自动机与形式逻辑

-期末大作业

NFA/DFA 的转换工具

项目设计文档

V1.0

院　　系：软件学院
学　　号：2009220700
学生姓名：李守勤

目录

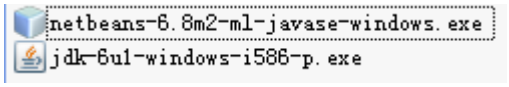
1. 项目设计简要介绍

1.1 项目设计要求

所选题目为 NFA/DFA 的转换工具
具体要求如下：
1.用 Java 编写，保证可扩展性。
2.检查某个字符串是否被指定的自动机接受。
3.判断语言是否为空，是否为无穷。
4.实现 NFA 到 DFA 的相互转换。
5.拟扩展实现将正则表达式转换为 NFA

1.2 开发环境

操作系统环境：Microsoft Windows XP
开发软件和开发包：



2. 概要设计

简述 NFA 转化为 DFA 的基本算法和实现思路。

2.1 把 NFA 转化为 DFA 的基本思想

当从NFA或ε-NFA开始并转化成DFA时，时间可能是NFA状态数的指数。首先，计算n个状态的ε-闭包要花费$O(n^3)$时间。必须从n种状态中每一个沿着所有标记ε的弧搜索。如果存在n种状态，则不可能超过n^2条弧。明智的簿记和设计良好的数据结构将确保能在$O(n^2)$时间里从每个状态中搜索。事实上，能用诸如沃舍尔（Warshall）算法的闭包算法来立刻计算整个ε-闭包。[1]

一旦计算出ε-闭包，就能用子集合构造来计算等价DFA。占支配地位的开销大体上是DFA的状态数，这可能是2^n。通过查询ε-闭包信息和每个输入符号的NFA转移表，就能在$O(n^3)$时间里对每种状态都计算出转移。换句话说，假设希望为DFA计算$\delta(\{q_1, q_2,\ldots, q_k\},a)$。从每个$q_i$沿ε标记路线可能有多达n个状态是可达的，这些状态中每个可能有至多n个弧标记为a。建立以状态为索引的数组，就能在与n^2成比例的时间里计算每个至多有n个状态的至多n个集合

[1] 对于传递闭包算法的讨论，参阅A·V·阿霍、J·E·霍普克罗夫特、和J·D·厄尔曼的*Data Structures and Algorithms*（《数据结构与算法》）一书，Addison-Wesley（爱迪生-卫斯理出版公司），1984。

的并。

以这种方式，就能对每个q_i计算出从q_i沿着路径标记a（可能包括ε）可达的状态集合。因为k≤n，至多存在n个状态要处理。对每个状态在$O(n^2)$时间里计算出可达状态。因此，计算可达状态所花费总时间是$O(n^3)$。可达状态集合的并只需要$O(n^2)$额外时间，结论是一个DFA转移的计算要花费$O(n^3)$时间。

注意假设输入符号数是常数且不依赖 n。因此，在对运行时间的这个估计和其他估计中，不考虑把输入符号数作为因子。输入字母表规模只影响“大 O”记号后面隐藏的常数因子，而不影响任何其他东西。

2.2．算法具体实现

一．子集构造法

1．先把 DFA M’中的 Q’和 F’置为空集；
2．M’的开始状态 q0’=[q0]，把[q0]置为未标记后加入到 Q’中；
3．如果 Q’中存在未标记的状态[q1,q2,…，qi]，则对每个 a∈Σ定义：δ’([q1,q2,…，qi],a)=[p1,p2,…,pi]当且仅当δ({q1,q2,…，qi},a)={p1,p2,…,pi}。
4．如果[q1,q2,…，qi]不在 Q’中，则把它置为为标记后加入到 Q’中；
5．如果 p1,p2,…,pi 中至少有一个是 M 的终态，则同时把[p1,p2,…,pi]加入到 F’中；然后给 Q’中所有的状态都标记为止；
6．重复执行上述（3）（4）（5）的步骤，直到不能向 Q’中加入新状态，并且 Q’中所有的状态都有标记为止；
7．重新命名 Q’中的状态，最后获得等价的 DFA M’。

二．对含 ε 变迁的 NFA 到 DFA 的转换

1．先把 DFA M’中的 Q’和 F’置为空集；

2．令 q_0’=[ε_CLOSURE({q_0})],并把[q_0]置为未标记后加入到 Q’中；

3．如果 Q’中存在未标记状态[q_1,q_2,…,q_i],则对每个 a∈Σ定义：d‘([q_1,q_2,…q_i],a)=[p_1,p_2,…,p_j]当且仅当 d({q_1,q_2,…q_i},a)={r_1,r_2,…,r_k}，ε_CLOSURE({r_1,r_2,…,r_k})= {p_1,p_2,…,p_j}。

4．如果[p_1,p_2,…,p_j]不在 Q’中，则把它置为未标记后加入到 Q’中；

5．如果 p_1,p_2,…,p_j 中至少有一个是 M 的终态，则同时把[p_1,p_2,…,p_j]加入到 F’中；然后给 Q’中的状态[q_1,q_2,…,q_i]加上标记；

6．重复执行上述（3）（4）（5）的步骤，直到不能向 Q’中加入新状态，并且 Q’中所有的状态都有标记为止；

7．重新命名 Q’中的状态，然后获得等价的 DFA M’

3．算法实现详细设计

3.1．算法实现的主要函数：

1. Find_epsilon_closure(q)
解释：从 NFA 的状态 q 出发，仅通过 ε 迁移能够到达的 NFA 的状态集合
2. Move(M,in)
解释：从一个集合 M 中包含的某个 NFA 的状态 q 出发，通过输入符号 in 迁移能够到达的 NFA 的状态集合。
3.2 具体实例实现步骤(此例子来源课本 P73)

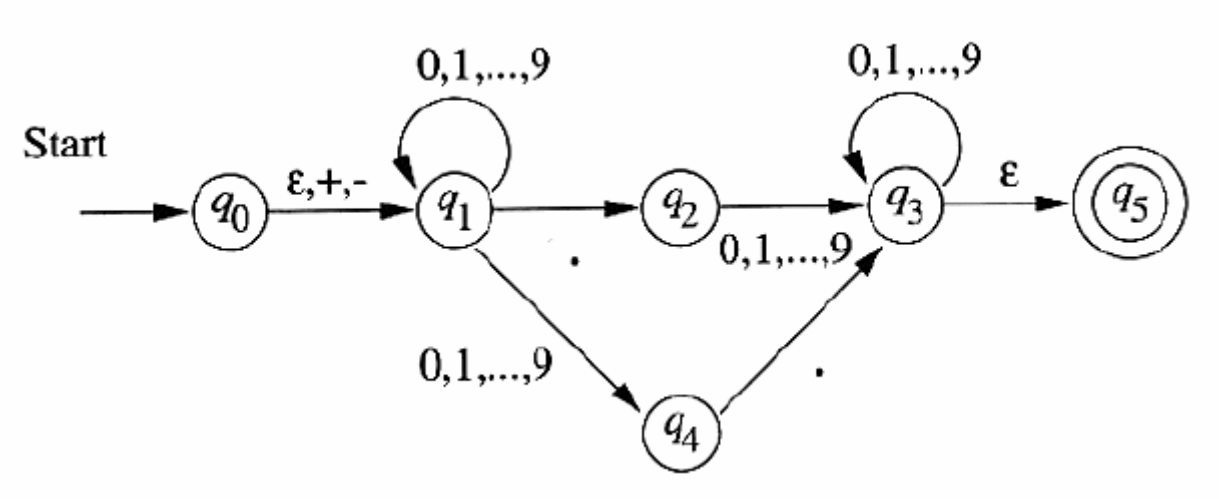


图 3.1 ε -NFA 状态转移图

初始仅有 Find_epsilon_closure (q0) ={q0, q1}。然后对于状态{q0, q1}，对于各种输入记号{+,-,.，0,1,…,9}，分别计算每种输入记号下的:
Find_epsilon_closure (Move({q0, q1}, +)) = Find_epsilon_closure ({q1}) = {q1};
Find_epsilon_closure (Move({q0, q1}, -)) = Find_epsilon_closure ({q1}) = {q1};
Find_epsilon_closure (Move({q0, q1}, 0,1,…,9)) = Find_epsilon_closure ({q1,q4}) = {q1,q4};
Find_epsilon_closure (Move({q0, q1},.)) = Find_epsilon_closure ({q1,q2}) = {q1,q2};
这样我们可以看到，得到了四个状态，分别是{q0, q1}，{q1}，{q1,q4}，{q1,q2}，其中{q0, q1}我们已经计算完毕，因此，计算新得到的其它三个状态的转移，直到计算完可能到达的状态，且没有新的状态产生。最后我们得到了下面的 DFA 的状态转移图：

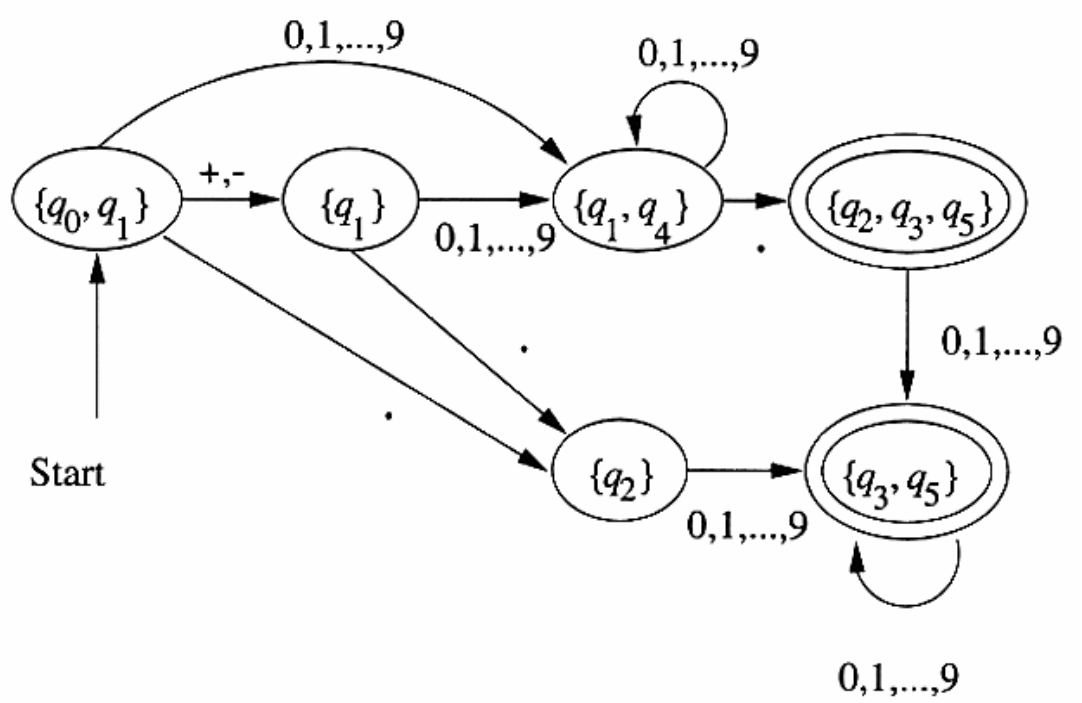


图 3.2 ε-NFA 转换后的 DFA 图

4.主要函数详解

Main类:

主函数运行类。

NFA类:

NFA 定义类。

类别	名称	详细描述
变量	int start	NFA 某个节点起始状态
	int end	NFA 某个节点结束状态

EDGE类:

EDGE 定义类。

类别	名称	详细描述
变量	int start	NFA/DFA 某个节点起始状态
	String input	转移字符串
	int end	NFA/DFA 某个节点结束状态

NFA2DFA类:

NFA 到 DFA 的转换实现类。（这里仅列举外部调用函数，程序中均有详细注释）

类别	名称	详细描述
外部接口	public void InitNFA(Vector<EDGE> edge,Vector<Integer> start,Vector<Integer> end)	将第一组同学设置的添加进来 NFA
	public void Process_NDATODFA()	执行 NFA 到 DFA 的转换
	public int getDFA_StartStatus()	获取转换后的 DFA 的起始状态
	public Vector<Integer> getDFA_EndStatus()	获取转换后的 DFA 的接受状态集合
	public Vector<EDGE> getDFA_TransStatus()	获取转换后的 DFA 的状态转移集合

5. 测试用例

测试用例：

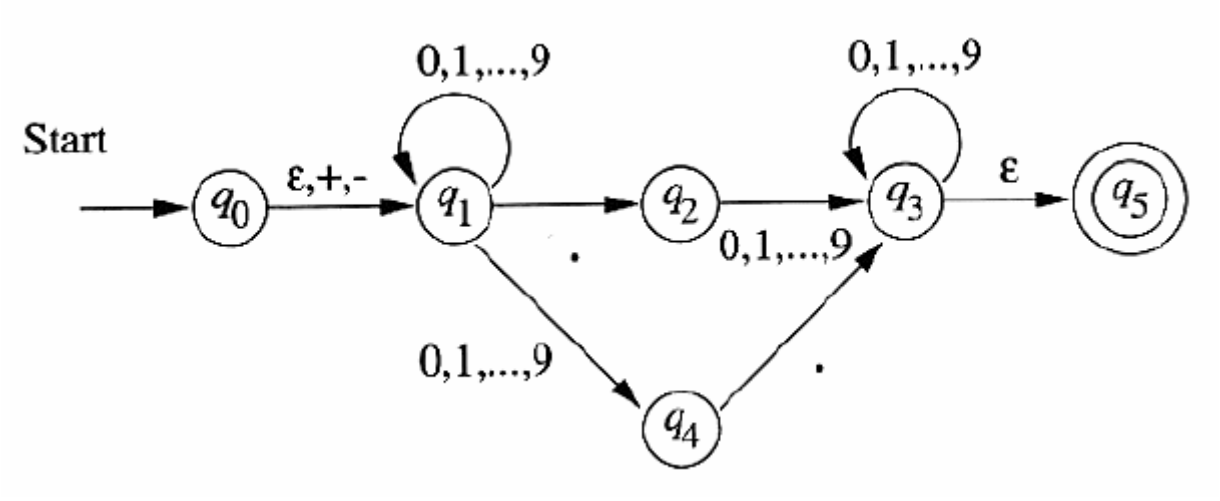


图 5.1 ε -NFA

6. 输出结果

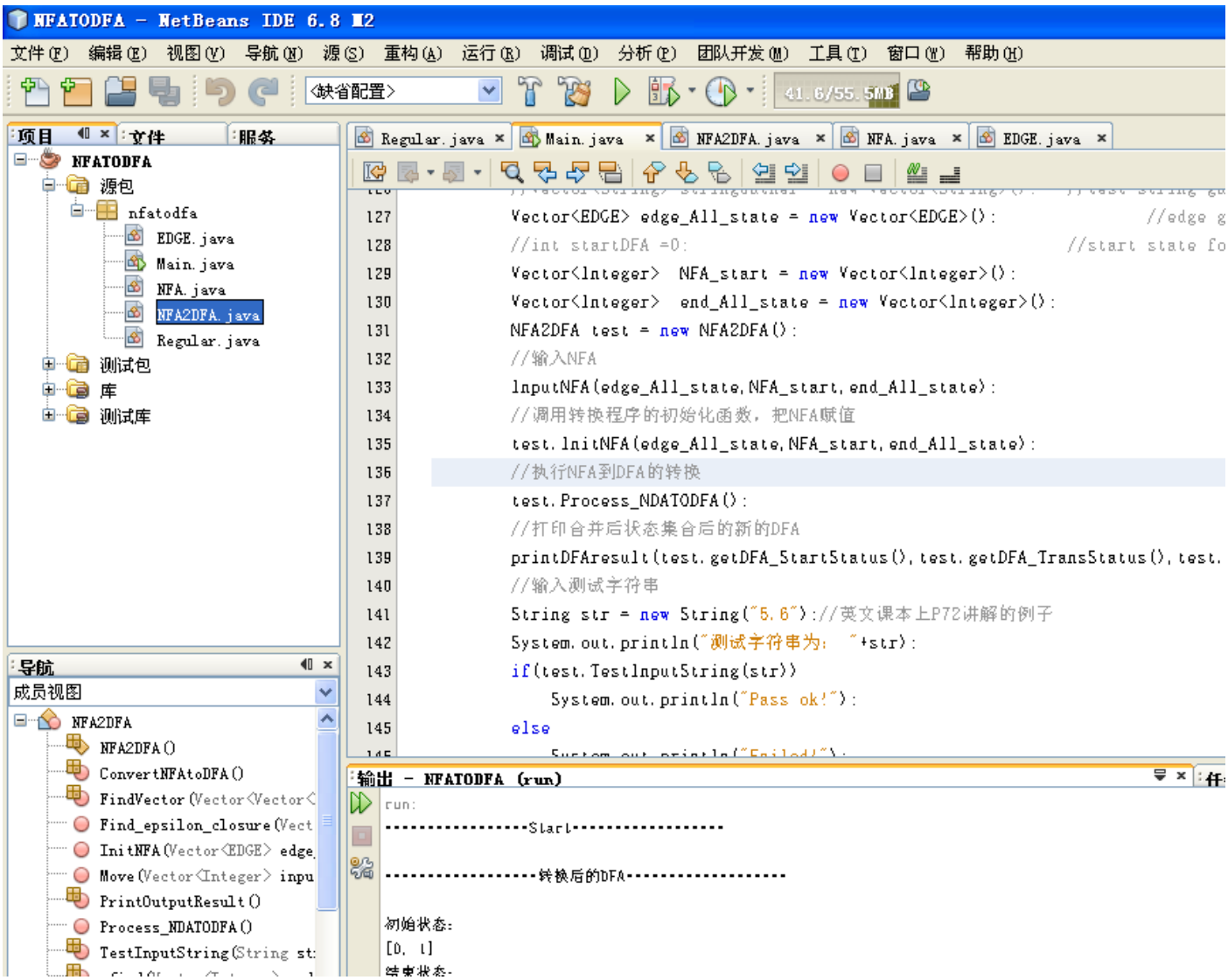


图 6 界面

NetBeans 开发环境下运行结果：（状态 q0 对应程序状态 0，以此类推）

```
==================转换后的 DFA====================
初始状态：
[0, 1]
结束状态：
[2, 3, 5]
[3, 5]
DFA 的状态转移表：
第 1 条边:
[0, 1]    +    [1]
第 2 条边:
[0, 1]    -    [1]
```

```
第 3 条边:
[0, 1]   0123456789   [1, 4]
第 4 条边:
[0, 1]   .   [2]
第 5 条边:
[1]   0123456789   [1, 4]
第 6 条边:
[1]   .   [2]
第 7 条边:
[1, 4]   0123456789   [1, 4]
第 8 条边:
[1, 4]   .   [2, 3, 5]
第 9 条边:
[2]   0123456789   [3, 5]
第 10 条边:
[2, 3, 5]   0123456789   [3, 5]
第 11 条边:
[3, 5]   0123456789   [3, 5]
==================End==================
```

```
==================重新命名 DFA 的状态==================
初始状态：
0
结束状态：
4
5
DFA 的状态转移表：
第 1 条边:
0   +   1
第 2 条边:
0   -   1
第 3 条边:
0   0123456789   2
第 4 条边:
0   .   3
第 5 条边:
1   0123456789   2
第 6 条边:
1   .   3
第 7 条边:
2   0123456789   2
第 8 条边:
2   .   4
```

```
第 9 条边:
3    0123456789    5
第 10 条边:
4    0123456789    5
第 11 条边:
5    0123456789    5
==================End==================
```

```
测试字符串为： 5.6
Pass ok!
```

7. 扩展功能实现：

将正则表达式转换为 NFA

算法 将正则表达式转换为 NFA(Thompson 构造法)

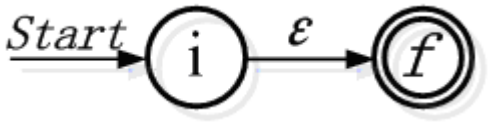
输入 字母表 Σ 上的正则表达式 r

输出 能够接受 *L(r)*的 NFA *N*

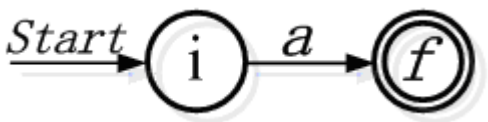
方法 首先将构成 *r* 的各个元素分解，对于每一个元素，按照下述**规则 1** 和**规则 2** 生成 NFA。

注意：如果 *r* 中记号 *a* 出现了多次，那么对于 *a* 的每次出现都需要生成一个单独的 NFA。之后依照正规表达式 r 的文法规则，将生成的 NFA 按照下述**规则 3** 组合在一起。

规则 1 对于空记号 ε，生成下面的 NFA。

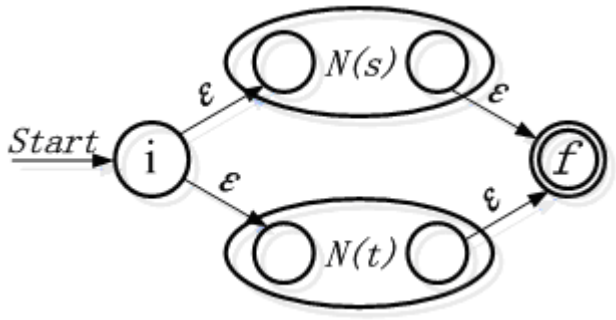


规则 2 对于 Σ的字母表中的元素 *a*，生成下面的 NFA。

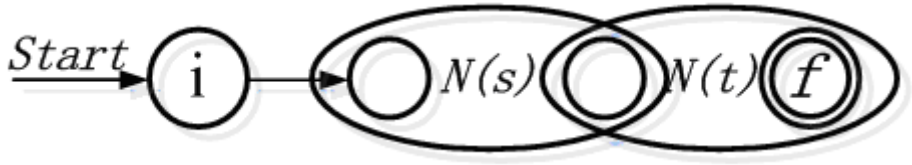


规则 3 令正规表达式 s 和 t 的 NFA 分别为 $N(s)$和 $N(t)$。

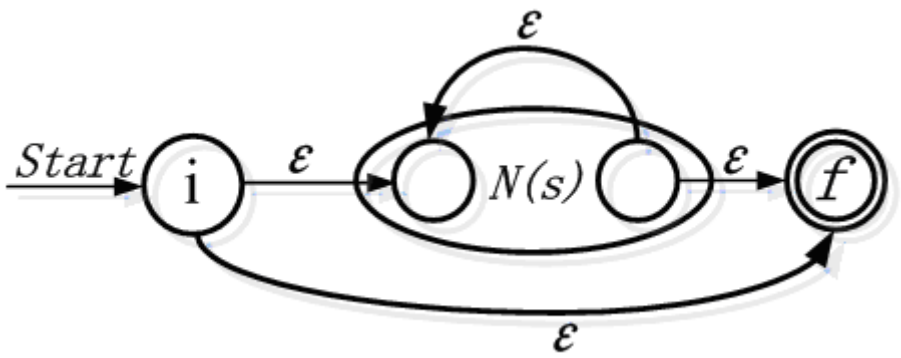
a) 对于 $s|t$，按照以下的方式生成 NFA $N(s|t)$。



b) 对于 st，按照以下的方式生成 NFA $N(st)$。



c) 对于 $s*$，按照以下的方式生成 NFA $N(s*)$。



d) 对于 (s)，使用 s 本身的 NFA $N(s)$。

程序输出结果如下：

```
==================输出 NFA==================
初始状态：
7
结束状态：
16
```

NFA 的状态转移表：
第 1 条边:
1 0 2
第 2 条边:
3 1 4
第 3 条边:
5 @ 1
第 4 条边:
5 @ 3
第 5 条边:
2 @ 6
第 6 条边:
4 @ 6
第 7 条边:
7 @ 5
第 8 条边:
6 @ 8
第 9 条边:
6 @ 5
第 10 条边:
7 @ 8
第 11 条边:
9 1 10
第 12 条边:
8 @ 9
第 13 条边:
11 0 12
第 14 条边:
13 1 14
第 15 条边:
15 @ 11
第 16 条边:
15 @ 13
第 17 条边:
12 @ 16
第 18 条边:
14 @ 16
第 19 条边:
10 @ 15
==========完毕==========

8 简单说明

1.本程序中的 ε 符号，均用字符 @ 来代替，这个符号也随时可以换成其他的，比如第一组同学说的 Empty。
2.正则表达式的运算符号有 * + ( )四个符号。
3.具体调用如下：（Main 类里面实现部分截取）

```
        //输入 NFA
        InputNFA(edge_All_state,NFA_start,end_All_state);
```

```
//把第一组定义的对应的 NFA 赋值给初始化函数，若是不能直接对接，只需要做简单转换。
比如：第一组同学要是定义的起始状态和结束状态是 String 类型，那么就做一个映射，就是 q0 对应状态 0，以此类推。
        //调用转换程序的初始化函数，把 NFA 赋值
        test.InitNFA(edge_All_state,NFA_start,end_All_state);
        //执行 NFA 到 DFA 的转换
        test.Process_NDATODFA();
        //打印合并后状态集合后的新的 DFA

        printDFAresult(test.getDFA_StartStatus(),test.getDFA_TransStatus(),test.getDFA
        _EndStatus());
        //输入测试字符串
        String str = new String("5.6");//英文课本上 P72 讲解的例子
        System.out.println("测试字符串为： "+str);
        if(test.TestInputString(str))
            System.out.println("Pass ok!");
        else
            System.out.println("Failed!");
        //add 2009.12.27 lishouqin
        //输入正则表达式
        Regular test_re = new Regular();
        String str_regular = new String("(0+1)*1(0+1)");//英文课本上 P100 讲解的例子
        test_re.InputRegular(str_regular);
        //输出结果
        test_re.process_re();
```