|  |  |
| --- | --- |
| **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**

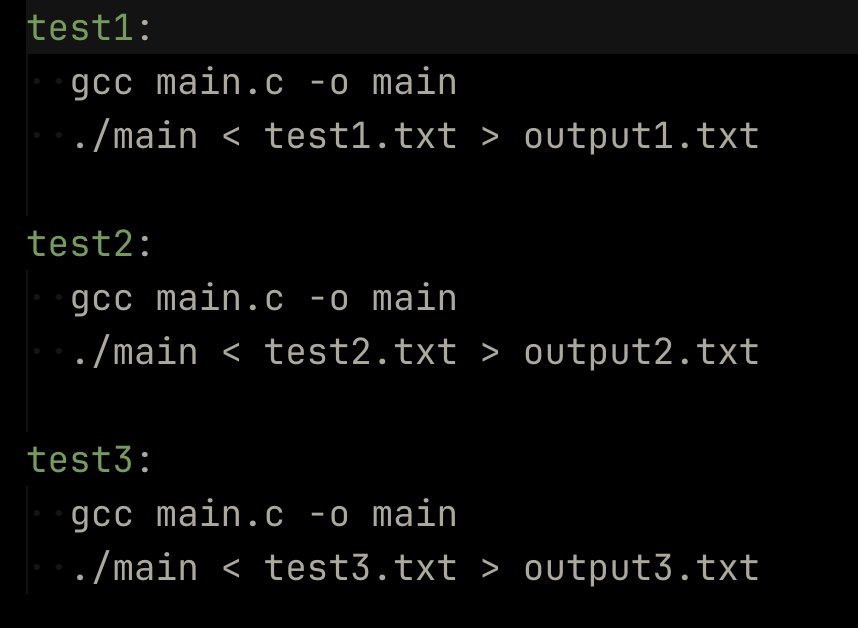
**使用C语言实现NFA到DFA的转换，实现输入数据 -> 构建DFA -> 核心逻辑（构建队列） -> 组装DFA -> 输出数据的流程。**

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。**

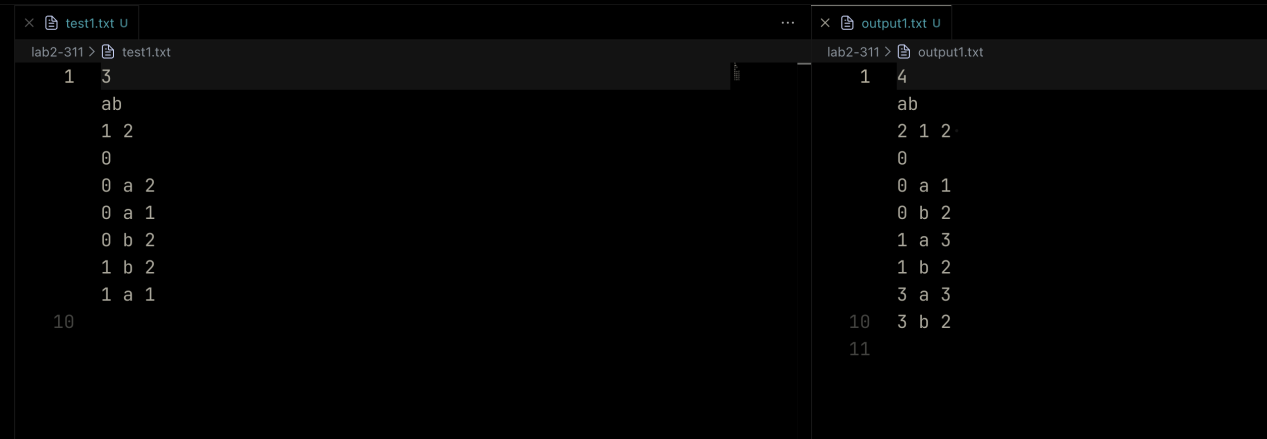
**和实验一的Java相比，使用C语言的主要挑战在于其更加底层（这种应用类的算法任务不适合用C语言），例如队列这种数据结构在本实验中使用数组进行简单的模拟，因为我们不关心算法的效率。**

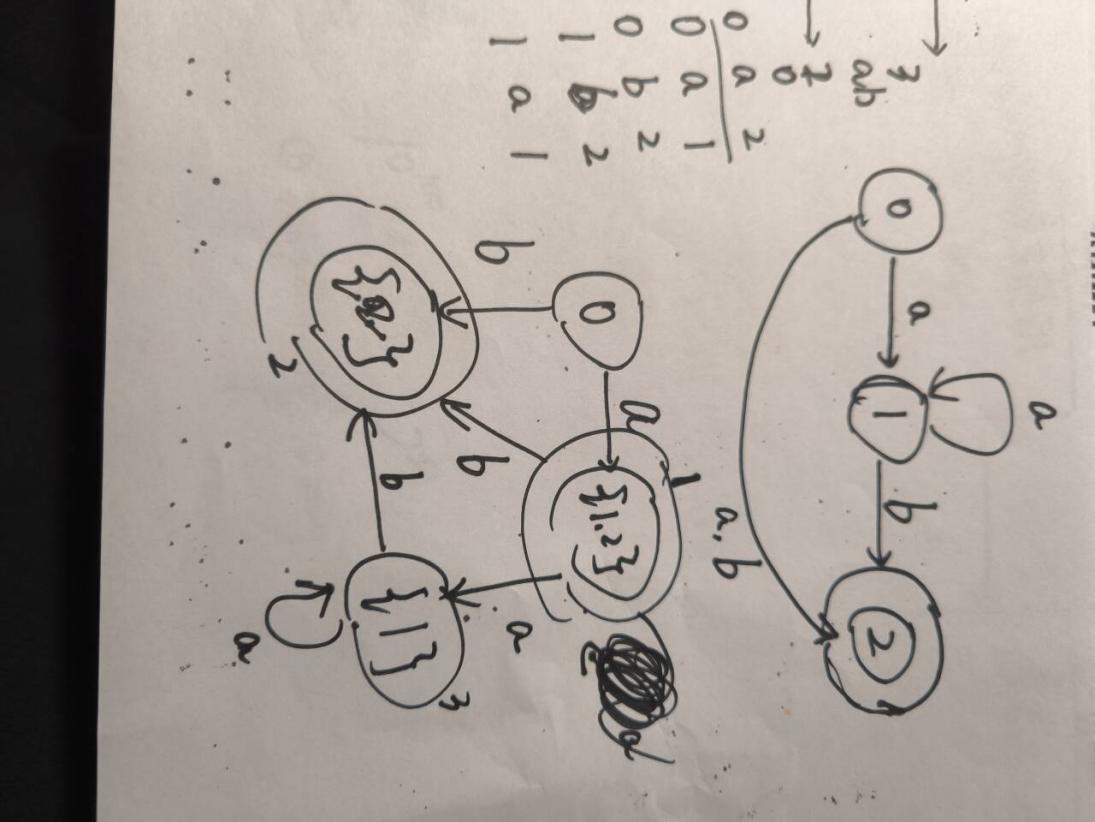
**使用shell的管道重定向功能，我们可以非常简单地构造出一组输入和输出，如下：**



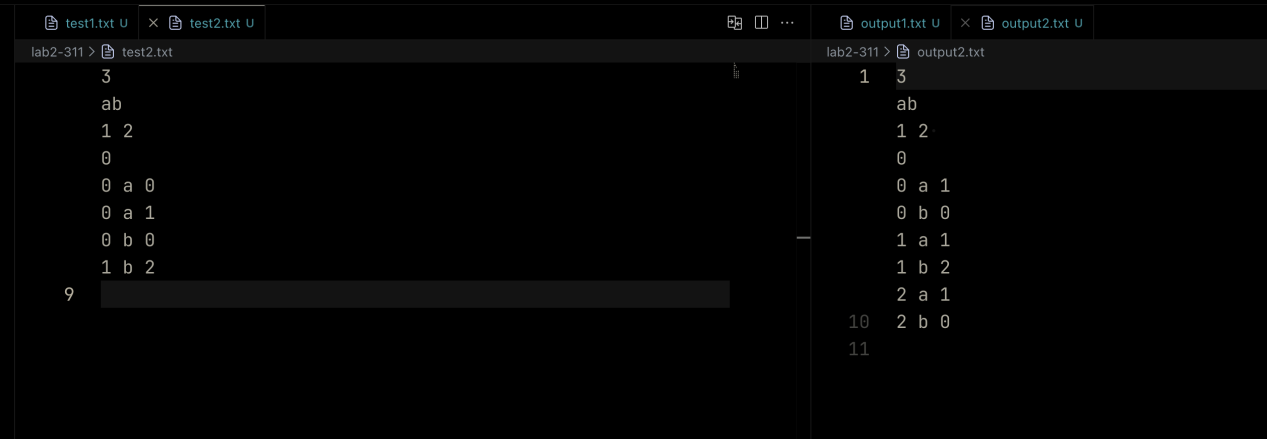
1. **实验步骤**
2. **构建基本结构体：NFA、DFA、Transition**
3. **设计输入、输出逻辑**
4. **绘制流程图（实际的NFA、DFA状态图），找到三个测试用例，梳理流程**
5. **编写核心逻辑**
6. **实验结果与分析**

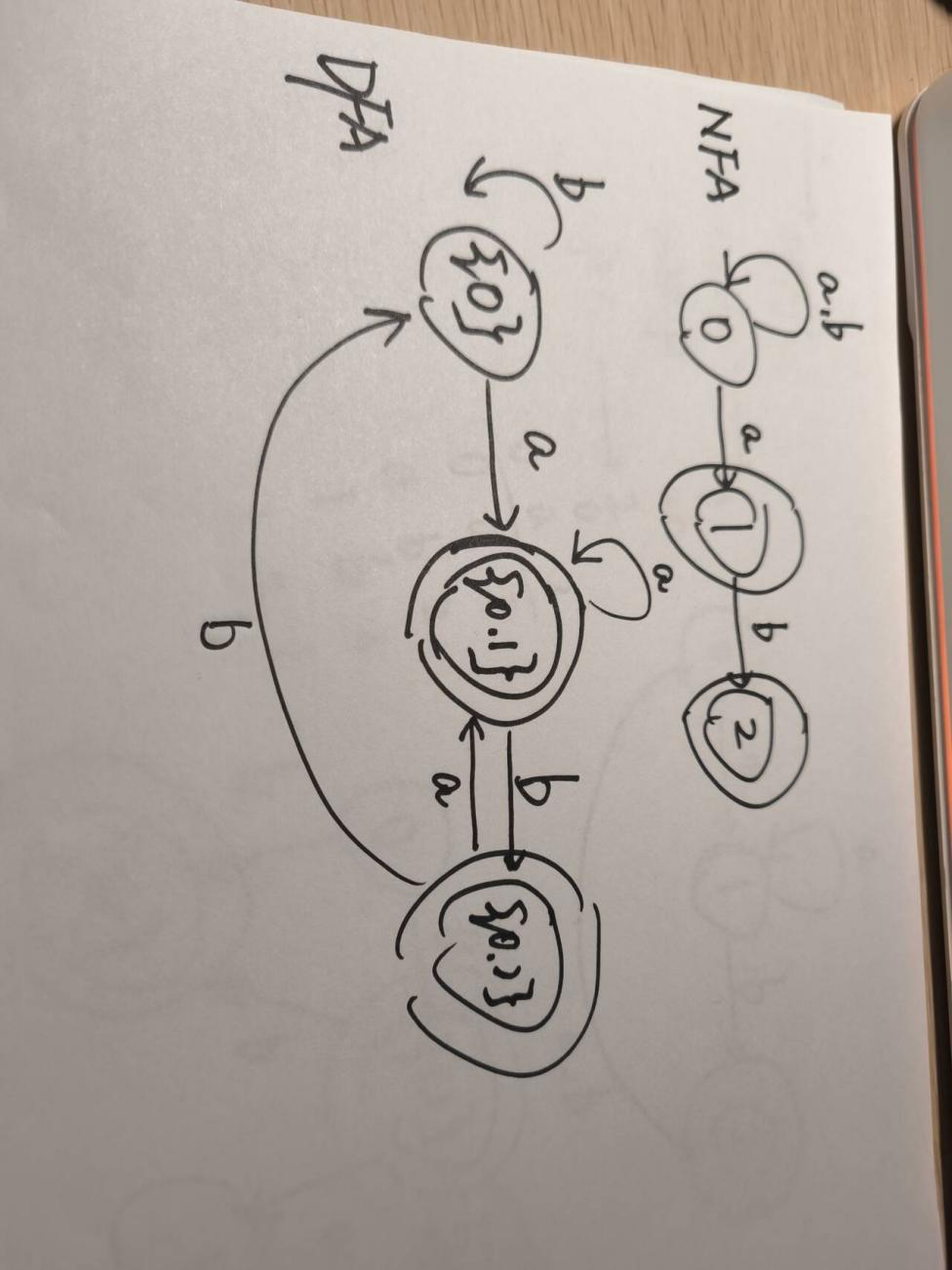
**测试数据1：**

****

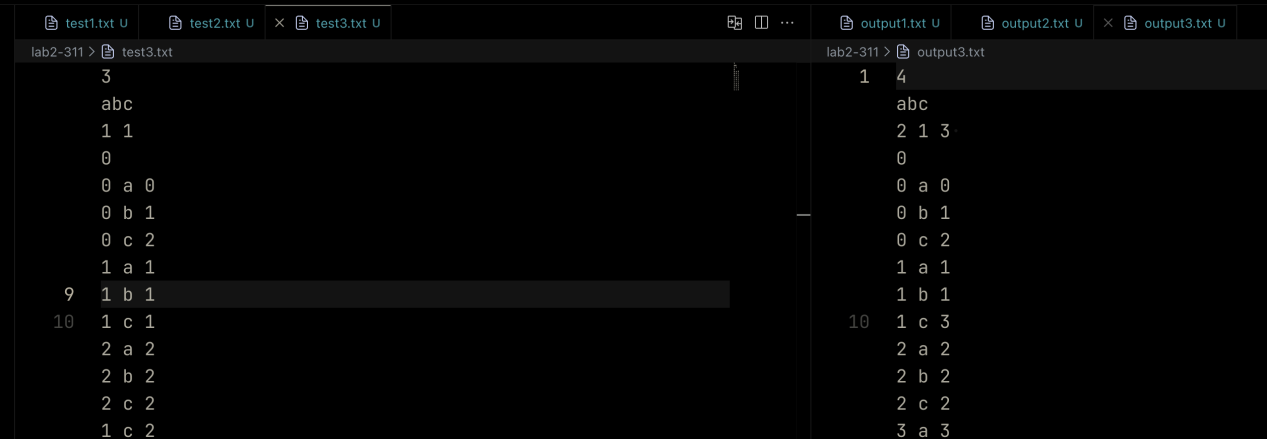
****

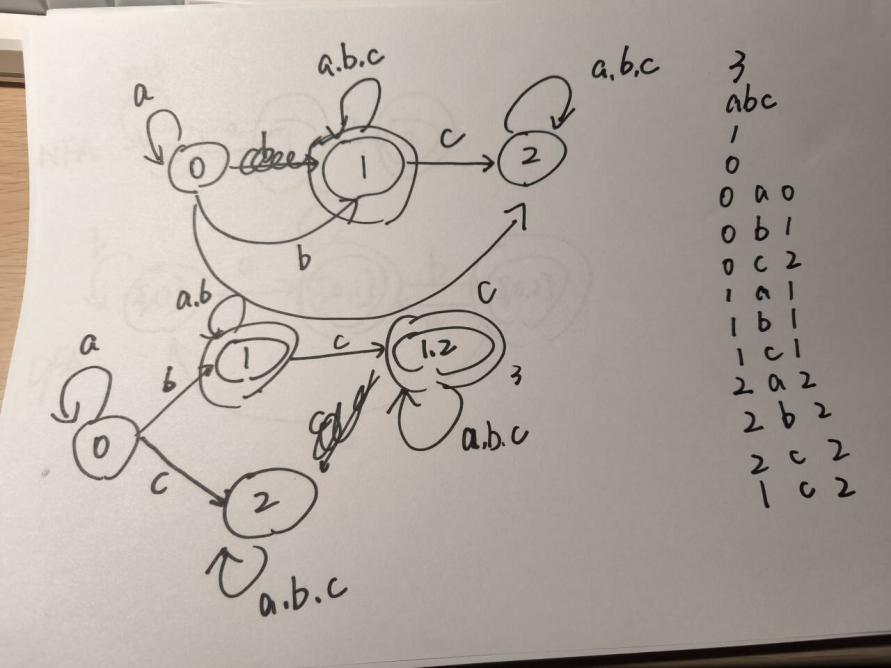
**测试数据2：**

****

****

**测试数据3：**

****

****

1. **核心代码清单（只写代码核心）**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define a structure to represent a transition

typedef struct {

int fromState;

char input;

int toState;

} Transition;

// Define a structure to represent an NFA

typedef struct {

int cardinality; // Number of states

char alphabet[256]; // Bitmap of input symbols

int finalStates[100]; // Set of final states

int finalStateCount;

int startState;

Transition transitions[1000];

int transitionCount;

} NFA;

// Define a structure to represent a DFA

typedef struct {

int cardinality;

char alphabet[256]; // Bitmap of input symbols

int finalStates[100];

int finalStateCount;

int startState;

Transition transitions[1000];

int transitionCount;

} DFA;

int isInAlphabet(char alphabet[256], char c) {

return alphabet[(unsigned char)c] == 1;

}

void addToAlphabet(char alphabet[256], char c) {

alphabet[(unsigned char)c] = 1;

}

int isFinalState(int finalStates[100], int finalStateCount, int state) {

for (int i = 0; i < finalStateCount; i++) {

if (finalStates[i] == state) {

return 1;

}

}

return 0;

}

DFA convertNFAToDFA(NFA nfa) {

DFA dfa;

memset(&dfa, 0, sizeof(DFA));

memcpy(dfa.alphabet, nfa.alphabet, 256);

int visited[1000][100] = {0};

int visitedCount = 0;

int queue[1000][100] = {0};

int queueSize[1000] = {0};

int queueFront = 0, queueRear = 0;

int dfaStateCount = 0;

int startSet[100] = {nfa.startState};

int startSetSize = 1;

memcpy(queue[queueRear], startSet, sizeof(int) \* startSetSize);

queueSize[queueRear++] = startSetSize;

int stateId[1000][100] = {0};

int stateSize[1000] = {0};

memcpy(stateId[dfaStateCount], startSet, sizeof(int) \* startSetSize);

stateSize[dfaStateCount++] = startSetSize;

dfa.startState = 0;

while (queueFront < queueRear) {

int\* currentSet = queue[queueFront];

int currentSize = queueSize[queueFront++];

int currentDfaState = -1;

// Find DFA state id

for (int i = 0; i < dfaStateCount; i++) {

if (stateSize[i] == currentSize &&

memcmp(stateId[i], currentSet, sizeof(int) \* currentSize) == 0) {

currentDfaState = i;

break;

}

}

// Final state check

for (int i = 0; i < currentSize; i++) {

if (isFinalState(nfa.finalStates, nfa.finalStateCount, currentSet[i])) {

if (!isFinalState(dfa.finalStates, dfa.finalStateCount, currentDfaState)) {

dfa.finalStates[dfa.finalStateCount++] = currentDfaState;

}

break;

}

}

for (int c = 0; c < 256; c++) {

if (!isInAlphabet(nfa.alphabet, c)) continue;

int nextSet[100] = {0};

int nextSetSize = 0;

for (int i = 0; i < currentSize; i++) {

for (int j = 0; j < nfa.transitionCount; j++) {

if (nfa.transitions[j].fromState == currentSet[i] &&

nfa.transitions[j].input == c) {

int exists = 0;

for (int k = 0; k < nextSetSize; k++) {

if (nextSet[k] == nfa.transitions[j].toState) {

exists = 1;

break;

}

}

if (!exists) {

nextSet[nextSetSize++] = nfa.transitions[j].toState;

}

}

}

}

if (nextSetSize == 0) continue;

// Check if this set already exists

int existingState = -1;

for (int i = 0; i < dfaStateCount; i++) {

if (stateSize[i] == nextSetSize &&

memcmp(stateId[i], nextSet, sizeof(int) \* nextSetSize) == 0) {

existingState = i;

break;

}

}

if (existingState == -1) {

memcpy(stateId[dfaStateCount], nextSet, sizeof(int) \* nextSetSize);

stateSize[dfaStateCount] = nextSetSize;

memcpy(queue[queueRear], nextSet, sizeof(int) \* nextSetSize);

queueSize[queueRear++] = nextSetSize;

existingState = dfaStateCount++;

}

dfa.transitions[dfa.transitionCount++] = (Transition){currentDfaState, (char)c, existingState};

}

}

dfa.cardinality = dfaStateCount;

return dfa;

}

void outputDFA(DFA dfa, char alphabetList[], int alphabetSize) {

printf("%d\n", dfa.cardinality);

for (int i = 0; i < alphabetSize; i++) {

putchar(alphabetList[i]);

}

printf("\n");

printf("%d ", dfa.finalStateCount);

for (int i = 0; i < dfa.finalStateCount; i++) {

printf("%d ", dfa.finalStates[i]);

}

printf("\n");

printf("%d\n", dfa.startState);

for (int i = 0; i < dfa.transitionCount; i++) {

printf("%d %c %d\n", dfa.transitions[i].fromState,

dfa.transitions[i].input, dfa.transitions[i].toState);

}

}

int main() {

NFA nfa;

memset(&nfa, 0, sizeof(NFA));

char alphabetInput[256];

char alphabetList[256];

int alphabetSize = 0;

printf("Enter number of states:\n");

scanf("%d", &nfa.cardinality);

printf("Enter alphabet (characters without spaces, e.g., abc):\n");

scanf("%s", alphabetInput);

memset(nfa.alphabet, 0, 256);

for (int i = 0; alphabetInput[i] != '\0'; i++) {

char c = alphabetInput[i];

if (!isInAlphabet(nfa.alphabet, c)) {

addToAlphabet(nfa.alphabet, c);

alphabetList[alphabetSize++] = c;

}

}

printf("Enter number of final states followed by the final states (e.g., 2 1 3):\n");

scanf("%d", &nfa.finalStateCount);

for (int i = 0; i < nfa.finalStateCount; i++) {

scanf("%d", &nfa.finalStates[i]);

}

printf("Enter start state:\n");

scanf("%d", &nfa.startState);

printf("Enter transitions (format: fromState inputSymbol toState), one per line. Enter an empty line to finish:\n");

getchar(); // consume newline after scanf

char line[256];

nfa.transitionCount = 0;

while (1) {

if (!fgets(line, sizeof(line), stdin) || strcmp(line, "\n") == 0) break;

int fromState, toState;

char input;

if (sscanf(line, "%d %c %d", &fromState, &input, &toState) == 3) {

nfa.transitions[nfa.transitionCount++] = (Transition){fromState, input, toState};

}

}

DFA dfa = convertNFAToDFA(nfa);

outputDFA(dfa, alphabetList, alphabetSize);

return 0;

}