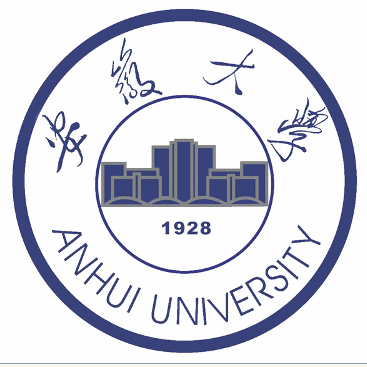
编译原理实验报告

****

**实验名称: 确定有穷自动机的最小化**

**实验时间： 2020/05/27**

**院 系： 计算机科学与技术学院**

**班 级： 计算机科学与技术一班**

**学 号： E01814055**

**姓 名： 张亚强**

1. **背景资料**

NFA 确定化的实质是以原有状态集上的子集作为 DFA 上的一个状态，将原 状态间的转换为该子集间的转换，从而把不确定有穷自动机确定化。经过确定化 后，状态数可能增加，而且可能出现一些等价状态，这时就需要简化。

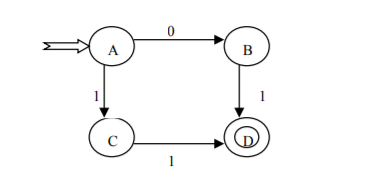
**二、实验目的要求**

输入： DFA。

输出： 最小化的 DFA。

**三、实验原理**

所谓自动机的化简问题即是对任何一个确定有穷自动机 DFA M，构造另一个确定有穷自动机 DFA M’，有 L（M）＝L（M’），并且 M’的状态个数不多于 M 的状态个数，而且可以肯定地说，能够找到一个状态个数为最小的 M’。 下面首先来介绍一些相关的基本概念。 设 Si 是自动机 M 的一个状态，从 Si 出发能导出的所有符号串集合记为 （Si）。 设有两个状态Si和 Sj，若有 L（Si）＝L（Sj），则称 Si和 Sj是等价状态。 下图所示的自动机中 L（B）＝L（C）＝{1}，所有状态 B 和状态 C 是等价状态。



又例如终态导出的符号串集合中必然包含空串 ε，而非终止状态导出的符 号串集合中不可能包含空串 ε，所以终态和非终止状态是不等价的。 对于等价的概念，我们还可以从另一个角度来给出定义。 给定一个 DFA M，如果从某个状态 P 开始，以字符串 w 作为输入，DFA M 将 结束于终态，而从另一状态 Q 开始，以字符串 w 作为输入，DFA M 将结束于非终 止状态，则称字符串 w 把状态 P 和状态 Q 区分开来。 把不可区分开来的两个状态称为等价状态。 设 Si是自动机 M 的一个状态，如果从开始状态不可能达到该状态 Si，则称 Si为无用状态。 设 Si是自动机 M 的一个状态，如果对任何输入符号 a 都转到其本身，而不 可能达到终止状态，则称 Si为死状态。

化简 DFA 关键在于把它的状态集分成一些两两互不相交的子集，使得任何 两个不相交的子集间的状态都是可区分的，而同一个子集中的任何两个状态都 是等价的，这样可以以一个状态作为代表而删去其他等价的状态，然后将无关 状态删去，也就获得了状态数最小的 DFA。 下面具体介绍 DFA 的化简算法：

（1） 首先将 DFA M 的状态划分出终止状态集 K1和非终止状态集 K2。 K＝K1∪K2 由上述定义知，K1和 K2是不等价的。

（2） 对各状态集每次按下面的方法进一步划分，直到不再产生新的划分。 设第 i 次划分已将状态集划分为 k 组，即： K＝K1 (i)∪K2 (i)∪…∪Kk (i) 对于状态集 Kj (i)（j=1,2,…,k）中的各个状态逐个检查，设有两个状态 Kj’、 Kj’’∈Kj (i)，且对于输入符号 a，有： F（Kj'，a）＝Km F（Kj''，a）＝Kn 如果 Km和 Kn属于同一个状态集合，则将 Kj’和 Kj’’放到同一集合中， 否则将 Kj’和 Kj’’分为两个集合。

（3） 重复第（2）步，直到每一个集合不能再划分为止，此时每个状态集 合中的状态均是等价的。

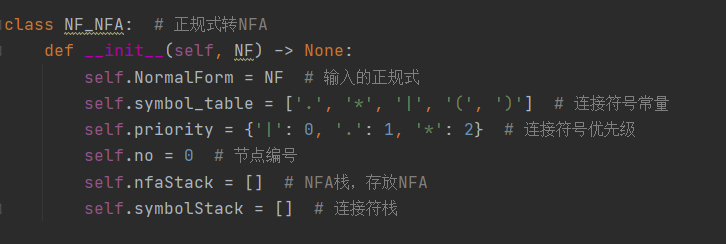
（4） 合并等价状态，即在等价状态集中取任意一个状态作为代表，删去其 他一切等价状态。

（5） 若有无关状态，则将其删去。 根据以上方法就将确定有穷自动机进行了简化，而且简化后的自动机是原 自动机的状态最少的自动机。

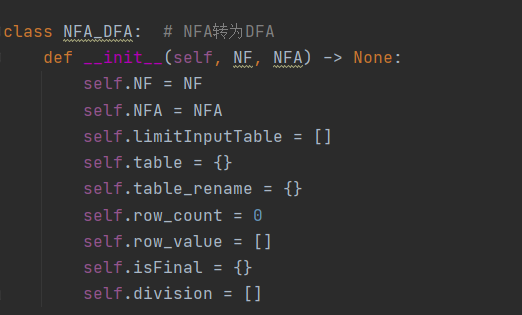
**四、实验内容**

本次实验使用python进行编程，读取一个文本文件，输出对应的NFA，DFA，及最小化DFA。

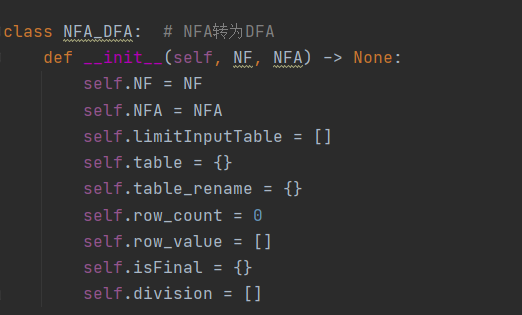
1.正规式转换成NFA



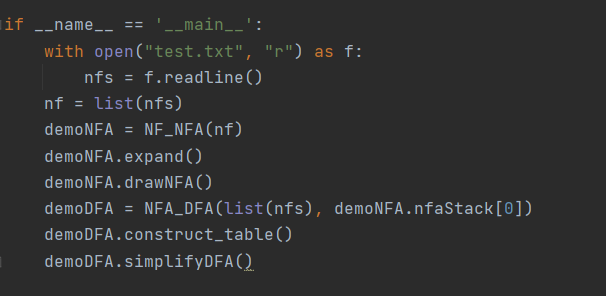
2. NFA到DFA的转换



3.子集法DFA最小化



4.获取NFA，DFA，最小化DFA并输出显示



**五、实验心得**

通过此次实验，我理解掌握了DFA子集法最小化，并用python实现了对任意正规式的处理，加深了对自动机的理解， 提高了编程能力。

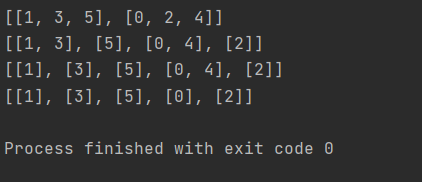
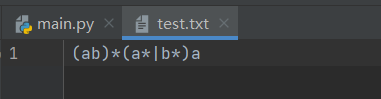
**六、实验代码与结果**

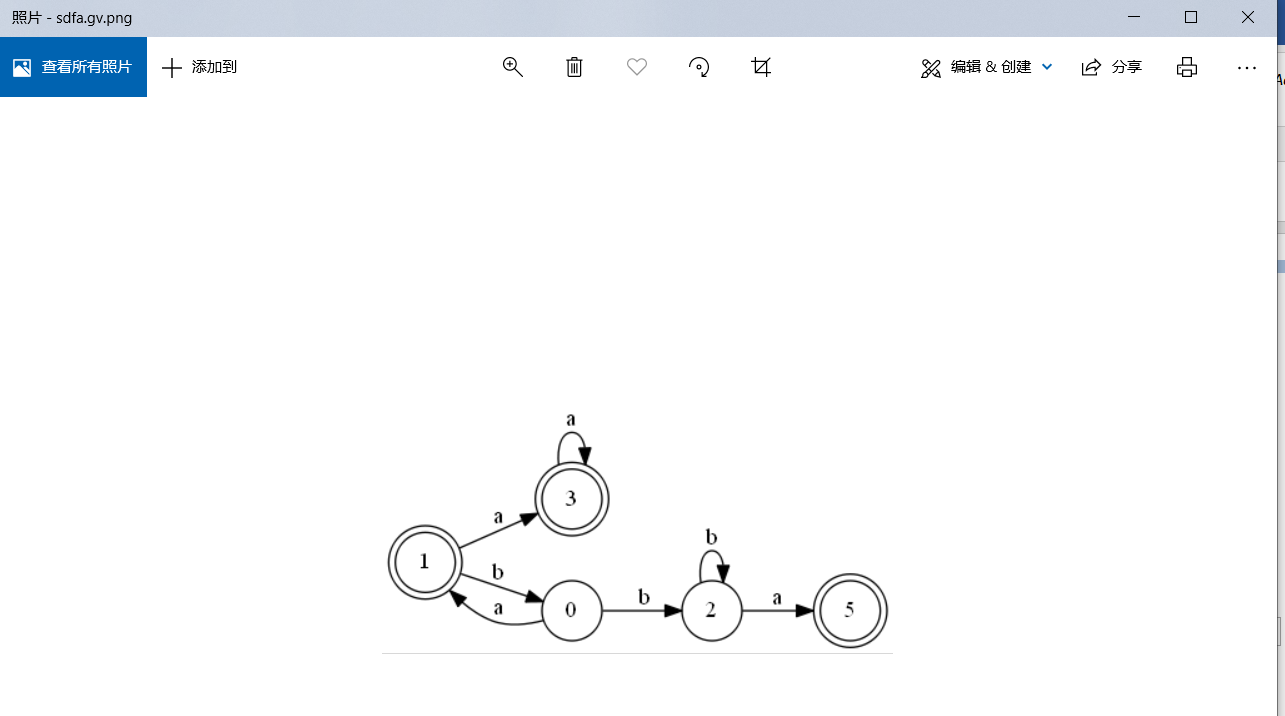
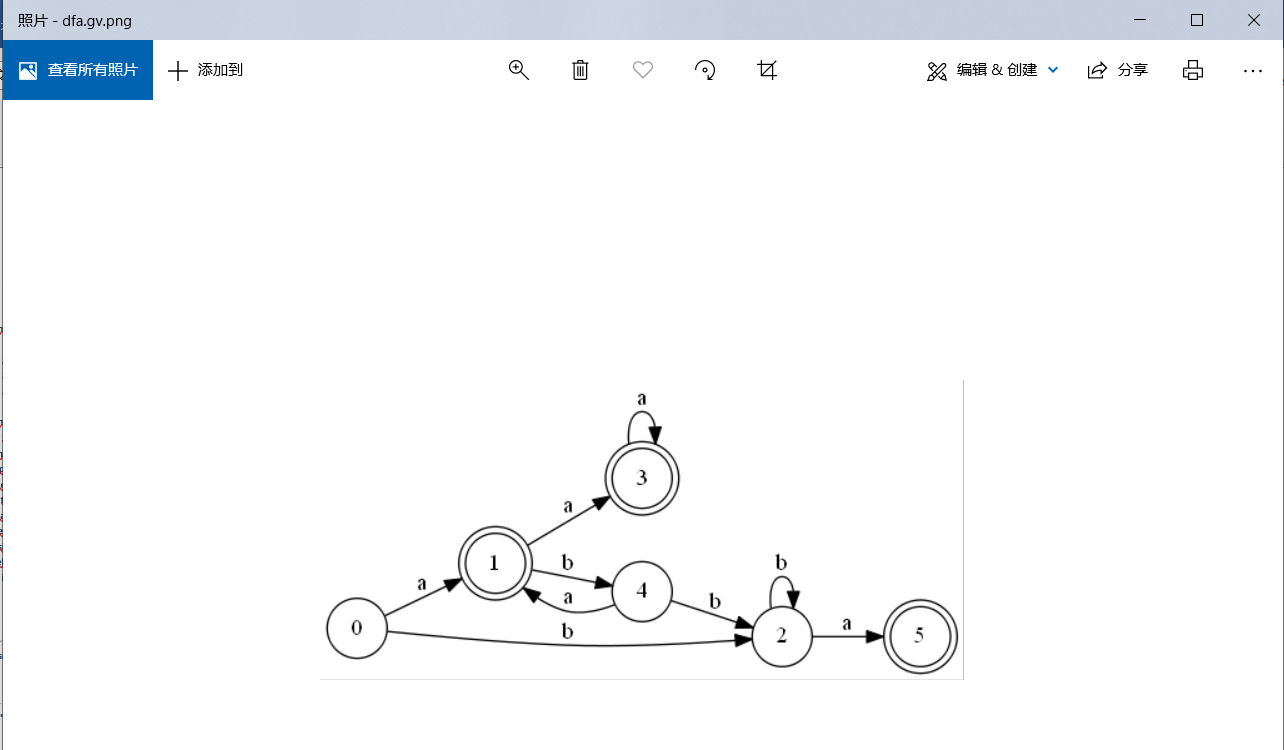
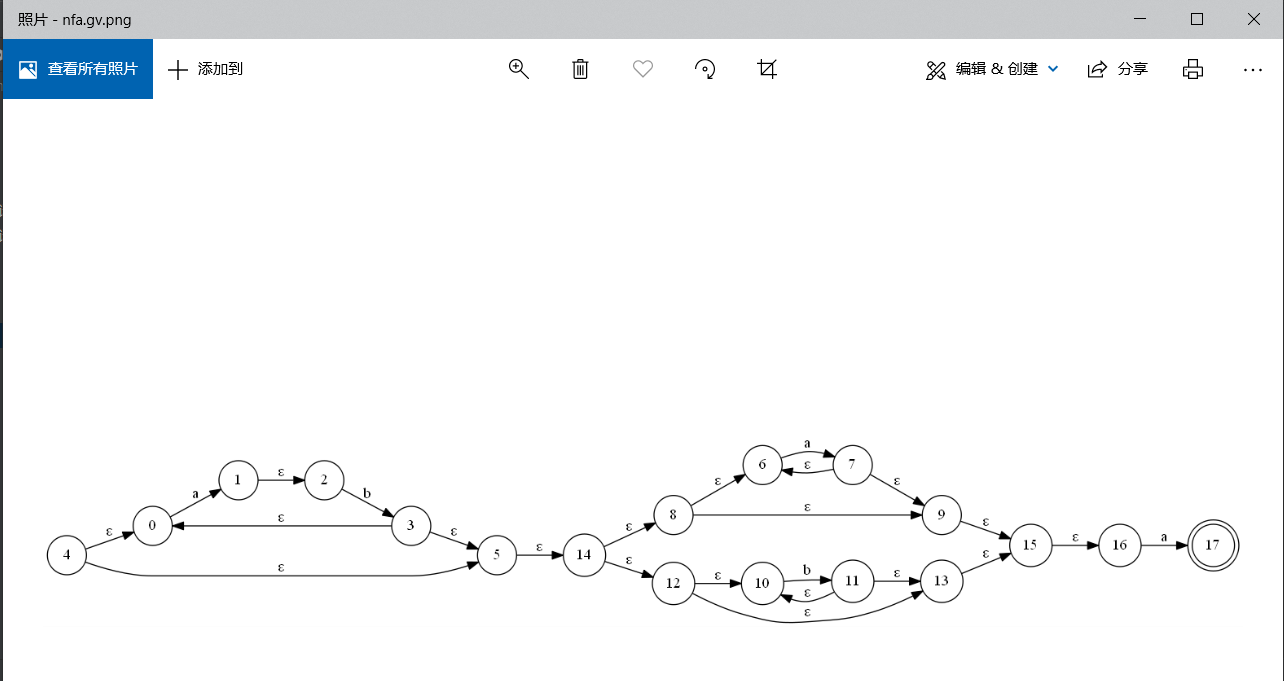
1.代码

**import graphviz  
  
  
class Node: # 节点类，描述节点的编号，节点的值，节点相连节点  
 def \_\_init\_\_(self, no, value) -> None:  
 self.no = no  
 self.value = value  
 self.friend = []  
  
  
class NFA: # NFA类，描述一个NFA的头节点和尾节点  
 def \_\_init\_\_(self, head, tail) -> None:  
 self.head = head  
 self.tail = tail  
  
  
class NF\_NFA: # 正规式转NFA  
 def \_\_init\_\_(self, NF) -> None:  
 self.NormalForm = NF # 输入的正规式  
 self.symbol\_table = ['.', '\*', '|', '(', ')'] # 连接符号常量  
 self.priority = {'|': 0, '.': 1, '\*': 2} # 连接符号优先级  
 self.no = 0 # 节点编号  
 self.nfaStack = [] # NFA栈，存放NFA  
 self.symbolStack = [] # 连接符栈  
  
 def expand(self):  
 i = 0  
 while i < len(self.NormalForm):  
 if self.NormalForm[i] not in self.symbol\_table or self.NormalForm[i] in [')', '\*']:  
 if i + 1 < len(self.NormalForm):  
 if self.NormalForm[i + 1] not in self.symbol\_table or self.NormalForm[i + 1] == '(':  
 self.NormalForm.insert(i + 1, '.')  
 i += 1  
 i += 1  
 print(self.NormalForm)  
  
 def NoAdd(self): # 节点编号增加  
 self.no += 1  
 return self.no - 1  
  
 def newNFAItem(self, value): # 创建新的NFA  
 head = Node(self.NoAdd(), 'ε')  
 tail = Node(self.NoAdd(), value)  
 head.friend.append(tail)  
 return NFA(head, tail)  
  
 def And(self, nfa1, nfa2): # 处理'.'符号，连接  
 nfa1.tail.friend.append(nfa2.head)  
 # print(nfa1.head.value, nfa2.tail.value)  
 return NFA(nfa1.head, nfa2.tail)  
  
 def Or(self, nfa1, nfa2): # 处理'|'符号，或  
 newHead = Node(self.NoAdd(), 'ε')  
 newTail = Node(self.NoAdd(), 'ε')  
 newHead.friend.append(nfa1.head)  
 newHead.friend.append(nfa2.head)  
 nfa1.tail.friend.append(newTail)  
 nfa2.tail.friend.append(newTail)  
  
 return NFA(newHead, newTail)  
  
 def closure(self, nfa): # 处理'\*'符号，闭包  
 newHead = Node(self.NoAdd(), 'ε')  
 newTail = Node(self.NoAdd(), 'ε')  
  
 newHead.friend.append(nfa.head)  
 nfa.tail.friend.append(newTail)  
 newHead.friend.append(newTail)  
 nfa.tail.friend.append(nfa.head)  
  
 return NFA(newHead, newTail)  
  
 def dealApart(self): # 对|和 . 进行分别处理  
 newNFA = None  
 ope = self.symbolStack.pop(-1)  
 nfa2 = self.nfaStack.pop(-1)  
 if ope == '|':  
 nfa1 = self.nfaStack.pop(-1)  
 newNFA = self.Or(nfa1, nfa2)  
 elif ope == '.':  
 nfa1 = self.nfaStack.pop(-1)  
 newNFA = self.And(nfa1, nfa2)  
 return newNFA  
  
 def paint(self, path): # 绘制NFA,  
 painter = graphviz.Digraph(path + '/nfa', format='png')  
 painter.attr(rankdir='LR')  
 startNode = self.nfaStack[0].head  
 endNode = self.nfaStack[0].tail  
 # painter.attr('node', shape='point')  
 # painter.node('start')  
 # painter.attr('node', shape='circle')  
 # painter.node(str(startNode.no))  
 # painter.edge('start', str(startNode.no), label=startNode.value)  
  
 paintStack = []  
 paintStack.append(startNode)  
 ban\_repeat = set()  
 ban\_repeat.add(startNode.no)  
  
 while len(paintStack) > 0:  
 currentNode = paintStack.pop(-1)  
 for node in currentNode.friend:  
 if node.no not in ban\_repeat:  
 paintStack.append(node)  
 ban\_repeat.add(node.no)  
 if node.no == endNode.no:  
 painter.attr('node', shape='doublecircle')  
 painter.node(str(node.no))  
 else:  
 painter.attr('node', shape='circle')  
 painter.node(str(node.no))  
 painter.edge(str(currentNode.no), str(node.no), label=node.value)  
 painter.view()  
  
 def drawNFA(self):  
 self.expand() # NFA绘制  
 for n in self.NormalForm:  
 if n in self.symbol\_table:  
 print(self.symbolStack, n, len(self.nfaStack))  
 newNFA = None  
 if n == ')':  
 while self.symbolStack[-1] != '(':  
 newNFA = self.dealApart()  
 self.nfaStack.append(newNFA)  
 self.symbolStack.pop(-1)  
 else:  
 if n == '\*':  
 newNFA = self.closure(self.nfaStack.pop(-1))  
 self.nfaStack.append(newNFA)  
 continue  
 if len(self.symbolStack) > 0:  
 if self.symbolStack[-1] != '(':  
 if n in self.priority:  
 while True:  
 if len(self.symbolStack) > 0 and self.priority[n] <= self.priority[  
 self.symbolStack[-1]]:  
 newNFA = self.dealApart()  
 self.nfaStack.append(newNFA)  
 else:  
 break  
 self.symbolStack.append(n)  
 else:  
 new = self.newNFAItem(n)  
 self.nfaStack.append(new)  
 while len(self.symbolStack) > 0: # 最后一个操作符处理  
 newNFA = self.dealApart()  
 if newNFA != None:  
 self.nfaStack.append(newNFA)  
 self.paint('/Users/bytedance/My/编译原理/compiler/src/views/NFADFA')  
  
  
class NFA\_DFA: # NFA转为DFA  
 def \_\_init\_\_(self, NF, NFA) -> None:  
 self.NF = NF  
 self.NFA = NFA  
 self.limitInputTable = []  
 self.table = {}  
 self.table\_rename = {}  
 self.row\_count = 0  
 self.row\_value = []  
 self.isFinal = {}  
 self.division = []  
  
 def getLimitTable(self): # 获取有穷输入列表  
 print(self.NF)  
 for n in self.NF:  
 if n not in ['(', ')', '|', '\*'] and n not in self.limitInputTable:  
 self.limitInputTable.append(n)  
  
 def expandNFA(self): # 在NFA头部和尾部加入节点  
 lastNo = self.NFA.tail.no  
 # newHead1 = Node(lastNo+1, 'ε')  
 newHead1 = Node(lastNo + 1, 'X')  
 # newHead2 = Node(lastNo+3, 'ε')  
 newTail2 = Node(lastNo + 2, 'ε')  
 newHead1.friend.append(self.NFA.head)  
 self.NFA = NFA(newHead1, self.NFA.tail)  
  
 self.NFA.tail.friend.append(newTail2)  
 self.NFA = NFA(self.NFA.head, newTail2)  
  
 def closure\_ε(self, target): # 求空弧闭包  
 result = []  
 result.append(target)  
 isChange = True  
 stack = []  
 stack.append(target)  
 stackno = []  
 stackno.append(target.no)  
 while len(stack) > 0:  
 top = stack.pop(-1)  
 for t in top.friend:  
 if t.value == 'ε' and t.no not in stackno:  
 stackno.append(t.no)  
 stack.append(t)  
 result.append(t)  
 return result  
  
 def move(self, row, arc): # 求move(I, a)  
 moveResult = []  
 moveResultNo = []  
 for r in row:  
 for f in r.friend:  
 if f.value == arc and f.no not in moveResultNo:  
 moveResultNo.append(f.no)  
 moveResult.append(f)  
 return moveResult  
  
 def getFirstRow(self): # 获取第一行的I  
 firstRow = self.closure\_ε(self.NFA.head)  
 return firstRow  
  
 def judgeFinal(self):  
 for row in self.table\_rename:  
 hasEndNode = False  
 for i in self.row\_value[row]:  
 if i.no == self.NFA.tail.no:  
 hasEndNode = True  
 break  
 if hasEndNode:  
 self.isFinal[row] = True  
 else:  
 self.isFinal[row] = False  
  
 def paint(self, path):  
 painter = graphviz.Digraph(path + '/dfa', format='png')  
 painter.attr(rankdir='LR')  
 for row in self.table\_rename:  
 hasEndNode = self.isFinal[row]  
 if hasEndNode:  
 painter.attr('node', shape='doublecircle')  
 painter.node(str(row), str(row))  
 else:  
 painter.attr('node', shape='circle')  
 painter.node(str(row), str(row))  
  
 for row in self.table\_rename:  
 for column in self.table\_rename[row]:  
 painter.edge(str(row), str(self.table\_rename[row][column]), label=column)  
  
 painter.view()  
  
 def construct\_table(self): # 维护状态转换表  
 self.getLimitTable()  
 self.expandNFA()  
  
 print(self.limitInputTable)  
  
 firstRow = self.getFirstRow()  
  
 self.row\_value.append(firstRow)  
 self.table[self.row\_count] = {}  
 self.row\_count += 1  
  
 isChange = True  
 while isChange: # 当表中信息不再改变时  
 isChange = False  
 tempRV = self.row\_value.copy()  
 for i in range(len(tempRV)):  
 for arc in self.limitInputTable:  
 if arc not in self.table[i]:  
 self.table[i][arc] = []  
 ###获取move(I, a)  
 moveResult = self.move(tempRV[i], arc)  
 ###获取closure\_ε  
 for m in moveResult:  
 closure = self.closure\_ε(m)  
 for c in closure:  
 if c not in self.table[i][arc]:  
 self.table[i][arc].append(c)  
 temp = self.table[i][arc].copy()  
 if len(temp) > 0:  
 if temp not in self.row\_value:  
 self.row\_value.append(temp)  
 self.table[self.row\_count] = {}  
 self.row\_count += 1  
 isChange = True  
 for row in self.table:  
 self.table\_rename[row] = {}  
 for column in self.table[row]:  
 for j, r in enumerate(self.row\_value):  
 if r == list(self.table[row][column]):  
 self.table\_rename[row][column] = j  
 break  
 print(self.table\_rename)  
 self.judgeFinal()  
 self.paint('D:\pycharm\pythonProject1')  
  
 def paintSDFA(self, address): # 绘制最小化的DFA  
 painter = graphviz.Digraph(address + '/sdfa', format='png')  
 painter.attr(rankdir='LR')  
 for d in self.division:  
 if self.isFinal[d[0]]:  
 painter.attr('node', shape='doublecircle')  
 painter.node(str(d[0]), str(d[0]))  
 else:  
 painter.attr('node', shape='circle')  
 painter.node(str(d[0]), str(d[0]))  
 for d in self.division:  
 tr = {}  
 for l in self.limitInputTable:  
 if l in self.table\_rename[d[0]]:  
 tresult = self.table\_rename[d[0]][l]  
 if [tresult] in self.division:  
 if tresult not in tr:  
 tr[tresult] = l  
 else:  
 tr[tresult] += ',' + l  
 for t in tr:  
 painter.edge(str(d[0]), str(t), label=tr[t])  
 painter.view()  
  
 def simplifyDFA(self): # DFA最小化  
 final = [] # 初始化非终态集  
 unfinal = [] # 初始化终态集  
 for row in self.isFinal:  
 if self.isFinal[row]:  
 final.append(row)  
 else:  
 unfinal.append(row)  
 self.division.append(final)  
 self.division.append(unfinal)  
  
 isChange = True  
 while isChange: # DFA状态不再改变时，停止循环  
 isChange = False  
 tempDivision = self.division.copy() # 浅复制  
 tempStore = []  
 print(tempDivision)  
 for divItem in tempDivision:  
 tempDivStore = []  
 if len(divItem) == 1: # 当子集长度为一时，不再划分  
 tempStore.extend([divItem])  
 continue  
 for limit in self.limitInputTable: # 当子集长度大于一时，继续判断  
 tryDiv = [] # 尝试划分列表  
 trRecord = {} # 记录相同状态  
 for div in divItem:  
 if limit in self.table\_rename[div]:  
 tr = self.table\_rename[div][limit]  
 if tr not in trRecord:  
 trRecord[tr] = []  
 trRecord[tr].append(div)  
 else:  
 if 'null' not in trRecord:  
 trRecord['null'] = []  
 trRecord['null'].append(div)  
 for key in trRecord:  
 if key != 'null':  
 temp = []  
 for t in trRecord[key]:  
 temp.append(t)  
 tryDiv.append(temp)  
 else:  
 for t in trRecord[key]:  
 tryDiv.append([t])  
 if len(tryDiv) > 1: # 如果得到的状态数大于一，则说明当前子集可再划分，退出当前循环  
 tempStore.extend(tryDiv)  
 tempDivStore.clear()  
 isChange = True  
 break  
 if len(tempDivStore) == 0: # 当前转换符时，不可再分  
 tempDivStore = tryDiv.copy()  
 if len(tempDivStore) > 0: # 遍历完所有转换符，都不可再分  
 tempStore.extend(tryDiv)  
 self.division.clear() # 浅复制  
 self.division = tempStore.copy()  
 tempStore.clear()  
 for i in range(len(self.division)):  
 if len(self.division[i]) > 1:  
 for t in self.table\_rename:  
 for j in self.table\_rename[t]:  
 if self.table\_rename[t][j] in self.division[i]:  
 self.table\_rename[t][j] = self.division[i][0]  
 self.division[i] = [self.division[i][0]]  
 print(self.division)  
 self.paintSDFA('/Users /My/编译原理')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with open("test.txt", "r") as f:  
 nfs = f.readline()  
 nf = list(nfs)  
 demoNFA = NF\_NFA(nf)  
 demoNFA.expand()  
 demoNFA.drawNFA()  
 demoDFA = NFA\_DFA(list(nfs), demoNFA.nfaStack[0])  
 demoDFA.construct\_table()  
 demoDFA.simplifyDFA()**

2.运行结果

（1）





（2）

