



**NFA到DFA的转化实验报告**

**学院：计算机学院（国家示范性软件学院）**

**班级： 2022211305**

**小组： 第七小组**

**组长： 张晨阳 2022211683**

**组员： 廖轩毅 2022211637**

**梁维熙 2022211124**

**金建名 2022211130**

**分工： 张晨阳：完成核心算法实现，优化报告**

**梁维熙：实现用户输入交互界面**

**廖轩毅：优化代码，完善结果输出**

**金建名：进行案例测试，完成报告编写**

**2024年4月6日**

**目录**

[一、实验环境描述及使用编程语言 1](#_Toc163402122)

[二、程序设计思路及核心算法 1](#_Toc163402123)

[1. 设计思路 1](#_Toc163402124)

[2. 核心算法 1](#_Toc163402125)

[3.程序输入描述 2](#_Toc163402126)

[4. 程序输出解释 3](#_Toc163402127)

[三、调试分析 3](#_Toc163402128)

[1. 算法复杂度分析 3](#_Toc163402129)

[2. 改进思路和方法 3](#_Toc163402130)

[四、测试结果 4](#_Toc163402131)

[测试集1： 4](#_Toc163402132)

[测试集2： 5](#_Toc163402133)

[测试集3： 5](#_Toc163402134)

# 实验环境描述及使用编程语言

操作系统：windows 11

调试软件：Visual Studio Code 1.88.0

代码实现语言：C++17

# 程序设计思路及核心算法

## 设计思路

将含有空转换的NFA转化为不含空转换的NFA，再将NFA转化为DFA。

## 核心算法

此程序的核心算法为将NFA转化为DFA。

**DFA的组成：**

1. DFA每个状态对应NFA的一个状态集合；
2. DFA的输入字符和NFA相同；
3. DFA的初始状态和NFA相同；
4. 状态转移表、函数为DFA中的每一个状态及每一个输出，定义一个转换新状态
5. DFA的终止状态：DFA中包含NFA至少一个终止状态的状态皆为终止状态

**数据结构设计：**

1. 状态：使用std::set<int>表示DFA中的一个状态；
2. 转换表：使用std::map<std::pair<std::set<int>,char>,std::set<int>>储存从一个状态集合通过一个输入符号到另一个状态集合的映射；
3. ε闭包：通过BFS算法，构造一个函数计算而得；

**算法步骤：**

1. 输入NFA；
2. 计算ε闭包；
3. 初始化DFA：使用NFA的初始状态的ε闭包作为DFA的初始状态；
4. 构建DFA：使用BFA算法，遍历每一个状态集合和输入符号，求得转换后的ε闭包，并且更新DFA转换表。若遇到新的状态集合，就将其添加到队列中；
5. 输出DFA；

## 3. 程序输入描述

以下描述中**字符串**均特指不包含空格、回车等特殊字符的 ASCII 字符序列。**要求输入是一个合法的 NFA**。

第一行输入若干数字，用空格分隔，设状态总数为n，使用数字 -1作为终止符，表示 NFA 的状态集合，**要求输入不出现重复状态。**

第二行输入若干字符串，用空格分隔，使用字符 # 作为终止符，表示 NFA 的字母表，**要求输入不出现重复符号**，**注意**，如果输入的是 *ε*-NFA，则需要在本行最后输入字符e表示*ε*。

接下来输入状态转换函数，若当前状态转换函数还没有输入完毕，则输入1表示还需要继续输入，若已经输入完毕，则输入0表示输入完毕。

以下输入具体的状态转化函数：

以三次输入为一组，第一次输入一个数字表示原状态，第二次输入一个转换操作符，第三次输入转换后的状态，由于NFA中转换后的状态不一定只有一个，则需要输入多个数字表示状态，用空格间隔，用 -1作为终止符。而后循环上述过程，直到状态转化函数输入完毕，输入0表示输入完毕。

接下来是初始状态输入，仅需输入一个数字表示初始状态即可。

最后是终止状态输入，由于可能不止一个终止状态，故输入一串数字表示状态，使用空格隔开，并使用 -1作为终止符。

## 程序输出解释

假设输出共有n行（n>=2），则输出的前n-2行表示生成的DFA中新的状态转化函数，e.g. [0] --a--> [0,1] 表示DFA中状态 [0] 经由操作符a转换后生成状态 [0,1] ；第n-1行表示此DFA的初始状态；第n行表示此DFA的所有终止状态。

# 调试分析

## 算法复杂度分析

1. ε-NFA到NFA的转换（即计算ε 闭包）

采用了BFS算法，时间复杂度最差为 ，空间复杂度为O（n）；

1. 从NFA状态集合 + 输入符号计算 DFA状态

时间复杂度最差为 ，空间复杂度为O（n）

## 改进思路和方法

由于 NFA 不合法的情况太多，难以一一判断，所以默认输入的是合法的 NFA，没有做过多处理；

解决方法：添加错误提示：若检测到错误的输入，清空当前缓存区，让用户重新输入；

对于程序输出还可以美化，例如使用表格表示状态转换表，增强输出可读性。

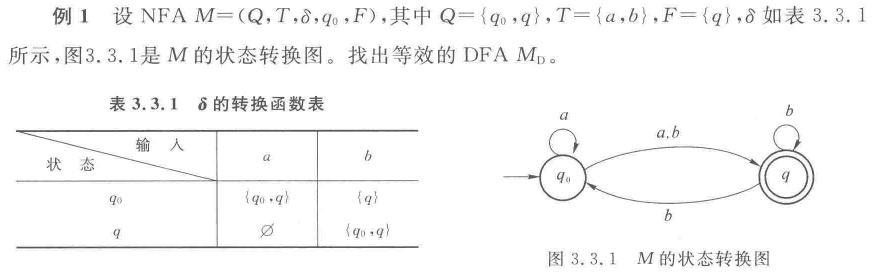
# 测试结果

### 测试集1：

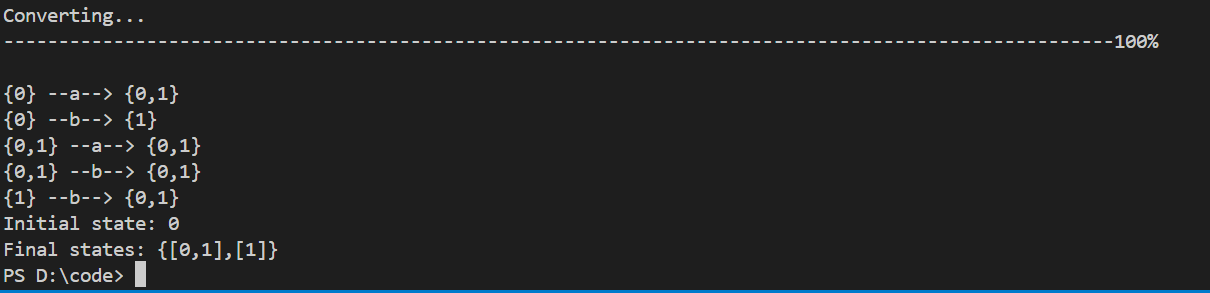
**输入：**

|  |
| --- |
| **0 1 -1** |
| **a b #** |
| **1** |
| **0** |
| **a** |
| **0 1 -1** |
| **1** |
| **0** |
| **b** |
| **1 -1** |
| **1** |
| **1** |
| **b** |
| **0 1 -1** |
| **0** |
| **0** |
| **1 -1** |

**输入解释：**

****

**输出：**

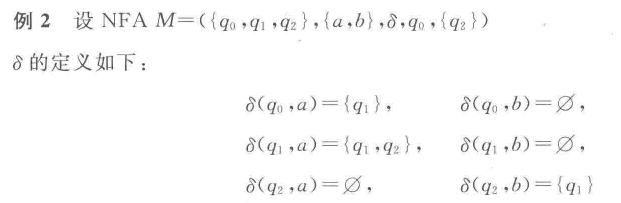


### 测试集2：

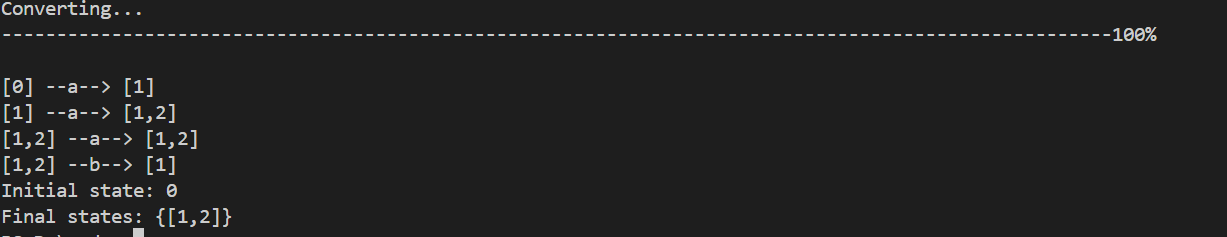
**输入：**

|  |
| --- |
| **0 1 2 -1** |
| **a b #** |
| **1** |
| **0** |
| **a** |
| **1 -1** |
| **1** |
| **1** |
| **a** |
| **1 2 -1** |
| **1** |
| **2** |
| **b** |
| **1 -1** |
| **0** |
| **0** |
| **2 -1** |

**输入解释：**



**输出：**

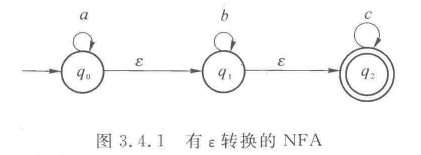


### 测试集3：

**输出：**

|  |
| --- |
| **0 1 2 -1** |
| **a b c e #** |
| **1** |
| **0** |
| **a** |
| **0 -1** |
| **1** |
| **0** |
| **e** |
| **1 -1** |
| **1** |
| **1** |
| **b** |
| **1 -1** |
| **1** |
| **1** |
| **e** |
| **2 -1** |
| **1** |
| **2** |
| **c** |
| **2 -1** |
| **0** |
| **0** |
| **2 -1** |

**输入解释：**

****

**输出：**

