【Experiment name】

Deterministic finite automaton minimization

【Purpose】

Input: DFA
Output: minimized DFA

【Experimental principle】

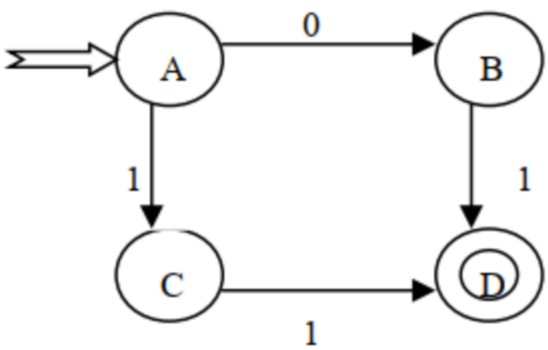
The essence of the determinization of NFA is to use a subset of the original state set as a state on the DFA, and convert
the transition between the original states into the transition between the subsets, so as to determinize the uncertain finite automata. After determinization, the number of states may increase, and some equivalent states may appear, so simplification is required.

The so-called automaton simplification problem is to construct another definite finite automaton DFA M' for any definite finite automaton DFA M', there is L(M)=L(M'), and the number of states of M' The number of states is not more than M, and it can be said with certainty that an M' with the smallest number of states can be found.

Let's first introduce some related basic concepts. Let S_i be a state of automaton M, and the set of all symbol strings that can be derived from S_i is denoted as $L(S_i)$.

There are two states S_i and S_j, if $L(S_i)=L(S_j)$, then S_i and S_j are said to be equivalent states.

In the automaton shown in the figure below, L(B)=L(C)={1}, all states B and C are equivalent states.



Another example is that the set of symbol strings derived from the terminal state must contain the empty string ε, but the set of symbol strings derived from the non-terminal state cannot contain the empty string ε, so the final state and the non-terminal state are not equivalent.

For the concept of equivalence, we can also give a definition from another angle.

Given a DFA M, if starting from some state P and taking a string w as input, DFA M will end in a final state, and starting from another state Q, taking a string w as input, DFA M will end in a non- The terminal state is called the string w to distinguish the state P from the state Q.

Two states that are indistinguishable are called equivalent states.

Let S_i be a state of automaton M, if it is impossible to reach this state $S_{i \text{ from the start state}}$, then

$S_{i \text{ is said}}$ to be a useless state.

Let S_i be a state of automaton M, if for any input symbol a goes to itself, and it is impossible to reach the terminal state, then S

i is said to be a dead state

to simplifying DFA is to divide its state set into some disjoint subsets, so that the states between any two disjoint subsets are distinguishable, and any two states in the same subset are Equivalent, in this way, one state can be used as a representative to delete other equivalent states, and then the irrelevant state can be deleted, and the DFA with the smallest number of states can be obtained.

The following is a detailed introduction to the simplification algorithm of DFA:

(1) First, divide the state of DFA M into a terminal state set K1 and a non-terminal state set K2.

K=K1∪K2

From the above definition, K1 and K2 are not equivalent.

(2) Each state set is further divided by the following method each time until no new division is generated.

Assume that the i-th division has divided the state set into k groups, namely:

K=K1(i)∪K2(i)∪…∪Kk(i)

For each state in the state set Kj(i) (j=1,2,...,k), check one by one, there are two states Kj', Kj''∈Kj(i), and for the input symbol a, there are:

F(Kj',a)=Km

F(Kj'', a) = Kn

If Km and Kn belong to the same state set, put Kj' and Kj'' in the same set,

Otherwise divide Kj' and Kj'' into two sets.

(3) Repeat step (2) until each set can no longer be divided, at this time each state set

The states in the combination are all equivalent.

(4) Merge equivalent states, that is, take any state in the equivalent state set as a representative, delete its

All other equivalent states.

(5) If there is an irrelevant state, delete it.

According to the above method, the definite finite automaton is simplified, and the simplified automaton is the original

An automaton with the fewest states.

【Experimental content】

According to the requirements of the experiment, the division processing function of dfa is designed to realize the minimization operation of DFA, which is realized by using python , especially using the numpy library .

Among them, python is used for programming, and in order to standardize the input and output formats of the model machine, json format is specially used for processing.

Finally, using D igraph in the graphviz library , the state diagrams of d fa and m dfa are output based on the files in json format .

core module

(1). Main function module (Main)

```python
def main():
    # 获取DFA.json
    dfa_input = "DFA2.json"
    (k_set, e_set, fc, s_set, z_set) = read(dfa_input)
    dfa = json.load(open(dfa_input, "r"))

    # 展示并存储dfa图像
    showMachine(dfa, 'dfa', 'dfa')

    # 获取dfa的表格
    dfa, s, z = create_dfa_matrix(k_set, e_set, fc, s_set, z_set)

    # dfa最小化
    mdfa_matrix, start, end = compress(dfa, s, z)

    # 生成mdfa(对应json格式)
    mdfa = create_mdfa(mdfa_matrix, e_set, start, end)

    # 写mdfa.json
    dfa_output = 'mdfa.json'
    write_dfa(mdfa, dfa_output)

    # 展示并存储mdfa的图像
    showMachine(mdfa, 'mdfa', 'mdfa')
```

Figure: Main function module (Main)

Ideas:

Mainly divided into 5 steps

①Read the d fa. json file (in a standardized automaton format, stored in json)

② Process to get the representation matrix of d fa
③Process the representation matrix in ②, minimize d fa , and obtain the f set, s set, and z set of m dfa (minDFA)
dfa. json based on the obtained sets of quintuples
⑤Display the state diagram of d fa and m dfa

(2) . Segmentation module (compress)

① Eliminate useless state

```python
def compress(dfa, s, z, k_set):
    """删去<<不可及，无所终>>的行"""
    # 获取有哪些终态节点
    element = list(map(lambda x: int(x), k_set))
    fin = False

    temp = []
    for i in range(len(z)):
        temp.append(z[i])
    # 设置bool型operation，根据dfa是否遍历完成，判断操作是否完成
    operation = False
    while (True):
        for i in range(dfa.shape[0]):
            for j in range(dfa.shape[1]):
                if dfa[i][j] in temp and dfa[i][0] not in temp:
                    temp.append(dfa[i][0])
                    operation = True
        if not operation:
            break
        else:
            operation = False

    # 删除不合规则的行
    k = 0
    for i in range(dfa.shape[0]):
        if dfa[i - k][0] not in temp:
            dfa = np.delete(dfa, i - k, axis=0)
            element = np.delete(element, i - k, axis=0)
            k = k + 1
    del temp

    for i in range(dfa.shape[0]):
        for j in range(dfa.shape[1]):
            if dfa[i][j] not in element:
                # return False
                raise Exception("dfa不规范")
```

Figure: process _dfa function module, eliminate useless state

Ideas:

①Take the s set as the initial entry, let the known final state nodes in d fa be "legal"

②Based on the state input, check whether the state output exists in the z set (it is the final state node), if so, mark the input as "legal"; if not, continue to traverse the output state

③If the state has been traversed, mark it as "traversed"

④Traverse the state matrix again, and update it according to the result of ③. If the corresponding output of a certain state has a ″legal″ mark (because ② may not be the final state node but can reach the final state node, it becomes ″legal″), then the result point

②Segmentation ″classification″ core algorithm

```python
"""各行归类"""
while (True):
    # 遍历矩阵，对照team归类字典，生成spy分类值矩阵
    for i in range(dfa_row):
        for j in range(dfa_col):
            for m in range(dfa_row):
                if dfa[m][0] == dfa[i][j]:
                    spy[i][j] = team[m]

    # 对spy分类值矩阵做处理，视一行中的各列相连生成一个二进制值，并以此生成team分类label
    num = 0
    for i in range(spy.shape[0]):
        for j in range(spy.shape[1]):
            num = num * 10 + spy[i][j]
        team[i] = num
        num = 0

    # 获取team中label总个数(过滤重复label)
    team_length = len(team)
    for i in range(len(team)):
        for j in range(i + 1, len(team)):
            if team[i] == team[j]:
                team_length = team_length - 1

    # 一旦本轮分类和上一轮分类情况相同，即无法再分类，跳出分类模块
    num_label_pre = num_label
    num_label = team_length
    if num_label_pre == num_label:
        break
```

Figure: Congress function module, split "classification" core

Ideas:

① Set empty s py, team matrix for subsequent use

② Generate a ″classification matrix″ s py corresponding to the same size as the d fa state matrix based on the label . At this time, label has two values 0 and 1, which are non-terminal symbols respectively. (For example, state 4 belongs to a non-terminal symbol, and the corresponding value is 0; state 5, belongs to a terminal symbol, and the corresponding value is 1)

③Analyze the data in s py , each line (i, :) is a unit, if there is any difference, it will be regarded as belonging to two different labels , and reclassified based on s py , and the label information corresponding to each state will be stored in team

team obtained in ③ , update s py to d fa (same as ②) , then analyze and update label (same as 3) , and repeat until no new label is generated.

③ Simplify the correlation matrix to obtain the final minimum d fa state matrix and the corresponding initial state and final state

```python
"""team正规化"""
# 由于team是对照original dfa生成的，最终难免有"k"归属同一类（即会有重复）
# 对team正规化删除重复行，最终所得结果恰为team中的独一无二的各种label

# 此时temp中数字都是(num=num*10+array[i])中产生的大数，将这些数按0,1,2...重置（更新label名称）
temp = []
for i in range(len(team)):
    if team[i] not in temp:
        temp.append(team[i])
for i in range(len(team)):
    team[i] = temp.index(team[i])
del temp
# 按新label名称更新spy
for i in range(dfa_row):
    for j in range(dfa_col):
        for m in range(dfa_row):
            if dfa[m][0] == dfa[i][j]:
                spy[i][j] = team[m]
# 设置sz标记"初态节点s","终态节点z"，格式如"1000222",1表示出发节点，2表示终态节点
sz = np.zeros(dfa_row).astype(np.int)
for i in range(dfa_row):
    for j in range(len(s)):
        if dfa[i][0] == s[j]:
            sz[i] = 1
    for j in range(len(z)):
        if dfa[i][0] == z[j]:
            sz[i] = 2
```

```python
    # 将spy和sz中重复行删除，获取最终最简的mdfa信息
    temp = []
    j = 0
    for i in range(dfa_row):
        if team[i] not in temp:
            temp.append(team[i])
        else:
            spy = np.delete(spy, i - j, axis=0)
            sz = np.delete(sz, i - j, axis=0)
            j = j + 1
    del temp

    """对sz做最终处理，由sz获取mdfa中的s_set, e_set"""
    start = []
    end = []
    for i in range(len(sz)):
        if sz[i] == 1:
            start.append(i)
        elif sz[i] == 2:
            end.append(i)
    print("spy:")
    print(spy)
    print("compress模块完成\n")

    return spy, start, end
```

Figure: congress module, simplifies the team that stores label

Ideas:

It can be known from the previous module that the final minimized dfa matrix can no longer be divided, and a new classification can be generated, so it can be output. In the final obtained team , the label is stored in numbers , the numbers are large and irregular, and the team needs to be normalized

①The label of t eam is changed to 0, 1, 2...

② Update s py (classification matrix) based on t eam

③Set s z to record the initial state and the corresponding state of the final state in the original d fa

③ Delete the repeated lines in s py and sz to realize the final minimization; from s z , it can be known which of the minimized d fa is the new initial state and which is the new final state

other modules

(3) create_dfa_matrix: Generate a state matrix based on the five-tuple read by json

```python
def create_dfa_matrix(k_set, e_set, fc, s_set, z_set):
    """获取dfa的表格，以np.array形式存储"""
    s = list(map(lambda x: int(x), s_set))
    z = list(map(lambda x: int(x), z_set))
    row = len(fc)
    col = len(e_set) + 1
    dfa = -np.ones((row, col)).astype(np.int)

    for i in range(len(k_set)):
        dfa[i][0] = k_set[i]
    for i in range(row):
        for j in range(1, col):
            dfa[i][j] = fc['%d' % (i + 1)][e_set[j - 1]]  ############
    print("dfa_matrix:")
    print(dfa)
    print("create_dfa_matrix模块完成\n")
    return dfa, s, z
```

Figure: State matrix creation module

Ideas:

matrix according to the five-tuple

② Fill in the status representation to the corresponding position

③ c reate_mdfa : Generate m dfa model information that can be stored in the . json file according to the five-tuple obtained after segmentation

```python
def create_mdfa(mdfa_matrix, e_set, start, end):
    """生成mdfa，对应json格式"""
    mdfa_row = mdfa_matrix.shape[0]
    mdfa_col = mdfa_matrix.shape[1]

    mdfa = {}
    mdfa["k"] = []
    mdfa["e"] = []
    mdfa["f"] = {}
    mdfa["s"] = []
    mdfa["z"] = []

    # k集导入
    for i in range(mdfa_row):
        mdfa["k"].append(str(mdfa_matrix[i][0]))
    # e集导入
    for i in range(len(e_set)):
        mdfa["e"].append(e_set[i])
    # f集导入
    for x in range(mdfa_row):
        a = str(x)
        mdfa["f"][a] = {}
        for i in range(len(e_set)):
            mdfa["f"][a][e_set[i]] = str(mdfa_matrix[x][i + 1])
    # s集导入
    for i in range(len(start)):
        mdfa["s"].append(str(start[i]))
    # z集导入
    for i in range(len(end)):
        mdfa["z"].append(str(end[i]))

    print("mdfa:")
    print(mdfa)
    print("create_mdfa模块完成\n")

    return mdfa
```

Figure: dfa 's .json format information generation module

Ideas:

①Based on the characteristics of the .json file , create k set, e set, s set, z set list , f set is { }

② Import characters to each episode

④File reading and writing module (read, write_dfa)

```python
def read(input):
    fa = json.load(open(input, "r"))
    for i in fa["f"]:
        if not i in fa["k"]:
            raise Exception("Set f contains iterms that not belongs to set k.")
        for j in fa["f"][i]:
            if not j in fa["e"] and not j == '#':
                raise Exception("Set f contains iterms that not belongs to set e.")
    return list(fa["k"]), list(fa["e"]), fa["f"], list(fa["s"]), list(fa["z"])
```

```python
def write_dfa(dfa, f):
    f = open(f, "w")
    f.write(json.dumps(dfa, indent=4))  # 基于json写文件
    f.close()
```

Figure: Read and write module (s howMachine)

⑤ State diagram creation module based on graohviz (showMachine)

```python
def showMachine(fa, name, imageput):
    """显示并存储基于json格式的自动机模型"""
    # 初始化
    dot = Digraph(name="%s" % name, comment="the test %s" % name, format="png")
    dot.attr(label=r'\%s' % name, fontsize='20')
    print("创建 %s 流程图" % name)

    # 创建节点
    num_k = len(fa["k"]) - 1  # num_k:最大元素数字(即元素个数减1)
    for i in range(num_k):
        dot.node(fa["k"][i])  # 0~k元素, 设置每个节点

    # 特殊部分
    # 尾节点double-circle形状
    for i in range(len(fa["z"])):
        dot.node(fa["z"][i], shape='doublecircle')
    # 指向首节点的箭头
    dot.node("fake", style='invisible')
    for i in range(len(fa["s"])):
        dot.edge("fake", fa["k"][i], style='bold')

    # 创建路径
    for i in range(num_k + 1):  # 0~k个元素, 设置每条路径
        begin = fa["k"][i]
        node_start = fa["k"][i]  # 设置路径起点
        if node_start in fa["f"]:
            for n in fa["e"] + ["#"]:  # 有 n 种路径可能
                if n in fa["f"][begin]:  # 判断该起点下有什么路径
                    num_f = len(fa["f"][begin][n])  # 获取需创建路径数目
                    for j in range(num_f):  # 有 j 种终点可能
                        node_end = fa["f"][begin][n][j]  # 获取终点
                        dot.edge(node_start, node_end, label="%s" % n)  # 创建路径, 基于“起点-路径-终点”
                        if num_f == 1:
                            break
    # 图像显示及存储
    # dot.view()
    dot.render(imageput, view=True)
```

Figure: State diagram creation module (s howMachine)

【Experimental Results】

graphviz - based model machine state diagram generation:

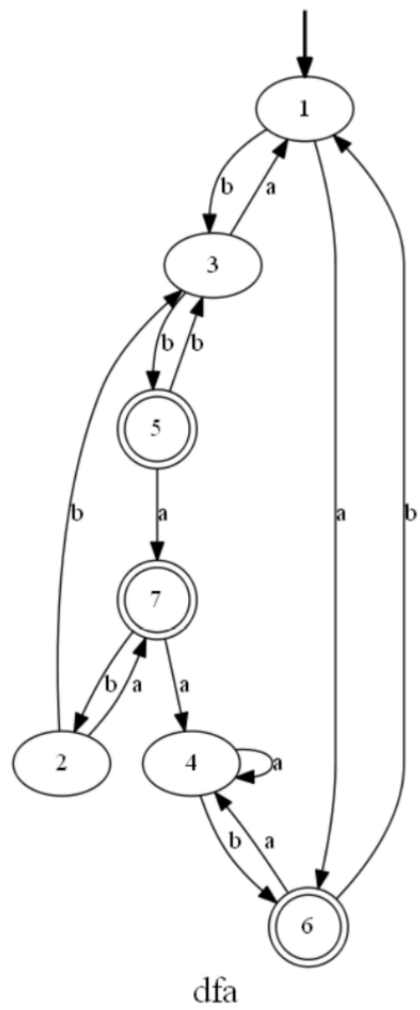
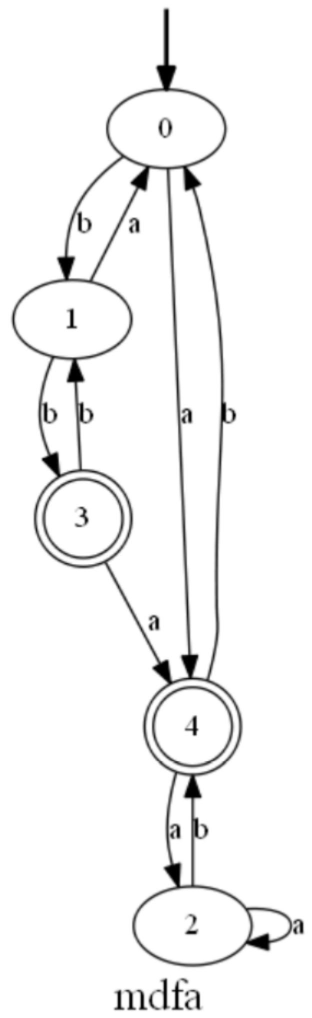
	

Figure: DFA (Example 1)	Figure: M in D FA (Example 1)

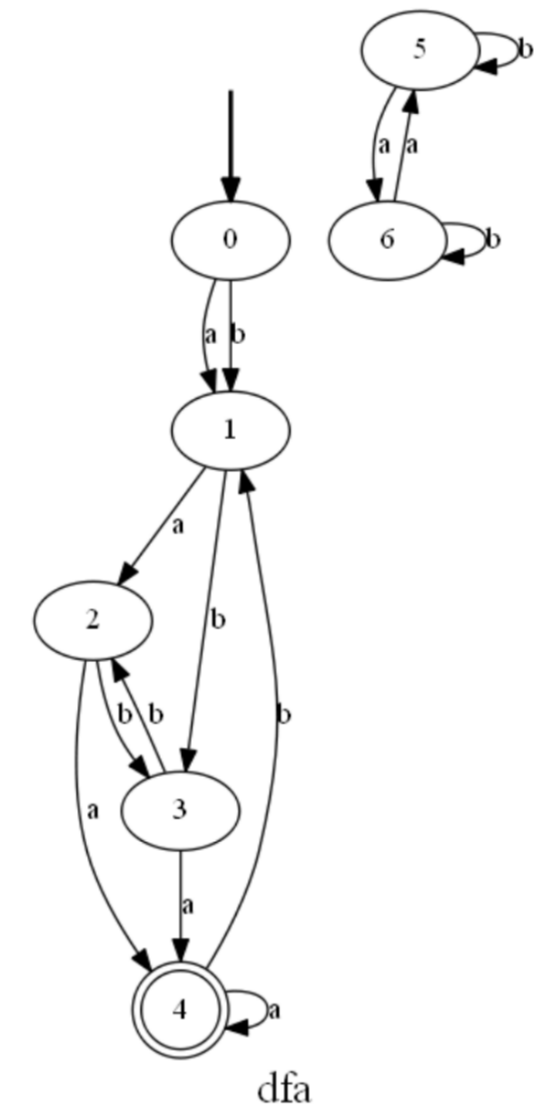
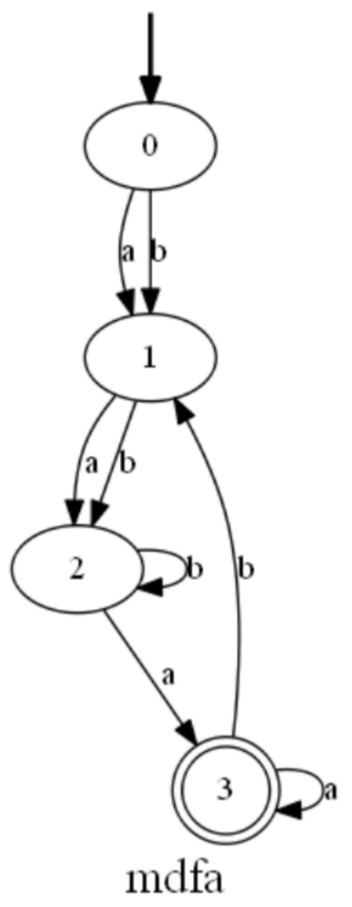
	

Figure: DFA (Example 2)	Figure: MinDFA (Example 2)

【Experiment Summary】

In this experiment, I first carefully reviewed the content of minimizing DFA in Chapter 3 "Lexical Analysis" of "Compiler Principles " .

There are two main processes of minimization, namely "eliminating useless states" and "merging equivalent states". Among them, a method called "segmentation method" is introduced in the book on merging equivalent states, which is convenient for practical use .

Among them, how to effectively realize the "segmentation method" is the most difficult. After careful thinking, I found a simple method to effectively achieve "segmentation". First, set the final state to 1, and the non-final state to 0. We call it label and mark all the states in the state matrix as shown in the figure below (as shown in blue next to each number in the matrix). Taking the example on page 54 of the book as an example, there are only 2 types of labels for the first division. After marking the state matrix, we refer to the method of converting binary to decimal, and connect the numbers from left to right in each row, and the obtained numbers are integrated. The starting node status and all path information can be expressed as one type, such as 1, 6, 3 is 0 , 10,010 is a kind of label , so that it can be reset and divided into 4 label in the second time , and in the third time Divide 5 labels . At this time, it can be clearly seen that even if the labels are divided for the fourth time, there are still 5 labels, which cannot be further divided, that is, the minimization has been achieved.

The above ideas are the core of the algorithm in this experiment. At the same time, in terms of code implementation, I divided 7 modules and designed all the codes. During the specific implementation process, I especially learned some methods about n umpy library, and better use list to realize data recording.

Finally, through this experimental practice, I have improved my understanding of the finite automaton (DFA) minimization method, better enhanced my code ability, and laid a solid foundation for the next experiment.

【Experiment code】

Appendix 1

Compile_3.py

```
import numpy as np
import json
from graphviz import Digraph


def read(input):
fa = json. load(open(input, "r"))
    for i in fa["f"]:
        if not i in fa["k"]:
            raise Exception("Set f contains iterms that not belongs to set k.")
        for j in fa["f"][i]:
            if not j in fa["e"] and not j == '#':
                raise Exception("Set f contains iterms that not belongs to set e.")
    return list(fa["k"]), list(fa["e"]), fa["f"], list(fa["s"]), list(fa["z"])


def showMachine(fa, name, imageput):
    """显示并存储基于 json 格式的自动机模型"""
    # 初始化
    dot = Digraph(name="%s" % name, comment="the test %s" % name, format="png")
dot.attr(label=r'\%s' % name, fontsize='20')
print("Create %s flowchart" % name)

# create node
num_k = len(fa["k"]) - 1 # num_k: the maximum number of elements (that is, the number of
elements minus 1)
for i in range(num_k):
dot.node(fa["k"][i]) # 0~k elements, set each node

# special part
# tail node double-circle shape
for i in range(len(fa["z"])):
dot.node(fa["z"][i], shape='doublecircle')
# Arrow pointing to the first node
dot.node("fake", style='invisible')
for i in range(len(fa["s"])):
dot.edge("fake", fa["k"][i], style='bold')

# create path
for i in range(num_k + 1): # 0~k elements, set each path
begin = fa["k"][i]
node_start = fa["k"][i] # set the starting point of the path
if node_start in fa["f"]:
```

```
for n in fa["e"] + ["#"]: # There are n possible paths
if n in fa["f"][begin]: # Determine what path exists under the starting point
num_f = len(fa["f"][begin][n]) # Get the number of paths to be created
for j in range(num_f): # There are j possible end points
node_end = fa["f"][begin][n][j] # Get the end point
dot.edge(node_start, node_end, label="%s" % n) # Create a path based on "start-path-end"
if num_f == 1:
break
# Image display and storage
# dot. view()
dot. render(imageput, view=True)


def compress(dfa, s, z, k_set):
"""Delete <<unreachable, no end>> line"""
# Get which final state nodes
element = list(map(lambda x: int(x), k_set))
fin = False

temp = []
for i in range(len(z)):
temp.append(z[i])
# Set bool type operation, judge whether the operation is completed according to whether the
traversal of dfa is completed
operation = False
while (True):
for i in range(dfa. shape[0]):
for j in range(dfa.shape[1]):
if dfa[i][j] in temp and dfa[i][0] not in temp:
temp.append(dfa[i][0])
operation=True
if not operation:
break
else:
operation = False

# delete outlier lines
k = 0
for i in range(dfa. shape[0]):
if dfa[i - k][0] not in temp:
dfa = np.delete(dfa, i - k, axis=0)
element = np.delete(element, i - k, axis=0)
k = k + 1
del temp
```

```
for i in range(dfa. shape[0]):
for j in range(dfa.shape[1]):
if dfa[i][j] not in element:
# return False
raise Exception("dfa is not standardized")

# Update the number of rows and columns of dfa
dfa_row = dfa.shape[0]
dfa_col = dfa.shape[1]

"""Create spy, team"""
"""Spy stores the classification value (shadow) of dfa, and team stores the class to which each row belongs, which can also be called label"""
spy = -np.ones((dfa_row, dfa_col)).astype(np.int)
team = np.zeros(dfa_row).astype(np.int)
for i in range(dfa_row): #team initialization
if dfa[i][0] in z:
team[i] = 1 # Initially, use 1 to represent the terminal class, and 0 to represent the non-terminal class
num_label = 2

"""Classify each row"""
while (True):
# Traverse the matrix, compare the team classification dictionary, and generate a spy classification value matrix
for i in range(dfa_row):
for j in range(dfa_col):
for m in range(dfa_row):
if dfa[m][0] == dfa[i][j]:
spy[i][j] = team[m]

# Process the spy classification value matrix, connect the columns in a row to generate a binary value, and use this to generate a team classification label
num = 0
for i in range(spy. shape[0]):
for j in range(spy. shape[1]):
num = num * 10 + spy[i][j]
team[i] = num
num = 0

# Get the total number of labels in the team (filter duplicate labels)
team_length = len(team)
for i in range(len(team)):
```

```
for j in range(i + 1, len(team)):
if team[i] == team[j]:
team_length = team_length - 1

# Once the current round of classification is the same as the previous round of classification, it can no longer be classified and jump out of the classification module
num_label_pre = num_label
num_label = team_length
if num_label_pre == num_label:
break

"""Team normalization"""
# Since the team is generated against the original dfa, it is inevitable that "k" will eventually belong to the same category (that is, there will be duplication)
# Normalize the team to delete duplicate lines, and the final result is exactly the unique labels in the team

# At this time, the numbers in temp are all large numbers generated in (num=num*10+array[i]), reset these numbers by 0, 1, 2... (update the label name)
temp = []
for i in range(len(team)):
if team[i] not in temp:
temp.append(team[i])
for i in range(len(team)):
team[i] = temp. index(team[i])
del temp
# Update spy by new label name
for i in range(dfa_row):
for j in range(dfa_col):
for m in range(dfa_row):
if dfa[m][0] == dfa[i][j]:
spy[i][j] = team[m]
# Set the sz mark "initial state node s", "final state node z", format such as "1000222", 1 represents the starting node, 2 represents the final state node
sz = np.zeros(dfa_row).astype(np.int)
for i in range(dfa_row):
for j in range(len(s)):
if dfa[i][0] == s[j]:
sz[i] = 1
for j in range(len(z)):
if dfa[i][0] == z[j]:
sz[i] = 2
# Delete duplicate lines in spy and sz to get the final and simplest mdfa information
temp = []
```

```
j = 0
for i in range(dfa_row):
if team[i] not in temp:
temp.append(team[i])
else:
spy = np.delete(spy, i - j, axis=0)
sz = np.delete(sz, i - j, axis=0)
j = j + 1
del temp

"""Do final processing on sz, get s_set, e_set in mdfa by sz"""
start = []
end = []
for i in range(len(sz)):
if sz[i] == 1:
start.append(i)
elif sz[i] == 2:
end.append(i)
print("spy:")
print(spy)
print("compress module completed\n")

return spy, start, end



def create_dfa_matrix(k_set, e_set, fc, s_set, z_set):
"""Get the table of dfa and store it in the form of np.array"""
s = list(map(lambda x: int(x), s_set))
z = list(map(lambda x: int(x), z_set))
row = len(fc)
col = len(e_set) + 1
dfa = -np.ones((row, col)).astype(np.int)

    for i in range(len(k_set)):
        dfa[i][0] = k_set[i]
    for i in range(row):
        for j in range(1, col):
            dfa[i][j] = fc[k_set[i]][e_set[j - 1]]    ###########
    print("dfa_matrix:")
    print(dfa)
    print("create_dfa_matrix 模块完成\n")
    return dfa, s, z
```

```
def create_mdfa(mdfa_matrix, e_set, start, end):
        """生成 mdfa，对应 json 格式"""
        mdfa_row = mdfa_matrix.shape[0]
        mdfa_col = mdfa_matrix.shape[1]

        mdfa = {}
        mdfa["k"] = []
        mdfa["e"] = []
mdfa["f"] = {}
mdfa["s"] =
mdfa["z"] =

# k 集导入
for i in range(mdfa_row):
mdfa["k"].append(str(mdfa_matrix[i][0]))
# e 集导入
for in in range(len(e_set)):
mdfa["e"].append(e_set[i])
# f 集导入
for x in range(mdfa_row):
a = str(x)
mdfa["f"][a] = {}
for in in range(len(e_set)):
mdfa["f"][a][e_set[i]] = str(mdfa_matrix[x][i+1]);
# s 集导入
for i in range(len(start)):
mdfa["s"].append(str(start[i]))
# z set import
for i in range(len(end)):
mdfa["z"].append(str(end[i]))

print("mdfa:")
print(mdfa)
print("create_mdfa module completed\n")

return mdfa


def write_dfa(dfa, f):
f = open(f, "w")
f.write(json.dumps(dfa, indent=4)) # Write files based on json
f. close()
```

```
def main():
# Get DFA.json
# dfa_input = "DFA2.json"
dfa_input = "DFA1.json"
(k_set, e_set, fc, s_set, z_set) = read(dfa_input)
dfa = json.load(open(dfa_input, "r"))

# Display and store the dfa image
showMachine(dfa, 'dfa', 'dfa')¶

# Get the form of dfa
dfa, s, z = create_dfa_matrix(k_set, e_set, fc, s_set, z_set)

# dfaminimization
mdfa_matrix, start, end = compress(dfa, s, z, k_set)

# Generate mdfa (corresponding to json format)
mdfa = create_mdfa(mdfa_matrix, e_set, start, end)

# write mdfa.json
dfa_output = 'mdfa.json'
write_dfa(mdfa, dfa_output)

# Display and store the image of mdfa
showMachine(mdfa, 'mdfa', 'mdfa')



if __name__ == '__main__':
main()
```

Appendix 2:

DFA1.json


```
{
"k": [
"0",
"1",
"2",
"3",
"4",
"5",
"6"
```

```
],
"e": [
"a",
"b"
],
"f": {
"0": {
"a": "1",
"b": "1"
},
"1": {
			"a": "2",
			"b": "3"
		},
		"2": {
			"a": "4",
			"b": "3"
		},
		"3": {
			"a": "4",
			"b": "2"
		},
		"4": {
			"a": "4",
			"b": "1"
		},
		"5": {
			"a": "6",
			"b": "5"
		},
		"6": {
			"a": "5",
			"b": "6"
		}
	},
	"s": [
		"0"
	],
	"z": [
		"4"
	]
}
```

DFA2.json


```
{
	"k": [
		"1",
		"2",
		"3",
		"4",
		"5",
		"6",
		"7"
	],
	"e": [
		"a",
		"b"
	],
	"f": {
		"1": {
			"a": "6",
			"b": "3"
		},
		"2": {
			"a": "7",
			"b": "3"
		},
		"3": {
			"a": "1",
			"b": "5"
		},
		"4": {
			"a": "4",
			"b": "6"
		},
		"5": {
			"a": "7",
			"b": "3"
		},
		"6": {
			"a": "4",
			"b": "1"
		},
		"7": {
			"a": "4",
```

```
                "b": "2"
            }
        },
        "s": [
            "1"
        ],
"z":
"5",
"6"
"7"
]
} }
```

附录 3:

MinDFA1.json

```
{
"k":
"0"
"1",
"2",
"3",
"4"
],
"e":
"a"
"b"
],
"f": {
"0": {
"a": "4",
                "b": "1"
            },
            "1": {
                "a": "0",
                "b": "3"
            },
            "2": {
                "a": "2",
                "b": "4"
            },
            "3": {
```

```
                "a": "4",
                "b": "1"
            },
            "4": {
                "a": "2",
                "b": "0"
            }
        },
        "s": [
            "0"
        ],
        "z": [
            "3",
            "4"
        ]
}
```


MinDFA2.json


```
{
        "k": [
            "0",
            "1",
            "2",
            "3"
        ],
        "e": [
            "a",
            "b"
        ],
        "f": {
            "0": {
                "a": "1",
                "b": "1"
            },
            "1": {
                "a": "2",
                "b": "2"
            },
            "2": {
                "a": "3",
                "b": "2"
            },
```

```
        "3": {
            "a": "3",
            "b": "1"
        }
    },
    "s": [
        "0"
    ],
    "z": [
        "3"
    ]
}
```