**《编译原理》实验报告**

|  |  |
| --- | --- |
| **实验名称** | 有限自动机的确定化和最小化 |
| 1. **实验目的：**   1. 理解有限自动机的作用，进一步理解有限自动机理论  2. 设计有限自动机的表示方式，采用合理的数据结构表示自动机的五个组成部分  3. 掌握ε闭包的求法和子集构造算法，以程序实现NFA到DFA的转换，并且最小化DFA，提高算法的理解和实现能力 | |
| 1. **实验过程及步骤：** 2. 将NFA存储在.txt文件中格式如图所示   程序的第一行标记了转移的条数为7条，第二条表示输入符号为a 和b，第三行到第九行构建了一个状态转换表，表示为如0a2 以为0号状态通过a到达2号状态   1. 新建一个java类，用来存储状态转换表 2. 编写求状态的空闭包closure函数 3. 编写求状态通过输入符号能到达的状态集move函数 4. 编写sort函数，用来对状态集进行排序，判断新的状态集是否要加入DFA 5. 使用closure、move、sort三个函数通过子集构造法求出DFA 6. 初次划分DFA的终结状态集合和非终结状态集合并分别存入一个List<String>中。 7. 使用求异法对两个状态集一直划分——如果一个状态集的元素通过a或者b到达的状态不在该状态机里面，则划分出该元素，直到状态集不再发生不变化为止。   **程序源代码如下：**   1. **edge.java**   **package** NFAtoDFA;  **public** **class** edge{  **private** String first;  **private** String second;  **private** String third;  **public** String getSecond() {  **return** second;  }  **public** **void** setSecond(String second) {  **this**.second = second;  }  **public** String getThird() {  **return** third;  }  **public** **void** setThird(String third) {  **this**.third = third;  }  **public** String getFirst() {  **return** first;  }  **public** **void** setFirst(String first) {  **this**.first = first;  }    }   1. **NFAtoDNA.java**   package NFAtoDFA;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.ArrayList;  import java.util.HashSet;  import java.util.List;  import NFAtoDFA.edge;  public class NFAtoDFA {  public String removeMethod(String s) { // 去重  StringBuffer sb = new StringBuffer();  int len = s.length();  int i = 0;  boolean flag = false;  for (i = 0; i < len; i++) {  char c = s.charAt(i);  if (s.indexOf(c) != s.lastIndexOf(c)) {  flag = false;  } else {  flag = true;  }  if (i == s.indexOf(c))  flag = true;  if (flag) {  sb.append(c);  }  }  return sb.toString();  }  public String closure(String ss, List<edge> ee) {  String l = "";  l += ss;  for (int i = 0; i < ss.length(); i++) {  char s = ss.charAt(i);  for (edge e : ee) {  if (e.getSecond().equals("#") && e.getFirst().charAt(0) == s) {  // ss.add(e.getThird());  l += e.getThird();  }  }  }  return removeMethod(l);  }  public String move(char c, List<edge> ee, String ss) {  // List<String> moveBych = new ArrayList<String>();  String l = "";  for (int i = 0; i < ss.length(); i++) {  char s = ss.charAt(i);  for (edge e : ee) {  if (e.getFirst().charAt(0) == s && e.getSecond().charAt(0) == c) {  // moveBych.add(e.getThird());  l += e.getThird();  }  }  }  l = removeMethod(l);  l = closure(closure(l, ee), ee);  return l;  }  public String sort(String list) {  char[] s1 = list.toCharArray();  // System.out.println(s1);  for (int i = 0; i < s1.length; i++) {  for (int j = 0; j < i; j++) {  if (s1[i] < s1[j]) {  char temp = s1[i];  s1[i] = s1[j];  s1[j] = temp;  }  }  }  return String.valueOf(s1);  }  public static void main(String[] args) {  int n = 0;  NFAtoDFA nfatodaf = new NFAtoDFA();  String pathname = "E://NFA.txt";  File filename = new File(pathname);  String all = null;// 总行数  String allcondition = null;// 转移字符  List<edge> allline = new ArrayList<edge>(); // NFA转换表  InputStreamReader read;  String myfinal = "";  try {  read = new InputStreamReader(new FileInputStream(filename));  BufferedReader br = new BufferedReader(read);  // 获得总行数  try {  all = br.readLine();  int all\_int = Integer.parseInt(all);  n = all\_int;  System.out.println(all\_int);  } catch (IOException e) {  e.printStackTrace();  }  // 获得转移字符  try {  allcondition = br.readLine();  // char a = allcondition.charAt(1);  System.out.println(allcondition);  } catch (IOException e) {  e.printStackTrace();  }  // 获得转换表  for (int i = 0; i < n; i++) {  try {  String oneline = br.readLine();  char first = oneline.charAt(0);  char second = oneline.charAt(1);  char third = oneline.charAt(2);  String sf = new String();  sf += first;  String ss = new String();  ss += second;  String st = new String();  st += third;  edge oline = new edge();  oline.setFirst(sf);  oline.setSecond(ss);  oline.setThird(st);  allline.add(oline);  } catch (IOException e) {  e.printStackTrace();  }  }  try {  myfinal = br.readLine();  } catch (IOException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  }  } catch (FileNotFoundException e) {  e.printStackTrace();  }  // 打印初始的每行变换  for (edge e : allline) {  System.out.print(e.getFirst());  System.out.print(e.getSecond());  System.out.print(e.getThird());  System.out.println();  }  edge firstline = allline.get(0);  String First = firstline.getFirst();// 得到起始节点  // System.out.println(First);  List<String> Dstates = new ArrayList<String>();  // 计算起始节点的空闭包，并存入Dststes  Dstates.add(nfatodaf.sort(nfatodaf.closure(nfatodaf.closure(First, allline), allline)));  // for (String t : T)  /\*  \* 测试sort和move成功 String L = T.get(0);  \* T.add(nfatodaf.sort(nfatodaf.move(allcondition.charAt(0), allline, L)));  \* System.out.println(550); for (String t : T) { System.out.println(t); }  \*/  for (int i = 0; i < 20; i++) {  String ss = "";  // myregister.set(i, 1);  for (int j = 0; j < allcondition.length(); j++) {  ss = nfatodaf.move(allcondition.charAt(j), allline, Dstates.get(i));  ss = nfatodaf.closure(nfatodaf.closure(ss, allline), allline);  ss = nfatodaf.sort(ss);  /\*  \* for (int k = 0; k < Dstates.size(); k++) { if (Dstates.get(i).equals(ss)) {  \* nothavenew = false; break; } }  \*/  // if (ss != "") {  Dstates.add(ss);  // myregister.add(0);  // }  }  // System.out.println(Dstates);  // i++;  }  // }  Dstates = new ArrayList<String>(new HashSet<String>(Dstates));// 去重  System.out.println("------DFA状态集合-------");  // 把空项删除  for (int i = 0; i < Dstates.size(); i++) {  if (Dstates.get(i).isEmpty()) {  Dstates.remove(i);  }  }  // 测试Dstates(DFA的first),成功  for (String t : Dstates) {  System.out.println(t);  }  System.out.println("---------DFA转移表------------");  // 根据first求出dfa  List<edge> dfas = new ArrayList<edge>();  for (int i = 0; i < Dstates.size(); i++) {  for (int j = 0; j < allcondition.length(); j++) {  edge dfa = new edge();  dfa.setFirst(Dstates.get(i));  String chartoString = "";  chartoString += allcondition.charAt(j);  dfa.setSecond(chartoString);  String lastmen = "";  lastmen = nfatodaf.move(allcondition.charAt(j), allline, dfa.getFirst());  lastmen = nfatodaf.closure(nfatodaf.closure(lastmen, allline), allline);  lastmen = nfatodaf.sort(lastmen);  dfa.setThird(lastmen);  dfas.add(dfa);  }  }  for (int i = 0; i < dfas.size(); i++) {  System.out.print(dfas.get(i).getFirst());  System.out.print(dfas.get(i).getSecond());  System.out.print(dfas.get(i).getThird());  System.out.println();  }  List<String> classnotfinal = new ArrayList<String>();  List<String> classfinal = new ArrayList<String>();  /\*  \* for (int i = 0; i < dfas.size(); i++) { for(int j = 0 ;  \* j<dfas.get(i).getFirst().length();j++) { if(dfas.get(i).getFirst().charAt(j)  \* == myfinal.charAt(0)) { classfinal.add(dfas.get(i).getFirst()); }else {  \* classnotfinal.add(dfas.get(i).getFirst()); } } }  \*/  for (int i = 0; i < Dstates.size(); i++) {  for (int j = 0; j < Dstates.get(i).length(); j++) {  if (Dstates.get(i).charAt(j) == myfinal.charAt(0)) {  classfinal.add(Dstates.get(i));  continue;  } else {  classnotfinal.add(Dstates.get(i));  }  }  }  classnotfinal = new ArrayList<String>(new HashSet<String>(classnotfinal));  // classfinal = new ArrayList<String>(new HashSet<String>(classfinal));  for (int i = 0; i < classfinal.size(); i++) {  for (int j = 0; j < classnotfinal.size(); j++) {  if (classfinal.get(i).equals(classnotfinal.get(j))) {  classnotfinal.remove(j);  }  }  }  System.out.println("初次划分DFA终结符号集合");  for (int i = 0; i < classfinal.size(); i++) {  System.out.print(classfinal.get(i));  System.out.println();  }  System.out.println("初次划分DFA非终结符号集合");  for (int i = 0; i < classnotfinal.size(); i++) {  System.out.print(classnotfinal.get(i));  System.out.println();  }  List<String> newcla = new ArrayList<String>();  List<String> realnewcla = new ArrayList<String>();  for (int j = 0; j < classfinal.size(); j++) {  for (int i = 0; i < allcondition.length(); i++) {  for (int k = 0; k < classfinal.size(); k++) {  if (nfatodaf.move(allcondition.charAt(i), allline, classfinal.get(j)).equals(classfinal.get(k))) {  newcla.add(classfinal.get(j));  }  }  }  }  for (int i = 0; i < newcla.size(); i++) {  for (int j = 0; j < classfinal.size(); j++) {  if (newcla.get(i).equals(classfinal.get(j))) {  classfinal.remove(j);  }  }  }  realnewcla = classfinal;  System.out.println("ppppppppppppppppppppppppppp");  newcla = new ArrayList<String>(new HashSet<String>(newcla));  for (int i = 0; i < realnewcla.size(); i++) {  System.out.print(realnewcla.get(i));  System.out.println();  }  System.out.println("ppppppppppppppppppppppppppp");  for (int i = 0; i < newcla.size(); i++) {  System.out.print(newcla.get(i));  System.out.println();  }      List<String> newnotcla = new ArrayList<String>();  List<String> realnewnotcla = new ArrayList<String>();  for (int j = 0; j < classnotfinal.size(); j++) {  for (int i = 0; i < allcondition.length(); i++) {  for (int k = 0; k < classnotfinal.size(); k++) {  if (nfatodaf.move(allcondition.charAt(i), allline, classnotfinal.get(j)).equals(classnotfinal.get(k))) {  newnotcla.add(classnotfinal.get(j));  }  }  }  }  for (int i = 0; i < newnotcla.size(); i++) {  for (int j = 0; j < classnotfinal.size(); j++) {  if (newnotcla.get(i).equals(classnotfinal.get(j))) {  classnotfinal.remove(j);  }  }  }  realnewnotcla = classnotfinal;  System.out.println("ssssssssssssssssssss");  newnotcla = new ArrayList<String>(new HashSet<String>(newnotcla));  for (int i = 0; i < realnewnotcla.size(); i++) {  System.out.print(realnewnotcla.get(i));  System.out.println();  }  System.out.println("sssssssssssssssssssssssss");  for (int i = 0; i < newnotcla.size(); i++) {  System.out.print(newnotcla.get(i));  System.out.println();  }  }  }  **测试程序如下：** | |
| 1. **实验结果：** | |
| 1. **实验总结：**   在本次实验中，我的程序还不是很完美，比如，如果状态是两个字符及以上则程序无法分辨，但是对于状态小于10个的转换表，该程序可以分析出实验结果。这次实验让我加深了对NFA,DFA的理解，实现编写NFA到DFA的转化以及DFA的化简过程中遇到了许多困难，让我不断地去查阅资料，增长了自己的知识，丰富了自己的实践经验。 | |