数据结构与算法（Python版）

表达式转换（下）

陈斌 北京大学 gischen@pku.edu.cn

通用的中缀转后缀算法

- 我们来讨论下通用的中缀转后缀算法
- 首先我们来看中缀表达式A+B*C，其对应的后缀表达式是ABC*+

  操作数ABC的顺序没有改变。

  操作符的出现顺序，在后缀表达式中反转了

  由于*的优先级比+高，所以后缀表达式中操作符的出现顺序与运算次序一致。

通用的中缀转后缀算法

- 在中缀表达式转换为后缀形式的处理过程中，操作符比操作数要晚输出

  所以在扫描到对应的第二个操作数之前，需要把操作符先保存起来

- 而这些暂存的操作符，由于优先级的规则，还有可能要**反转**次序输出。

  在A+B*C中，+虽然先出现，但优先级比后面这个*要低，所以它要等*处理完后，才能再处理。

- 这种**反转**特性，使得我们考虑用**栈**来保存暂时**未处理**的**操作符**

通用的中缀转后缀算法

- 再看看(A+B)*C，对应的后缀形式是AB+C*

  这里+的输出比*要早，主要是因为括号使得+的优先级提升，高于括号之外的*

- 回顾上节的“**全括号**”表达式，后缀表达式中操作符应该出现在左括号对应的右括号位置

  所以遇到左括号，要标记下，其后出现的操作符优先级提升了，一旦扫描到对应的右括号，就可以马上输出这个操作符

通用的中缀转后缀算法

- 总结下，在从左到右扫描逐个字符扫描中缀表达式的过程中，采用一个**栈**来暂存未处理的操作符
- 这样，**栈顶**的操作符就是**最近**暂存进去的，当遇到一个新的操作符，就需要跟栈顶的操作符比较下优先级，再行处理。

通用的中缀转后缀算法：流程

- 后面的算法描述中，约定中缀表达式是由空格隔开的一系列单词（token）构成，
  操作符单词包括*/+-()
  而操作数单词则是单字母标识符A、B、C等。
- 首先，创建空栈opstack用于**暂存操作符**，空表postfixList用于保存后缀表达式
- 将中缀表达式转换为单词（token）列表

```
A+B*C =split=> ['A', '+', 'B', '*', 'C']
```

通用的中缀转后缀算法：流程

❖ 从左到右扫描中缀表达式单词列表

如果单词是操作数，则直接添加到后缀表达式列表的末尾

如果单词是左括号“(”，则压入opstack栈顶

如果单词是右括号“)”，则反复弹出opstack栈顶操作符，加入到输出列表末尾，直到碰到左括号

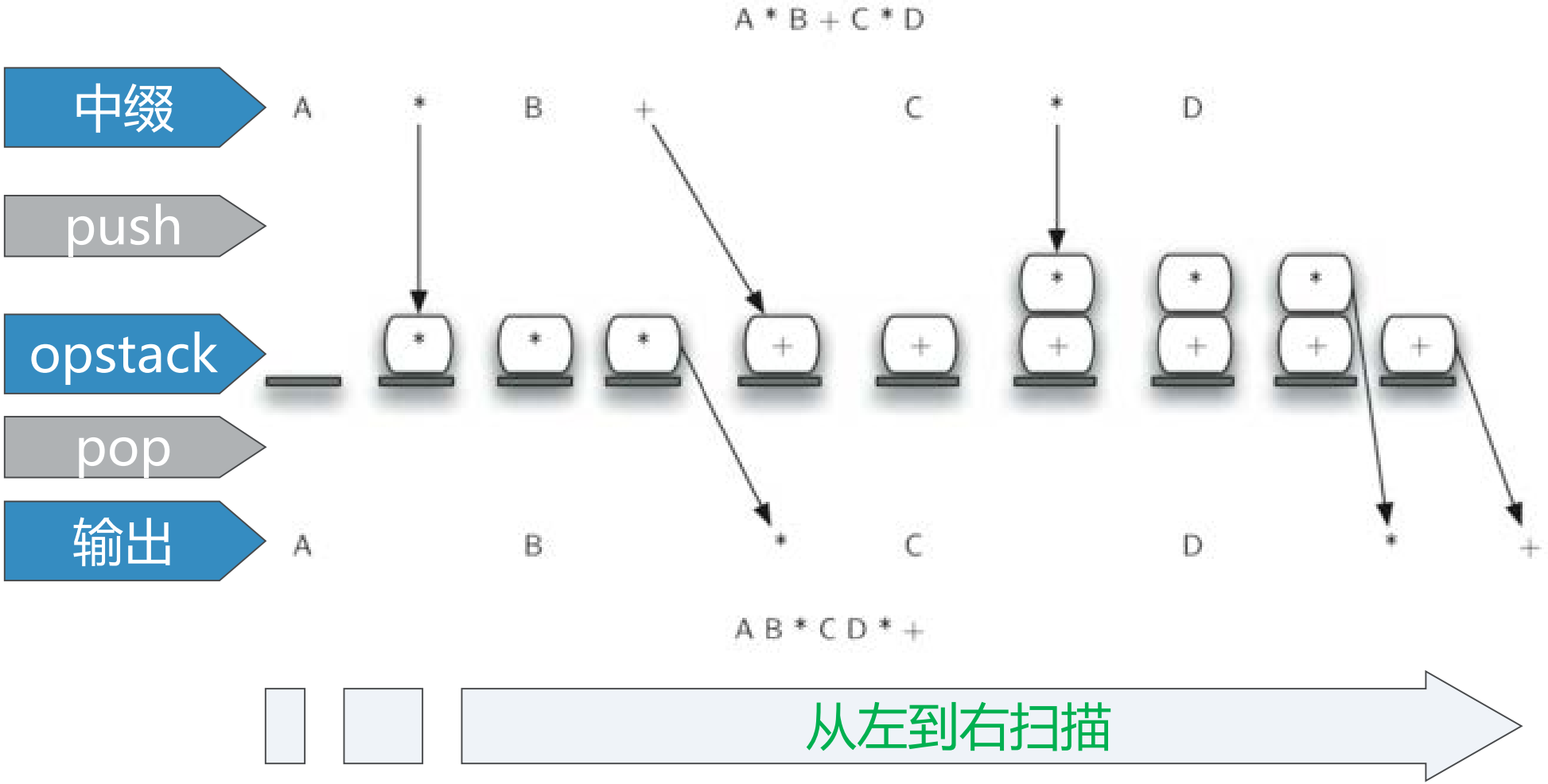
如果单词是操作符“*/+-”，则压入opstack栈顶

- 但在压入之前，要比较其与栈顶操作符的优先级
- 如果栈顶的高于或等于它，就要反复弹出栈顶操作符，加入到输出列表末尾
- 直到栈顶的操作符优先级低于它

通用的中缀转后缀算法：流程

- 中缀表达式单词列表扫描结束后，把opstack栈中的所有剩余操作符**依次弹出**，添加到输出列表**末尾**
- 把输出列表再用join方法合并成后缀表达式字符串，算法结束。

通用的中缀转后缀算法：实例



代码


```
from pythonds.basic.stack import Stack

def infixToPostfix(infixexpr):
    prec = {}
    prec["*"] = 3
    prec["/"] = 3
    prec["+"] = 2
    prec["-"] = 2
    prec["("] = 1
    opStack = Stack()
    postfixList = []
    tokenList = infixexpr.split()

    for token in tokenList:
        if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" or token in "0123456789":
            postfixList.append(token)
        elif token == '(':
            opStack.push(token)
        elif token == ')':
            topToken = opStack.pop()
            while topToken != '(':
                postfixList.append(topToken)
                topToken = opStack.pop()
        else:
            while (not opStack.isEmpty()) and \
               (prec[opStack.peek()] >= prec[token]):
                  postfixList.append(opStack.pop())
            opStack.push(token)

    while not opStack.isEmpty():
        postfixList.append(opStack.pop())
    return " ".join(postfixList)
```

记录操作符优先级

解析表达式到单词列表

操作数

(

)

操作符

操作符

合成后缀表达式字符串

通用的中缀转后缀算法