

院系	年级专业	姓名	学号	实验日期
计算机学院	2019计科	吴家隆	1915404063	2021.9.6

编程语言: *python3.9*

实验1.MYT 算法的实现 (REGEX2NFA)

实验内容

将待转换的正则表达式存放在.txt中，主程序读取该正则表达式将其转化为NFA，并以三元组的格式输出到output.txt中，并指明开始状态和接受状态。

实验步骤

☑ 构建一个FA类

存放states、symbol、transitions、startstate、finalstates五类元素

states	symbol	transitions	startstate	finalstates
存放已有的状态	输入符号表	状态之间的映射关系	开始状态	接受状态

并加入以下函数

setStart(self,state)	addFinal(self,state)	addTransition(self, fromstate, tostate, inputch)
-----------------------------	-----------------------------	---

☑ 构建一个Regex2NFA类

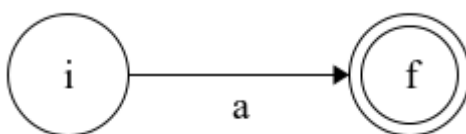
在buildNFA初始化中完成以下内容：

1. 显式地为正则式添加连接符，加入`.`作为连接符
2. 将正则表达式中缀表达式转换为后缀表达式，去除括号
3. 由后缀表达式构建NFA

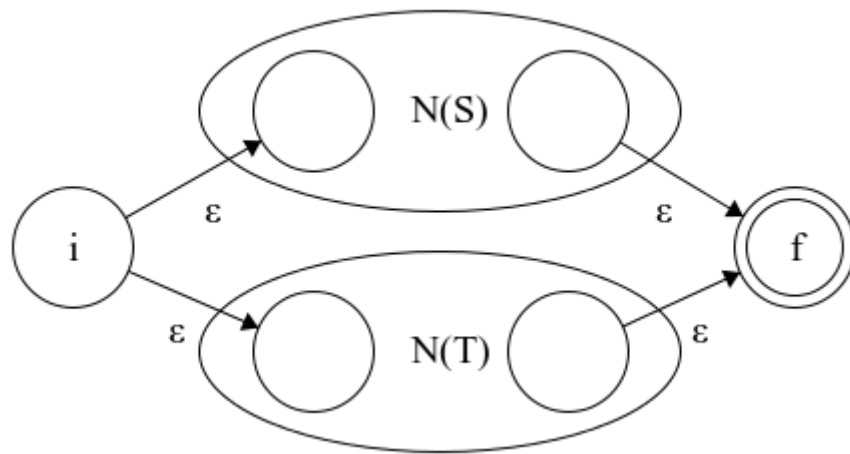
在后缀表达式构造NFA中，根据栈中的符号对元素(FA类)进行运算 (McNaughton-Yamada-Thompson算法)

basicstruct(inputch)	linestruct(a, b)	dotstruct(a, b)	starstruct(a)
Regex = a \rightarrow NFA	Regex = a b \rightarrow NFA	Regex = a · b \rightarrow NFA	Regex = a* \rightarrow NFA

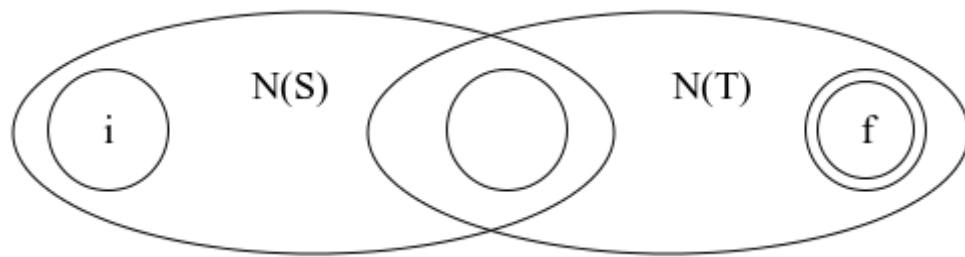
- basic



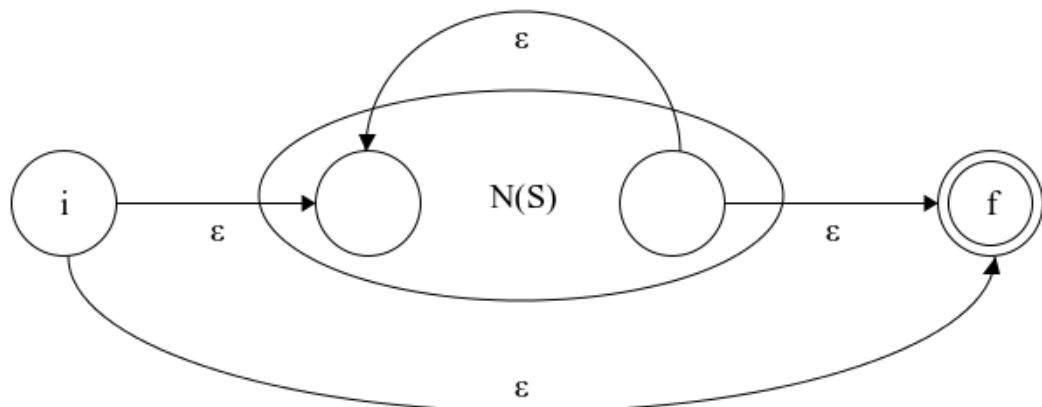
- line



◦ dot



◦ star



☒ 完成实验的输入输出

输入input.txt

输出output.txt

实验结果

以 $(a|b)^*abb$ 为例

input.txt

```
test_thompson_nfa.py × output.txt × input.txt ×
1 (a|b)*abb
```

主程序

```

if __name__ == '__main__':
    with open("input.txt", "r", encoding="utf-8") as x:
        regex = x.read()
        a = Regex2NFA(regex)
        a = NFA2DFA(a)
        a.displayminDFA()

```

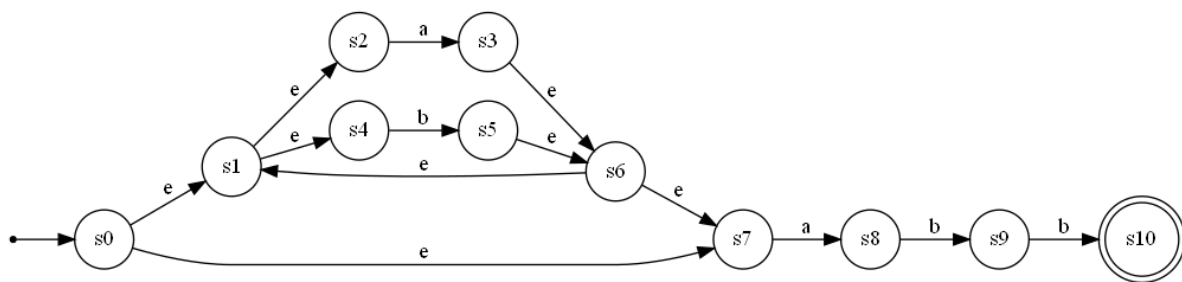
output.txt

```

start state:0
accepting states:10
0   e   1
0   e   7
6   e   7
6   e   1
1   e   2
1   e   4
3   e   6
5   e   6
2   a   3
4   b   5
7   a   8
8   b   9
9   b  10

```

将output绘图检验正确性



验证结果正确

实验2.子集构造算法的实现(NFA2DFA)

实验内容

通过读取实验1获得NFA，将NFA通过自己构造法完成DFA的转化，并输出到output.txt中

实验步骤

- ✓ 如实验1相同，构造NF类，在实验1基础上增加以下函数

getMove(self, state, skey)	getEpsilonClosure(self, findstate)
获取NFA从state状态出发，通过skey能到达的所有状态的集合	从NFA的状态集合findstate内每个状态出发，只用e转换就能到达的状态的集合

✓ 构造NFA2DFA类

完成buildDFA(self, nfa)的构造

子集构造法

- 输入: 一个NFA N
- 输出: 一个DFA D
- D的转换表: Dtran, 状态集: Dstates
- 如果D的某个状态B包含一个N的接收状态, 那么V是D的一个接受状态

✓ 完成实验的输入输出

输入input.txt

输出output.txt

实验结果

以 $(a|b)^*abb$ 为例

input.txt

```
start state:0
accepting states:10
0 e 1
0 e 7
6 e 7
6 e 1
1 e 2
1 e 4
3 e 6
5 e 6
2 a 3
4 b 5
7 a 8
8 b 9
9 b 10
```

主程序

```
if __name__ == '__main__':
    adic = defaultdict(defaultdict)
    with open("input.txt", "r", encoding="utf-8") as x:
        lines = x.readlines()
    startsta = int(lines[0].split(":")[1])
    finalstates = [int(lines[1].split(":")[1])]
    sta = set([])
    symbol = set([])
    for i in range(2, len(lines)):
        lines[i] = lines[i].strip()
        a, b, c = lines[i].split()
        if b != "e":
```

```

symbol.add(b)
sta.add(int(a))
sta.add(int(c))
adic[int(a)][int(c)] = b
P = FA(symbol)
P.init(sta,symbol,adic,startsta,finalstates)
D = NFA2DFA(P)
D.displayDFA()
D.dfa.mywrite()

```

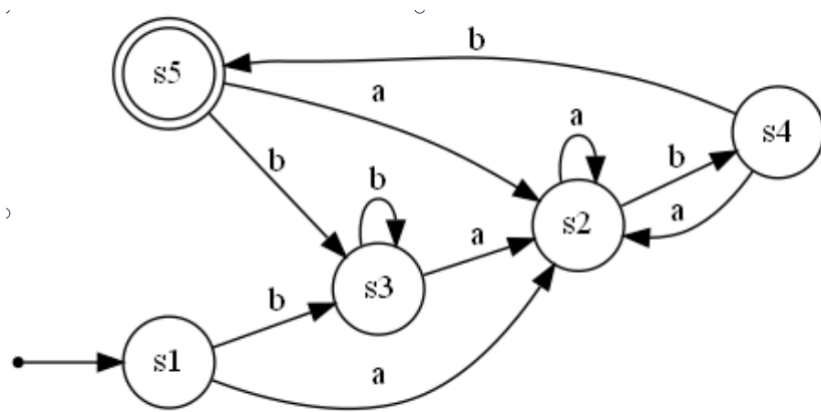
output.txt

```

start state:1
accepting states:5
1  a  2
1  b  3
3  a  2
3  b  3
2  a  2
2  b  4
4  a  2
4  b  5
5  a  2
5  b  3

```

将output绘图检验正确性



检验结果正确

实验3.DFA 最小化

实验内容

通过读取上一实验获得的DFA，将DFA进行最小化

实验步骤

- ✓ 在实验1构造的NF类中加入newBuildFromEqualStates(self, equivalent, pos)，在最小化合并状态后修改状态的表示数字
- ✓ 构造MinDfa类

在MinDfa类中完成分割最小化 minimise(self)

把一个DFA（不含多余状态）的状态分割成一些不相交的子集，并且任意两个子集之间的状态都是可区别状态，同一子集内部的状态都是等价状态。

步骤：

1. I_0 = 非状态元素构成的集合， I_1 = 终态元素构成的集合
2. 经过多次划分后，要保证，任意一个 I_k 中的元素通过move(I_k , 某个字符)的结果都同属于一个 I_z ，这时候划分完成。否则把状态不同的单独划分出去
3. 重复上一步，直至没有新的 I 子集增加。
4. 从子集中任选一个代替整体，画出最简DFA。

✅ 完成实验的输入输出

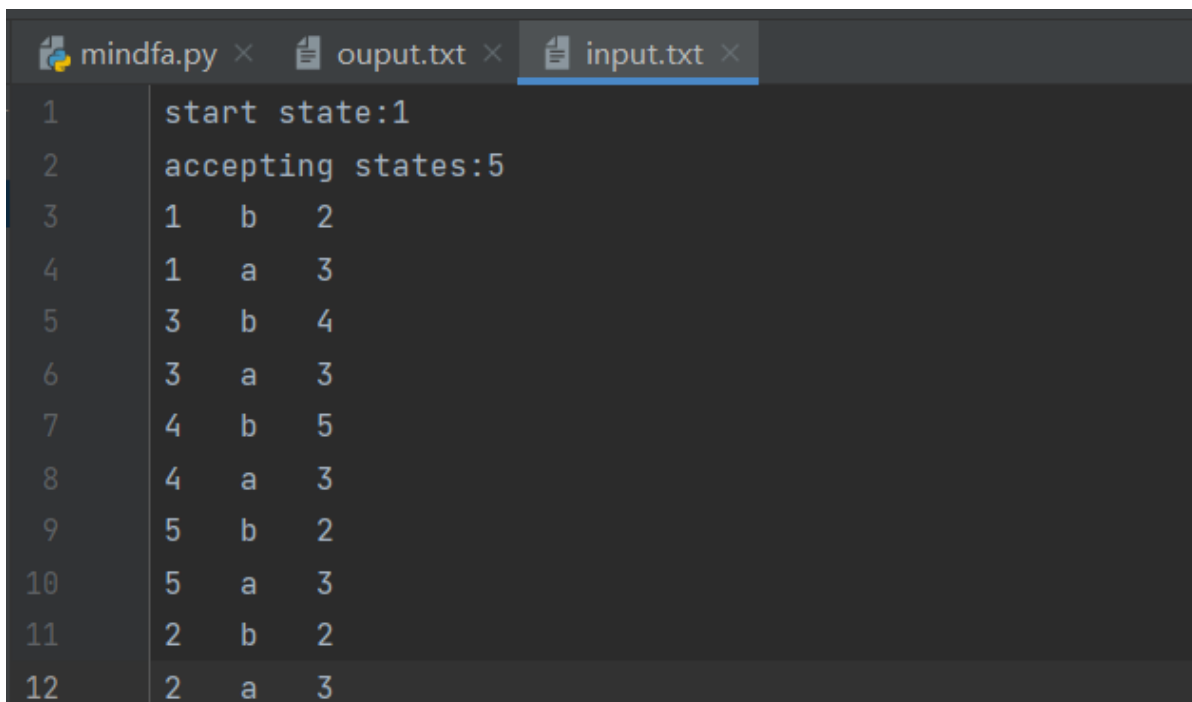
输入input.txt

输出output.txt

实验结果

以 $(a|b)^*abb$ 为例

input.txt



```
mindfa.py x ouput.txt x input.txt x
1 start state:1
2 accepting states:5
3 1 b 2
4 1 a 3
5 3 b 4
6 3 a 3
7 4 b 5
8 4 a 3
9 5 b 2
10 5 a 3
11 2 b 2
12 2 a 3
```

主程序

```
if __name__ == '__main__':
    adic = defaultdict(defaultdict)
    with open("input.txt", "r", encoding="utf-8") as x:
        lines = x.readlines()
        startsta = int(lines[0].split(":")[1])
        finalstates = [int(lines[1].split(":")[1])]
        sta = set([])
        symbol = set([])
        for i in range(2, len(lines)):
            lines[i] = lines[i].strip()
            a, b, c = lines[i].split()
            if b != "e":
                symbol.add(b)
```

```

sta.add(int(a))
sta.add(int(c))
adic[int(a)][int(c)] = b
P = FA(symbol)
P.init(sta, symbol, adic, startsta, finalstates)
D = MinDfa(P)
D.minimise()
D.minDFA.mywrite()

```

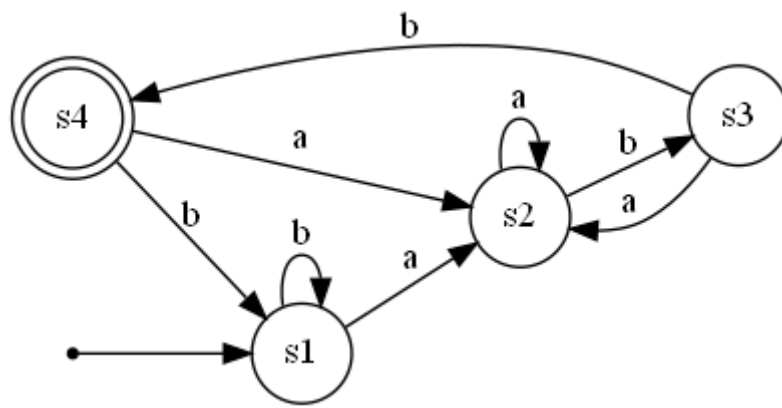
output.txt

```

mindfa.py x ouput.txt x
1 start state:1
2 accepting states:4
3 1 b 1
4 1 a 2
5 2 b 3
6 2 a 2
7 3 b 4
8 3 a 2
9 4 b 1
10 4 a 2

```

将output绘图检验正确性



检验结果正确

文件运行说明

将压缩包解压缩，分别为三个子文件夹

regex2nfa文件夹为MYT算法的实现，欲检验算法的正确性，只需要修改input.txt的regex，然后运行主程序re2bnfa.py，运行结果在output.txt中显示

nfa2dfa文件夹为子集构造算法的实现，欲检验算法的正确性，只需要修改input.txt，然后运行主程序nfa2dfa.py，运行结果在output.txt中显示

mindfa文件夹为子集构造算法的实现，欲检验算法的正确性，只需要修改input.txt，然后运行主程序mindfa.py，运行结果在output.txt中显示

以上的input.txt全部以**(a|b)*abb** 为例

在该readme中，主程序部分只摘取了main部分，可以解压文件运行查看主程序源文件

注意：为方便起见，用字符**e**表示空串，并且假设输入的正则表达式不涉及符号**e**。