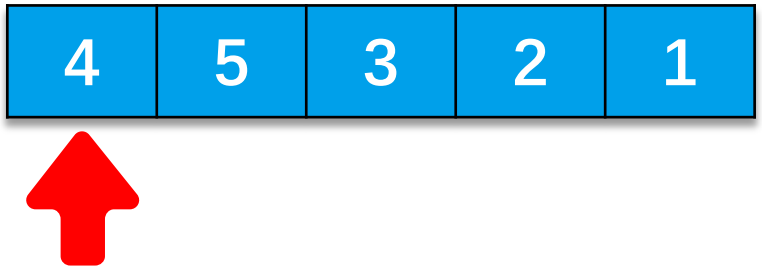
遍历一次入一次stack

左边操作中



stack

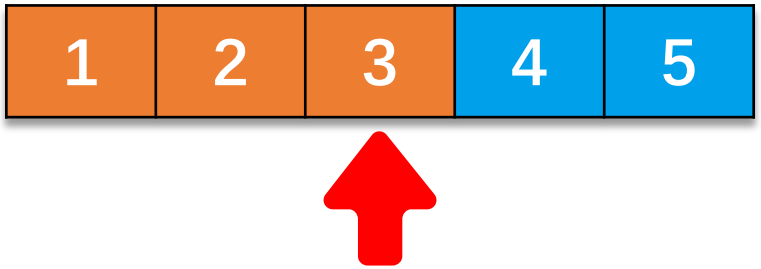
popped序列



右边操作中

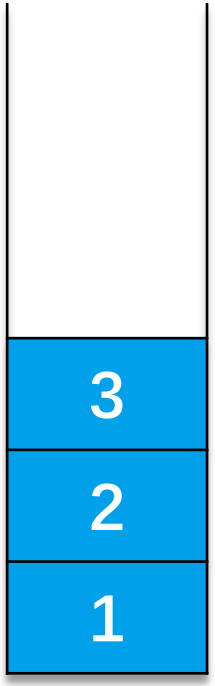
LeetCode-946 验证栈序列

pushed序列



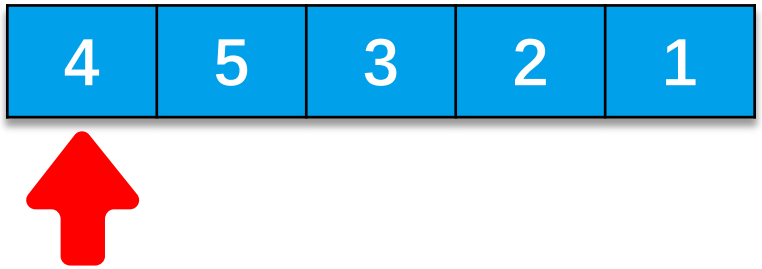
遍历一次入一次stack

左边操作中



stack

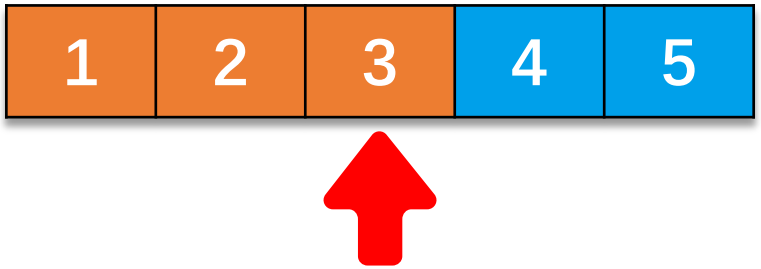
popped序列



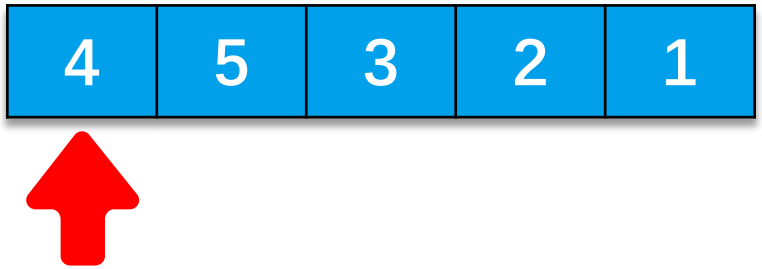
右边操作中

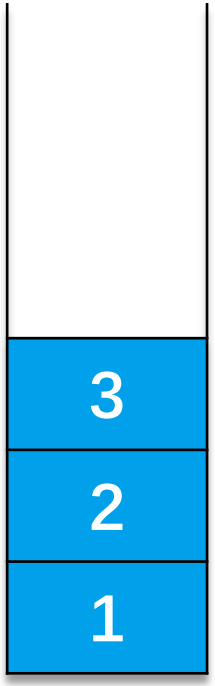
LeetCode-946 验证栈序列

pushed序列



popped序列





stack

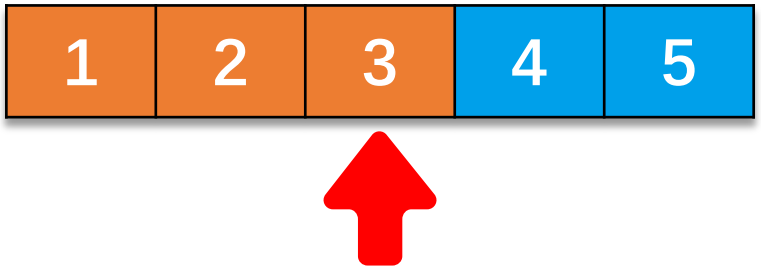
stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

左边操作中

右边操作中

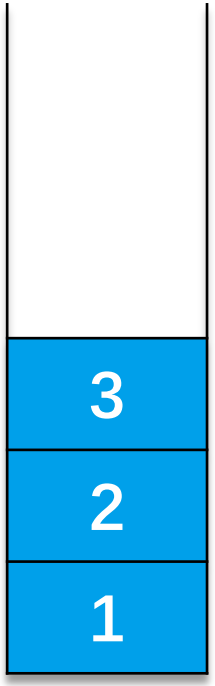
LeetCode-946 验证栈序列

pushed序列



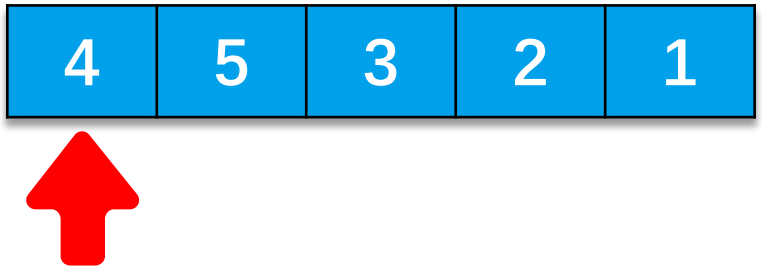
遍历一次入一次stack

左边操作中



stack

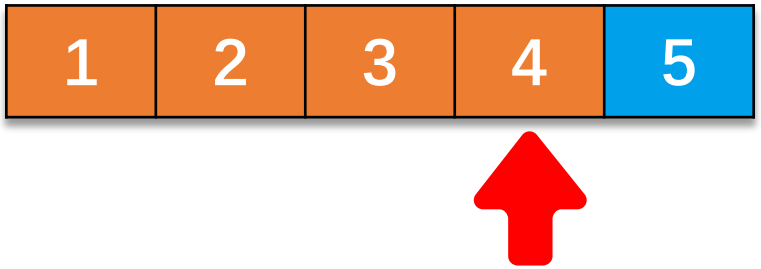
popped序列



右边操作中

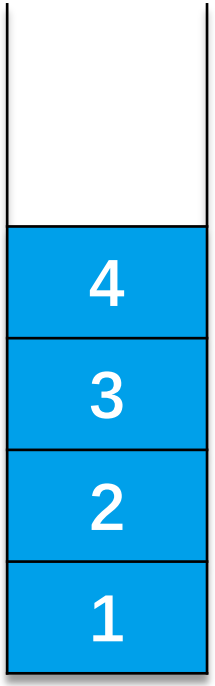
LeetCode-946 验证栈序列

pushed序列



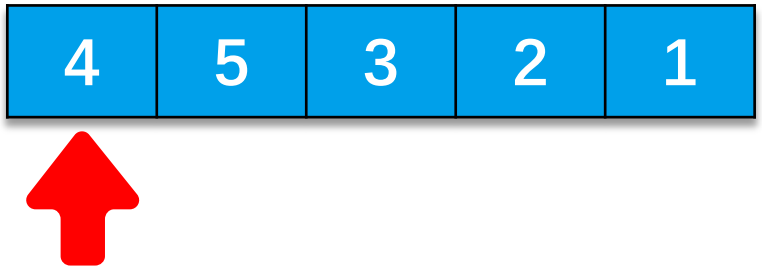
遍历一次入一次stack

左边操作中



stack

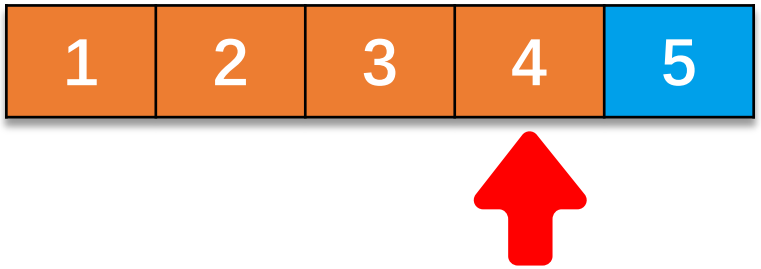
popped序列



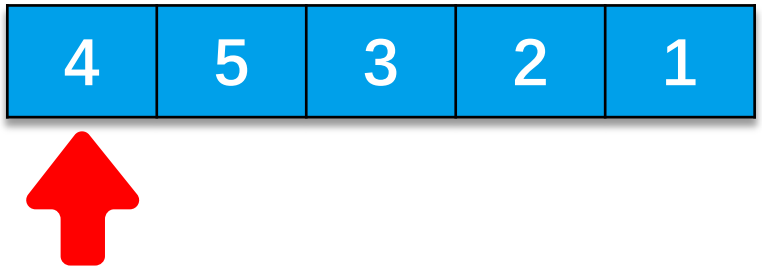
右边操作中

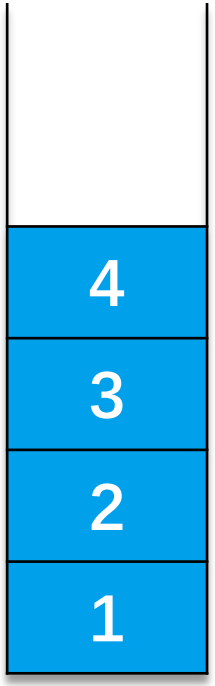
LeetCode-946 验证栈序列

pushed序列



popped序列





stack

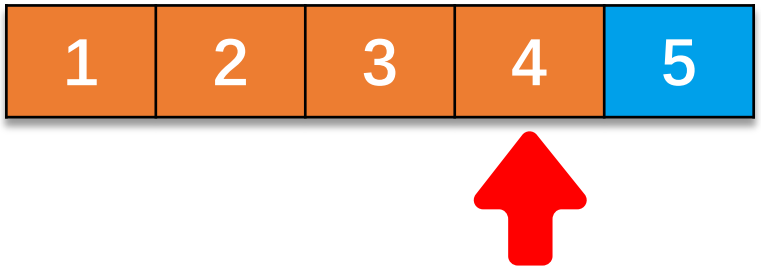
stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

左边操作中

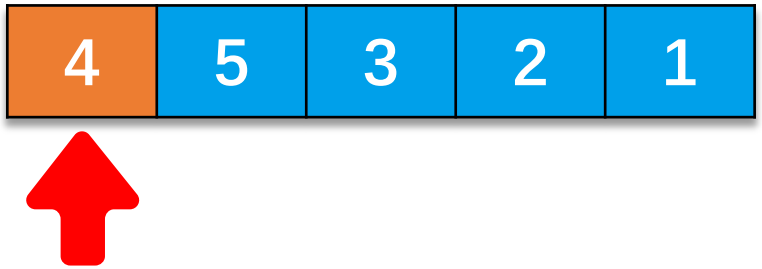
右边操作中

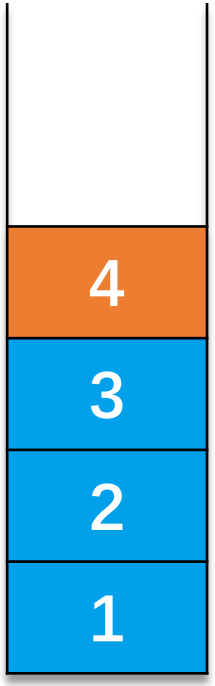
LeetCode-946 验证栈序列

pushed序列



popped序列





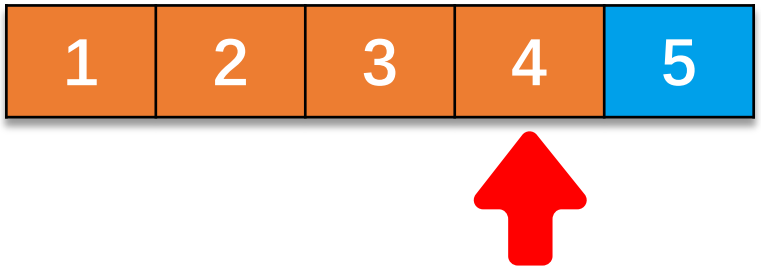
stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

左边操作中

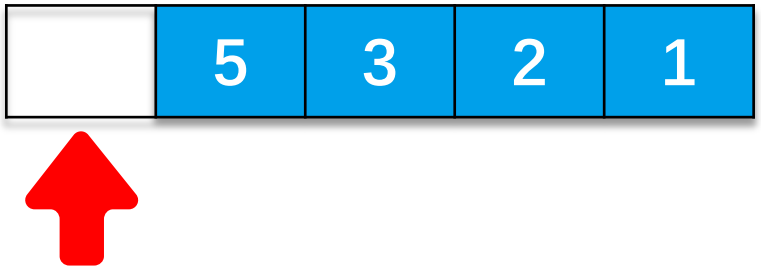
右边操作中

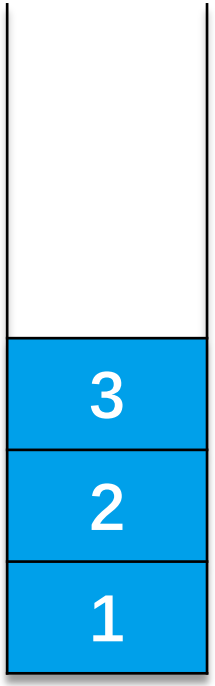
LeetCode-946 验证栈序列

pushed序列



popped序列





stack

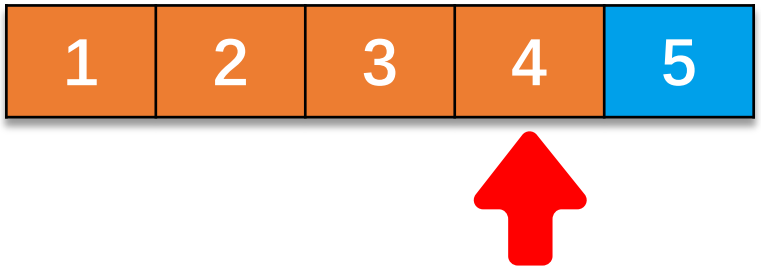
stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

左边操作中

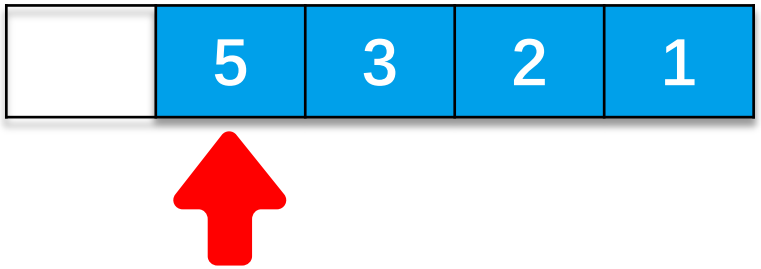
右边操作中

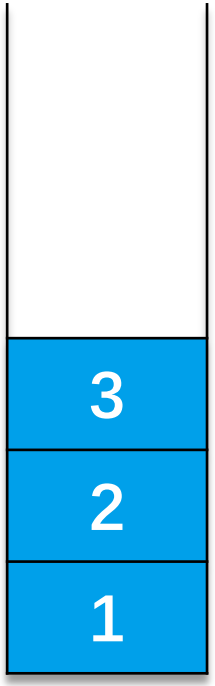
LeetCode-946 验证栈序列

pushed序列



popped序列





stack

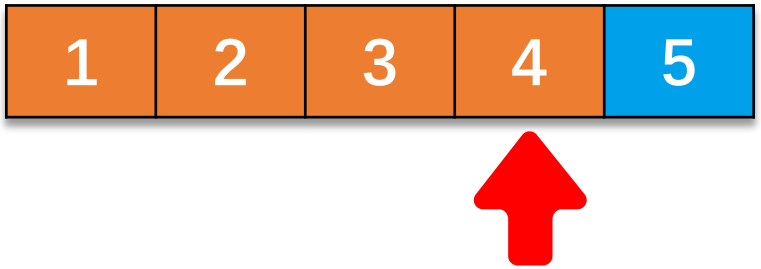
stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

左边操作中

右边操作中

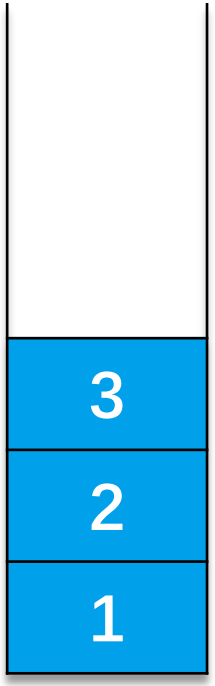
LeetCode-946 验证栈序列

pushed序列



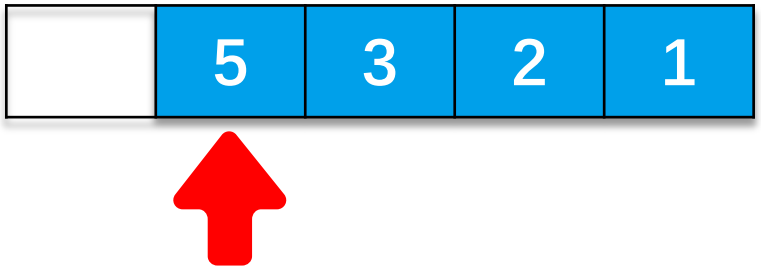
遍历一次入一次stack

左边操作中



stack

popped序列

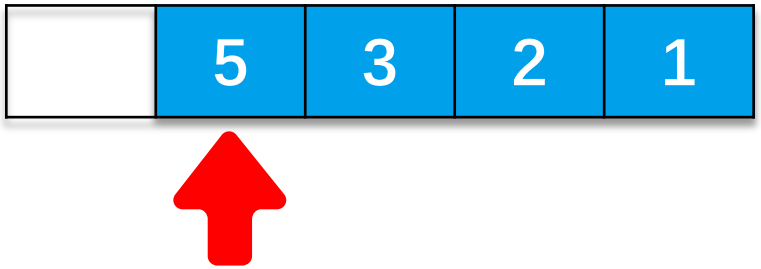


右边操作中

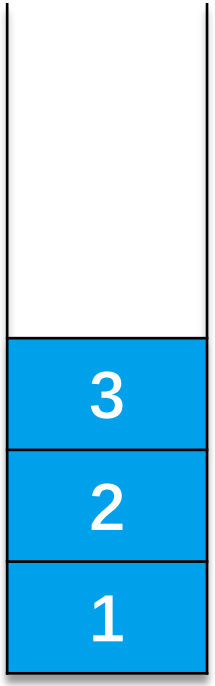
LeetCode-946 验证栈序列

pushed序列

popped序列



遍历一次入一次stack



stack

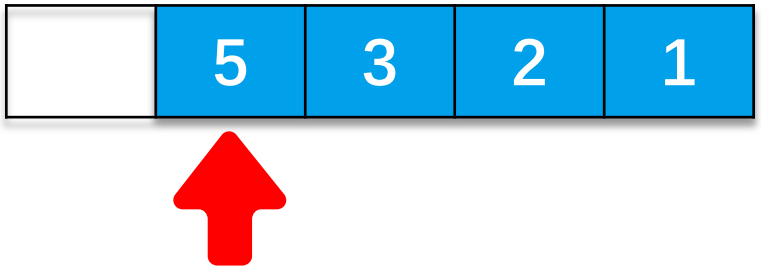
左边操作中

右边操作中

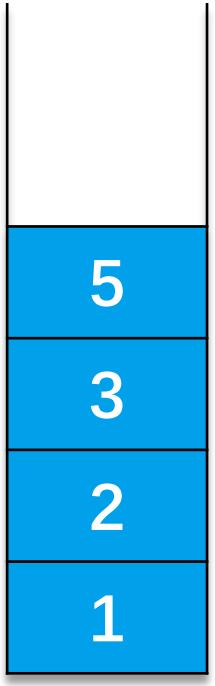
LeetCode-946 验证栈序列

pushed序列

popped序列



遍历一次入一次stack



stack

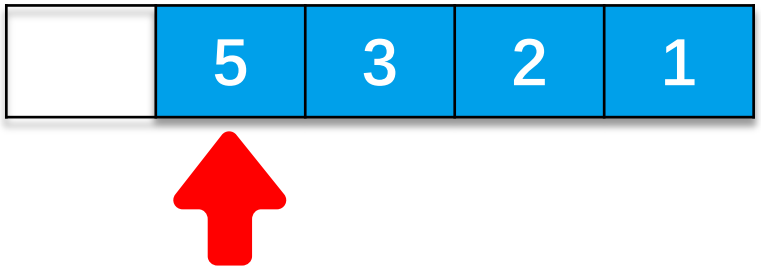
左边操作中

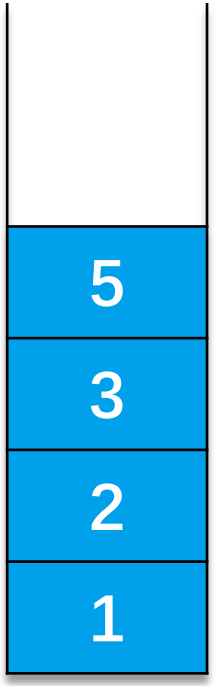
右边操作中

LeetCode-946 验证栈序列

pushed序列

popped序列





stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

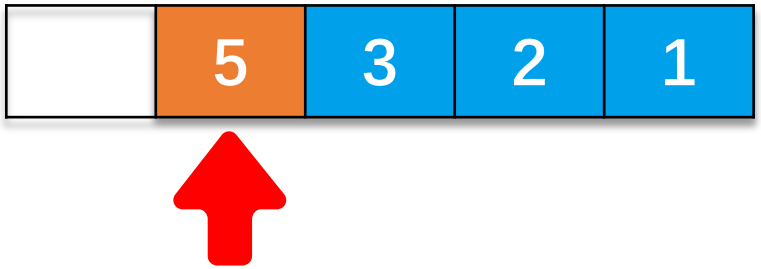
左边操作中

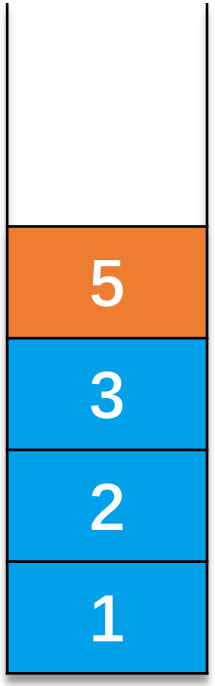
右边操作中

LeetCode-946 验证栈序列

pushed序列

popped序列





stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

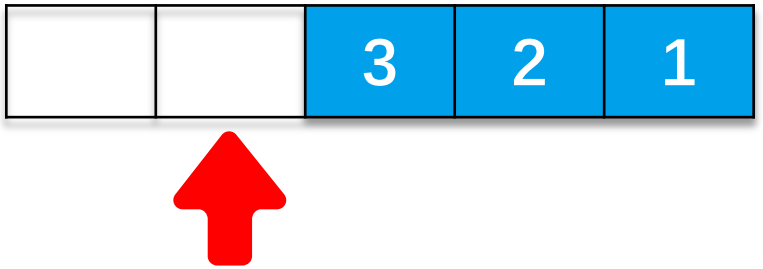
左边操作中

右边操作中

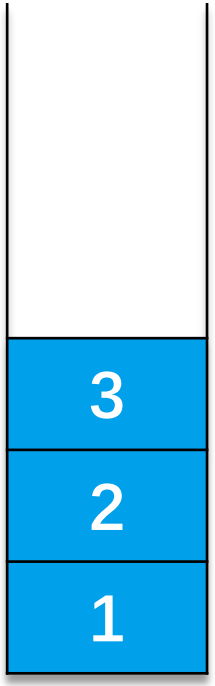
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束



stack

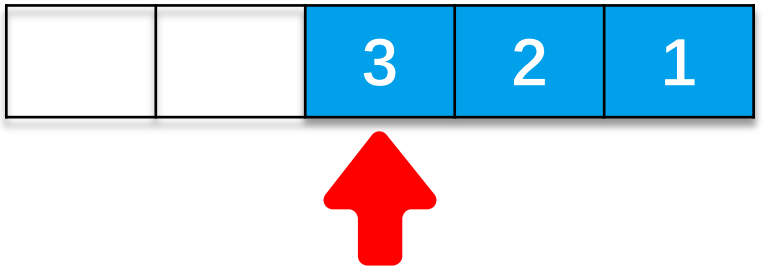
左边操作中

右边操作中

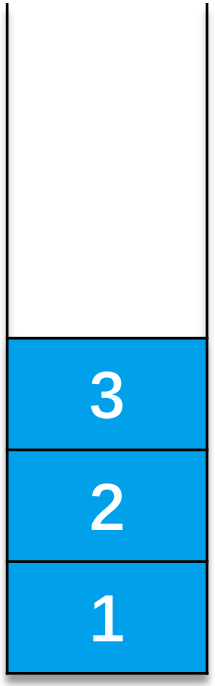
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束



stack

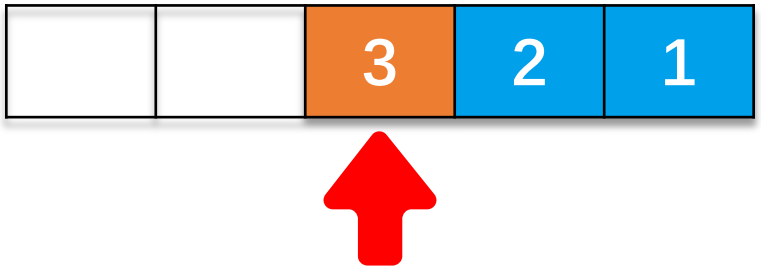
左边操作中

右边操作中

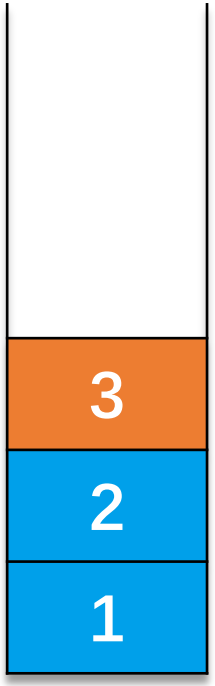
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束



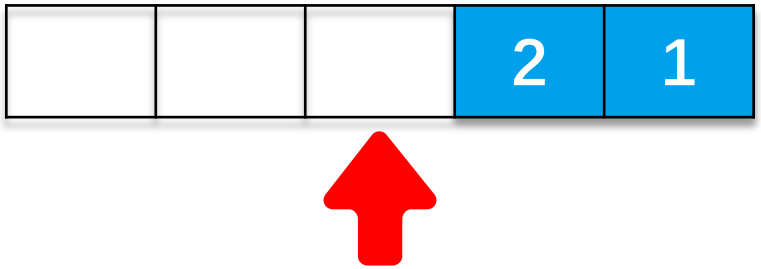
左边操作中

右边操作中

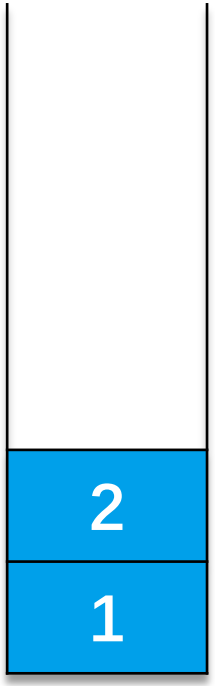
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束



stack

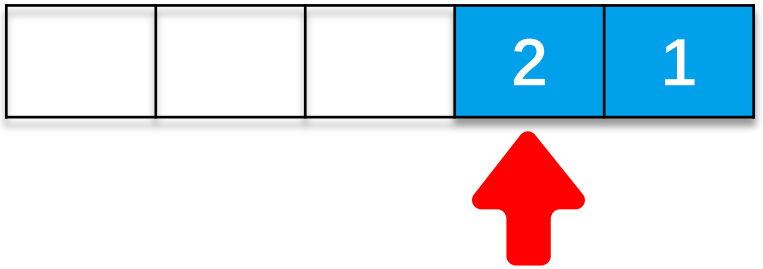
左边操作中

右边操作中

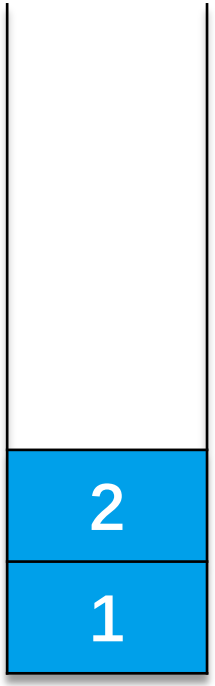
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束



stack

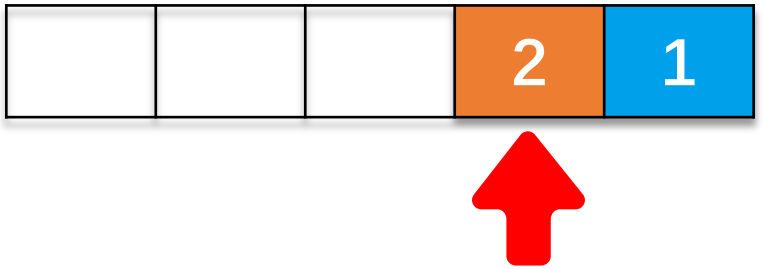
左边操作中

右边操作中

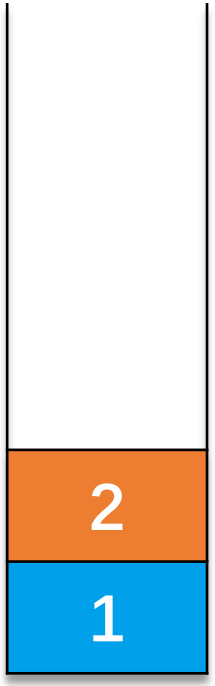
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束



stack

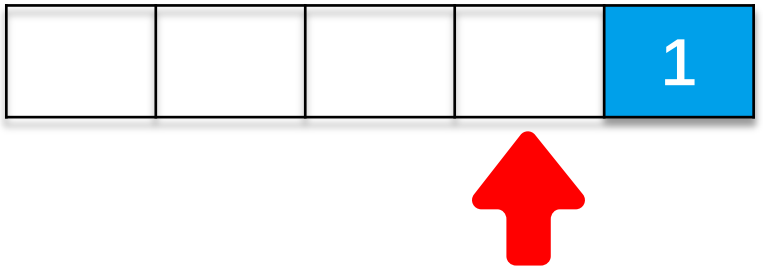
左边操作中

右边操作中

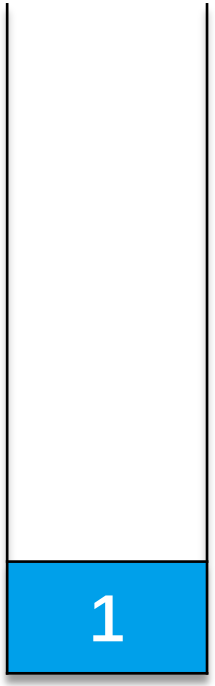
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束



stack

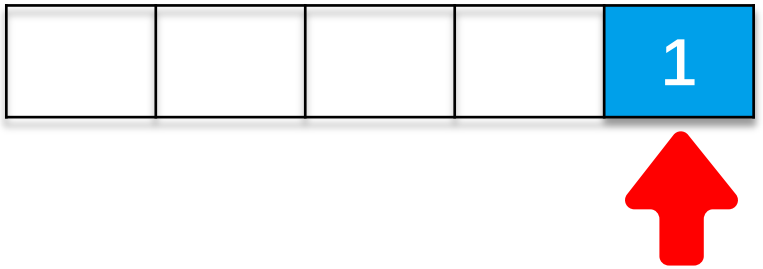
左边操作中

右边操作中

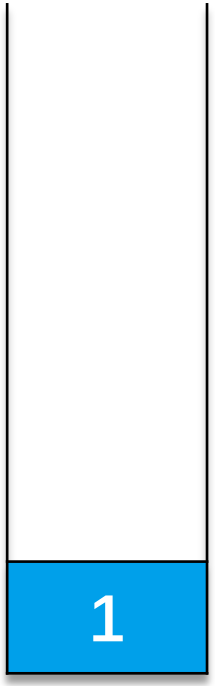
LeetCode-946 验证栈序列

pushed序列

popped序列



stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

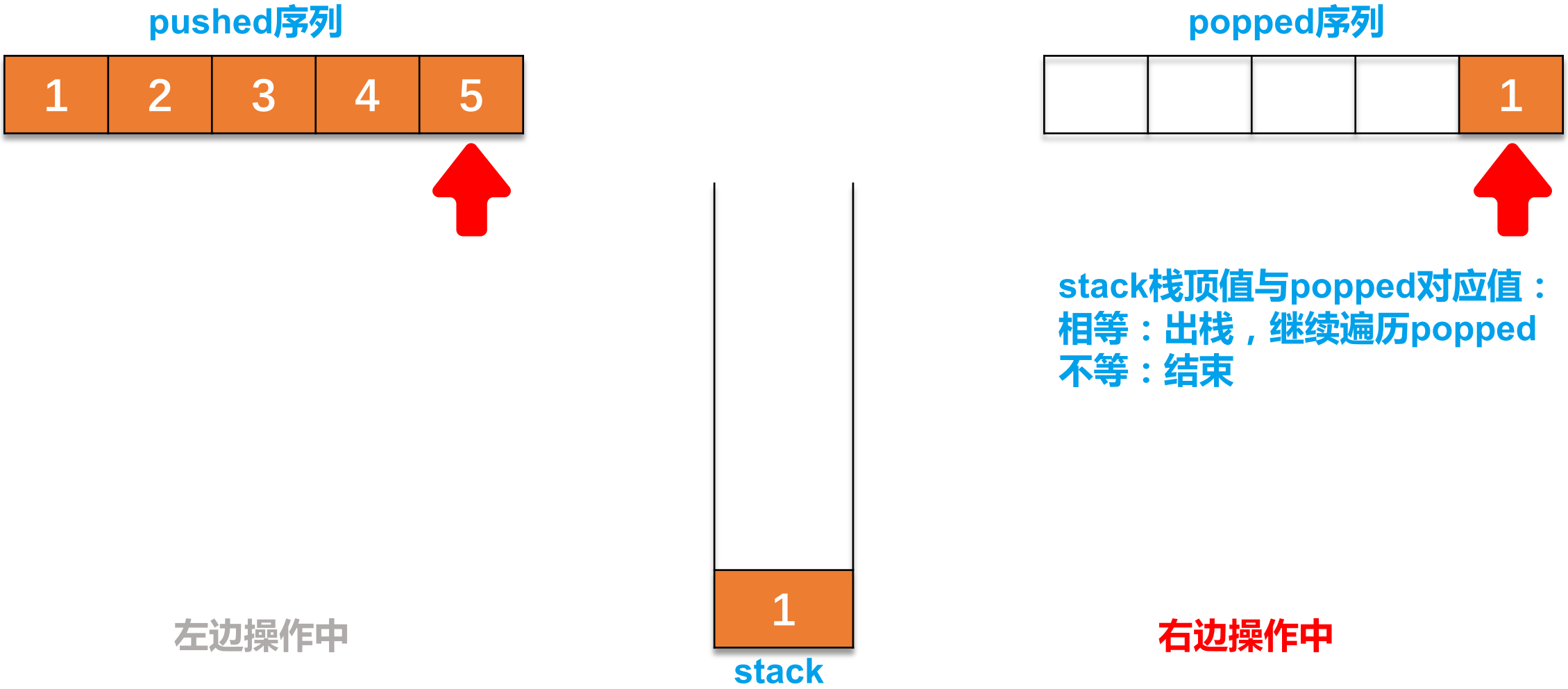


stack

左边操作中

右边操作中

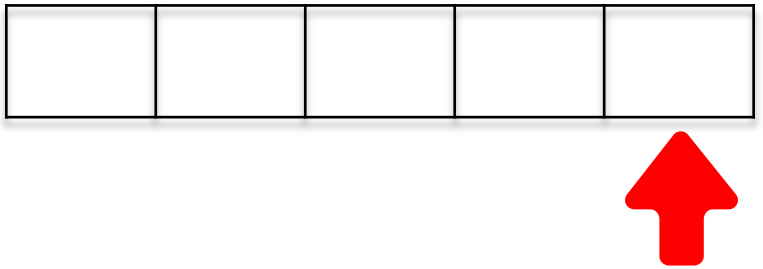
LeetCode-946 验证栈序列

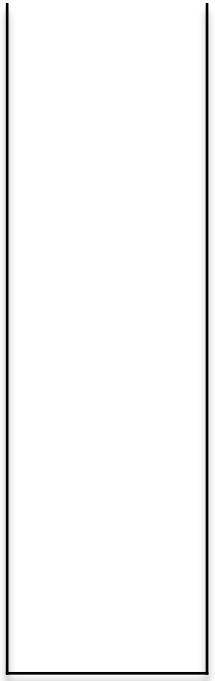


LeetCode-946 验证栈序列

pushed序列

popped序列





stack

stack栈顶值与popped对应值：
相等：出栈，继续遍历popped
不等：结束

左边操作中

右边操作中

经典面试题-栈结构扩展应用

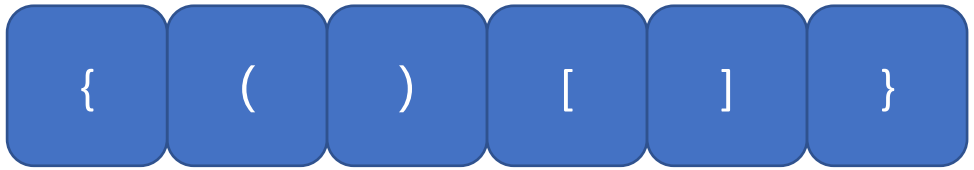
大约用时：（ 60 mins ）

下一部分：经典面试题-智力发散题

20.有效的括号

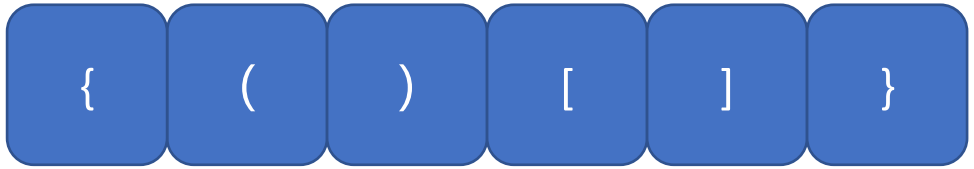
门徒计划，带你开启算法精进之路

LeetCode-20 有效括号



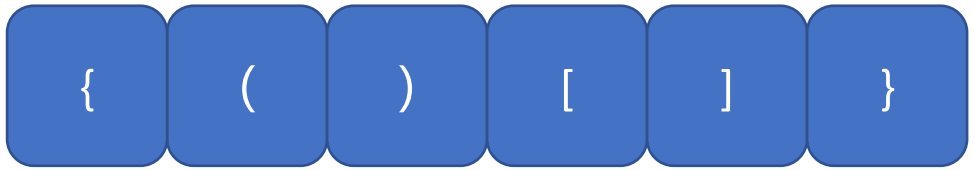
给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ，判断字符串是否有效。
有效字符串需满足：
1.左括号必须用相同类型的右括号闭合。
2.左括号必须以正确的顺序闭合。

LeetCode-20 有效括号



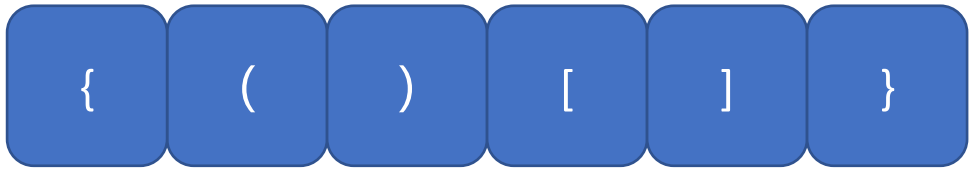
我们遍历给字符串s。
当我们遇到一个左括号时，我们会希望在后面的字符中，有一个相同类型的右括号将其闭合。
由于后遇到的左括号要先闭合，因此我们可以将这个左括号放入栈顶。

LeetCode-20 有效括号



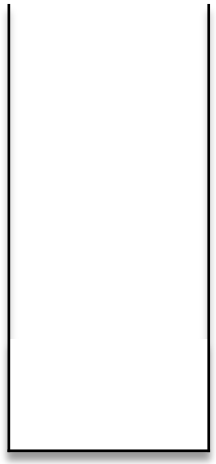
当我们遇到一个右括号时，我们就可以去出栈顶的左括号，判断两个括号是否能闭合。
如果不是相同类型的括号或者栈中没有括号，我们这个字符串就是无效的。

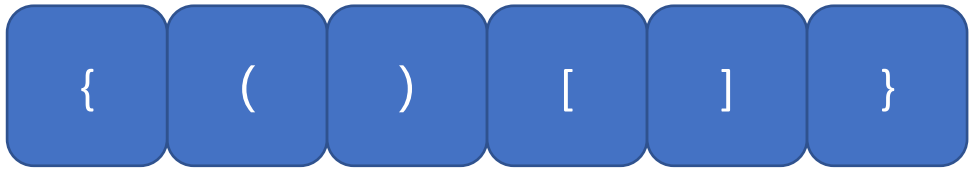
LeetCode-20 有效括号



为了能快速的判断括号的类型，我们可以创建一个Map来帮助我们。
创建一个栈来存储我们的左括号。

LeetCode-20 有效括号

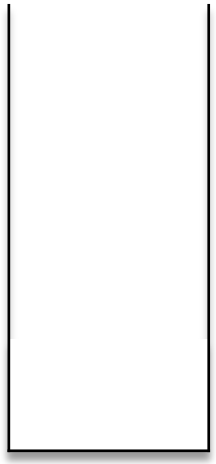


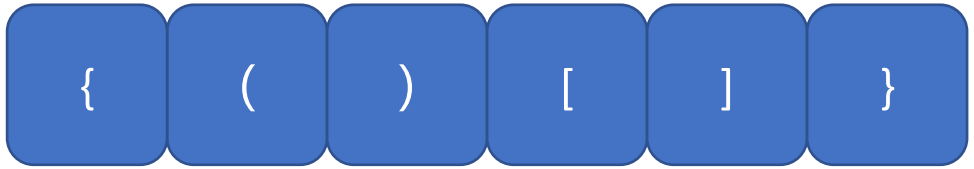


key	val
)	(
]	[
}	{

为了能快速的判断括号的类型，我们可以创建一个Map来帮助我们。
创建一个栈来存储我们的左括号。

LeetCode-20 有效括号

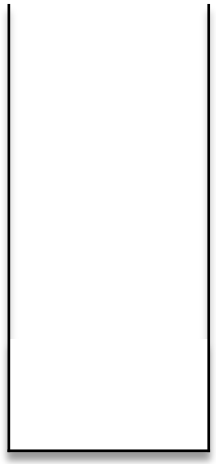


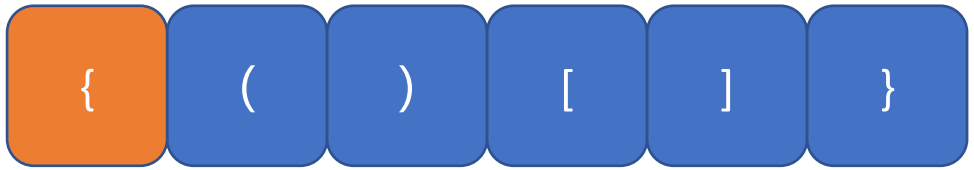


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

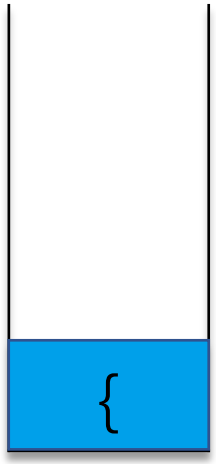


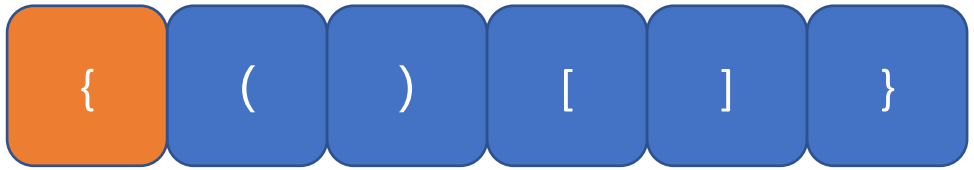


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

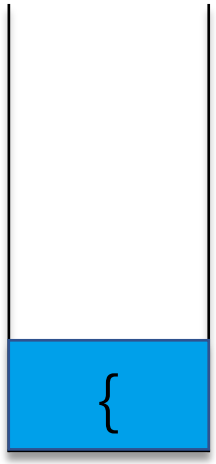


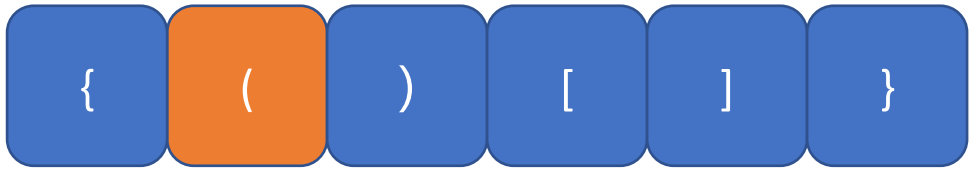


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

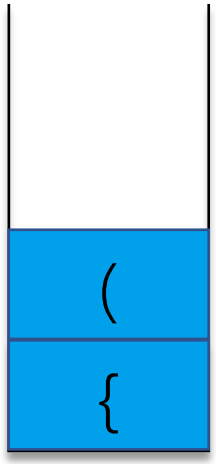


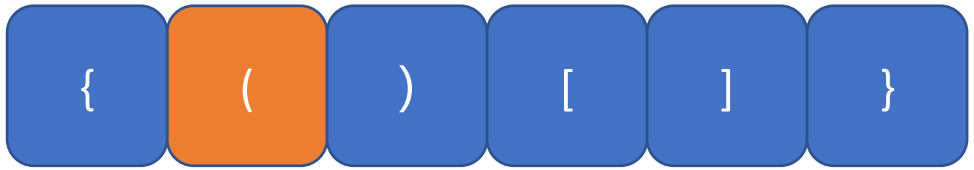


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

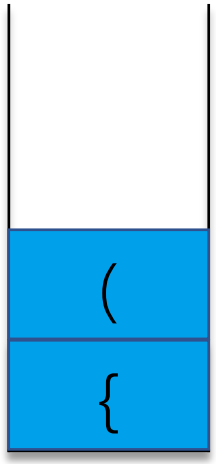


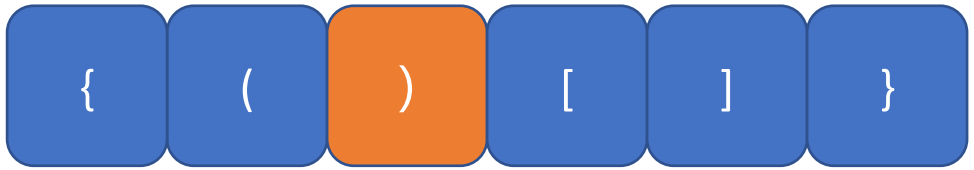


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

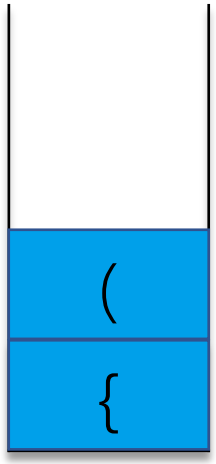


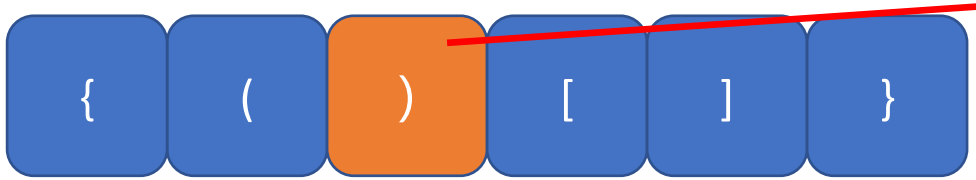


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

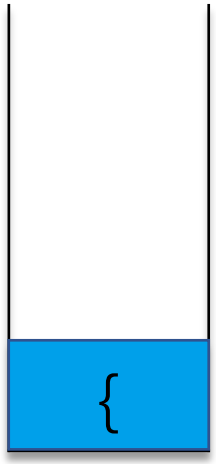


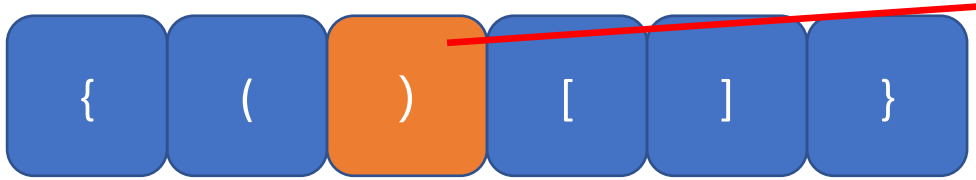


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

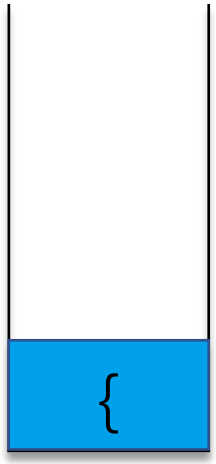


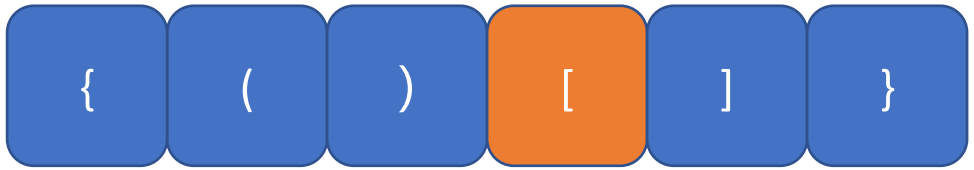


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

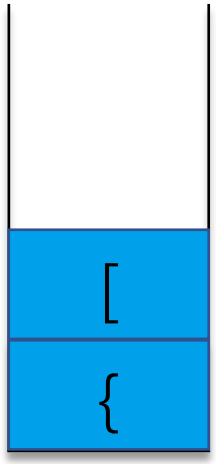


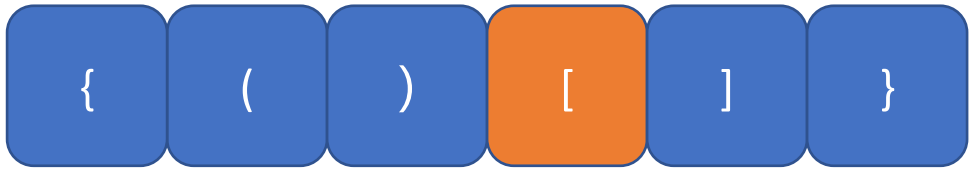


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

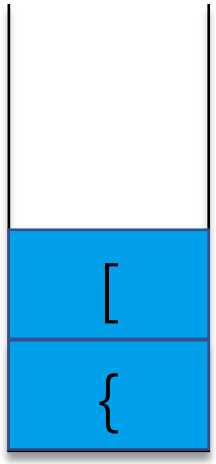


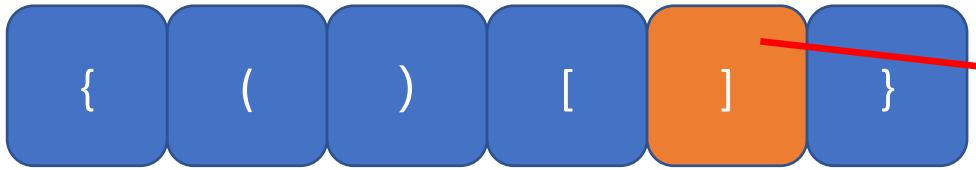


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

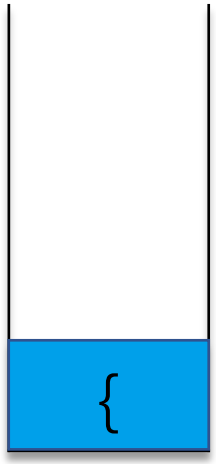


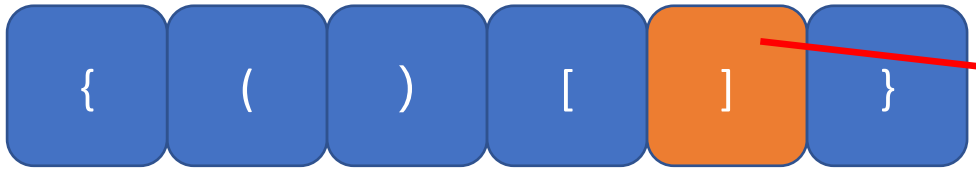


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

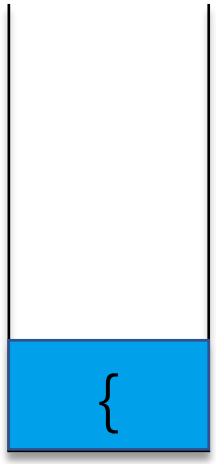


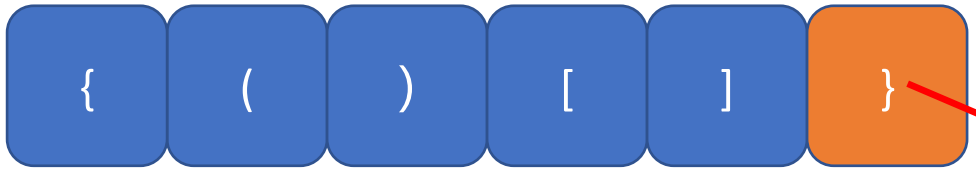


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

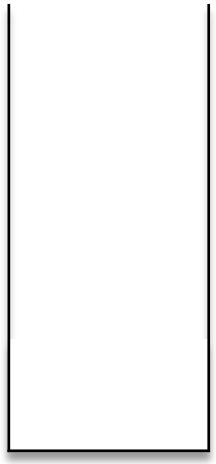


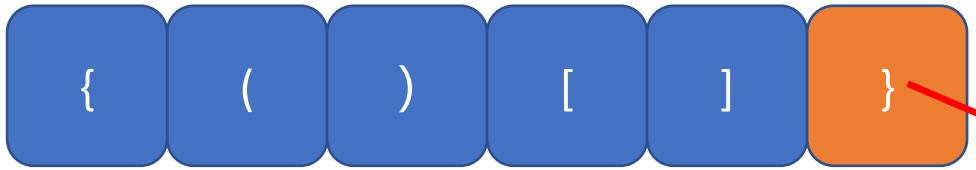


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号

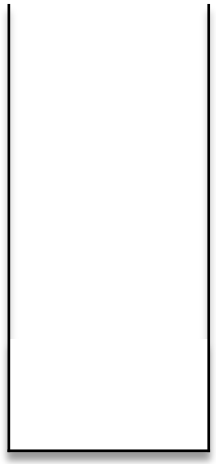


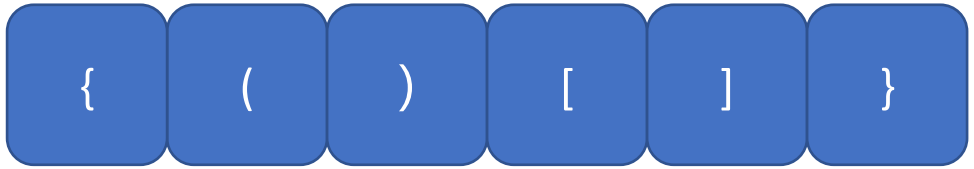


key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

LeetCode-20 有效括号





key	val
)	(
]	[
}	{

开始遍历我们的字符串，如果是左括号，我们就将该字符加入到栈中。
如果是右括号，我们就将判断栈顶的元素和当前的元素是否能完全闭合。

1021.删除最外层的括号

门徒计划，带你开启算法精进之路

LeetCode-1021 删除最外层的括号

1249.移除无效的括号

门徒计划，带你开启算法精进之路

LeetCode-1249 移除无效的括号

145.二叉树的后续遍历

门徒计划，带你开启算法精进之路

LeetCode-145 二叉树的后续遍历

331.验证二叉树的前序序列化

门徒计划，带你开启算法精进之路

LeetCode-331 验证二叉树的前序序列化

227.基本计算器II

门徒计划，带你开启算法精进之路

LeetCode-227 基本计算器II

经典面试题-智力发散题

大约用时：（40 mins）

下一部分：答疑解惑-留作业

636.函数的独占时间

门徒计划，带你开启算法精进之路

LeetCode-636 函数的独占时间

1124.表现良好的最长时间段

门徒计划，带你开启算法精进之路

LeetCode-1124 表现良好的最长时间段

答疑解惑-留作业

大约用时：（5 mins）

下一部分：大家晚安

每天都想干翻这个世界
到头来，被世界干的服服帖帖

大家晚安
--船长