**西安电子科技大学**

**信息存储与检索 课程实验报告**

**实验名称 上机实验报告4**

经济与管理学院 2106011 班

成 绩

姓名 赵红玉 学号 21069100225

同作者 无

实验日期 2024 年 05 月 20 日

|  |
| --- |
| 指导教师评语：  实验内容：  实验效果：  实验报告：  考勤情况：  其他情况：  指导教师：  年 月 日 |

# 一、实验目的：

* 加深理解逆波兰、准波兰转换的基本原理。
* 熟悉检索式的逆波兰、准波兰转换过程。
* 熟练运用自己熟悉的软件工具或编程语言，实现布尔提问式的逆波兰、波兰转换方法并对运行结果进行分析。

# **二、机器环境（硬件、软件配置，编程语言）**

操作系统：macOS

软件配置：Visual Studio Code

编程语言：Python

# **三、算法或原理：**

 在布尔检索系统中，为了实现用户的检索需求，首先需要对用户输入的布尔逻辑检索式进行必要的加工和变换处理。布尔检索是以倒排文档为基础。常用的变换处理方法有逆波兰变换法、准波兰变换法等。

## 3.1. 逆波兰变换

  逆波兰变换法又称福岛方法，该方法使用最早、影响也最大。

  处理要点：先将布尔检索式（即提问式）转换成逆波兰表示形式，然后将逆波兰表示形式翻译成一组检索指令。（即两步处理）（后续的检索处理过程，就是执行一组检索指令）

（1）逆波兰表达式

  例如：A加B之和，再乘以C。

  中缀表示法：(A+B)\* C（其中，运算符在两个运算项的中间）

  运算规则类似于算术运算，一般是从左到右进行，遇到括号时优先括号内的运算。但是其括号无法去掉，书写并非最简洁 。

  注：不同的算符规定了不同的运算优先次序（即算符优先级）

 后来，波兰的一位学者卢卡西维兹提出两种不用括号的表示法，称为波兰表示法。

  例如：对 (A+B) \* C ，有两种表示。

前缀表示法：算符在前，为 \*+ABC。又称正波兰表示法。

后缀表示法：算符在后，为 AB+C\*。又称逆波兰表示法。

三种表达式例示比较：

分析前述三种表示法，有以下共同点：

运算符个数保持不变；

运算项个数不变，且出现的先后次序不变。

又有以下不同点：

正波兰和逆波兰表示法都是无括号的表示法，结构上比中缀表示法更简洁；

逆波兰表示法又比正波兰表示法运算处理更简单方便，符合常规的执行顺序。

这里，给出对表达式的逆波兰（后缀）表示法的运算规则：

从左向右扫描后缀表示形式，当遇到运算项时，就把它保存起来；当遇到运算符时，就取出其前面紧接着的两个运算项依次进行运算处理，并把其结果当作一个新的运算项保存起来；再继续向右检查及扫描其它符号，直至结束。

（2）布尔检索式的逆波兰变换

   为了实现提问式的逆波兰变换，首先需要在计算机内存设置以下3个工作区域 ：

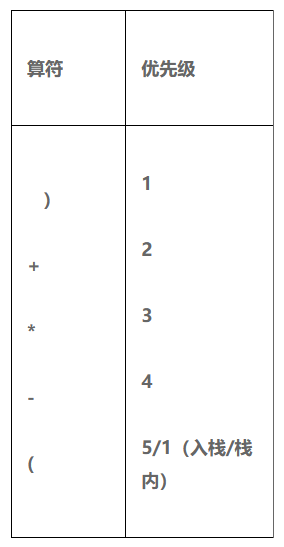
  逆波兰输出区；算符栈；检索词表区。

逆波兰输出区——存放经变换处理后提问式的逆波兰形式，即算项和算符形成的序列；

算符栈——一个“后进先出”的表区，主要用来暂存运算符，并重新排列运算符，以便确定运算顺序。（临时使用的堆栈）

检索词表区——将提问式中的检索词列表，并定义对应地址编号。在逆波兰输出区中，使用地址编号来代替作为运算项的检索词。

逆波兰变换过程中对运算符的处理，是通过各自的优先级来控制的。算符优先级高低分别由数值大小表示。

算符优先级列表

提问式示例:(A+B)\*(C+D)．（归结为两类：检索词、算符）

提问式的逆波兰变换处理规则：

  从左向右逐个扫描提问式的字符，根据不同情形予以相应的处理。

①如果是检索词（运算项），则将其置入检索词表区中，并将相应的词表地址送入逆波兰输出区中。

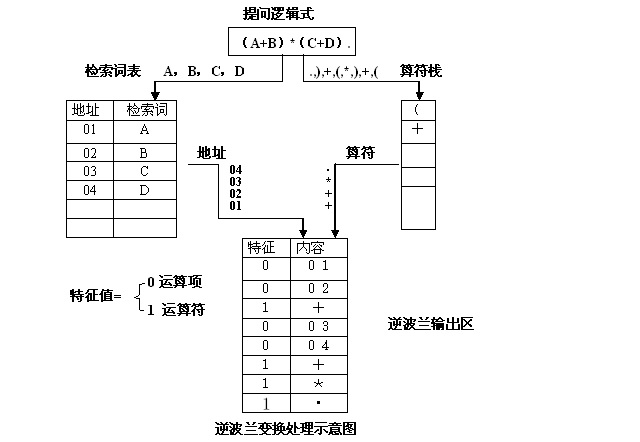
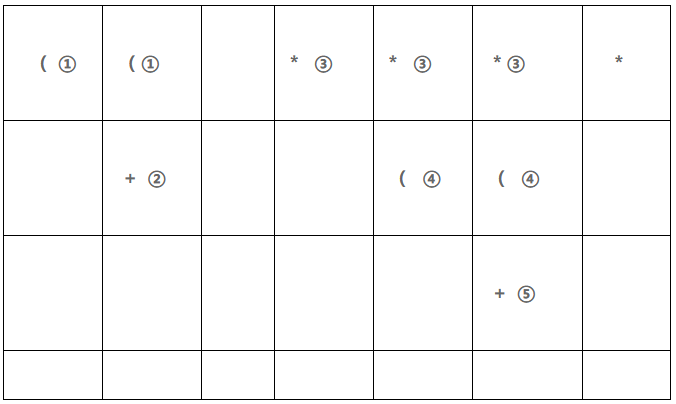
②如果是运算符+，\*，-，则将它与算符栈的栈顶算符按照优先级进行比较：若它的优先级比栈顶算符的优先级高，则把它压入算符栈内；若相等或低于栈顶算符的优先级，则退出（或弹出）栈顶算符，转送入逆波兰输出区，然后再与新的栈顶算符比较优先级，以此类推。

③如果是左括号“（”，由于其入栈优先级为5，即优先级最高，应无条件入栈。进栈后其优先级变为最低（栈内优先级），即变为1。

④如果是右括号“）”，则将与之配对的“（”之间的所有算符从算符栈中按“后进先出”次序依次弹出，移入逆波兰输出区，并清除这对括号。

⑤ 若为检索式结束标志“．”，则将留在算符栈中的算符按“后进先出”次序全部移入逆波兰输出区中，最后将“．”亦置入其中。

注意：三个工作区域的初始状态均为空；逆波兰区的特征值：0表示运算项，1表示算符。



逆波兰变换处理示意图大致反映出变换过程中各个表区的状态。

下表示出提问式(A+B)\*(C+D).

变换过程中算符栈变化情况:

(其中， ①、 ②、③等表示进栈顺序，每列最上方为栈底)

（3）检索指令表的生成

当逆波兰变换结果（即逆波兰形式）形成之后，下一步需要生成一组检索指令。

这一过程需设置3个表区：

  检索指令表、工作区、工作区管理表。

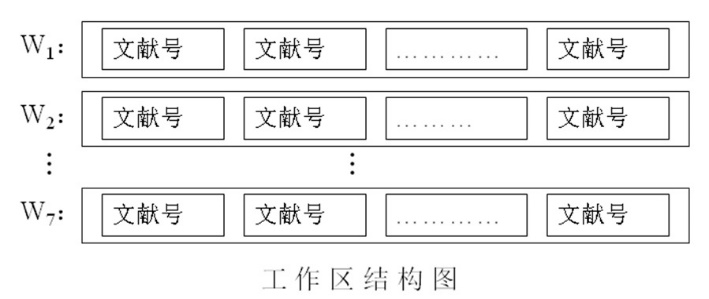
  检索指令表：放置由逆波兰输出区转换而来的一条条检索指令。

  工作区：存放包含某个检索词的文献号，和运算的结果（含中间结果、最终结果）。通常放在内存，一般设7个工作区。

  一个工作区的单元数即所存放的文献号个数（有时也称为工作区长度），其大小视检索系统的规模而定。

  工作区管理表：管理、控制工作区的使用。（结构为7行、2列）

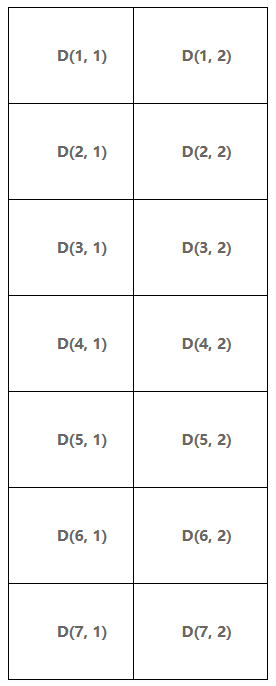
工作区的区号为1~7。设置原则：工作区单元数不小于倒排档中任一检索词之后可能出现的文献号个数（考虑最大个数，定长设置）。



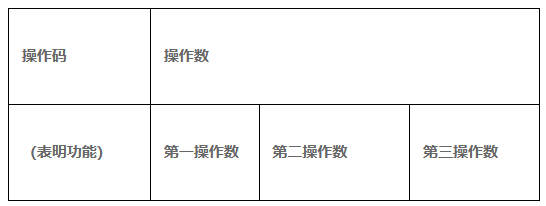
工作区管理表的解释：

  D(j，1)，即第1列。 D(j，1)=1，表示第j个工作区被占用；D(j，1)=0，表示第j个工作区未被占用（即空闲）。  其中， j取值1~7。

  D(j，2)，即第2列，存放第j个工作区的运算次序值（初值为0、加1递增，相当于一个统一的计数器）。



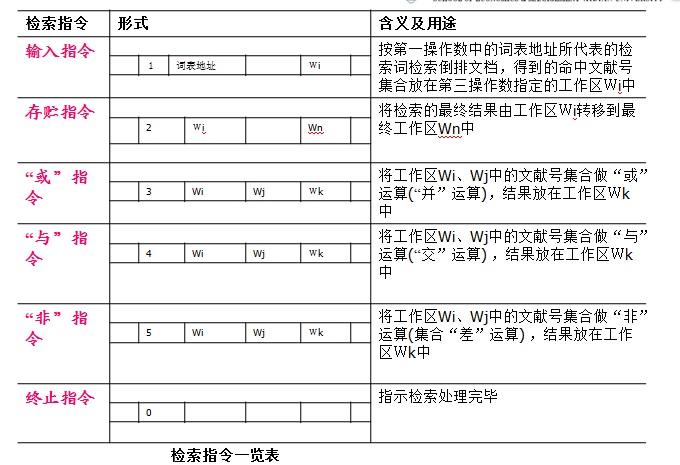
检索指令表用来存放若干条检索指令。按检索指令的功能，共有6种检索指令(操作)。每条检索指令的构成形式如下所示：



6种检索指令:

输入指令、存贮指令、“或”指令、“与”指令、“非”指令、终止指令。

1、2、3 、4、5、0。

下面讨论如何生成检索指令（表）。

注意：初态时各表区清空/清0；工作区号是由小到大使用；严格区分检索指令表中的地址编号（01、02等）和工作区号码（1-7）！

由逆波兰输出区内容生成检索指令表，遵循以下处理顺序和规则:

从逆波兰输出区的第一行开始逐行扫描，同时按以下顺序给出检索指令。

① 遇到检索词表地址(运算项)：由工作区管理表中找出可用（即空闲）的工作区(号)，将词表地址作为第一操作数，某个工作区(号)作为第三操作数，做出“输入指令”，置入检索指令表中。（置运算次序值：+1递增）

② 遇到算符：由工作区管理表中取出两个运算次序最高的(即最近被占用的)工作区号码，依其占用的先后分别作为第一、二操作数，再在工作区管理表中找出一个“未被占用”的工作区的号码作为第三操作数，按算符性质做出“或”、“与”、“非”检索指令之一，将指令下推进检索指令表。

需要注意的是：当实际执行完该运算指令后，由第三操作数所确定的工作区仍“被占用”，而由第一、二操作数所确定的工作区则要获得释放，即改为“未被占用”，这一点要及时在工作区管理表中反映出来。

③ 当扫描到结束标志“．”时，说明逆波兰输出区已扫描完毕，且工作区管理表中必定显示只有一个工作区“被占用”。这时，将此工作区的号码作为第一操作数，最终工作区的号码作为第三操作数，做出“存贮指令”，送入检索指令表中。

④ 当形成“存贮指令”之后，做出“终止指令”，下推进检索指令表。

后面我们以提问式（A+B）\*（C+D）.为例，说明将逆波兰输出区内容如何变换为检索指令表。

当生成检索指令表之后，将按检索指令表中的顺序逐条执行检索指令，即可完成布尔检索过程。

## **3.2．检索处理过程**

使用表区：检索指令表、检索词表、工作区。

按照检索指令表中的指令顺序，逐条执行检索指令：

①遇输入指令，由第一操作数所示地址从检索词表内取出检索词，进而查询倒排档，取出包含该检索词的文献号集合，存放到第三操作数所指示的工作区。

②遇运算指令，将第一、二操作数对应的文献号集合（放在两个工作区）进行规定的集合运算，得到新的文献号集合并存放到第三操作数所指示的工作区。

③遇存贮指令，将文献号集合转贮到最终工作区。

倒排文档检索举例

简化的文献记录集合（包括6篇文献）：

001专家系统在情报检索中的应用(标引词:专家系统;智能检索系统)

002一种新的倒排档溢出处理算法(标引词:倒排档;溢出处理)

003情报检索专家系统的特点与发展(标引词:专家系统;智能检索系统)

004提问式中的位置算符(标引词:提问逻辑式;位置算符)

005提问式准波兰变换算法的研究(标引词:提问逻辑式;准波兰变换)

006智能检索系统的设计与开发(标引词:智能检索系统)

  以“”为记录分隔符，上面6个文献记录的顺序文档（主文档）的机器存放形式如下：

001专家系统在情报检索中的应用002一种新的倒排档溢出处理算法003情报检索专家系统的特点与发展004提问式中的位置算符005提问式准波兰变换算法的研究006智能检索系统的设计与开发

相应的主题词倒排档的形式如下：

倒排档  002

提问逻辑式 004，005

位置算符 004

溢出处理 002

智能检索系统 001，003，006

专家系统 001，003

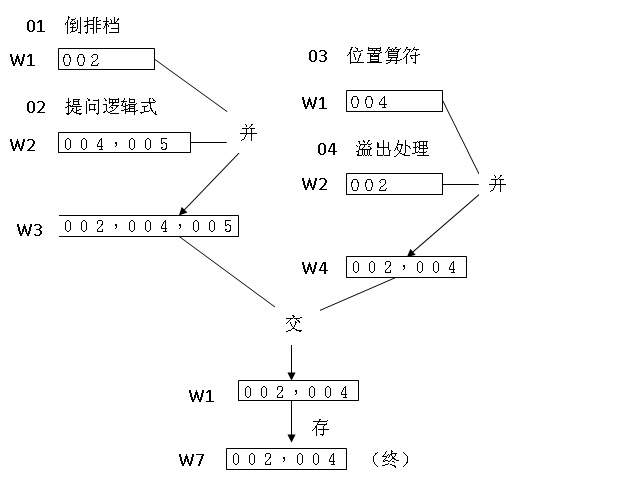
准波兰变换 005

对如下布尔检索式进行逆波兰变换：

（倒排档＋提问逻辑式）\*（位置算符＋溢出处理）.



检索处理过程示意（即执行检索指令序列）：



可以看出，对布尔检索式

（倒排档＋提问逻辑式）\*（位置算符＋溢出处理）的处理结果为：集合{002，004}

 即该布尔检索式对应的命中文献号是：002，004

 于是，可由命中文献号查找顺排文档（主文档）：

 002 一种新的倒排档溢出处理算法

 004 提问式中的位置算符

## **3.3 准波兰变换法**

针对福岛法存在的逆波兰变换对系统内存空间要求过高的问题，我国学者提出了一种改进的方法——准波兰变换法。

思路：利用二叉树表示检索式，并对二叉树进行一定的调整，再通过二叉树的操作实现检索式的变换。

目的：检索处理所需内存工作区个数最少。

工具：采用表达式二叉树。

表达式二叉树：将一个表达式用一棵二叉树表示，其中运算项作为树的叶子结点，运算符作为树或子树的根结点。

如二个检索式： ① (A+B)\*(C+D).   ② A+B\*(C+D).

方法：将检索式表示为二叉树；调整二叉树的结构使大枝在左、小枝在右；对调整后的二叉树进行后序遍历。

分析A+B\*(C+D)对应的二叉树，其后序遍历结果：ABCD+\*+。   此时按照福岛法的变换处理过程，需要使用5个工作区。这里依据的是逆波兰表达式的运算规则，即遇到运算项必须保存下来（用工作区）。

而如果先处理大的分枝，后处理小的分枝，即调整树结构，使其后序遍历结果变成： CD+B\*A+ ，此时就只需要使用3个工作区。

通过分析对应二叉树，可以发现：

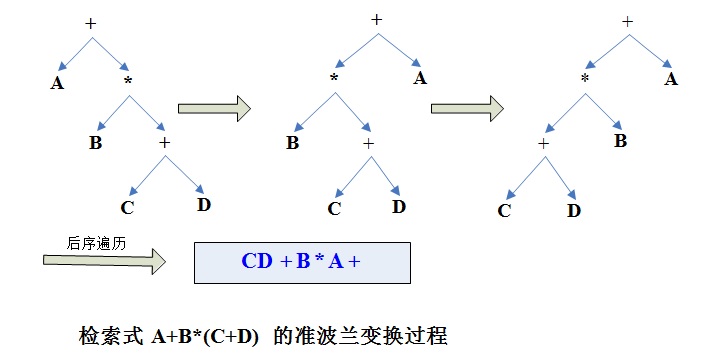
当树的两棵子树不对称时，先处理较大分支，占用工作区较少。

于是给出对检索式的准波兰变换规则:

(1)创建检索式的二叉树表示形式；

(2)比较二叉树的每一层次上的左、右子树是否对称。若不对称，则把大的分枝保留或调整到左边，小的分枝保留或调整到右边，直到全部节点的左右子树窦处理完毕。

(3)后序遍历该二叉树，结点输出序列就是检索式的准波兰形式。



准波兰变换法的分析：

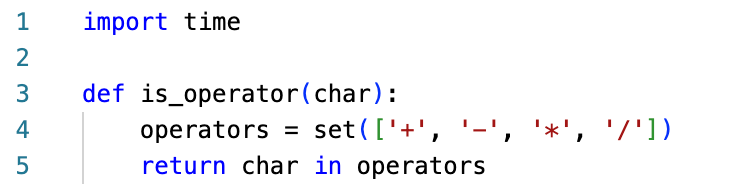
（1）该方法可以将内存工作区从7个降低至5个，其算法比逆波兰变换稍微复杂一些。统计认为，提问式中的检索词个数在20个以内，都是适用的。

（2）对于已采用逆波兰变换法的检索系统，只要增加一个重组模块，即可获得理想的准波兰变换结果。

## 四、程序、代码内容

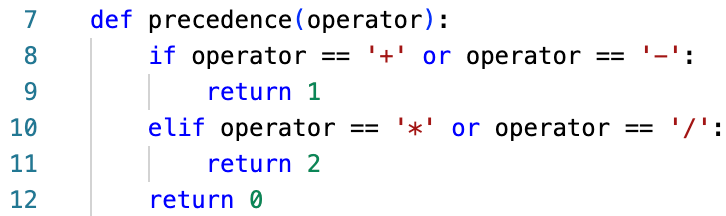
## **4.1 逆波兰转换**

**（1）代码解释**

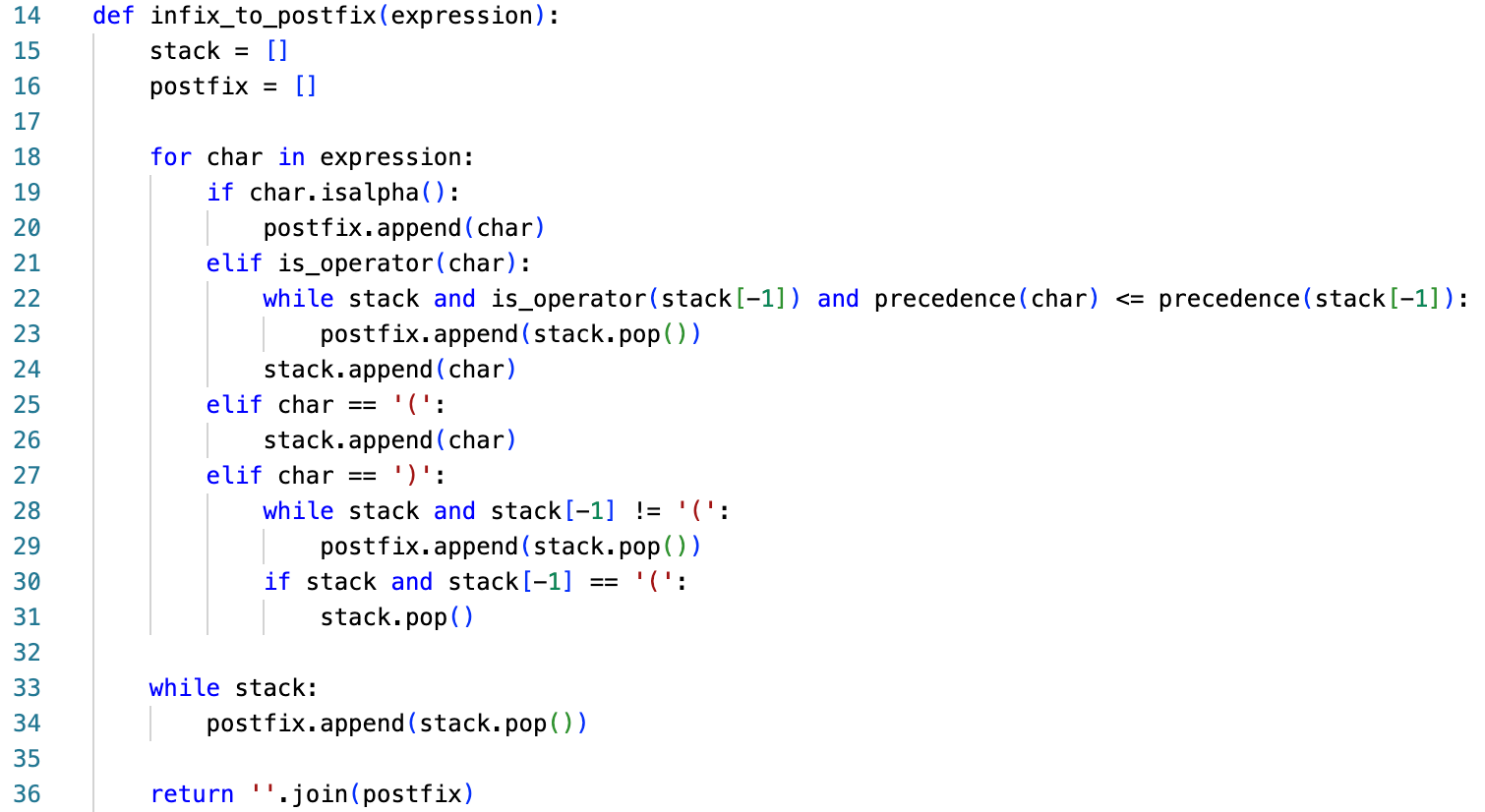


引入计时器。

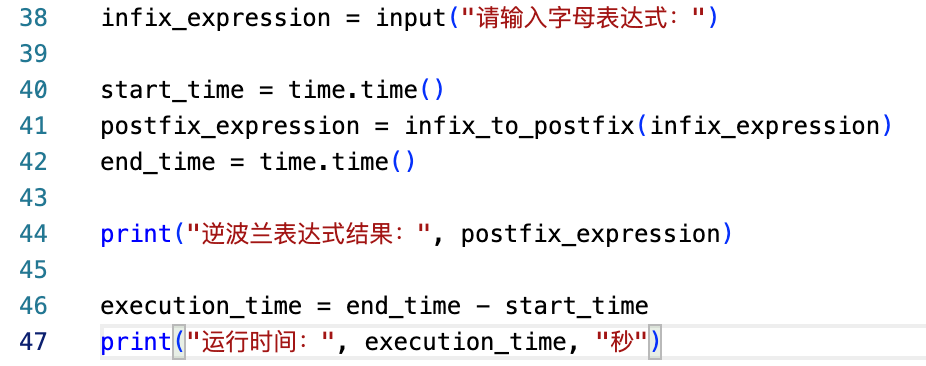
is\_operator(char)函数用于检查字符是否为运算符。它将运算符存储在一个集合中，并检查给定的字符是否是集合中的元素。



precedence(operator)函数用于确定运算符的优先级。在这个示例中，将'+', '-', '', '/'这四个运算符分为两个级别：级别1表示'+', '-'，级别2表示'', '/'，其他运算符的优先级为0。



infix\_to\_postfix(expression)函数用于将中缀表达式转换为后缀表达式。它遍历表达式中的每个字符，如果是字母，则直接添加到后缀表达式列表中；如果是运算符，则将其与栈顶的运算符进行比较，如果栈顶运算符的优先级大于等于当前运算符，则将栈顶运算符弹出并添加到后缀表达式列表中，重复此过程直到栈为空或者栈顶运算符的优先级小于当前运算符。然后将当前运算符压入栈中。如果遇到'('，则将其压入栈中；如果遇到')'，则将栈中的运算符依次弹出并添加到后缀表达式列表中，直到遇到'('为止。最后，将栈中剩余的运算符依次弹出并添加到后缀表达式列表中。



用户输入一个字母表达式，使用input()函数获取用户输入的表达式。

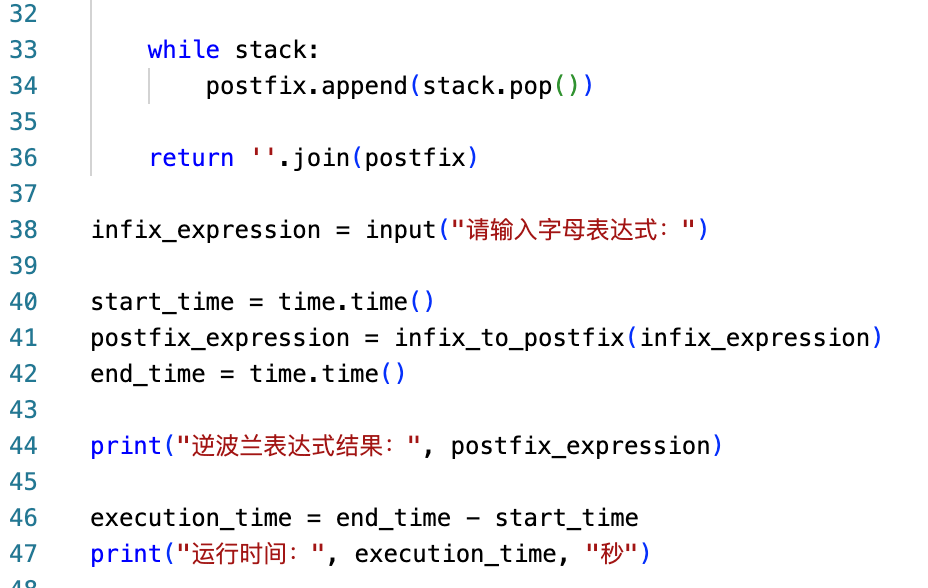
调用infix\_to\_postfix(infix\_expression)将中缀表达式转换为后缀表达式。

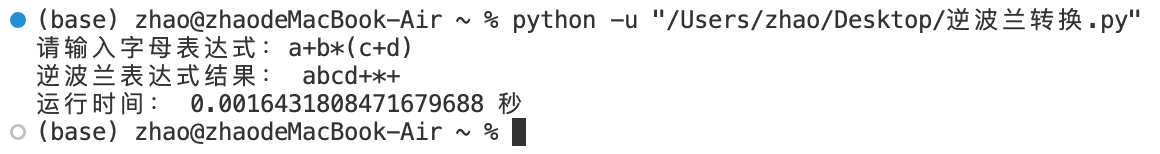
使用time.time()函数获取程序开始执行的时间，将其保存在start\_time变量中。

使用time.time()函数获取程序执行结束的时间，将其保存在end\_time变量中。

计算运行时间，将end\_time减去start\_time，得到执行时间，保存在execution\_time变量中。

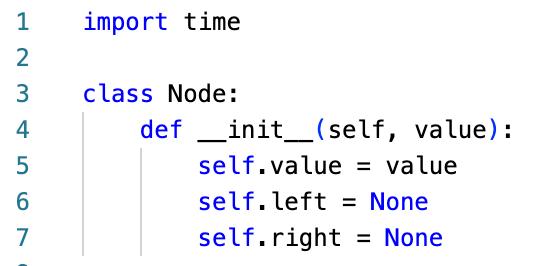
1. 整体代码



（3）运行结果

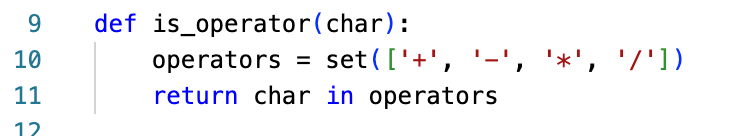
## **4.2 准波兰转换**

1. **代码解释**

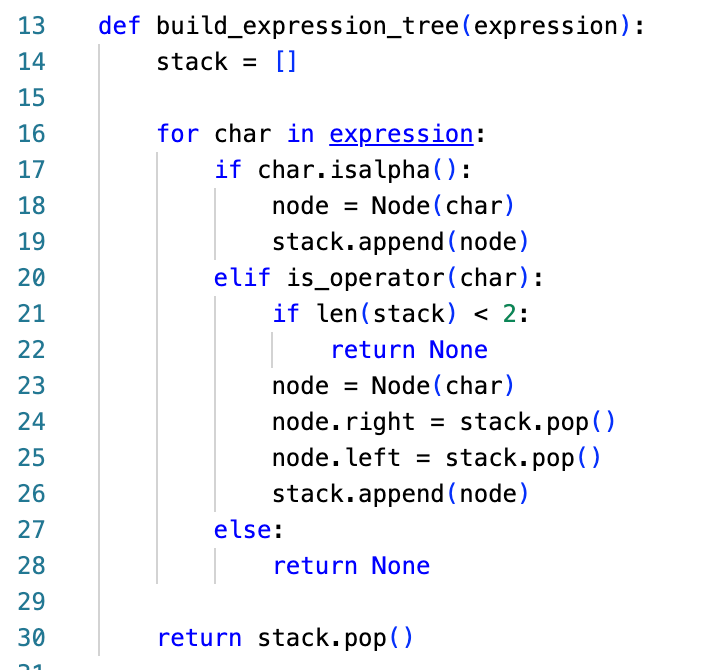


引入计时器。

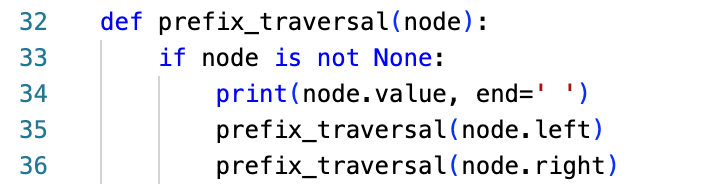
class Node：这是一个简单的节点类，表示二叉树的节点。每个节点包含一个值、左子树和右子树。



is\_operator(char)：这个函数用于检查给定的字符是否是运算符（+、-、\*、/）。



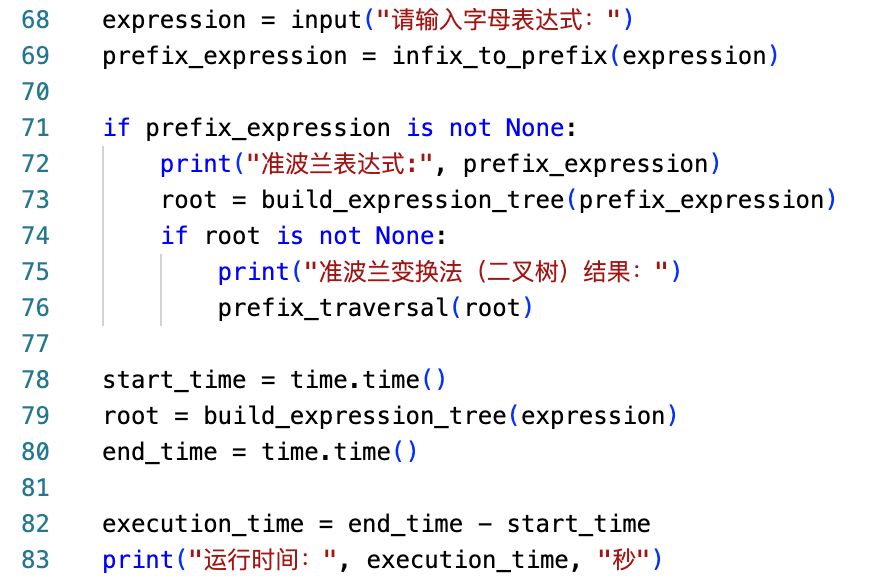
build\_expression\_tree(expression)：这个函数接收一个后缀表达式，并使用准波兰变换法构建相应的二叉树。它遍历后缀表达式中的每个字符，如果是字母，则创建一个节点并将其推入栈中。如果是运算符，则从栈中弹出两个节点，将它们作为右子节点和左子节点设置给新的节点，并将新节点推入栈中。最后，返回栈中剩余的节点，即根节点。



prefix\_traversal(node)：这个函数通过前序遍历方式打印出二叉树的节点值。它首先输出当前节点的值，然后递归地遍历左子树和右子树。



infix\_to\_prefix(expression)：这个函数将中缀表达式转换为前缀表达式。它使用两个栈，一个用于操作符，一个用于输出结果。它遍历中缀表达式中的每个字符，如果是字母，则直接添加到输出结果中。如果是左括号，则将其推入操作符栈中。如果是右括号，则将操作符栈中的操作符弹出并添加到输出结果中，直到遇到左括号。如果是操作符，则将操作符栈中的操作符弹出并添加到输出结果中，直到遇到优先级更低的操作符或栈为空。最后，将输出结果转换为字符串并返回。



prefix\_expression = infix\_to\_prefix(expression)：这行代码将中缀表达式转换为前缀表达式。

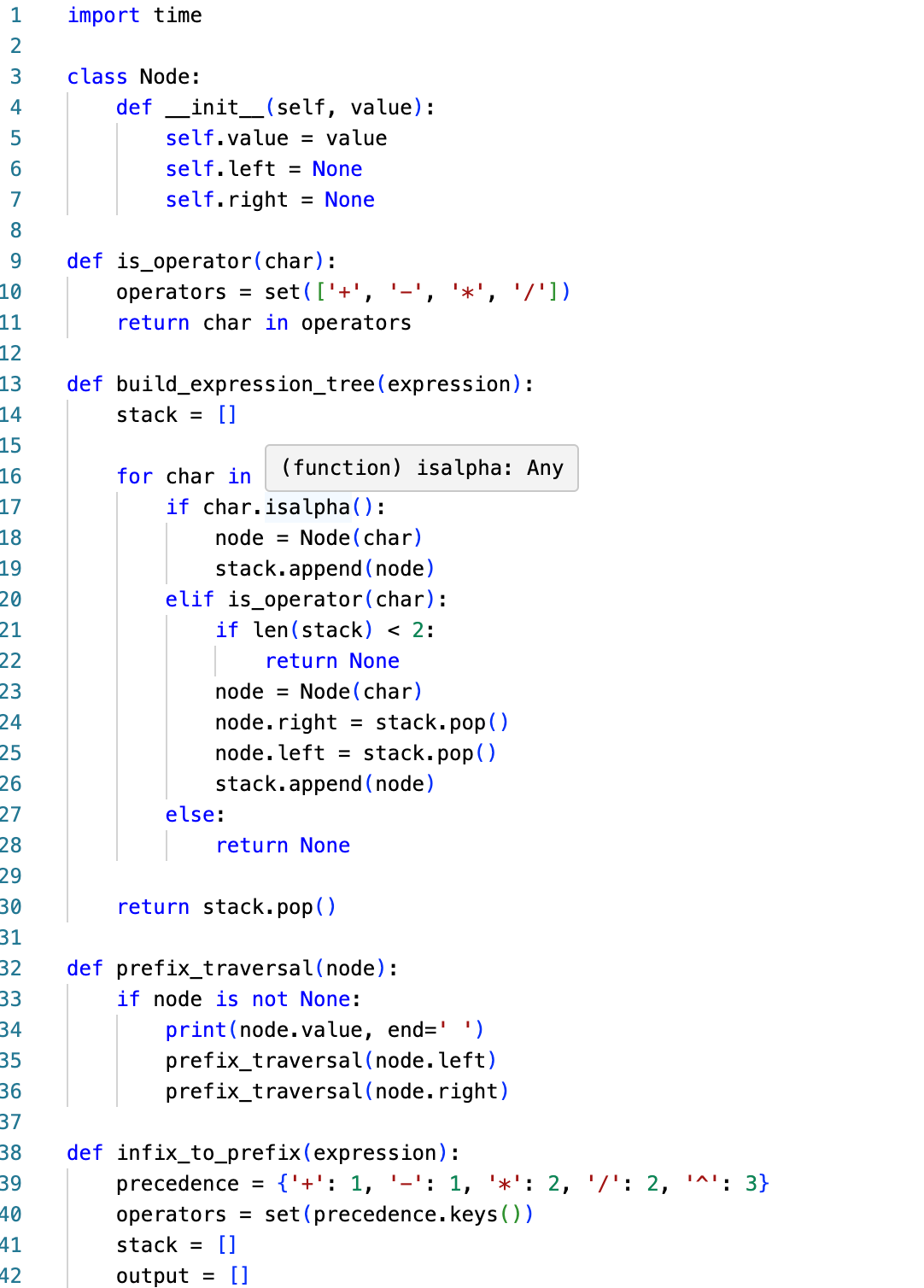
root = build\_expression\_tree(prefix\_expression)：使用前缀表达式构建二叉树，并将根节点赋值给root变量。

prefix\_traversal(root)：通过前序遍历方式打印二叉树的节点值。

root = build\_expression\_tree(expression)：使用原始的中缀表达式构建二叉树，并将根节点赋值给root变量。

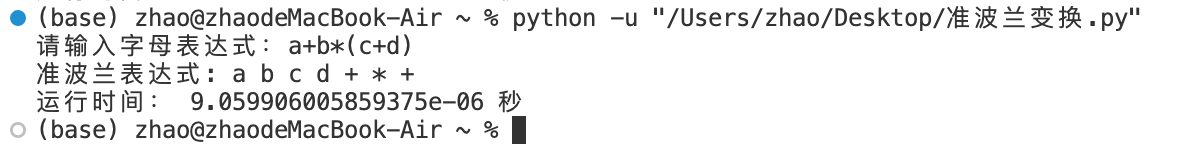
start\_time = time.time()：记录开始时间。end\_time = time.time()：记录结束时间。execution\_time = end\_time - start\_time：计算运行时间。

（2）整体代码





（3）运行结果



# 结论及分析：

结论：

1. 逆波兰转换法的有效性。实验验证了逆波兰转换法（福岛方法）在布尔检索中的应用是有效的。通过将中缀表达式转换为逆波兰表达式，可以简化检索指令的生成和执行过程。

2. 准波兰变换法的优势。实验展示了准波兰变换法相较于逆波兰变换法在内存使用上的优势。通过调整二叉树结构，准波兰变换法能够减少所需的内存工作区数量，从而在处理复杂的布尔检索式时更加高效。

3. 算法实现的可行性。实验中实现的Python代码表明，逆波兰和准波兰转换算法是可行的，并且可以成功地应用于布尔检索式的转换。

4. 时间效率。实验通过计时器记录了程序的运行时间，显示了算法执行的效率。对比明显得出准波兰变换法快于逆波兰变换法。

分析：

1. 逆波兰变换法。

通过实验，了解到逆波兰变换法通过消除括号并简化表达式结构，使得检索指令的生成变得更加直接和简便。实验中的逆波兰变换处理规则和过程，展示了如何有效地将中缀表达式转换为逆波兰表达式。

2. 准波兰变换法。

准波兰变换法利用二叉树的结构来优化检索式的处理，通过调整树结构，使得大的分支先被处理，从而减少了所需的工作区数量。实验中提出的准波兰变换规则，展示了如何构建二叉树，调整其结构，并进行后序遍历以生成准波兰形式的检索式。

3. 算法实现：

实验中的代码实现了中缀表达式到逆波兰表达式的转换，并通过构建和遍历二叉树来实现准波兰变换。代码中的函数设计清晰，易于理解，展示了算法实现的具体步骤。

4. 性能考量：

实验通过计时器对算法的运行时间进行了测量，体现了准波兰变换法的性能优于逆波兰变换法。对于更大规模的检索式，准波兰变换法由于其内存优势，可能提供更好的性能。

5. 实用性：

逆波兰和准波兰变换法在信息检索领域的应用表明，这些方法不仅在理论上是有意义的，而且在实际的检索系统中也是实用的。