**北京邮电大**

**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：实验一: NFA到DFA的转化**

**项目完成人：**

**负责人姓名：**郭栩源 **学号：**2021213586

**姓名：**姜杨皓 **学号：**2021213650

**姓名：**胡栋康 **学号：**2021210166

**姓名：**邢智恺 **学号：**2021213585

**指导教师：杨正球 助教：杨惠元**

**日 期： 2023年 4月 10日**

**一 . 项目目的和要求**

**1.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。

**2.实验要求**

1. 采用分组实验
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
3. 要求程序运行正确，设计风格好，文档描述清晰

**二 .项目开发环境**

语言使用C++，在visual studio code中使用mingw编译和gdb调试；最后使用头歌教学云平台进行测试验证代码。

**三 .项目需求及设计方案**

**1.项目需求：**

有限状态自动机是描述控制过程有力工具。有限状态自动机有不同的类型，例如，确定有限状态自动机（DFA）和不确定有限状态自动机（NFA）。这些不同类型的自动机之间可以等价转化。我们在实际应用中，可以利用某种类型的自动机更加方便刻画实际系统，然后再利用等价转化算法实现不同类型的自动机转化。

本实验要求编程实现NFA到DFA的自动转化。输入自己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

**2.设计方案**

（1）NFA转化为DFA的基本原理—子集构造法

设NFA Mn=(Q,T,δ,q0,F0),构造一个等效DFA Md=(Qd,T,,q0d,Fd)，其中：

Qd是Q的幂集，Qd中元素为[q1,q2,…,qn]，{q1,q2,…,qn}Q;

q0d=[q0];

Fd的每个状态包含F0中的一个状态；

定义为：

([q1,q2,…,qn],a)=[p1,p2,…,pn]

当且仅当：

({q1,q2,…,qn},a)= {p1,p2,…,pn}

（2）算法思路

先读取输入，确定起始状态和终止状态。存储状态转移函数；

void init();*//初始化函数*

生成一个DFA状态表示NFA中的起始状态，并将该状态放入队列中，并标记为已访问。

随后每次取出队列的队首元素x，设状态x输入0、1时，分别生成状态y、z，若y、z未被访问过，则生成一个新状态表示它，并将该状态放入队列，并标记为已访问。

void createDfa();*//完善所有的dfa状态和dfa的deduce数组*

最后遍历DFA，删除不可达元素，最后完整输出。

void dfs(int *a*);*//记录一遍dfs，经过为1，否则为0；在后续输出中为0则不输出*

void printdfa();

（3）伪代码

1. 参考数据结构和函数接口

#define nfaStateNum 10 //nfa的状态数量，假设为10

int nfaDeduce0[nfaStateNum][nfaStateNum] //二维数组，其中nfaDeduce[i]是一个一维数组，存储在NFA中通过状态qi输入0能到达的状态

int nfaDeduce1[nfaStateNum][nfaStateNum] //二维数组，其中nfaDeduce[i]是一个一维数组，存储在NFA中通过状态qi输入1能到达的状态

int dfaStateNum //dfa的状态数量，初始为0

struct dfaStateNode{

bool isStart,isFinal

int nfaState[nfaStateNum] //存储一个dfa状态中包含的nfa状态，以-1结尾

}dfaState[2^ nfaStateNum]

int dfaDeduce0[2^ nfaStateNum] //存储DFA中qi输入0时能到达的状态

int dfaDeduce1[2^ nfaStateNum] //存储DFA中qi输入1时能到达的状态

queue<dfaState> q; //存储DFA状态的队列

void getStatus(String str,int[] state) //从字符串中提取其包含的状态，通过数组state返回

int findStatus(int[] state) //判断一个包含多个NFA状态的数组是否在DFA状态中出现过，若无则返回-1，否则返回其下标

void deduce(int[] fromState,int[] toState,int input) // fromState数组输入input时推倒出toState数组，通过toState返回

1. 输入以及预处理

input(“\t\t0\t1\n”) //读入第一行

string str

while(input(str)!=EOF) //每次读一个字符串，遇到\t或\n结束，此处读每行中的第一个字符串

int state[nfaStateNum]，toState[nfaStateNum] //包含多个NFA状态的数组

getStatus(str,state)

if(str[0] equal ‘(‘) //如果读到小括号，说明此行为起始或终止状态

if(str[1] equal ‘s’)

//生成一个DFA状态表示NFA中的起始状态

dfaState[dfaStateNum].isStart=1

mamcpy(dfaState[dfaStateNum].nfaState,state,sizeof(state))

else

//生成一个DFA状态表示NFA中的终止状态

dfaState[dfaStateNum].isFinal=1

mamcpy(dfaState[dfaStateNum].nfaState,state,sizeof(state))

else

//生成一个DFA状态表示NFA中的状态

mamcpy(dfaState[dfaStateNum].nfaState,state,sizeof(state))

//状态加入队列

q.push(dfaState[dfaStateNum++])

input(str) //读入每行中的第二的字符串

getStatus(str,toState)

memcpy(nfaDeduce0[state[0]],stateTo,sizeof(toState)) //存储状态转移函数

input(str) //读入每行中的第三的字符串

getStatus(str,toState)

memcpy(nfaDeduce1[state[0]],stateTo,sizeof(toState)) //存储状态转移函数

1. 循环处理队首元素

dfaStateNode curState=q.front()

q.pop()

int toState[nfaStateNum]

deduce(curState.nfaState,toState,0)

if(findStatus (toState) equal -1) //若未访问过

//生成一个DFA状态

mamcpy(dfaState[dfaStateNum].nfaState, toState,sizeof(toState))

q.push(dfaState[dfaStateNum])

//存储DFA状态转移表

dfaDeduce0[curState.id]= dfaStateNum++

else

dfaDeduce0[curState.id]= findStatus (toState)

deduce(curState.nfaState,toState,1)

if(findStatus (toState) equal -1) //若未访问过

//生成一个DFA状态

mamcpy(dfaState[dfaStateNum].nfaState, toState,sizeof(toState))

//存储DFA状态转移表

dfaDeduce1[curState.id]= dfaStateNum++

q.push(dfaState[dfaStateNum++])

else

dfaDeduce1[curState.id]= findStatus (toState)

**四 .实际代码**

1.实际代码中的数据结构和函数接口：

//1. 数据结构和函数接口

//1.1数据结构

const int maxiNum = 100; //nfa的状态数量最大值，也即小状态的数量，假设为100

const int maxNum = 1000;//2 ^ maxiNum的值，dfa中元素最多有多少个

int nfaStateNum;//nfa的状态数量，初始为0

int dfaStateNum;//dfa的状态数量，初始为0

int nfaDeduce0[maxiNum][maxiNum]; //二维数组，其中nfaDeduce[i]是一个一维数组，存储在NFA中通过状态qi输入0能到达的状态

int nfaDeduce1[maxiNum][maxiNum]; //二维数组，其中nfaDeduce[i]是一个一维数组，存储在NFA中通过状态qi输入1能到达的状态

int nfaStateCondition[maxiNum];//记录nfa状态是否为首尾，0是一般正常，1是首节点，2是尾节点

typedef struct dfaStateNode {

int l = 0;

int isStart=0;

int isFinal=0;

int nfaNode[maxiNum]={0}; //存储一个dfa状态中包含的nfa状态，以-1结尾

}dfaStateNode;

dfaStateNode dfaState[maxNum];//数组形式存储所有dfa状态

int dfaDeduce0[maxNum];//存储DFA中qi输入0时能到达的状态

int dfaDeduce1[maxNum]; //存储DFA中qi输入1时能到达的状态

int ifDfaDeduce[maxNum];//存储dfa中状态qi是否已经进行处理

queue<dfaStateNode> q;//存储DFA状态的队列

queue<int>qnum;//存储q队列中DFA状态的下标，所有入队出队操作与q队列一起进行

int dfsState[maxNum];//记录dfs过程中经过的dfa节点，经过则置为1，否则为0

//1.2函数接口

//1.2.1输入以及预处理

void init();//初始化函数

//1.2.2生成dfa状态及转移表

int find(dfaStateNode node);//判断一个dfaStateNode在DFA状态中是否出现过，若无则返回-1，若空返回-2，否则返回其下标

void handle(int handleNum);//处理dfa状态的第几个

void createDfa();//完善所有的dfa状态和dfa的deduce数组

//1.2.3删除不可达元素

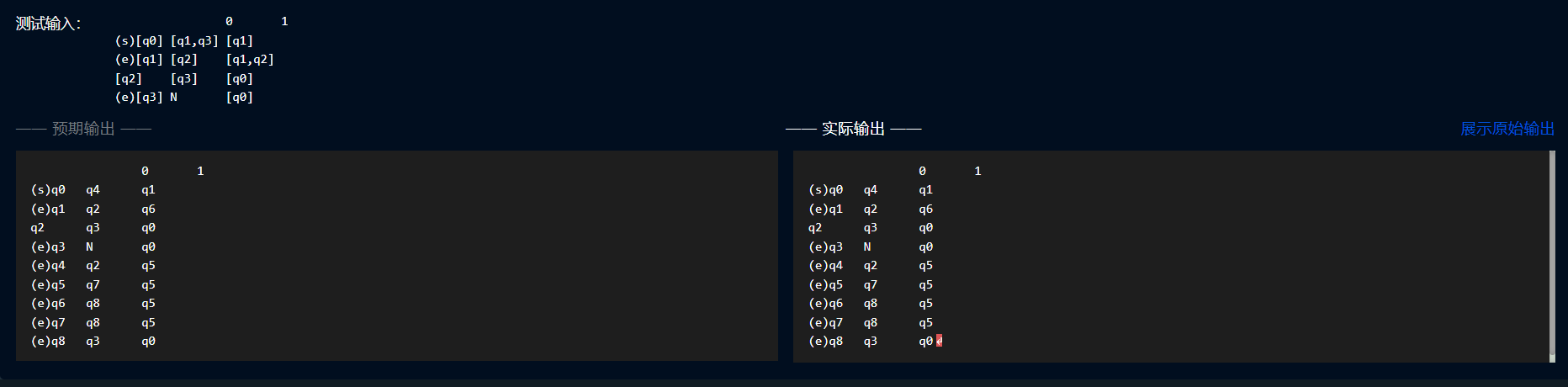
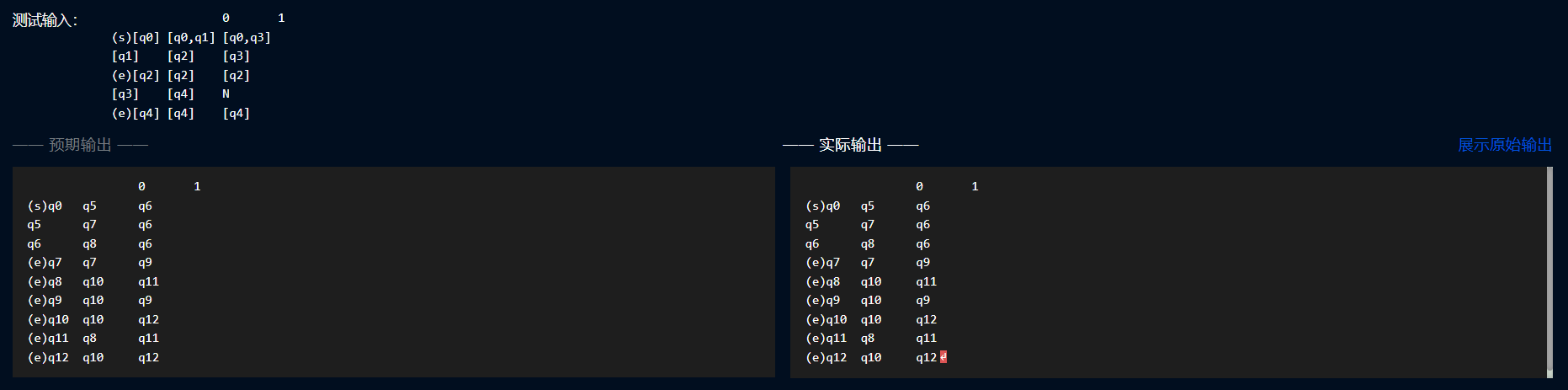
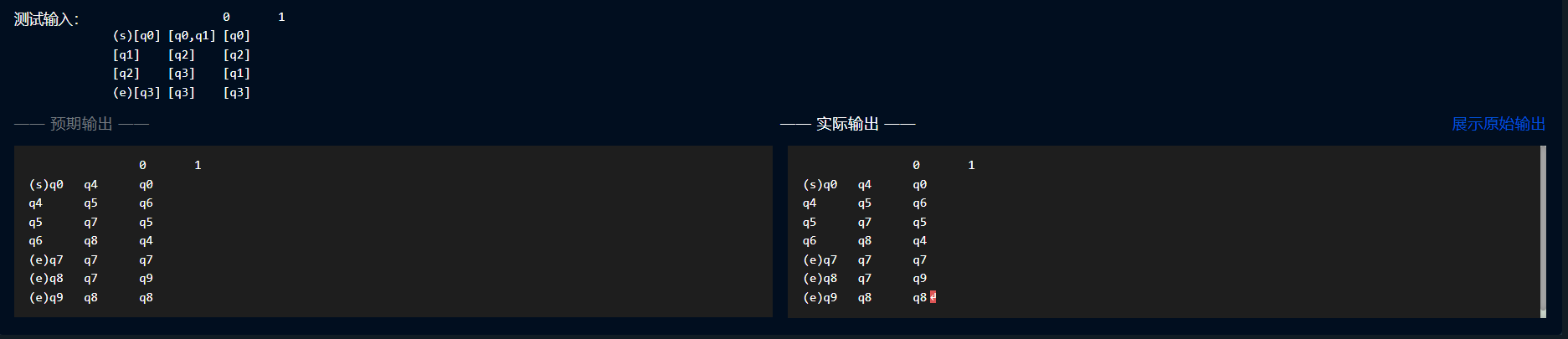
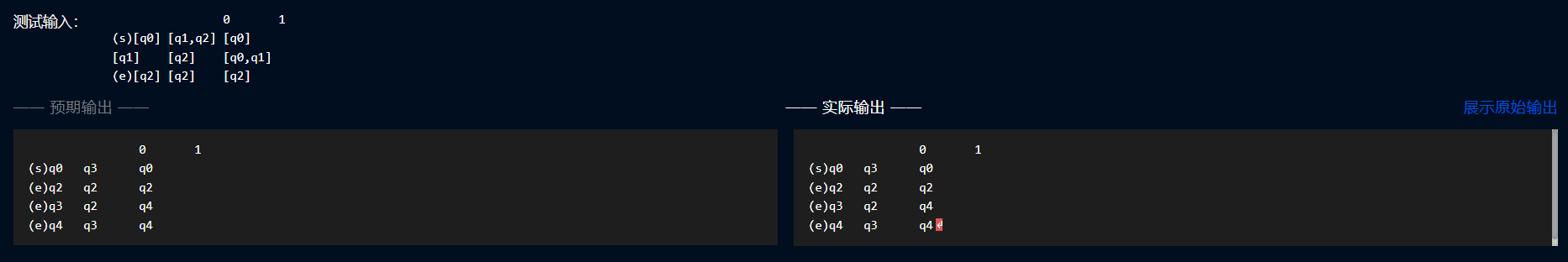
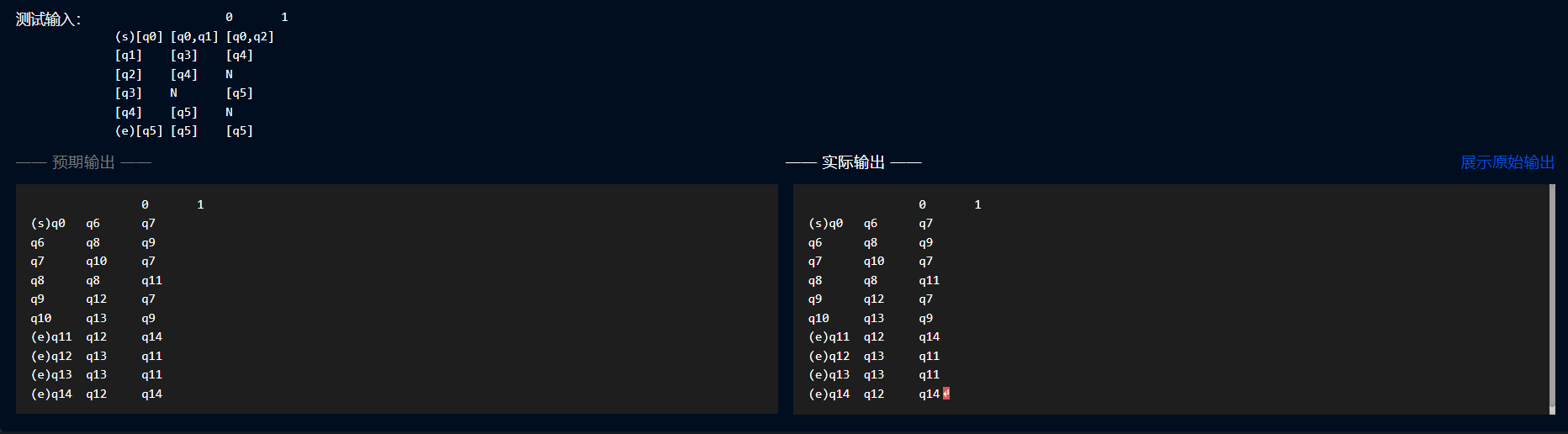
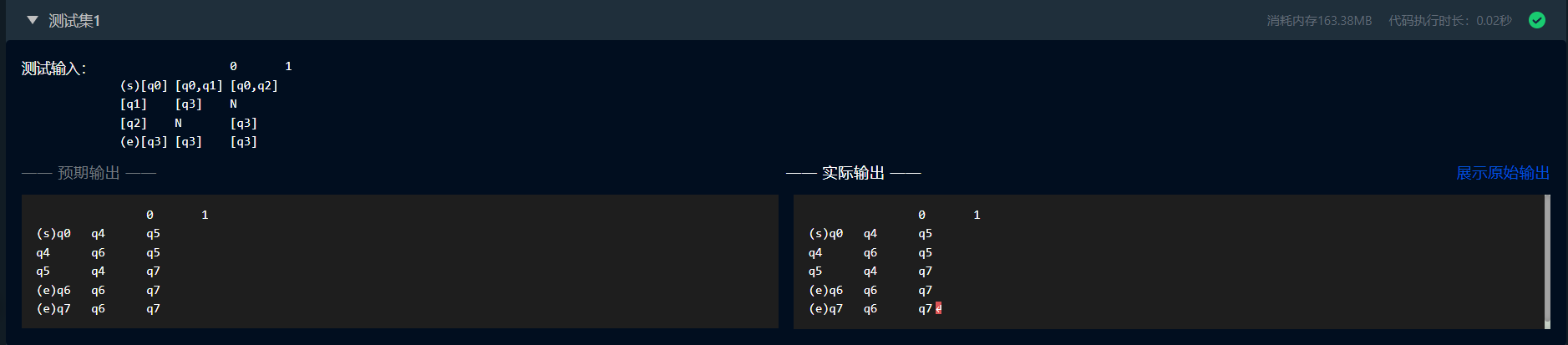
void dfs(int a);//记录一遍dfs，经过为1，否则为0；在后续输出中为0则不输出

//1.2.4输出

void printdfa();

实际代码中函数功能的实现与伪代码中略有不同，但基本思路一样，具体实际代码参考提交的cpp文件。

**五 .项目结果及分析**

****

****

其中第六组样例出错，通过分析NFA转化为DFA中每个状态具体命名规则，检查并修改代码，最后通过所有样例。

**六 .项目人员、进度安排及完成过程**

郭栩源负责代码框架，整体算法思路以及伪代码

姜杨皓和胡栋康负责代码编写，代码测试

邢智恺负责实验报告总结

全部实验按进度完成。