正则表达式到NFA和DFA的转换

院系	年纪专业	姓名	学号	实验日期	指导老师
计科院	20计科	柯骅	2027405033	2022.09.07	李军辉

MYT算法的实现（正则表达式转NFA）

实验要求

从input.txt中读取正则表达式，编写程序让它转化成NFA，按照“起始状态编号 状态转换符号 转换后状态编号”的格式输出到ouput.txt文件中，并指明起始状态与结束状态

实验步骤

1.将正则表达式转化成后缀表达式

- [x] 将正则表达式在需要的地方添加 " . "作为连接符，便于转化

```
def add_dot(s):
    ans=''
    for i in range(len(s)-1):
        if ('a'<=s[i]<='z' or s[i]==')' or s[i]=='*') and ('a'<=s[i+1]<='z'
or s[i+1]=='('):
            ans=ans+s[i]+'.'
        else:
            ans=ans+s[i]
    ans+=s[-1]
    return ans
```

- [x] 根据优先级，利用栈结构存储符号，得到后缀表达式

```
def suffix(s):
    b=list()
```

```
    ans=''
    for i in s:
        if 'a'<=i<='z':
            ans+=i
        elif i==')':
            while(len(b)>0 and b[-1]!='('):
                ans+=b[-1]
                b.pop()
            b.pop()
        elif i=='(':
            b.append('(')
        else:
            while(len(b)!=0 and get_por(b[-1])>get_por(i)):
                ans+=b[-1]
                b.pop()
            b.append(i)

    while(len(b)!=0):
        ans+=b[-1]
        b.pop()
    return ans
```

2.用得到的后缀表达式转化成NFA

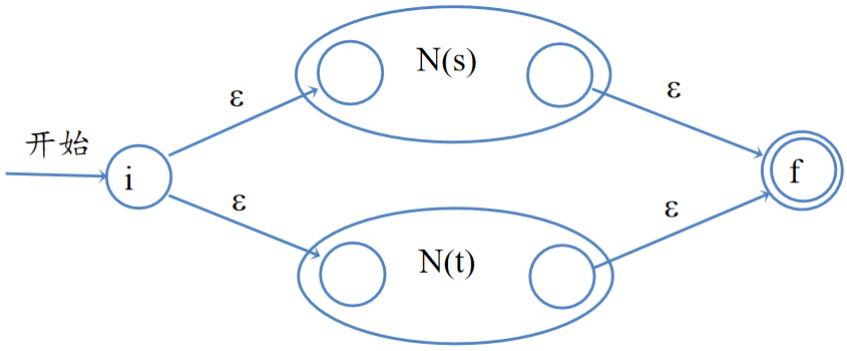
- [x] 按顺序读取后缀表达式，如果是字符，那么压入栈b，如果是符号，那么从根据不同的符号从栈顶取出字符处理成“一个模块”，并纪录起始编号和结束编号方便下次操作，再次压入栈中，等待下一次处理。

  最终栈中剩下的一个模块就是我们所需的答案，这个模块的起始编号与结束编号就是NFA的开始状态和结束状态

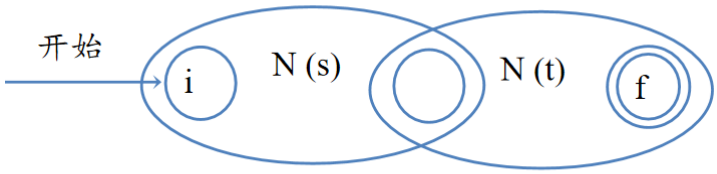
```
def suf_to_nfa(suf):
    ans_list=[]
    b=[]
    k=0 #k:已经用过多少个了
    for i in range(len(suf)):
        ch=suf[i]
        if 'a'<=ch<='z':
            b.append((ch,-1,-1))
        elif ch=='*':
            if(b[-1][1]==-1):
                k+=2
                node=(b[-1][0],k-1,k)
                b.pop()
                b.append(node)
                ans_list.append((k-1,'e',k))
            node=b[-1]
            b.pop()
            ans_list.append((node[2], 'e', k + 2))
            ans_list.append((k+1, 'e', k + 2))
            ans_list.append((node[2], 'e', node[1]))
            ans_list.append((k+1, 'e', node[1]))
            b.append(('!',k+1,k+2))
            k+=2
        elif ch=='|':
            if b[-1][1]==-1 and b[-2][1]==-1:
                node1=b[-1]
```

```
                node2=b[-2]
                b.pop()
                b.pop()
                ans_list.append((k+1,node1[0],k+2))
                ans_list.append((k+1,node2[0],k+3))
                ans_list.append((k + 2, 'e', k + 4))
                ans_list.append((k + 3, 'e', k + 4))
                ans_list.append((k + 4, 'e', k + 6))
                ans_list.append((k + 5, 'e', k + 1))
                b.append(('!',k+5,k+6))
                k+=6
            else:
                if (b[-1][1] == -1):
                    node = b[-1]
                    b.pop()
                    b.append((node[0], k + 1, k + 2))
                    ans_list.append((k + 1, node[0], k + 2))
                    k += 2
                if (b[-2][1] == -1):
                    node = b[-2]
                    b.pop()
                    b.append((node[0], k + 1, k + 2))
                    ans_list.append((k + 1, node[0], k + 2))
                    k += 2
                node1=b[-1]
                node2=b[-2]
                b.pop()
                b.pop()
                ans_list.append((k+1,'e',node1[1]))
                ans_list.append((k+1,'e',node2[1]))
                ans_list.append((node1[2],'e',k+2))
                ans_list.append((node2[2],'e',k+2))
                b.append(('!',k+1,k+2))
                k+=2
        else:
            node1=b[-1]
            node2=b[-2]
            b.pop()
            b.pop()
            if node1[1]==-1:
                node1=(node1[0],k+1,k+2)
                ans_list.append((k+1,node1[0],k+2))
                k+=2
            if node2[1]==-1:
                node2=(node2[0],k+1,k+2)
                ans_list.append((k+1,node2[0],k+2))
                k+=2
            ans_list.append((node2[2],'e',node1[1]))
            b.append(('!',node2[1],node1[2]))
    return (ans_list,b[0][1],b[0][2])
```

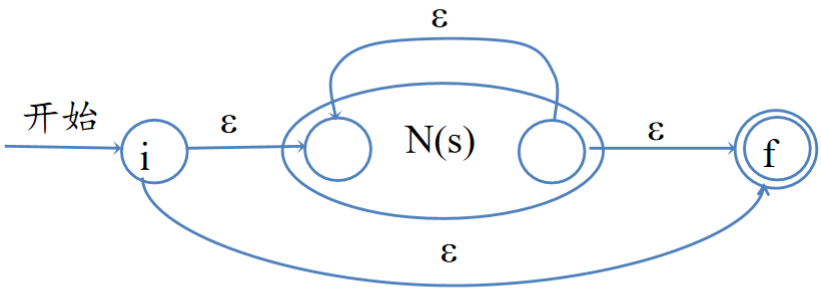
☑ 参考模块：



识别正则式 s|t 的NFA



识别正则式 st 的NFA



识别正则式 s^* 的NFA

实验结果

output.txt - 记事本

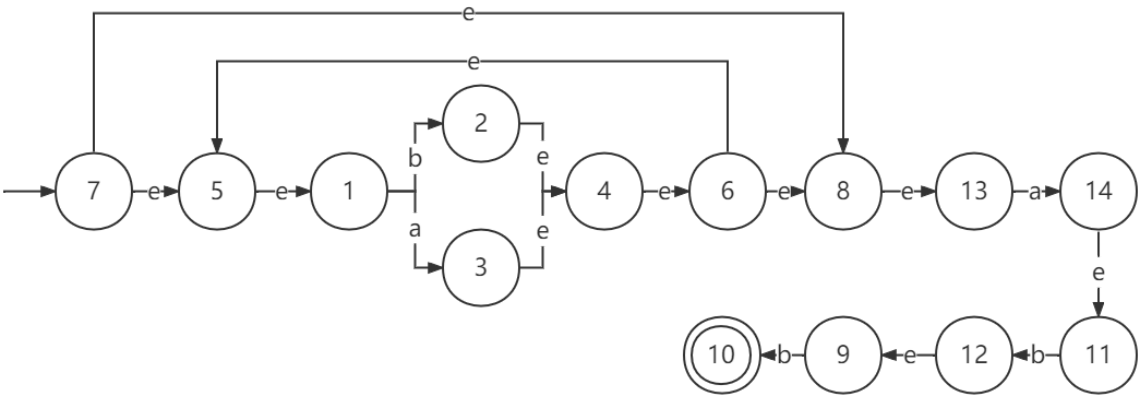
文件(F) 编辑(E) 格式(O) 查看

```
1 b 2
1 a 3
2 e 4
3 e 4
4 e 6
5 e 1
6 e 8
7 e 8
6 e 5
7 e 5
9 b 10
11 b 12
12 e 9
13 a 14
14 e 11
8 e 13
start state: 7
accepting state: 10
```

input.txt - 记事本

文件(F) 编辑(E) 格式(O)

```
(a|b)*abb
```



子集构造算法的实现

实验要求

将MYT算法的输出作为输入，利用子集构造法将NFA转化成DFA，输出格式同上

实验步骤

- [x] 1. 初始化：将开始状态集放入saved_idxs，并作为当前状态
  2. 枚举符号集marks中的所有符号，寻找当前状态的mark的出边，作为待处理状态，执行操作3
  3. 如果待处理状态在saved_idxs已存在，那么添加一条新边**“当前状态编号 mark 待处理状态编号”**

     如果待处理状态在saved_idxs不存在，那么将待处理状态作为新状态append到saved_idxs中

4. 将saved_idxs中的下一状态作为当前状态，再次执行操作2，直到saved_idxs中的左右状态全部处理完

- [x] 关键函数nfa_to_dfa

```
def nfa_to_dfa(nfa_list,beginn):
    m=dict()
    marks=set()
    for i in nfa_list:
        m[i[0]]=m.get(i[0],list())
        m[i[0]].append((i[1], i[2]))
        marks.add(i[1])
    marks.discard('e')
    marks=list(marks)#去除e的符号集
    turns=[]
    k=0
    same_set.clear()#同系的点
    get_category(m, beginn)#获取和beginn同系的点
    saved_idxs=[copy.deepcopy(same_set),]#等待去处理的点的集合的列表
    next_idxs=set()
    t=0#当前状态编号
    while(t<len(saved_idxs)):
        same_idxs=saved_idxs[t]
        for mark in marks:#枚举符号
            next_idxs.clear()
            for same_idx in same_idxs:#枚举本系的点，寻找mark的出路
                next_idx=get_next(m,same_idx,mark)
                if(next_idx!=-1):
                    next_idxs.add(next_idx)
            same_set.clear()
            for i in next_idxs:
                get_category(m,i)
            flag=0
            for i in range(len(saved_idxs)):
                if saved_idxs[i]==same_set:
                    turns.append((t,mark,i))
                    flag=1
            if(flag==0):
                saved_idxs.append(copy.deepcopy(same_set))
                turns.append((t,mark,k+1))
                k+=1
        t+=1
    return saved_idxs,turns
```

实验结果

input.txt - 记事本

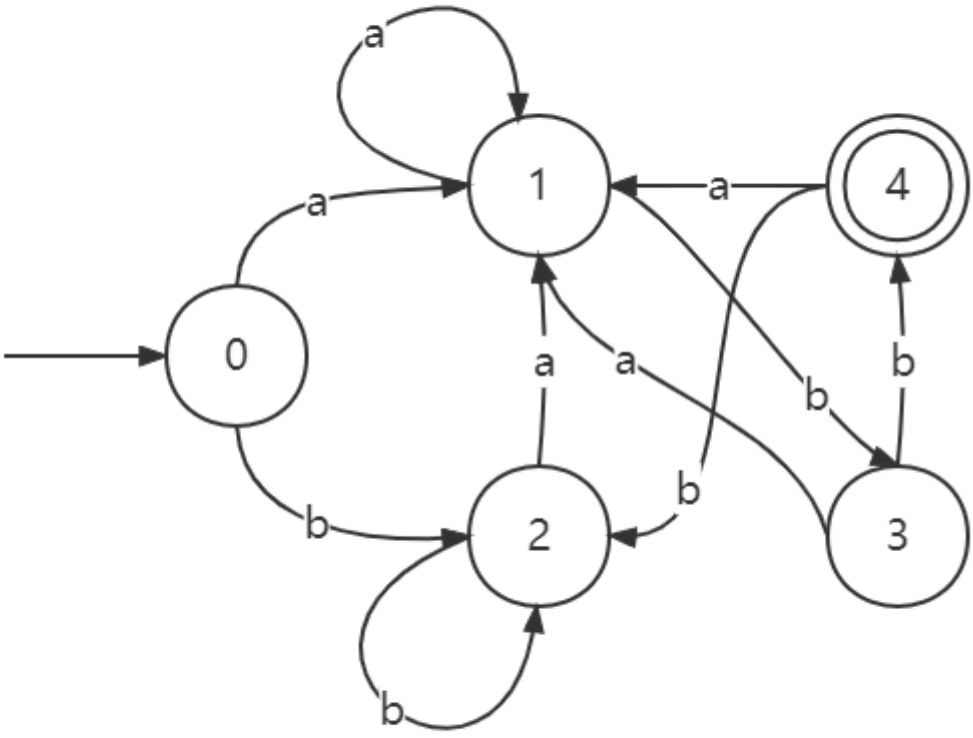
文件(F) 编辑(E) 格式(O)

```
1 b 2
1 a 3
2 e 4
3 e 4
4 e 6
5 e 1
6 e 8
7 e 8
6 e 5
7 e 5
9 b 10
11 b 12
12 e 9
13 a 14
14 e 11
8 e 13
start state: 7
accepting state: 10
```

output.txt - 记事本

文件(F) 编辑(E) 格式(O)

```
0 a 1
0 b 2
1 a 1
1 b 3
2 a 1
2 b 2
3 a 1
3 b 4
4 a 1
4 b 2
start state: 0
accepting state: 4
```



DFA最小化

实验要求

将子集构造法的输出作为输入，将DFA最小化，输出格式同上

实验步骤

- [x] 1. 将初始DFA的状态划分为两个集合，分别为：**非终止状态集I0**、**终止状态集I1**，存储在types中
  2. 不断地划分，直到在一个**状态集合IS**中的所有原始状态经过相同的符号mark的结果都属于一个**状态集合IT**。

     否则，将根据不同的结果把**状态集合IS**再次划分，直到符合上面的条件
  3. 将处理好后的**状态的划分**重新命名，并得出对应转换关系
- [x] 关键函数：

```
def min_DFA(dfa_list,begin_idx,end_idx):
    m = dict()
    marks = set()
    n=-1
    for i in dfa_list:
        m[i[0]] = m.get(i[0], list())
        m[i[0]].append((i[1], i[2]))
        marks.add(i[1])
        n=max(n,i[0],i[2])
    types=[]
    a=set()
    b=set()
    for i in range(n+1):
        if i not in end_idx:
            a.add(i)
        else:
            b.add(i)
    types.append((None,a))
    types.append((None,b))#一种当前最优划分
    while(True):
        new_types=[]
        for type in types:#枚举各个状态，检查每个状态中是否有违规的（不一样）
                          #在一个状态中，不断分出新的状态，作为下一次的types
            new_types_temp=[]
            for i in type[1]:
                point=[]
                for mark in marks:
                    point.append(get_next(m,i,mark,types))
                flag=0#表示没有找到相同的point
                for p in range(len(new_types_temp)):#枚举当前type中已经存在的指向，如果新指向已经存在，那么将这个i放到new--types中对应的type中
                                    #否则（新指向不存在），那么添加一个新type，将这个i放到new--types的新的type中
                    if new_types_temp[p][0]==point:
                        new_types_temp[p][1].add(i)
                        flag=1
                        break
                if(flag==0):
                    new_types_temp.append((point,{i}))
            new_types=new_types+new_types_temp
        if(types==new_types):
```

```
            break
        else:
            types=new_types
    return types_to_turns(types,marks,begin_idx,end_idx)
```

实验结果

input.txt - 记事本

文件(F) 编辑(E) 格式(O)

```
0 a 1
0 b 2
1 a 1
1 b 3
2 a 1
2 b 2
3 a 1
3 b 4
4 a 1
4 b 2
start state: 0
accepting state: 4
```

output.txt - 记事本

文件(F) 编辑(E) 格式(C

```
0 b 0
0 a 1
1 b 2
1 a 1
2 b 3
2 a 1
3 b 0
3 a 1
start state: 0
accepting state: 3
```