编译原理实验一

实验报告

151250093

林文烨

## 词法分析

#### Motivation/Aim

编写一个词法分析程序，通过编写词法分析程序，掌握记号、模式与单词，掌握正规式与正规集，掌握有限自动机，掌握如何从正规式到词法分析器的各种算法。

#### Content description

1)Input

Stream of characters

REs(The number of REs is decided by yourself)

2)Output

Sequence of tokens

3)Classes of words are defined by yourself

4)Error handling is included

#### Ideas/Methods

实验采用第二种方法，实现自己的Lex，输入正则表达式，自动输出对应的token，

G -> RE -> NFA -> DFA -> 最小化DFA -> 完成

程序需要对输入的字符串进行判断，是否被最小化DFA接受，若是，则输出token，不是，则程序结束。

#### Assumption

Token分为标识符、关键字、操作符、注释符、整数。

关键字：

**'and'**, **'del'**, **'from'**, **'not'**, **'while'**, **'as'**, **'elif'**, **'global'**, **'or'**, **'with'**, **'assert'**, **'else'**, **'if'**, **'pass'**, **'yield'**,  
**'break'**, **'except'**, **'import'**, **'print'**, **'class'**, **'exec'**, **'in'**, **'raise'**, **'continue'**, **'finally'**, **'is'**, **'return'**, **'def'**,  
**'for'**, **'lambda'**, **'try'**

操作符：

**'<'**, **'<='**, **'='**, **'<>'**, **'>'**, **'>='**, **'+'**, **'-'**, **'/'**, **'+='**, **'-='**

注释符：#

另外，空格、\n、\t、\r在词法分析阶段忽略。

#### Related FA descriptions

将Token构造正则表达式：正则表达式不支持扩展表达

标识符：

**(A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|\_)((A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|\_)|(0|1|2|3|4|5|6|7|8|9))\*'**

关键字：

**'and'**, **'del'**, **'from'**, **'not'**, **'while'**, **'as'**, **'elif'**, **'global'**, **'or'**, **'with'**, **'assert'**, **'else'**, **'if'**, **'pass'**, **'yield'**,  
**'break'**, **'except'**, **'import'**, **'print'**, **'class'**, **'exec'**, **'in'**, **'raise'**, **'continue'**, **'finally'**, **'is'**, **'return'**, **'def'**,  
**'for'**, **'lambda'**, **'try'**

操作符：

**'<'**, **'<='**, **'='**, **'<>'**, **'>'**, **'>='**, **'+'**, **'-'**, **'/'**, **'+='**, **'-='**

注释符：#

整数：

**(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)\***

## Description of important Data Structures

## NFA: 使用Thompson算法构造

NFAnode 为每个nfa结点，NFAchar为单个字符时构造的子NFA，NFAor, NFAcat, NFAclosure 分别对应“|”，“连接”，“\*”的正则表达式运算符构建的NFA

**class NFAnode:** # maybe the set next method should be include, thus no need to set the next node everywhere in the code  
 **def \_\_init\_\_(**self, edge**=**None**):** self.edge **=** edge  
 self.next1 **=** None  
 self.next2 **=** None  
 self.state **=** get\_state**()  
  
  
class NFAchar:  
 """  
 NFA of character  
 """  
  
 def \_\_init\_\_(**self, edge**):** self.start **=** NFAnode**(**edge**)** self.end **=** NFAnode**()** self.start.next1 **=** self.end  
  
  
**class NFAor:  
 """  
 operator :|  
 use ~ to represent ε  
 """  
  
 def \_\_init\_\_(**self, nfa1, nfa2**):** self.start **=** NFAnode**('~')** self.start.next1 **=** nfa1.start  
 self.start.next2 **=** nfa2.start  
  
 self.end **=** NFAnode**()** nfa1.end.edge **= '~'** nfa1.end.next1 **=** self.end  
 nfa2.end.edge **= '~'** nfa2.end.next1 **=** self.end  
  
  
**class NFAcat:  
 """  
 operator: `  
 """  
  
 def \_\_init\_\_(**self, nfa1, nfa2**):** nfa1.end.edge **=** nfa2.start.edge  
 nfa1.end.next1 **=** nfa2.start.next1  
 nfa1.end.next2 **=** nfa2.start.next2  
  
 nfa2.start **=** nfa1.end  
  
 self.start **=** nfa1.start  
 self.end **=** nfa2.end  
  
  
**class NFAclosure:  
 """  
 operator: \*  
 """  
  
 def \_\_init\_\_(**self, nfa**):** self.start **=** NFAnode**('~')** self.end **=** NFAnode**()** self.start.next1 **=** nfa.start  
 self.start.next2 **=** self.end  
  
 nfa.end.edge **= '~'** nfa.end.next1 **=** nfa.start  
 nfa.end.next2 **=** self.end

## DFA

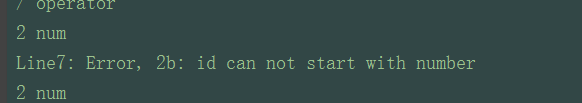
DFAnode 是每一个结点，StateSet中dfa中是一个dfa结点链表，通过每个结点中存储的next来表示结点的链接，edge表示通过什么符号到达下个结点，edge中的元素位置与next中存储的下一个结点的元素位置一一对应。

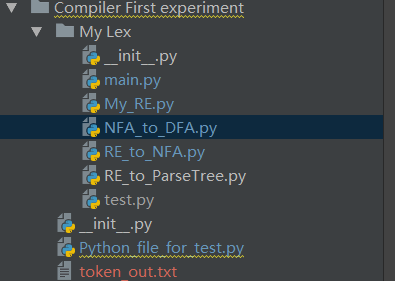
**class DFAnode:  
 def \_\_init\_\_(**self**):** self.state **=** get\_state**()** self.next **= []** # multiple next node, like LinkedList<LinkedList<>> in java  
 self.edge **= []** # save with what char to turn to next  
  
 **def add\_edge(**self, next\_node, one\_edge**):** self.next.append**(**next\_node**)** self.edge.append**(**one\_edge**)  
  
  
class StateSet:  
 def \_\_init\_\_(**self**):** self.state\_set\_list **= []** # this is a list of set; just like list(set1,set2 ...,etc)  
 self.marker **= []** # this marks whether the state\_set in the list has calculate the e-closure(move(A,char))  
 self.dfa **= []** self.terminals **= {}** # {state:token}  
  
 **def is\_all\_marked(**self**):  
 if** 0 **not in** self.marker**:  
 return** True  
 **else:  
 return** False  
  
 **def get\_not\_marked(**self**):  
 for** i, item **in** enumerate**(**self.marker**):  
 if** item **==** 0**:  
 return** i  
  
 **def add\_without\_mark(**self, new\_state\_set**):** self.state\_set\_list.append**(**new\_state\_set**)** self.marker.append**(**0**)** self.dfa.append**(**DFAnode**())  
  
 def DFA\_to\_DFAo(**self, nfa\_end, end\_state\_token**):** final **= []** states\_non\_term **= []** states\_term **= []  
 for** nfa\_nodes, dfa\_node **in** zip**(**self.state\_set\_list, self.dfa**):** is\_terminal **=** False  
 **for** node **in** nfa\_nodes**:  
 if** node.state **==** nfa\_end.state**:** is\_terminal **=** True  
 **if** is\_terminal**:** states\_term.append**(**dfa\_node**)  
 else:** states\_non\_term.append**(**dfa\_node**)** s\_non\_term **=** self.\_euqal**(**states\_non\_term**)** s\_term **=** self.\_euqal**(**states\_term**)** final.extend**(**s\_non\_term**)** final.extend**(**s\_term**)** # remove extra and replace by the reprensent  
 **for** equal\_set **in** final**:** represent **=** equal\_set**[**0**]  
 for** extra **in** equal\_set**:  
 if not** extra **==** represent**:** self.\_remove\_extra**(**represent, extra**)  
  
 for** i, nodes **in** enumerate**(**self.state\_set\_list**):  
 for** every\_node **in** nodes**:  
 for** every\_nfa\_node, token **in** end\_state\_token.items**():  
 if** every\_nfa\_node **==** every\_node**:** self.terminals**[**self.dfa**[**i**]**.state**] =** token  
  
 **def \_remove\_extra(**self, represent, extra\_node**):** self.state\_set\_list.pop**(**self.dfa.index**(**extra\_node**))** self.dfa.remove**(**extra\_node**)  
 for** node **in** self.dfa**:  
 if** extra\_node **in** node.next**:** ii **=** node.next.index**(**extra\_node**)** node.next**[**ii**] =** represent  
  
 **def \_euqal(**self, dfa\_node\_list**):** res **= []** visted **=** set**()  
 for** i **in** range**(**len**(**dfa\_node\_list**)):  
 if** i **in** visted**:  
 continue** inner **= [**dfa\_node\_list**[**i**]]  
 for** j **in** range**(**i **+** 1, len**(**dfa\_node\_list**)):  
 if** self.\_compare**(**dfa\_node\_list**[**i**]**, dfa\_node\_list**[**j**]):** inner.append**(**dfa\_node\_list**[**j**])** visted.add**(**j**)** res.append**(**inner**)  
 return** res  
  
 **def \_compare(**self, dfa\_node1, dfa\_node2**):  
 if** dfa\_node1.next **==** dfa\_node2.next **and** dfa\_node1.edge **==** dfa\_node2.edge**:  
 return** True  
 **else:  
 return** False

#### Description of core Algorithms

本算法的要求是从字符串表示的代码段中识别具有独立意义的单词、符号，要做到根据扫描到单词符号的第一个字符的种类，拼出相应的单词符号。关键字也作为标识符。当扫描程序扫描到标识符时，优先查找一张事先建立好的关键字表，若能匹配，则将该单词识别为关键字，否则为一般性质的标识符。

识别出源程序中的各个单词符号，并转换成内部编码形式，各个单词符号分为：reserve, id, operator, num这几种。删除无用的空白字符回车字符以及其他非实质性字符。删除注释。进行词法检查，报告所发现的错误。

错误检测：实现定位到错误发生的代码行。‘2b’为输入文件中的错误字段。



Main 实现输入输出及调用其他方法，My\_RE事实上是作为输入文件，存储想使用的正则表达式，其他的文件作用如名字所示

#### Use cases on running

输入程序：Python\_file\_for\_test.py

**from** numpy **import** sum  
**import** random  
  
  
**class Proxies(**object**):** a **=** 3 **+** 15 **/** 2  
 2b **=** 1  
 **def \_\_init\_\_(**self**):** self.proxies **=** list**()** # self.maintain\_ip()  
  
 # def maintain\_ip(self):  
 # self.proxies = list()  
 # soup = page\_read.page\_read\_nolog("http://www.free-proxy-list.net/")  
 # tbody = soup.find\_all('tbody')[0]  
 # for row in tbody.find\_all('tr'):  
 # proxytem = {"http": "http://" + row.find\_all('td')[0].get\_text() + ":" + row.find\_all('td')[1].get\_text()}  
 # self.proxies.append(proxytem)  
  
 **def random\_ip(**self**):  
 if** len**(**self.proxies**) <** 20**:  
 return** self.proxies**[**random.randint**(**0, 20**)]**

**输出结果：**

from FROM  
numpy id  
import IMPORT  
sum id  
import IMPORT  
random id  
class CLASS  
Proxies id  
object id  
a id  
= operator  
3 num  
+ operator  
15 num  
/ operator  
2 num  
Line7: Error, 2b: id can not start with number  
2 num  
= operator  
1 num  
def DEF  
\_\_init\_\_ id  
self id  
self id  
proxies id  
= operator  
list id  
def DEF  
random\_ip id  
self id  
if IF  
len id  
self id  
proxies id  
< operator  
20 num  
return RETURN  
self id  
proxies id  
random id  
randint id  
20 num

#### Problems occurred and related solutions

主要还是实践度不够，缺乏动手实践，需要多加练习。

#### Your feelings and comments

通过本实验的完成，我加深了对词法分析原理的理解，实践了由表达式到最小化DFA的说算法设计和技术。同时对课程的知识有了自己的发现，比如求最小集过程：不就是先分成结束集和非结束集，一次性比较是否所有后续状态都一样嘛！一样则加入同一集合当中，这样就不用像上课讲的求循环那么繁复了。同时还加深了对python的理解。多态以及值传递。实验很好！