**实验1：中缀与正则表达式解析**

1. **实验环境**

Windows/Linux，Python2.7

1. **预备知识**
2. **中缀与后缀表达式**

中缀表达式是一个通用的算术或逻辑公式表示方法， 操作符是以中缀形式处于[操作数](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E6%95%B0/7658270" \t "https://baike.baidu.com/item/%E4%B8%AD%E7%BC%80%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)的中间（例：3 + 4），中缀表达式是人们常用的算术表示方法。

一个表达式E的后缀形式可以如下定义：

1. 如果E是一个变量或常量，则E的后缀式是E本身。
2. 如果E是E1 op E2形式的表达式，这里op是任何二元操作符，则E的后缀式为E1'E2' op，这里E1'和E2'分别为E1和E2的后缀式。
3. 如果E是（E1）形式的表达式，则E1的后缀式就是E的后缀式。

如：我们平时写a+b，这是中缀表达式，写成后缀表达式就是：ab+

**2、中缀转后缀表达式算法**

将一个中缀表达式转换为后缀表达式的一般算法是：

首先需要分配2个栈，一个作为临时存储运算符的栈S1（含一个结束符号），一个作为存放结果（后缀表达式）的栈S2（空栈），S1栈可先放入优先级最低的运算符#，注意，中缀式应以此最低优先级的运算符结束。可指定其他字符，不一定非#不可。从中缀式的左端开始取字符，逐序进行如下步骤：

|  |
| --- |
| 1. 若取出的字符是操作数，则分析出完整的运算数，该操作数直接送入S2栈。 2. 若取出的字符是运算符，则将该运算符与S1栈栈顶元素比较，如果该运算符(不包括括号运算符)优先级高于S1栈栈顶运算符（包括左括号）优先级，则将该运算符进S1栈，否则，将S1栈的栈顶运算符弹出，送入S2栈中，直至S1栈栈顶运算符（包括左括号）低于（不包括等于）该运算符优先级时停止弹出运算符，最后将该运算符送入S1栈。 3. 若取出的字符是“（”，则直接送入S1栈顶。 4. 若取出的字符是“）”，则将距离S1栈栈顶最近的“（”之间的运算符，逐个出栈，依次送入S2栈，此时抛弃“（”。 5. 重复上面的1~4步，直至处理完所有的输入字符。 6. 若取出的字符是“#”，则将S1栈内所有运算符（不包括“#”），逐个出栈，依次送入S2栈。 |

**3、正则表达式与有限自动机**

正则表达式是对字符串（包括普通字符和特殊字符操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。正则表达式是一种文本模式，该模式描述在搜索文本时要匹配的一个或多个字符串。

正则引擎主要可以分为两大类：一种是DFA，一种是NFA。

DFA 引擎在线性时状态下执行，因为它们不要求回溯（并因此它们永远不测试相同的字符两次）。DFA 引擎还可以确保匹配最长的可能的字符串。但是，因为 DFA 引擎只包含有限的状态，所以它不能匹配具有反向引用的模式；并且因为它不构造显示扩展，所以它不可以捕获子表达式。

传统的 NFA 引擎运行所谓的“贪婪的”匹配回溯算法，以指定顺序测试正则表达式的所有可能的扩展并接受第一个匹配项。因为传统的 NFA 构造正则表达式的特定扩展以获得成功的匹配，所以它可以捕获子表达式匹配和匹配的反向引用。但是，因为传统的 NFA 回溯，所以它可以访问完全相同的状态多次（如果通过不同的路径到达该状态）。因此，在最坏情况下，它的执行速度可能非常慢。因为传统的 NFA 接受它找到的第一个匹配，所以它还可能会导致其他（可能更长）匹配未被发现。

**4、元字符**

|  |  |
| --- | --- |
| **元字符** | **描述** |
| \ | 将下一个字符标记符、或一个向后引用、或一个八进制转义符。例如，“\\n”匹配\n。“\n”匹配换行符。序列“\\”匹配“\”而“\(”则匹配“(”。即相当于多种编程语言中都有的“转义字符”的概念。 |
| ^ | 匹配输入字行首。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。 |
| $ | 匹配输入行尾。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置。 |
| \* | 匹配前面的子表达式任意次。例如，zo\*能匹配“z”，也能匹配“zo”以及“zoo”。\*等价于{0,}。 |
| + | 匹配前面的子表达式一次或多次(大于等于1次）。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。 |
| ? | 匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”。?等价于{0,1}。 |
| {*n*} | *n*是一个非负整数。匹配确定的*n*次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。 |
| {*n*,} | *n*是一个非负整数。至少匹配*n*次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o\*”。 |
| {*n*,*m*} | *m*和*n*均为非负整数，其中*n*<=*m*。最少匹配*n*次且最多匹配*m*次。例如，“o{1,3}”将匹配“fooooood”中的前三个o为一组，后三个o为一组。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。 |
| ? | 当该字符紧跟在任何一个其他限制符（\*,+,?，{*n*}，{*n*,}，{*n*,*m*}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少地匹配所搜索的字符串，而默认的贪婪模式则尽可能多地匹配所搜索的字符串。例如，对于字符串“oooo”，“o+”将尽可能多地匹配“o”，得到结果[“oooo”]，而“o+?”将尽可能少地匹配“o”，得到结果 ['o', 'o', 'o', 'o'] |
| . | 匹配除“\n”和"\r"之外的任何单个字符。要匹配包括“\n”和"\r"在内的任何字符，请使用像“[\s\S]”的模式。 |
| (pattern) | 匹配pattern并获取这一匹配。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0…$9属性。要匹配圆括号字符，请使用“\(”或“\)”。 |
| x|y | 匹配x或y。例如，“z|food”能匹配“z”或“food”(此处请谨慎)。“[z|f]ood”则匹配“zood”或“food”。 |
| [xyz] | 字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。 |
| [^xyz] | 负值字符集合。匹配未包含的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin”任一字符。 |
| [a-z] | 字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。  注意:只有连字符在字符组内部时,并且出现在两个字符之间时,才能表示字符的范围; 如果出字符组的开头,则只能表示连字符本身. |

**5、Python中正则表达式库Re**

1）re.match

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。函数语法：

re.match(pattern, string, flags=0)

2）re.search

re.search 扫描整个字符串并返回第一个成功的匹配。函数语法：

re.search(pattern, string, flags=0)

3）re.findall

在字符串中找到正则表达式所匹配的所有子串，并返回一个列表，如果没有找到匹配的，则返回空列表。注意： match 和 search 是匹配一次 findall 匹配所有。语法格式为：

re.findall(string[, pos[, endpos]])

1. **实验要求**

**1、实现中缀表达式转化为后缀表达式**

如将9 - 5 + 2转化为9 5 - 2 +，要求能够中缀表达式中包含四则运算符和括号。

1. **正则表达式练习**
2. 匹配

* 验证jpg,jpeg,gif,bmp格式的文件名，大小写均可?
* 匹配日期格式：2/31/2006
* 电话号码(区号提取)：(0512) 68078800-6852

2）提取（需要提取的内容在 re\_example/ 文件夹中）

* 提取list.html中的所有超链接
* 提取content中的标题与正文

**建议用时：**2课时

**参考资料**

1. Python中的Re模块介绍：https://www.runoob.com/python/python-reg-expressions.html
2. 正则表达式30分钟入门教程：https://deerchao.cn/tutorials/regex/regex.htm