数据结构和算法
(Python描述)

郭 炜

微信公众号

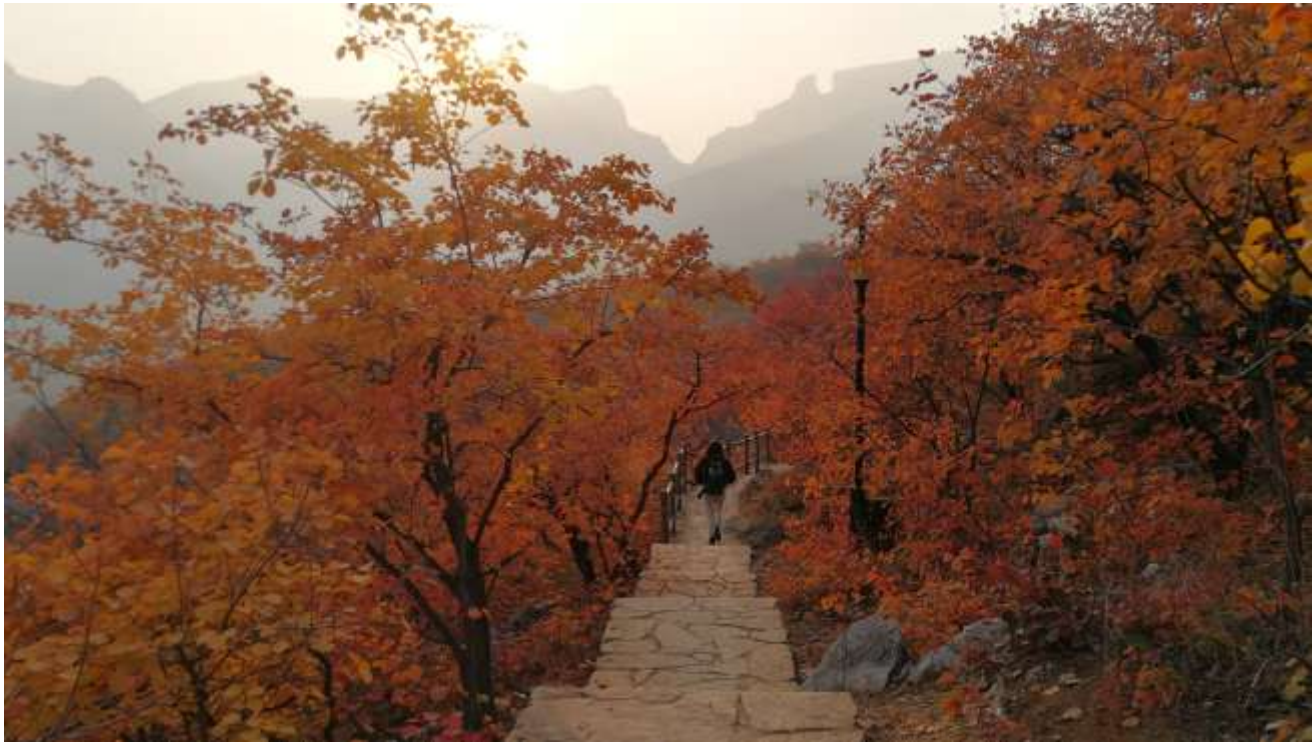
微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

栈的应用

栈的概念和实现



北京坡峰岭红叶

栈的概念

- 类似于子弹匣，后压进去的子弹，先射出去

- 支持四种操作：

top()	返回栈顶元素
push(x)	将x压入栈中
pop()	弹出并返回栈顶元素
isEmpty ()	看栈是否为空

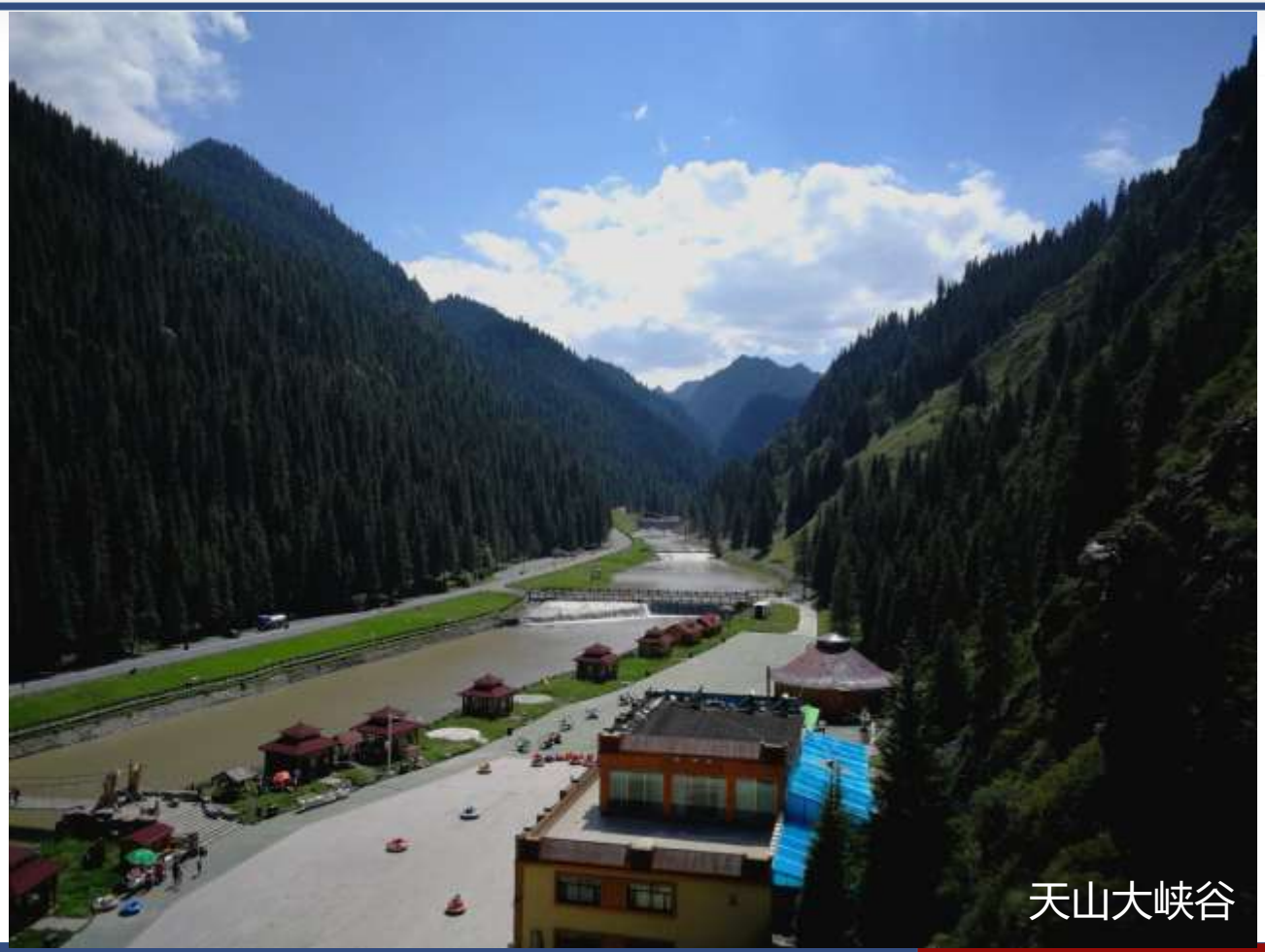
- 要求上面操作复杂度都是O(1)

栈的实现

- 用列表可以实现栈
- 四种操作的实现(stack为一个列表)：

top()	stack[-1]
push(x)	stack.append(x)
pop()	stack.pop()
isEmpty ()	len(stack) == 0

例题1
字符串中的括号配对



例题1：字符串中的括号配对

●字符串中可能有三种成对的括号，"( )"、"[ ]"、"{ }"。判断字符串的括号是否都正确配对了(不存在括号也算正确配对）。括号交叉算不正确配对，例如"12{34[78}ab]"。但是一对括号被包含在另一对括号里面，例如"12{ab[8]}"不影响正确性。

解题思路：

从头到尾扫描字符串，碰到左括号就入栈。碰到右括号，就要求栈顶必须是一个和它配对的左括号，如果不是，则断定字符串不符合要求。如果是，则弹出栈顶。

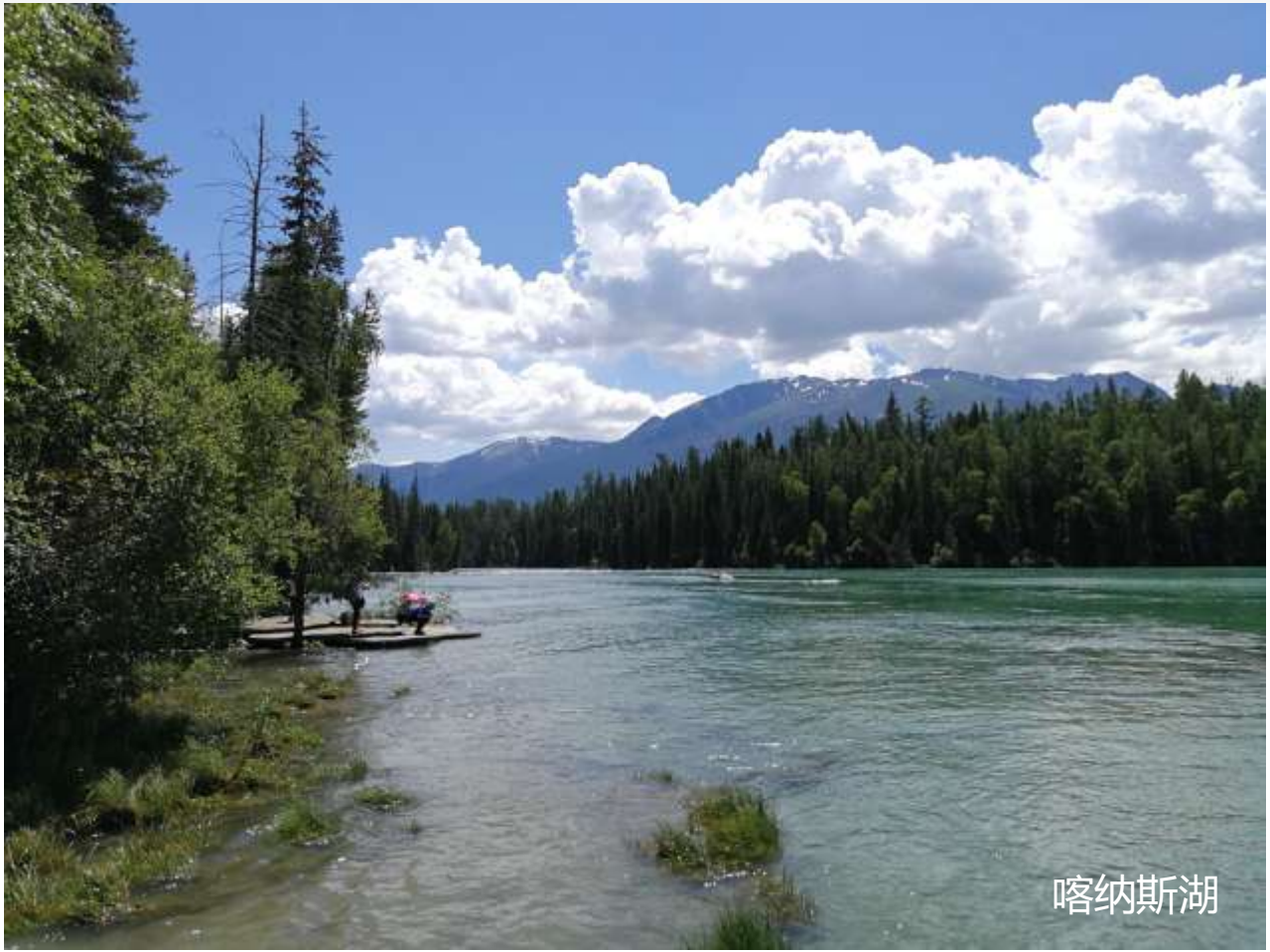
字符串扫描结束时，栈为空则为正确，不为空则为错误

例题1：字符串中的括号配对

```
def match(s):  #复杂度O(n)
    stack = []
    pairs = {")":"(","]":"[","}":"{" }
    for x in s:
        if x in "([{":
            stack.append(x)
        elif x in ")]}":
            if len(stack) == 0 or stack[-1] != pairs[x]:
                return False
            stack.pop()
    return len(stack) == 0

print(match(input()))
```

例题2
后序表达式求值



例题2: 后序表达式求值

●运算符：+ - * / 其中 * / 优先级高于 + -

●原子：整数或者小数

●后序表达式递归定义：

1）一个原子是一个后序表达式

2）两个后序表达式a、b加上一个运算符c，是一个后序表达式。

a b c 的计算方法和传统的中序表达式 (a) c (b) 一样

●以下都是后序表达式(原子、运算符之间用空格分隔)

```
3.4
5
5 3.4 +          等价于 5 + 3.4
5 3.4 + 6 /      等价于 (5+3.4)/6
5 3.4 + 6 * 3 +  等价于 (5+3.4)*6 + 3
```

例题2: 后序表达式求值

- 求给定后序表达式的值

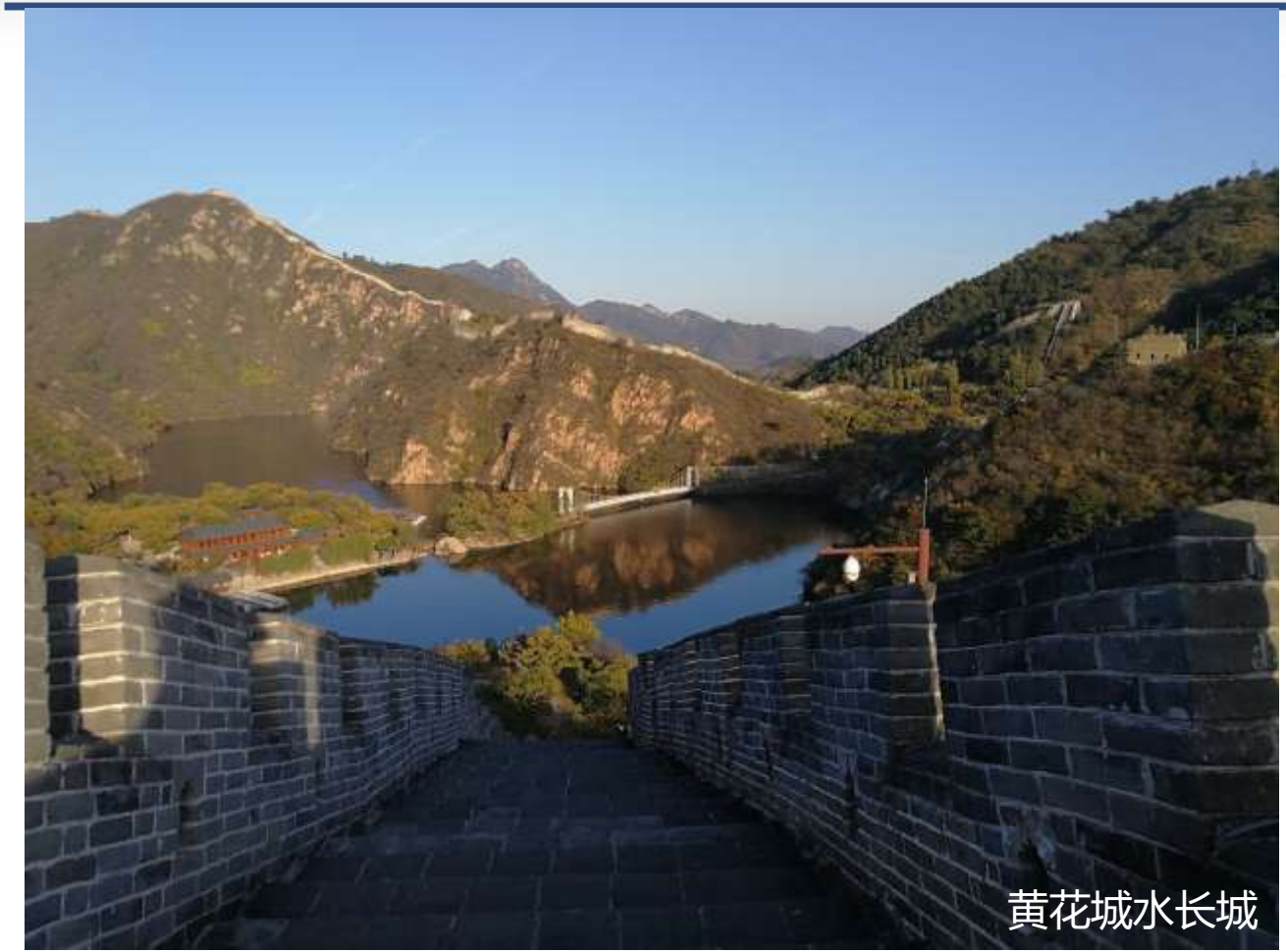
从左到右扫描一遍后序表达式，碰到原子就入栈，碰到运算符，就取出栈顶两个元素进行运算，并将结果压入栈中。

扫描结束时，栈里应该只有一个元素，就是后序表达式的值

例题2: 后序表达式求值


```
def countSuffix(s):  #计算后序表达式s的值，复杂度O(n)
    s = s.split()
    stack = []
    for x in s:
        if x in "+-*/":
            a,b = stack.pop(),stack.pop()
            stack.append(eval(str(b) + x + str(a)))
        else:
            stack.append(float(x))
    return stack[0]
```

例题3
中序表达式转换成后序表达式



例题3: 中序表达式转换成后序表达式

给定一个中序表达式，求转成后序表达式。
原子、运算符和括号之间用空格分隔。

输入样例:

```
3 * ( 4 + 5 ) * ( 6 + 7 ) + ( 5 - 2 ) * 1 + 3
```

输出样例:

```
3 4 5 + * 6 7 + * 5 2 - 1 * + 3 +
```

例题3: 中序表达式转换成后序表达式

思路:

① 一个最简单的两原子中序表达式，转换成后序表达式时，原子顺序不变。 因此复杂的中序表达式，转换成后序表达式时，原子顺序也不变。

② 用列表result存放转换的结果，则扫描中序表达式时，碰到原子，就添加到result

③ 每个运算符都有两个操作数，操作数为原子或带括号的中序表达式。扫描时碰到运算符，不能直接将其添加到result（因为其第二个操作数还未出现），应存放在某处(栈中），以便合适的时候找出来添加到result

④ 扫描到运算符a时，如果发现栈顶的运算符优先级不低于a，则将栈顶运算符添加到result（此刻result的尾部应该是一个原子或者一个刚生成的后序表达式），直到栈为空或者栈顶运算符优先级低于 a或栈顶为"("，然后将a入栈

⑤ 扫描结束时，将栈中所有元素取出添加到result

例题3: 中序表达式转换成后序表达式

括号表达式的处理：

1) 括号表达式当作一个中序表达式处理

2) 碰到"("则将其入栈，然后开始处理中序表达式的过程。碰到")"时，认为该中序表达式处理结束，应该将栈顶元素不停弹出添加到result，直到“("出栈

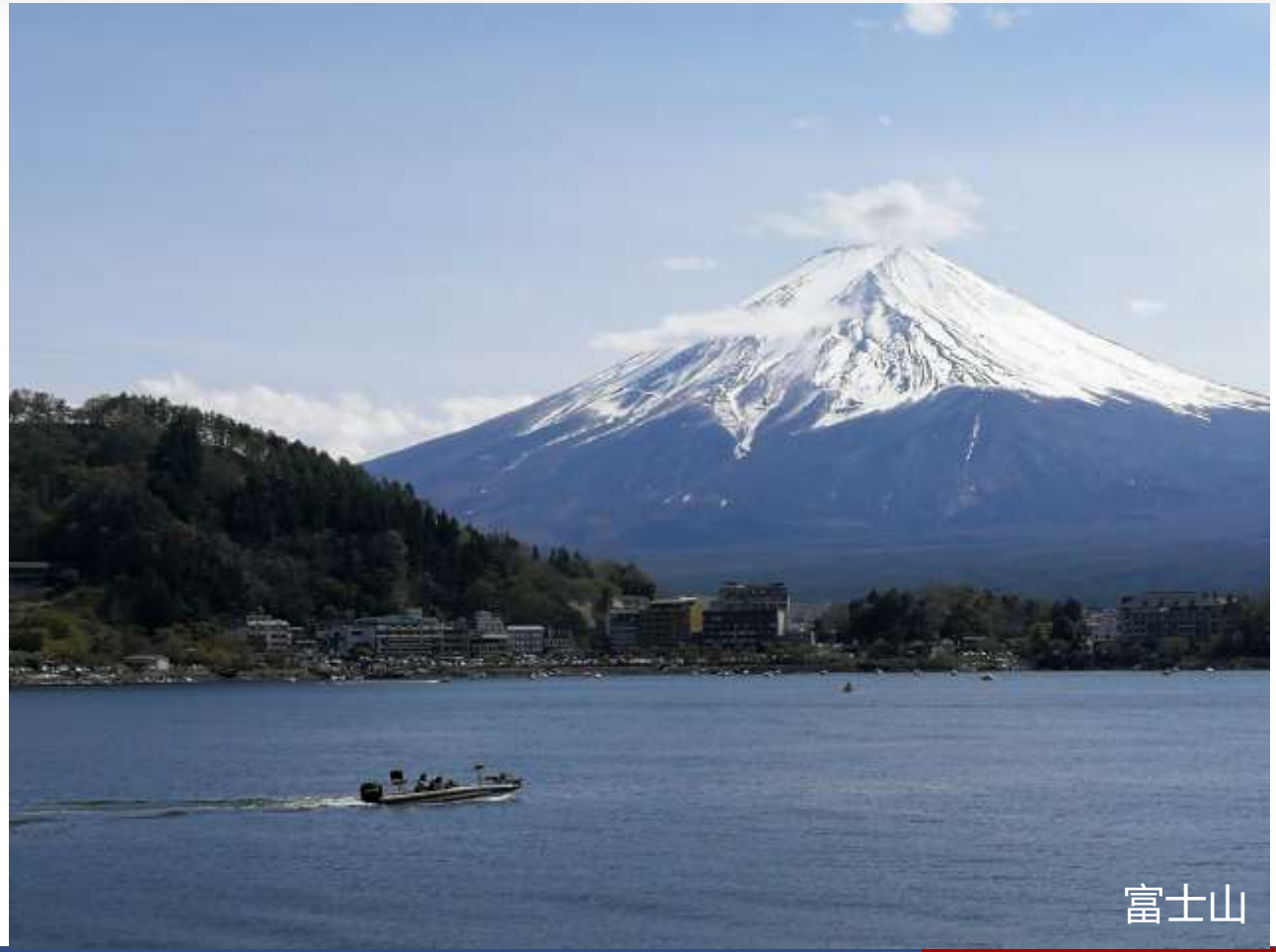
例题3: 中序表达式转换成后序表达式


```
def midToSuffix(s):
    s = s.split()
    stack,result = [],[]
    priority = {"/": 1, "*": 1, "+": 2, "-": 2}
    for x in s:
        if x == "(":
            stack.append(x)
        elif x == ")":
            while stack[-1] != "(":
                result.append(stack.pop())
            stack.pop()
        elif x in "/*+-":
            while len(stack) >= 1 and \
                  stack[-1] != '(' and priority[stack[-1]] <= priority[x]:
                result.append(stack.pop())
            stack.append(x)
        else:
            result.append(x)
```

例题3: 中序表达式转换成后序表达式


```
while stack != []:
    result.append(stack.pop())
return " ".join(result)
```

例题4
中序表达式求值



例题4: 中序表达式求值

要求不能转换成后序表达式再求。

① 维护原子栈stkNum和运算符栈stkOp两个栈

② 碰到原子就加入原子栈

③ 碰到运算符a，如果stkOp顶部运算符x优先级小于a，则x的第二个操作数应该是a运算进行后的结果，此时还没有出现，因此应该将a压入stkOp；如果stkOp顶部运算符优先级不小于a，则取出该运算符，并取出stkNum顶部两个原子进行运算，运算结果压回stkNum。直到stkOp顶部运算符优先级小于a,或stkOp顶部为"("，则将a压入stkOp

④ 扫描结束后，从stkOp不停弹出运算符进行运算，直到stkOp为空，则stkNum中应该只剩下一个元素，即整个表达式的值

⑤ "("压入运算符栈，然后开始一个中序表达式的处理过程。碰到")"视为一个中序表达式扫描结束，要对stkOp进行退栈并运算，直到"("被弹出

例题4: 中序表达式求值


```
def countMid(s):
    s = s.split()
    stkNum,stkOp = [],[] #原子栈、运算符栈
    priority = {"/": 1, "*": 1, "+": 2, "-": 2}
    for x in s:
        if x == "(":
            stkOp.append(x)
        elif x == ")":
            while stkOp[-1] != "(":
                op = stkOp.pop()
                a,b = stkNum.pop(),stkNum.pop()
                stkNum.append(eval(str(b) + op + str(a)))
            stkOp.pop()
        elif x in "/*+-":
            while len(stkOp) >= 1 and \
                  stkOp[-1] != '(' and priority[stkOp[-1]] <= priority[x]:
                op = stkOp.pop()
```

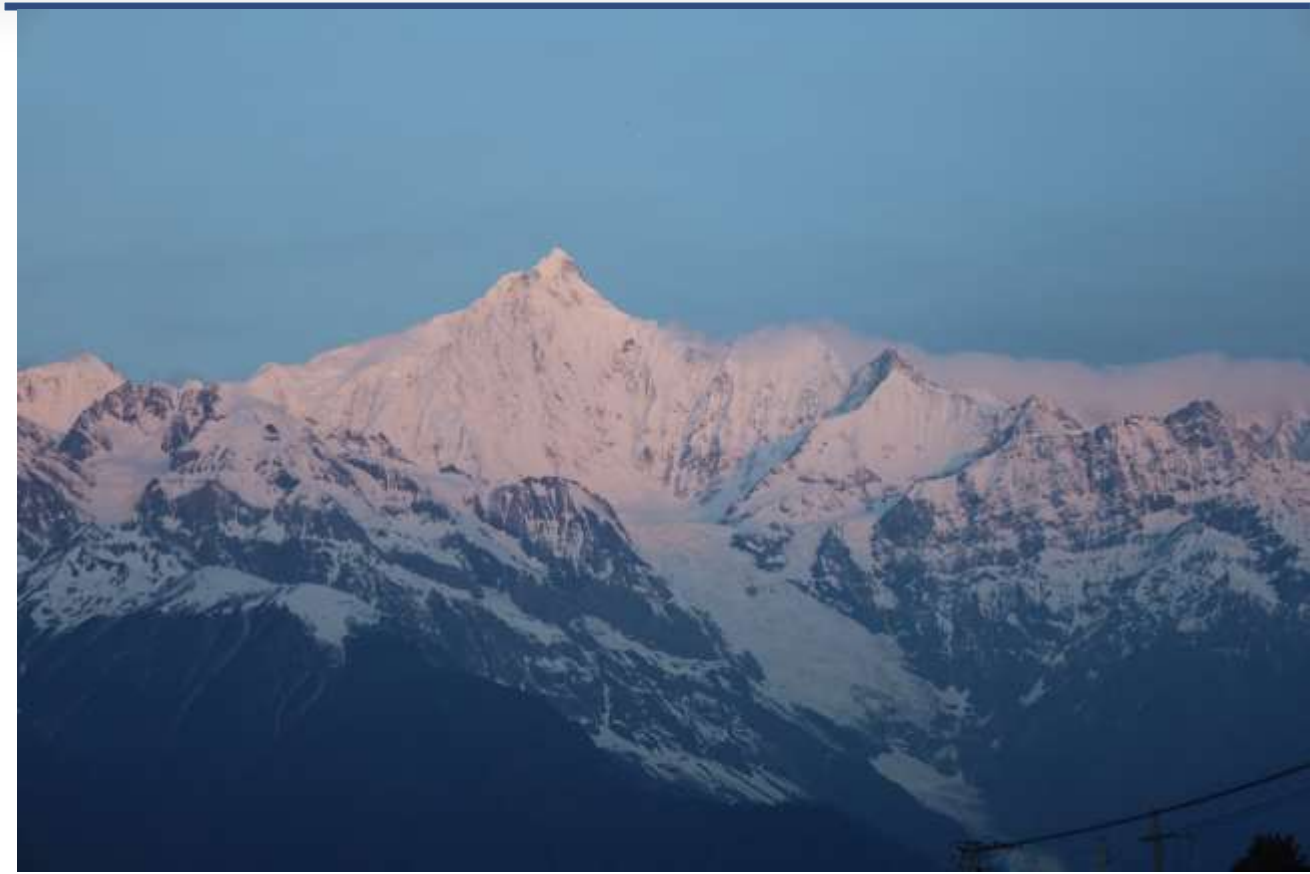
例题4: 中序表达式求值


```
                a,b = stkNum.pop(),stkNum.pop()
                stkNum.append(eval(str(b) + op + str(a)))
            stkOp.append(x)
        else:
            stkNum.append(float(x))
    while len(stkOp) > 0:
        op = stkOp.pop()
        a,b = stkNum.pop(),stkNum.pop()
        stkNum.append(eval(str(b) + op + str(a)))
    return stkNum[0]

print(countMid("3 * ( 4 + 5 ) * ( 6 + 7 ) + ( 5 - 2 ) * 1 + 3"))
```

例题5
合法的出栈序列

★



梅里雪山

例题5: 合法的出栈序列

给定一个由大小写字母和数字构成的，没有重复字符的长度不超过62的字符串x，现在要将该字符串的字符依次压入栈中，然后再全部弹出。
要求左边的字符一定比右边的字符先入栈，出栈顺序无要求。
再给定若干字符串，对每个字符串，判断其是否是可能的x中的字符的出栈序列。

输入：第一行是原始字符串x。后面有若干行(不超过50行)，每行一个字符串。所有字符串长度不超过100。

输出：对除第一行以外的每个字符串，判断其是否是可能的出栈序列。如果是，输出"YES"，否则，输出"NO"

样例输入:

```
abc
abc
bca
cab
```

样例输出

```
YES
YES
NO
```

例题5: 合法的出栈序列

解题思路：

用一个栈，将s1中的字符从左到右依次入栈，入栈过程中，或s1全部入栈后，都可以进行出栈操作，看能否得到s2。

设置一个作用于s2的下标p2，初始值为0。s2[p2]表示下一个应该出栈的字符。每步操作要么将s1的下一个字符入栈，要么就是执行出栈操作。如果栈顶的元素和s2[p2]相同，则必须执行出栈操作，且将p2加1;如果栈为空或栈顶元素和s2[p2]不同，则只能执行将s1的下一个字符入栈的操作。

例题5: 合法的出栈序列


```
def isPopSeq(s1,s2):#判断s2是不是s1经出入栈得到的出栈序列
    stack = []
    if len(s1) != len(s2):
        return False
    else:
        L = len(s1)
        stack.append(s1[0])
        p1,p2 = 1,0
        while p1 < L:
            if len(stack) > 0 and stack[-1] == s2[p2]:
                stack.pop()
                p2 += 1
            else:
                stack.append(s1[p1])
                p1 += 1
        return "".join(stack[::-1]) == s2[p2:]
```

例题5: 合法的出栈序列


```
s1 = input()
while True:
    try:
        s2 = input() #如果输入数据结束，再执行input()会产生异常
    except:
        break  #输入数据结束
    if isPopSeq(s1,s2):
        print("YES")
    else:
        print("NO")
```