|  |  |
| --- | --- |
| **成绩** |  |

**2025年《程序设计语言原理》**

**实验报告**

**班级： 34082301**

**姓名： 高浚然**

**学号： 2023215319**

**指导老师： 陈霖**

**实验名称： Lab1**

**实验地点： A503**

## 实验一：

1. **实验题目**

NFA转换DFA

1. **实验目的**

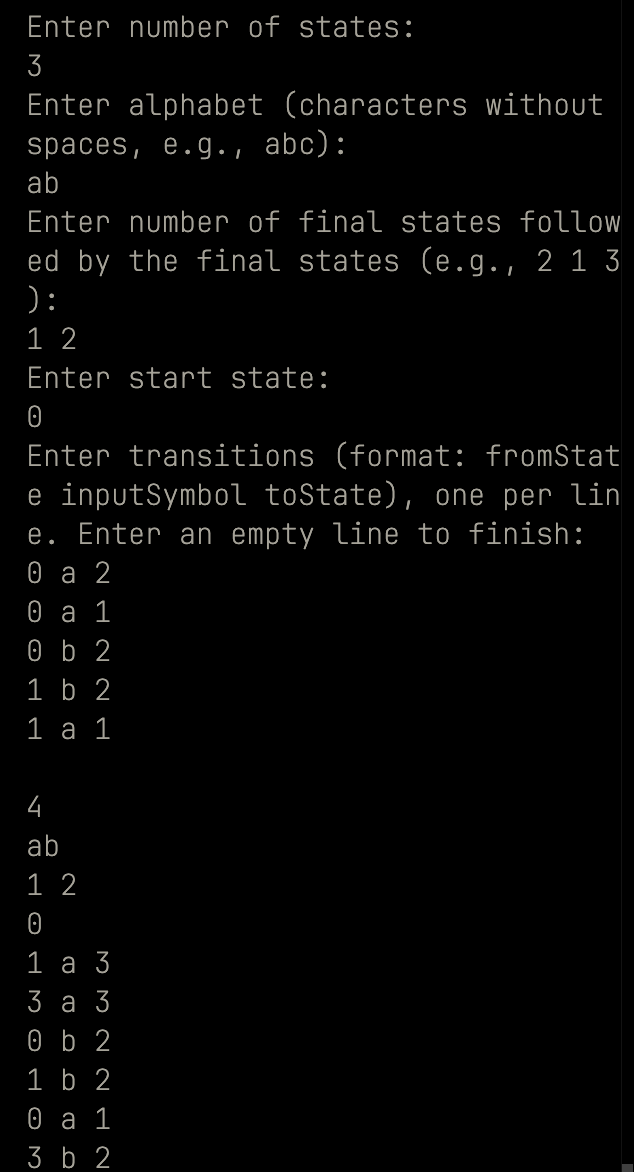
**使用Java实现NFA到DFA的转换，实现输入数据 -> 构建DFA -> 核心逻辑（构建队列） -> 组装DFA -> 输出数据的流程。**

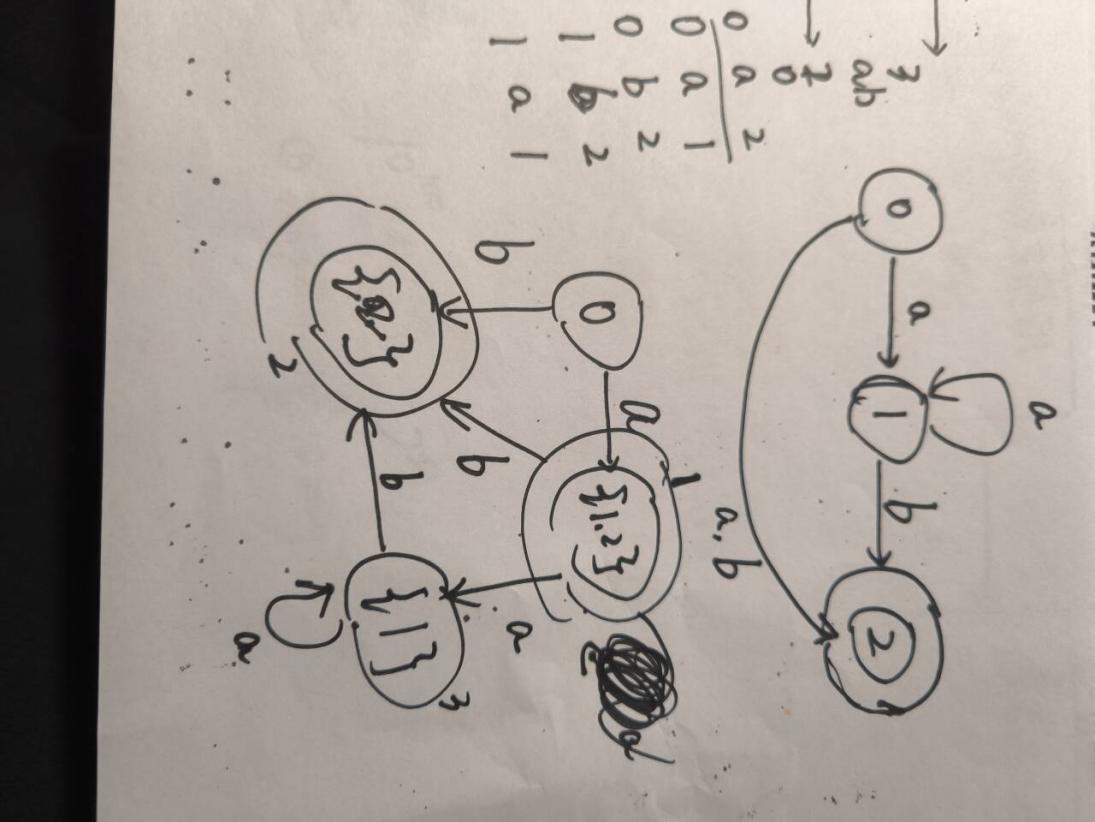
1. **实验内容**

**核心步骤：我们遍历字母表，从初始状态开始，对每个字母的输入找到得到状态的集合（对初始状态的集合中的每个字母输入得到的集合取并集）。得到的集合将被转换成一个从0开始的编号，作为DFA中的状态。对于不在现有状态集中的集合，将被放回循环的下一轮，重复操作直到所有集合均在现有的状态集中。**

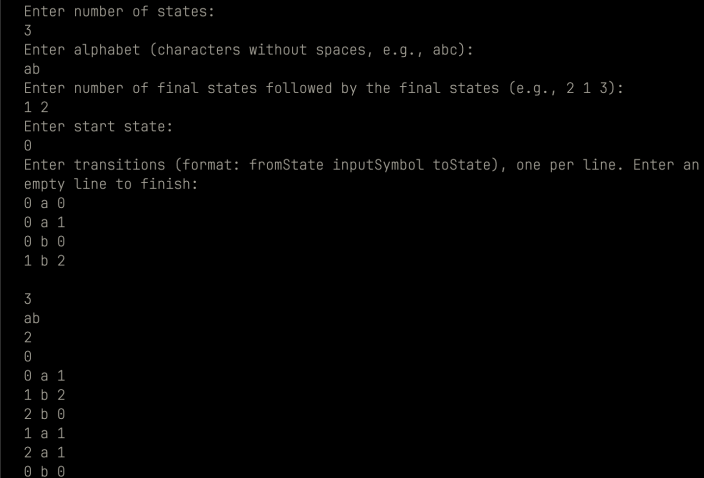
1. **实验步骤**
2. **构建基本类：NFA、DFA、Transition**
3. **设计输入、输出逻辑**
4. **绘制流程图（实际的NFA、DFA状态图），找到三个测试用例，梳理流程**
5. **编写核心逻辑**
6. **实验结果与分析**

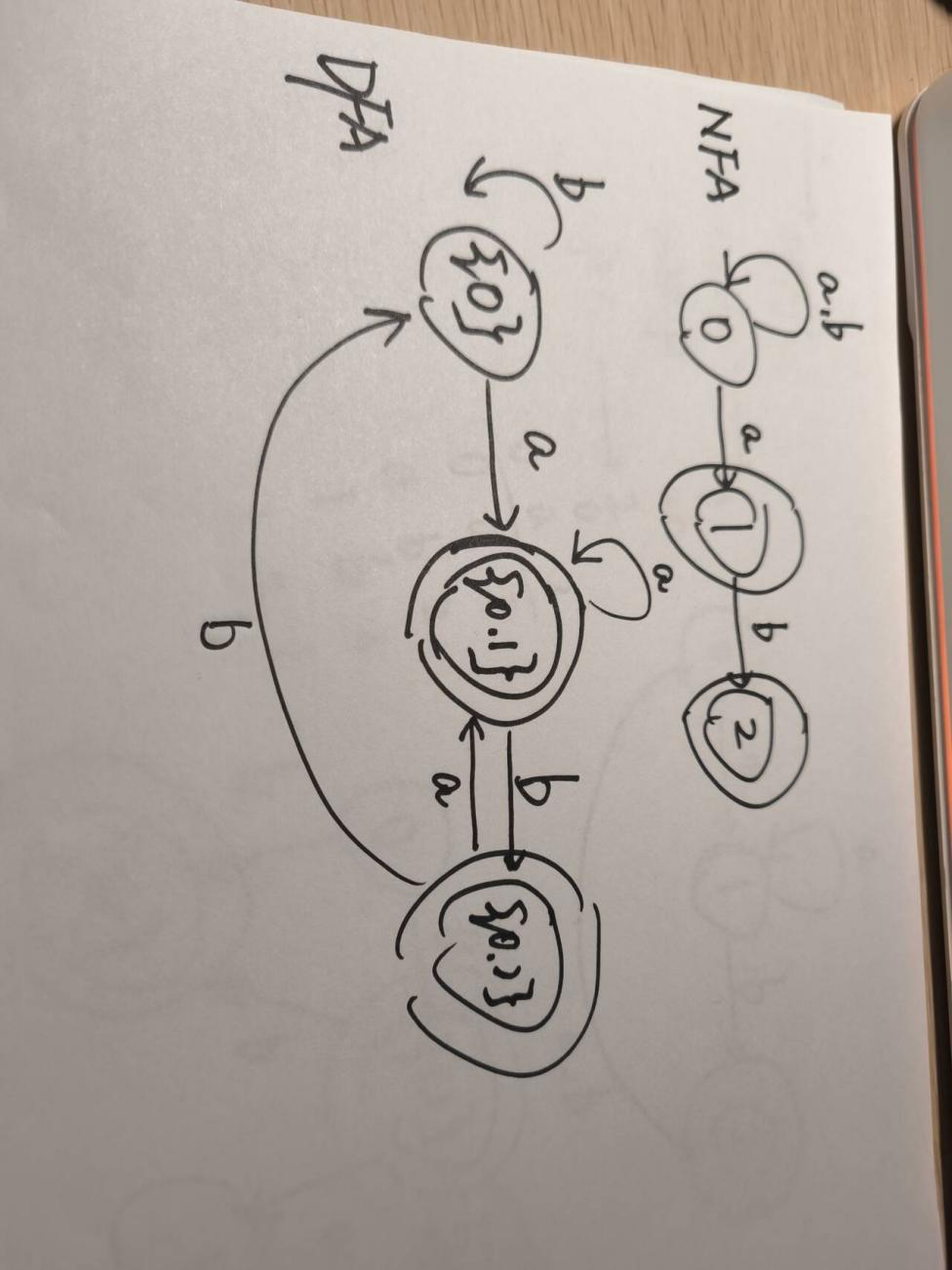
**测试数据1：**

****

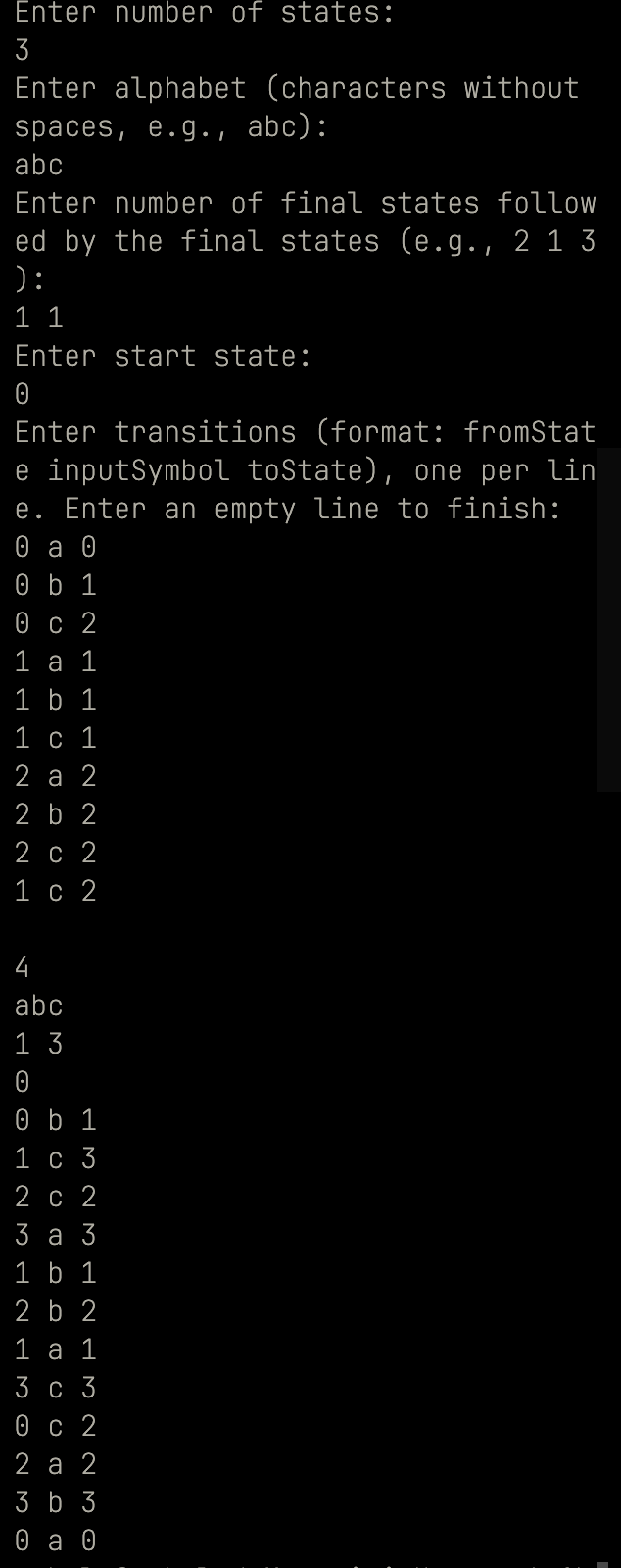
****

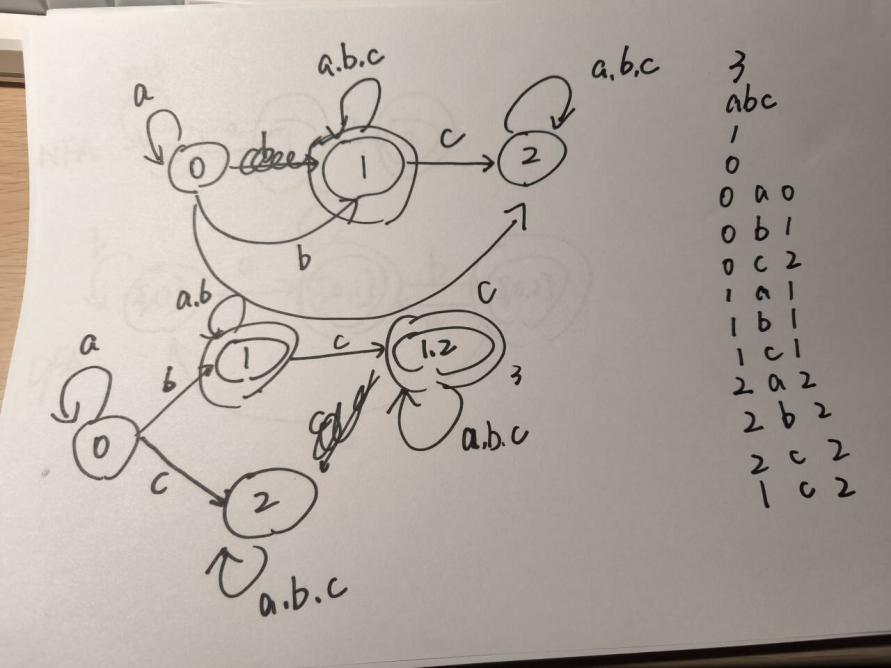
**测试数据2：**



****

**测试数据3：**

****

****

1. **核心代码清单（只写代码核心）**

// package javasrc;

import java.util.\*;

/\*\*

\* Represents a Non-Deterministic Finite Automaton (NFA).

\* This class encapsulates the structure and behavior of an NFA.

\*/

class NFA {

int cardinality; // Number of states in the NFA

Set<Character> alphabet; // Set of input symbols

Set<Integer> finalStates; // Set of final states

int startState; // Start state of the NFA

Set<Transition> transitions; // Set of transitions

/\*\*

\* Constructs an NFA with the given parameters.

\*

\* @param cardinality Number of states in the NFA

\* @param alphabet Set of input symbols

\* @param finalStates Set of final states

\* @param startState Start state of the NFA

\* @param transitions Set of transitions

\*/

public NFA(int cardinality, Set<Character> alphabet, Set<Integer> finalStates, int startState,

Set<Transition> transitions) {

this.cardinality = cardinality;

this.alphabet = alphabet;

this.finalStates = finalStates;

this.startState = startState;

this.transitions = transitions;

}

}

/\*\*

\* Represents a transition in an automaton.

\* This class encapsulates the from state, input symbol, and to state of a

\* transition.

\*/

class Transition {

int fromState; // Source state of the transition

char input; // Input symbol of the transition

int toState; // Destination state of the transition

/\*\*

\* Constructs a transition with the given parameters.

\*

\* @param fromState Source state of the transition

\* @param input Input symbol of the transition

\* @param toState Destination state of the transition

\*/

public Transition(int fromState, char input, int toState) {

this.fromState = fromState;

this.input = input;

this.toState = toState;

}

}

/\*\*

\* Represents a Deterministic Finite Automaton (DFA).

\* This class encapsulates the structure and behavior of a DFA.

\*/

class DFA {

int cardinality; // Number of states in the DFA

Set<Character> alphabet; // Set of input symbols

Set<Integer> finalStates; // Set of final states

int startState; // Start state of the DFA

Set<Transition> transitions; // Set of transitions

/\*\*

\* Constructs a DFA with the given parameters.

\*

\* @param cardinality Number of states in the DFA

\* @param alphabet Set of input symbols

\* @param finalStates Set of final states

\* @param startState Start state of the DFA

\* @param transitions Set of transitions

\*/

public DFA(int cardinality, Set<Character> alphabet, Set<Integer> finalStates, int startState,

Set<Transition> transitions) {

this.cardinality = cardinality;

this.alphabet = alphabet;

this.finalStates = finalStates;

this.startState = startState;

this.transitions = transitions;

}

/\*\*

\* Outputs the DFA structure.

\* Prints the number of states, alphabet, final states, start state, and

\* transitions.

\*/

public void output() {

System.out.println(cardinality);

for (char c : alphabet) {

System.out.print(c);

}

System.out.println();

for (int state : finalStates) {

System.out.print(state + " ");

}

System.out.println();

System.out.println(startState);

for (Transition t : transitions) {

System.out.println(t.fromState + " " + t.input + " " + t.toState);

}

}

}

/\*\*

\* Converts an NFA to a DFA.

\* This class provides a method to convert an NFA to an equivalent DFA using the

\* subset construction algorithm.

\*

\* @author Gao Junran

\*/

public class NfaToDfa {

/\*\*

\* Converts an NFA to a DFA.

\*

\* @param nfa The input NFA to be converted

\* @return The resulting DFA

\*/

public static DFA convertNFAToDFA(NFA nfa) {

Set<Transition> dfaTransitions = new HashSet<>(); // Set of transitions for the resulting DFA

Map<Set<Integer>, Integer> stateMapping = new HashMap<>(); // Maps NFA state sets to DFA state IDs

Queue<Set<Integer>> queue = new ArrayDeque<>(); // Queue to process subset states

// Initialize the start state of the DFA

Set<Integer> startSet = new HashSet<>();

startSet.add(nfa.startState); // Initial DFA state from NFA start state

queue.add(startSet);

stateMapping.put(startSet, 0);

Set<Integer> dfaFinalStates = new HashSet<>(); // Set of final states in the DFA

Set<Character> dfaAlphabet = nfa.alphabet; // Alphabet of the DFA is the same as the NFA

int stateCounter = 1; // Counter for DFA state IDs

// Process each subset state

while (!queue.isEmpty()) {

Set<Integer> currentStateSet = queue.poll(); // Get the current subset state

int dfaStateId = stateMapping.get(currentStateSet); // Get the DFA state ID for the current subset state

// Check if the current DFA state contains any NFA final states

if (!Collections.disjoint(currentStateSet, nfa.finalStates)) {

dfaFinalStates.add(dfaStateId);

}

// Process each input symbol

for (char symbol : dfaAlphabet) {

// Compute the set of next NFA states for the current input symbol and current

// NFA state

Set<Integer> nextStateSet = new HashSet<>();

for (int state : currentStateSet) {

for (Transition t : nfa.transitions) {

if (t.fromState == state && t.input == symbol) {

nextStateSet.add(t.toState);

}

}

}

// If there are next states, add a transition to the DFA

if (!nextStateSet.isEmpty()) {

if (!stateMapping.containsKey(nextStateSet)) {

// Assign a new DFA state ID to the next NFA state set

stateMapping.put(nextStateSet, stateCounter++);

queue.add(nextStateSet);

}

dfaTransitions.add(new Transition(dfaStateId, symbol, stateMapping.get(nextStateSet)));

}

}

}

// Construct and return the resulting DFA

return new DFA(

stateMapping.size(), // Total number of states in DFA

dfaAlphabet, // Same alphabet as NFA

dfaFinalStates, // Computed final states in DFA

stateMapping.get(startSet), // Start state of DFA

dfaTransitions // DFA transitions

);

}

/\*\*

\* Main method to read NFA input and convert it to a DFA.

\*

\* @param args Command-line arguments (not used)

\*/

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter number of states:");

int cardinality = Integer.parseInt(scanner.nextLine()); // Number of states

System.out.println("Enter alphabet (characters without spaces, e.g., abc):");

Set<Character> alphabet = new HashSet<>();

for (char c : scanner.nextLine().toCharArray()) {

alphabet.add(c); // Set of input symbols

}

System.out.println("Enter number of final states followed by the final states (e.g., 2 1 3):");

Set<Integer> finalStates = new HashSet<>();

String[] finalStatesInput = scanner.nextLine().split(" ");

int finalStateCount = Integer.parseInt(finalStatesInput[0]); // First number is the count

for (int i = 1; i <= finalStateCount; i++) { // Read next `finalStateCount` numbers

finalStates.add(Integer.parseInt(finalStatesInput[i]));

}

System.out.println("Enter start state:");

int startState = Integer.parseInt(scanner.nextLine()); // Start state

System.out.println("Enter transitions (format: fromState inputSymbol toState), one per line. Enter an empty line to finish:");

Set<Transition> transitions = new HashSet<>();

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

if (line.isEmpty())

break; // End of transitions

String[] parts = line.split(" ");

transitions.add(new Transition(Integer.parseInt(parts[0]), parts[1].charAt(0), Integer.parseInt(parts[2]))); // Read transitions

}

scanner.close();

// Create NFA and convert to DFA

NFA nfa = new NFA(cardinality, alphabet, finalStates, startState, transitions);

DFA dfa = convertNFAToDFA(nfa);

// Output the resulting DFA

dfa.output();

}

}