**中山大学数据科学与计算机学院本科生实验报告**

课程名称：编译器构造实验 任课教师：陈炬桦 教学助理（TA）：

|  |  |  |  |
| --- | --- | --- | --- |
| 学年度 | 2018-2019 | 学期 | 第二学期 |
| 年级 | 2016级 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337327 | 姓名 | 郑映雪 |
| 电话 | 18956073450 | Email | 646352420@qq.com |
| 开始日期 | 2019.3.22 | 完成日期 | 2019.3.23 |

**实验题目**

## 1. Description

输入字母表，非确定εNDFA 确定化ＦＡ状态集，映射集；用造表法算法。

输出确定ＦＡ状态集，映射集；

## 2. Input

输入字母个数，字母表

状态个数，状态表（状态名称，开始状态，终止状态：0否1是），空格符分隔；

映射个数，映射表（起，终，字母），空格符分隔，k表示ε；

## 3. Output

Determine State:

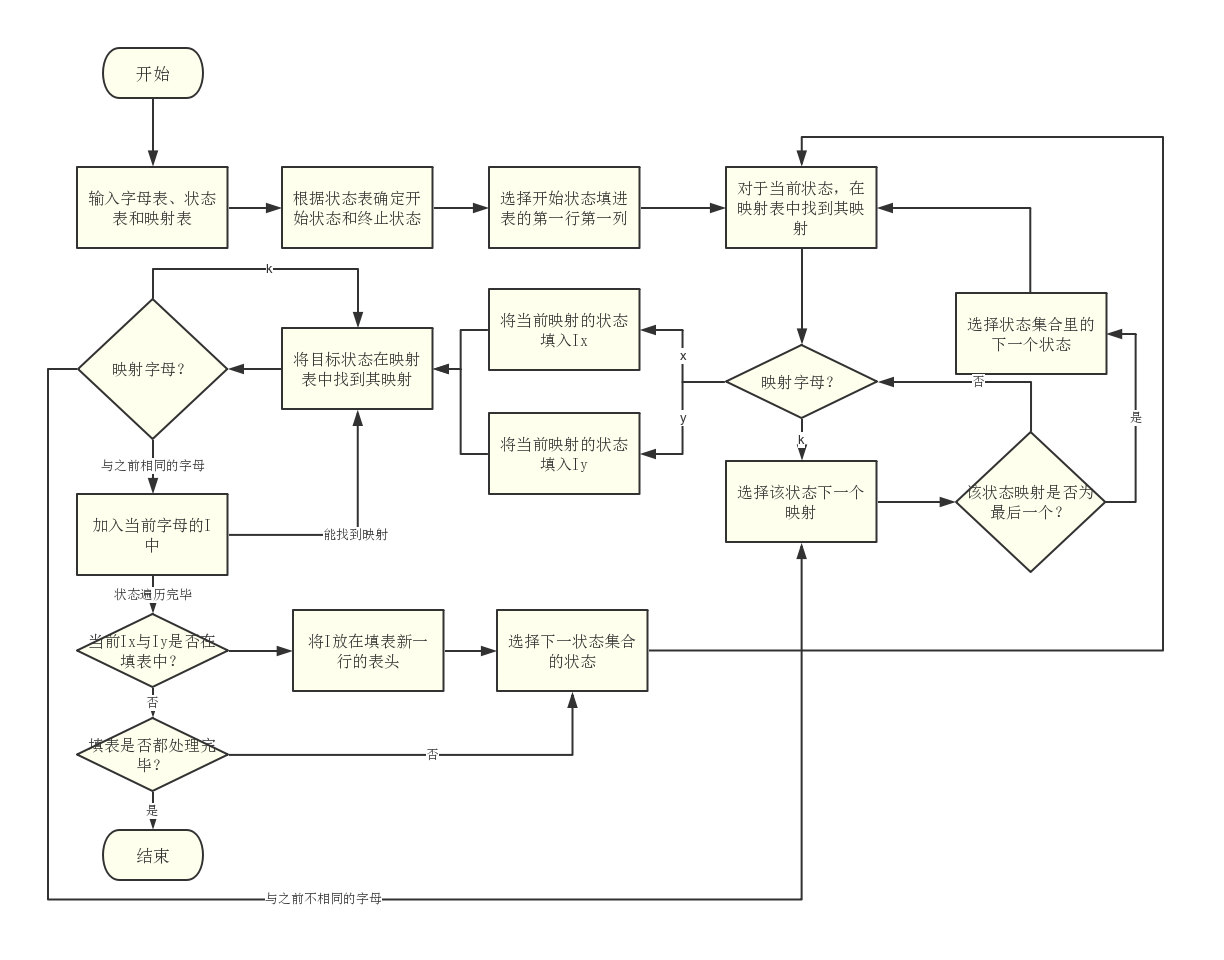
状态表: 状态名称，开始状态，终止状态，[子集]

Determine Mapping:

映射表: 起，终，字母,(顺序号

**算法描述**

## 1. 流程总览

本流程图以test中字母表为xy示为示例。流程如下：

## 2.各模块功能

由于本次实验操作如果单纯用C++实现的话非常复杂（老师上课也说python比较方便，pascal比较合适，但是西西里没法用python，pascal我也因为很久没用忘得差不多了），所以流程图很复杂，可能会让人眼花缭乱。下面我将详细叙述每一部分的编程方法。

**a.初始化模块**

本次实验的输入是以矩阵的形式进行的，由于是C++编程，所以采用string类型矩阵比较合适（vector也可）。读入状态矩阵时，以字符串的形式存储第一列节点的名称，同时根据后面两列数据是否为1来设置好起始节点和终止节点。

**b.确定化模块（以本实验为例）**

这是本次实验的算法主体部分。下面是确定化的步骤：

①将起始节点作为第一个确定化的节点。

②判断当前节点的Ix和Iy。在映射表中搜索有当前状态的组，通过判断第三列是x还是y将其加入自己Ix或Iy中。如遍历完毕没有符合条件的目标状态，则Ix或Iy置空。

③如搜索成功，则以上次搜索到的目标状态在映射表里继续搜索（DFS深度搜索原理）。如果搜索到该状态的表列且第三列为k，则以新的目标状态在映射表里搜索，重复此步直到搜索完毕或无符合条件的搜索结果。

④将搜索完毕的Ix和Iy进行标准化。即进行去除重复状态和状态排序的操作。

⑤以当前状态的Ix和Iy与之前已经存在的新状态集合进行对比，如果未在之前的状态集合里出现过，则将其加入下一空行的第一列，作为等待确定化的新的状态集合，如果已经出现过，则不做任何处理。

⑥如果所有表中的状态集合都已搜索完毕，且没有新的状态集合需要处理，则结束确定化的过程，确定化结束。

确定化的算法在手写和算法描述是比较简单的，但是C++操作比较麻烦，所以代码还是比较长的。采用string类型最方便的是I中状态的加入只需要简单的进行+操作即可。而且在重复状态判断时string类型也是重载了==符号的，使用方便。

**c.输出部分**

填表结束后，还要根据填表输出新的状态表和映射表。

新的状态表第一列只需要对填表中每一个状态集合进行新状态编号。同时对状态集合遍历判断是否有原本的开始状态和终止状态，如果有则将状态表中该表项置为1。

编号之后，按序输出填表中的映射。如果Ix或Iy为空，则不输出该映射。最后对所有映射进行编号处理并输出。

**测试数据(2组)**

|  |  |  |
| --- | --- | --- |
| 测试数据 | 测试结果 | 结果判定 |
|  |  | 正确 |
|  |  | 正确 |

**程序清单**

#include <iostream>

#include <string>

#include <cstring>

using namespace std;

//将I去除重复的状态以及将状态排序

string tran(string str) {

    string newstr;

    for (int i = 0; i < str.size(); ++i) {

        int flag = 0;

        for (int j = i + 1; j < str.size(); ++j) {

            if (str[i] == str[j]) flag = 1;

        }

        if (!flag) newstr += str[i];

    }

    for (int i = 0; i < newstr.size(); ++i) {

        for (int j = 0; j < newstr.size() - i - 1; ++j) {

            if (newstr[j] > newstr[j + 1]) {

                char tmp = newstr[j];

                newstr[j] = newstr[j + 1];

                newstr[j + 1] = tmp;

            }

        }

    }

    return newstr;

}

//确定化的过程

void determin(string alphabet, string state[][4], string reflec[][3], int statelen, int reflen, string start, string end) {

    string table[1000][3];

    table[0][0] = start;

    int maxlen = 1, oldlen = 0;

    //搜索过程

    while (true) {

        for (int k = oldlen; k < maxlen; ++k) {

            for (int i = 0; i < table[k][0].size(); ++i) {

                for (int j = 0; j < reflen; ++j) {

                    //判断属于Ix还是Iy

                    if (table[k][0][i] == reflec[j][0][0])

                        if (reflec[j][2][0] == 'x') {

                            table[k][1] += reflec[j][1];

                            string tmp = reflec[j][1];

                            //搜索字母为k的状态的节点

                            for (int k2 = 0; k2 < reflen; k2++) {

                                if (reflec[k2][0][0] == tmp[0] && reflec[k2][2][0] == 'k') {

                                    table[k][1] += reflec[k2][1];

                                    tmp = reflec[k2][1][0];

                                    k2 = 0;

                                }

                            }

                        }

                        else if (reflec[j][2][0] == 'y') {

                            table[k][2] += reflec[j][1];

                            string tmp = reflec[j][1];

                            //搜索字母为k的状态的节点

                            for (int k2 = 0; k2 < reflen; k2++) {

                                if (reflec[k2][0][0] == tmp[0] && reflec[k2][2][0] == 'k') {

                                    table[k][2] += reflec[k2][1];

                                    tmp = reflec[k2][1][0];

                                    k2 = 0;

                                }

                            }

                        }

                }

                //删去重复节点与节点排序

                table[k][1] = tran(table[k][1]);

                table[k][2] = tran(table[k][2]);

            }

            //如果状态为新状态，则加入表中

            oldlen = maxlen;

            string prex = table[k][1], prey = table[k][2];

            int flag1 = 0, flag2 = 0;

            for (int j = 0; j < oldlen; j++) {

                if (table[j][0] == prex) flag1 = 1;

                if (table[j][0] == prey) flag2 = 1;

            }

            if (prex == "") flag1 = 1;

            if (prey == "") flag2 = 1;

            if (!flag1) {

                table[maxlen][0] = prex;

                maxlen++;

            }

            if (!flag2) {

                table[maxlen][0] = prey;

                maxlen++;

            }

        }

        //如果没有新状态添加且所有旧状态处理完毕，则退出

        if (oldlen == maxlen && table[maxlen][0] == "") break;

    }

    //输出状态表，如果状态名里含有起始节点或者终止节点，则赋值为1

    cout << "Determine State:" << endl;

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

        int ifstart = 0, ifend = 0;

        cout << i << ' ';

        for (int j = 0; j < 3; j++) {

            for (int l = 0; l < table[i][0].size(); ++l) {

                if (table[i][0][l] == start[0]) ifstart = 1;

                if (table[i][0][l] == end[0]) ifend = 1;

            }

        }

        if (ifstart) cout << 1 << ' '; else cout << 0 << ' ';

        if (ifend) cout << 1 << ' '; else cout << 0 << ' ';

        cout << '[' << table[i][0] << ']' << endl;

    }

    //将填表里的节点集转化为新的节点

    int newtable[1000][3] = { 0 };

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

        newtable[i][0] = i;

        for (int j = 0; j < maxlen; ++j) {

            if (table[j][0] == table[i][1]) newtable[i][1] = j;

            if (table[j][0] == table[i][2]) newtable[i][2] = j;

        }

    }

    //根据新的节点输出新的映射表

    cout << "Determine Mapping:" << endl;

    int num = 0;

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

        if (newtable[i][1] != 0) {

            cout << newtable[i][0] << ' ' << newtable[i][1] << " x (" << num << endl;;

            num++;

        }

        if (newtable[i][2] != 0) {

            cout << newtable[i][0] << ' ' << newtable[i][2] << " y (" << num << endl;

            num++;

        }

    }

}

//初始化部分

int main() {

    int allen, statelen, reflen;

    string alphabet = "";

    cin >> allen;

    //输入字母表和状态表、映射表

    for (int i = 0; i < allen; i++) {

        char tmp;

        cin >> tmp;

        alphabet += tmp;

    }

    cin >> statelen;

    string state[1000][4];

    string start, end;

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

        for (int j = 0; j < 3; ++j) {

            cin >> state[i][j];

        }

        //判断开始和终止节点

        if (state[i][1] == "1") start = state[i][0];

        if (state[i][2] == "1") end = state[i][0];

    }

    string reflec[1000][3];

    cin >> reflen;

    for (int i = 0; i < reflen; i++)

        for (int j = 0; j<3; ++j)

            cin >> reflec[i][j];

    //确定化过程

    determin(alphabet, state, reflec, statelen, reflen, start, end);

    system("pause");

}