北京邮电大学
计算机学院(国家示范性软件学院)
2022-2023 学年第 2 学期
实验项目总结报告

课程名称：形式语言与自动机

项目名称：**NFA 转化为 DFA**

项目完成人：

负责人姓名：张梓良 学号：**2021212484**

姓名：张梓良 学号：**2021212484**

姓名：苗雨 学号：**2021212492**

姓名：杨晨 学号：**2021212171**

指导教师：杨正球 助教：杨惠元、句康

日　期：　**2023** 年　**4** 月　**5** 日

目录

一．项目目的和要求

实验目的

编程实现 NFA 到 DFA 的转化，理解不同自动机的转化过程。

实验要求

（1）采用分组实验，每组学生 3~4 人，班内自由组合，培养学生团队合作能力。
（2）编程语言要求使用 C/C++，需要使用头歌教学云平台进行测试验证代码，每组只需一人在平台上测试提交即可
（3）要求程序运行正确，设计风格好，文档描述清晰，并且按期提交小组实验报告，个人实验报告，源代码，及可执行程序。文件命名方式：组长班级+组长姓名+文件类型（报告/代码/程序），个人实验报告命名格式：班级+学号+姓名+个人实验报告。文件打包提交，命名方式：实验一+组长班级+组长姓名。
（4）实验报告至少包含以下内容：

① 小组成员，班级，姓名，学号；成员分工。
② 实验环境描述：所使用的语言等。
③ 程序的设计思路及核心算法。
④ 程序的输入，输出，以及执行效果（可截图）
⑤ 改进思路和方法（可选）

二 .项目开发环境

Windows 11 22H2

Visual Studio Code1.77

c++11

gcc version 10.2.0

三 .项目需求及设计方案

问题描述

本实验要求编程实现 NFA 到 DFA 的自动转化。输入设定的不确定有限自动机描述格式，输出对应的确定有限自动机。

设 NFA M=({q0，q1，q2……qn},{0,1}, δ ,q0,F)

1.输入 NFA 的转换函数表

2.将 NFA 转化为 DFA

3.去掉 DFA 中的不可达状态

4.输出 DFA 的转换函数表

其中，第一行输出 2 个"\t"，然后输出 0"\t"1。设 n 为转化后 DFA 的状态数，以下 n+1 行，每行输出为：状态，0 转移，1 转移，之间用"\t"间隔，第一行状态为起始状态 q0（用(s)q0 表示），之后状态如果是 NFA 中单独的状态则保留，产生的新状态按照其在输出的状态转移函数表的出现顺序（先 0 后 1，先上后下）命名为 qn,qn+1，qn+2……。状态顺序由小到大，终止状态前用(e)标识, ∅ 用 N 表示，NFA 的状态集用“[]”

样例输入			样例输出		
	0	1		0	1
(s)[q0]	[q1]	N	(s)q0	q1	N
[q1]	[q1,q2]	N	q1	q3	N
(e)[q2]	N	[q1]	(e)q3	q3	q1

设计方案

1.输入格式的处理

第一行的数据是固定的，因此并不是有效输入。第一列的原始状态下标是按行数递增的，因此不用特地存储，用数组或 vector 向量下标代替即可。

对于存储 NFA 的状态，以及其对应的的转移状态，考虑到有 2 个要素“自身”“输入 0\1 的转移状态”。可以用一个三维 vector 类型的变量 Q1 来存储，其中 Q1 的 size 就是 NFA 的状态个数。Q1[0][0]存储 q0 输入‘0’后转移的状态，Q1[0][1]存储 q0 输入‘1’后转移的状态，Q1[1][0]存储 q1 输入‘0’后转移的状

态……其中，若转移后的状态为‘N’，则用‘-1’来代替。

在实际处理过程中，新开两个临时变量 `vector<vector<int>> temp1;`和 `vector<int> temp2;`分别用于临时存储一个状态的所有转移后的状态集合以及每一个转移后的状态集合。

对于读入的每一行转移函数，采用 `getline(cin, s)`的方式以 string 的形式读入。对于读入的每一行字符串，首先使用 string 类的内置 find()函数查找‘e’判断是否为终止状态，再通过 find()函数查找‘]’快速定位到转移后的状态处，对转移后的状态进行处理，将它们的下标存储到 temp2 对应的位置中。

对于状态下标的处理，由于是采用字符串的形式读入的，因此需要将一串数字字符转换成对应的 int 型整数，首先采用 isdigit()函数判断当前字符是否为数字字符，再采用 `num = num * 10 + (int)(s[loc] - '0')`方式将数字字符转换成对应的数字。

对于读到空集，采用简单 if 语句判断当前字符是否为‘N’即可。若为‘N’，说明读到空集，将-1 存储到 temp2 对应的位置中。

对于区分输入‘0’和‘1’的不同转移状态，采用判断当前字符是否为‘]’的方式来判断是否已经将一个状态转移集合[q0,q1,q2...]读完，若已经读完，则将 temp2 整体存储到 temp1 中，并清空 temp2 用于下一个转移状态集合的存储。此时需要注意，读到空集属于特殊情况，不能采用判断是否读到‘]’的方式来判断一个状态集合是否已经读完，而应是读到‘N’就认为一个转移状态集合已经读完，再按上述方法进行处理。

最终，每处理完一行，将 temp1 整体存储到 Q1 中，并清空 temp1 和 temp2，进行下一行的处理。

2.转移函数的设计

起先想借鉴 NFA 转移函数的存储方法，即用 vector 嵌套。但是考虑到 DFA 中的状态经转移之后具有唯一性，形式比较简单，故采用以三元组（设计为结构体）为元素的一维 vector 来实现。

一个普遍的思路是对于 N 个 NFA 中的状态，找出所有可能包含的 2 的 N 次方个 DFA 状态，并推出状态之间的转换关系，再删去其中不可达的部分。但这种思路不利于新状态的编号，效率也较低。故采用一个工作队列，起先将 q0 入队。每次都从队头取出状态，按照规则进行 0\1 状态转移，若生成了未出现过的新状态，则将新状态加入状态列表和工作队列队尾，记录产生的状态转移规则并

加入状态转移函数，直到队列空为止。

每通过转移函数得到一个状态时需要判断其是否曾经出现。为了方便 DFA 中已有状态的查询，特别设计了状态列表 StateList，存储着 DFA 当前已经出现的所有状态（包括编号和对应的 NFA 中的状态集）。

考虑到 DFA 会出现新的状态，因而用一个 int 类型的变量 newnumber 来实时更新新状态的下标。显然，newnumber 要初始化为 Q1.size。

我们用 temp1 来存储每次的队头元素，其中 temp1.State 就是所包含的转移集合（转移后的编号集），temp1.number 是该状态的编号。

用 temp3 变量来记录转移后的状态信息。

接下来的 for 循环就是根据已有的状态转移集合，计算新的转移集合，并将新的转移集用 State 存储。起先选用 set 类来存储对应 NFA 中状态的集合，因为在进加入 NFA 中状态时不需要判重（set 类具有无重复元素、有序的良好性质）。但是考虑到 set 类在访问其中元素时需要使用迭代器，不方便状态集的比较和元素查询，故最终还是选用 vector 类来存储状态集。

之后，判断 State 是否是新出现的状态集，如果是已有的，则赋予已有的编号，否则赋予新编号，并检查是否是终结状态（有一个状态是终结状态，则该集合也是终结状态）。之后，将状态加入 temp3 形成转移后的状态信息。

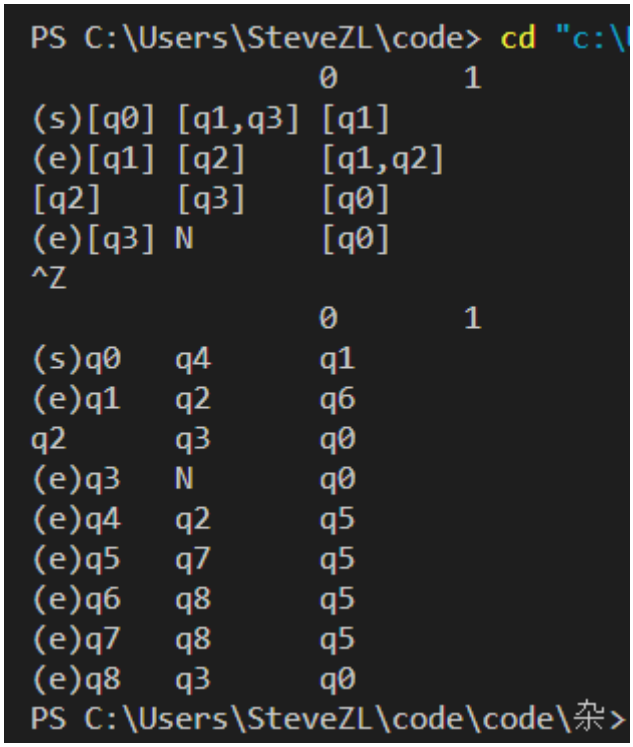
将 temp3 加入队列，继续循环。

3.输出函数的设计

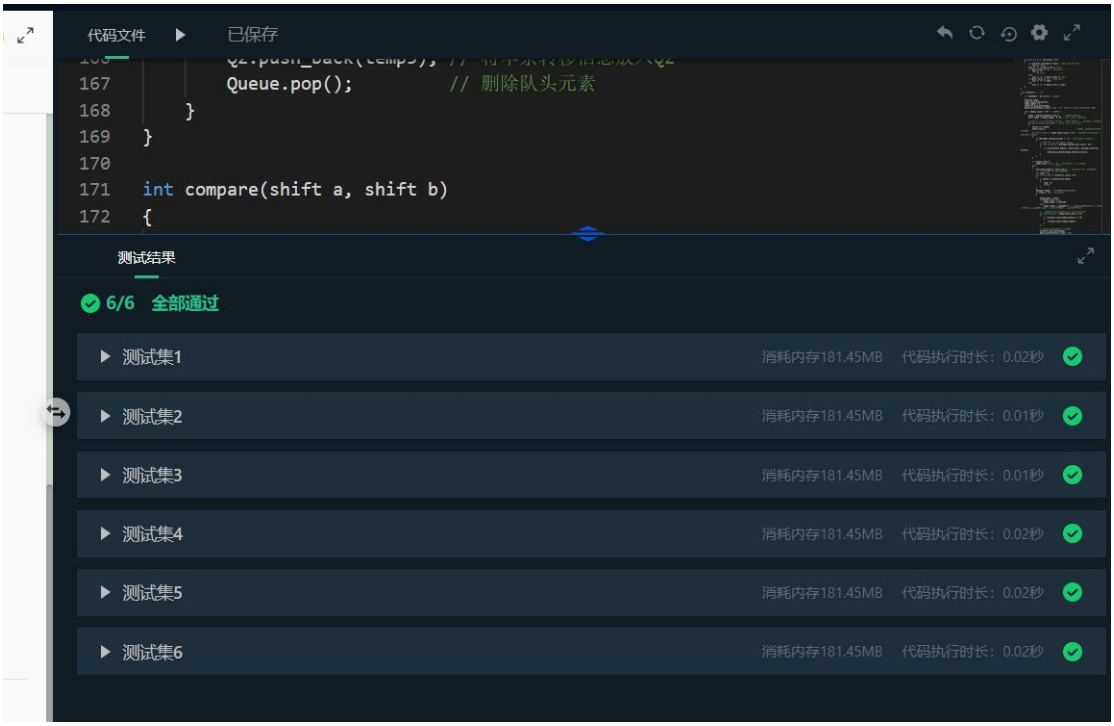
按照题目要求，将 Q2 中的状态集编号排序后，以表格的形式依次输出即可。需要注意的是，起始状态需要额外输出(s)，而终结状态需要额外输出(e)，遇到‘-1’需要输出‘N’。在每一行的输出中，首先判断原始状态是否为终止状态，通过对存储终止状态的集合 isfinal2 使用 set 内置函数 count()进行判断。其次再判断转移后的状态是否为空，通过使用 if 语句判断 Q2[i].to[j]是否为‘-1’进行判断。

四 .项目结果及分析

运行结果



头歌平台提交结果



五 .项目人员、进度安排及完成过程

程序的输入、输出设计：张梓良
NFA 到 DNA 转换函数的设计：苗雨
实验报告撰写：杨晨

六．改进的思路和方法（可选）

1. 可以考虑使用更具描述性的变量名，以便于理解代码。例如，可以将 Q1 和 Q2 改为 nfa_transitions 和 dfa_transitions。
2. 可以考虑使用更高效的数据结构来存储状态转移信息。例如，可以使用哈希表来存储状态转移信息，这样可以提高查找速度。
3. 可以考虑使用更高效的算法来实现 NFA 到 DFA 的转换。例如，可以使用子集构造法来实现转换。
4. 由于状态下标一定为非负整数，因此可以采用 unsigned int 的方式存储，这样非负整数的数据范围相较于采用 int 的方式存储扩大了一倍，对于大规模输入数据而言，更能防止溢出和错误。
5. 灵活应用 C++STL 容器进行数据存储，并采用其内置函数对数据进行处理，增强代码的可阅读性，降低程序的复杂性。