【Experiment name】

Determinization of Uncertain Finite Automata

【Purpose】

Input: Non-deterministic finite (poor) state automata.
Output: Determinized finite (poor) state automata

【Experimental principle】

A deterministic finite automaton (DFA) M can be defined as a quintuple, M = (K, Σ, F, S, Z), where:

(1) K is a finite non-empty set, and each element in the set is called a state;

(2) Σ is a finite alphabet, and each element in Σ is called an input symbol;

(3) F is a single-valued conversion function from K×Σ→K, that is, F(R, a)=Q, (R, Q∈K) means that the current state is R, and if the character a is input, it will go to the state Q, state Q is called the successor state of state R;

(4) S∈K is the only initial state;

(5) Z ⊆K, is a final state set.

It can be seen from the definition that a deterministic finite automaton has only one initial state, but can have multiple final states, and each state has at most one successor state for any input symbol in the alphabet.

For DFA M, if there is a path from a certain initial state node to a certain final state node, it is said that the string formed by connecting the markers of all arcs on this path can be accepted by DFA M. If the initial state node of M is also the final state node, it is said that ε can be accepted (or recognized) by M, and the set of all character strings (words) that DFA M can accept is denoted as L(M) .

An uncertain finite automaton (NFA) M can be defined as a quintuple, M = (K, Σ, F, S, Z), where:

(1) k is a finite non-empty set, and each element in the set is called a state;

(2) Σ is a finite alphabet, and each element in Σ is called an input symbol;

(3) F is a transformation function from a subset of K×Σ→K;

(4) S⊆ K, is a non-empty initial state set;

(5) Z⊆ K, is a final state set.

It can be seen from the definition that the main difference between an uncertain finite automaton NFA and a deterministic finite automaton DFA is:

state;

(2) NFA allows states to have the same symbol on an output side, that is, there can be multiple successor states for the same input symbol . That is, F in DFA is a single-valued function, while F in NFA is a multi-valued function.

Therefore, the deterministic finite automaton DFA can be regarded as a special case of the deterministic finite automaton NFA. Like DFA, NFA can also be represented by matrices and state transition diagrams.

For NFA M, if there is a path from an initial state node to a certain final state node, it is said that the string formed by the connection of all arc labels (except ε) on this path can be accepted by M. The set of all character strings (words) that NFA M can accept is denoted as L(M).

Since DFA is a special case of NFA, the symbol strings that can be accepted by DFA must be acceptable by NFA.

Suppose M1 and M2 are finite automata on the same letter set Σ, if L(M1)=L(M2), then the
finite automata M1 and M2 are said to be equivalent.

From the above definition, two automata are said to be equivalent if they can accept the same language
. DFA is a special case of NFA, so for every NFA M1 there always exists a DFA M2 such that
L(M1)=L(M2). That is, a language that can be accepted by an uncertain finite automaton can always find an equivalent
deterministic finite automaton to accept the language.

NFA determined into DFA

The same string α can be generated by multiple paths, and in practical applications, as an automaton describing the control process, it is usually a deterministic finite automaton DFA, so it is necessary to convert the uncertain finite automaton into an equal The deterministic finite automaton of the price, this process is called the determinization of the uncertain finite automata,

That is, NFA is deterministically transformed into DFA.

The following introduces a deterministic algorithm for NFA, which is called the subset method:

(1) If all the initial states of NFA are S1, S2, ..., Sn, then let the initial state of DFA be:

S = [S1, S2, ..., Sn],

The square brackets are used to indicate a certain state composed of several states.

(2) Let there be a state in the state set K of DFA as [Si, Si+1, ..., Sj], if for a symbol a∈Σ, there is F ({ Si, Si+1, ..., Sj } ,

a)={Si', Si+1', ..., Sk'}

Then let F({Si, Si+1, ..., Sj}, a)={Si', Si+1', ..., Sk'} be a conversion function of DFA. If [Si', Si+1', ..., Sk'] is not in K, it will be added into K as a new state.

(3) Repeat step 2 until no new state is added in K.

(4) All states obtained above constitute the state set K of DFA, the transition function constitutes F of DFA, and the alphabet of DFA is still the alphabet Σ of NFA.

(5) Any state in DFA that contains the final state of NFA is the final state of DFA.

For the above-mentioned deterministic algorithm of NFA—the subset method, another description method with stronger operability can also be used, and we give its detailed description below. First, two related definitions are given.

Suppose I is a subset of NFA M state set K (ie I∈K), then define ε-closure(I) as:

(1) If Q∈I, then Q∈ ε-closure(I);

(2) If Q∈I, any state Q' that can be reached from Q through any ε-arc, then Q' ∈ ε-closure(I).

The state set ε-closure(I) is called the ε-closure of state I.

Suppose NFA M = (K , Σ , F , S , Z), if I∈K , a∈Σ , then define Ia = ε-closure (J), where J is all starting from ε-closure (I), after The set of states reached by an arc a.

The essence of NFA determinization is to use the subset of the original state set as a state on the DFA, and the original state

The transition between states is the transition between the subsets, thus determinizing the uncertain finite automata. After confirming

The number of states may increase, and some equivalent states may appear, which requires simplification.

【Experimental content】

move and closure is designed to realize the conversion from NFA to DFA, which is realized by python .

Among them, in the storage of n fa , since python is used , the related information of n fa and d fa is stored in .json format.

Main function module (Main)

```python
def main():
    nfa_input = "NFA_1.json"      # 例1 NFA路径
    dfa_output = "DFA_1.json"     # 例1 DFA路径
    # nfa_input = "NFA_2.json"    # 例2 NFA路径
    # nfa_input = "DFA_2.json"    # 例2 DFA路径
    (k_set, e_set, f, s_set, z_set) = read(nfa_input)  # 五元组-input
    nfa = json.load(open(nfa_input,"r"))
    showMachine(nfa, e_set, 'nfa')     #输出图像
    dfa = process_dfa(k_set, e_set, f, s_set, z_set)    # 创建-dfa
    write_dfa(dfa, dfa_output)                          # 输出-dfa
    showMachine(dfa, e_set, 'dfa')     #输出图像
```

Figure: Main function module (Main)

Ideas:

① Obtain the json address of n fa, dfa

② get n fa

③ generate d fa

④ Write d fa

⑤ Use Graphviz to generate n fa, dfa directed flow chart

Dfa generation module

① Subject

```python
def process_dfa(k_set, e_set, f, s_set, z_set):
    dfa = creat_dfa(e_set)              # 初始化dfa，含路径信息 list（a,b）
    dfa_set = []                        # 集合T0,T1,T2...存储
    memo = creat_memo(e_set)            # 缓存区-memo
    ep = ep_closure(f, memo["#"], s_set, '#')     # closure(起点)
    #Attention here. Has to be a list
    queue = deque([ep])                 # 创建deque队列[ep],即T0

    dfa_set.append([ep])                # [ep]进队列dfa_set,即T0存入
    dfa["k"].append("0")                # 元素集-添0(dfa起点，元素)
    dfa["s"].append("0")                # 开始集-添0(dfa起点)
    if not len(ep&z_set) == 0:
        dfa["z"].append("0")            # 特例:作为起点
    i = 0

    while queue:
        T = queue.popleft()           # 取Ti
        j = ""
        index = str(i)                # 新符i(从0开始)
        i = i + 1
        dfa["f"][index] = {}
        for s in e_set:
            t = ep_closure(f, memo["#"], move(f, memo[s], T, s), '#')  # 先move，后closure，结果集t
            try:
                j = str(dfa_set.index(t))
            except ValueError:        # 若参数无效
                queue.append(t)       # 结果集t-append-queue
                j = str(len(dfa_set))     # 取dfa_set长度并str，表示一个新元素(用数字表示)
                dfa_set.append(t)     # 结果集t-append-dfa_set
                dfa["k"].append(j)    # "k"-append-新元素
            dfa["f"][index][s] = j    # function-index-路径()
            if not len(t&s_set) == 0:
                dfa["s"].append(j)    # 特例:作为起点
            if not len(t&z_set) == 0:
                dfa["z"].append(j)    # 特例:作为终点

    return dfa
```

Figure: process _dfa function module, dfa generates the main body

Ideas:

moving in the closure first in the textbook . First obtain the initial state node, and then obtain the T0 data set, then start from the T0 data set, call the move and closure functions , and obtain T 1 , T2 , T3... according to the guidance of the path symbol , and finally carry out the obtained data set Process and generate the required dfa information .

②move function _

```python
def move(f, memo, c_set, arc):
    res = set()                # res: set集-拆分字符
    for s in c_set:            # s: 入口集
        if not s in memo:
            memo[s] = set()    # s为新集
            if s in f:
                if arc in f[s]:
                    memo[s] = set(f[s][arc])    # 路径结果
        res |= memo[s]         # s为旧集
    return res
```

Figure: move function

Ideas:

Use memo to store temporary data.

Read the departure data set, and generate the destination data set according to the path guidance.

③c loss function

```python
def ep_closure(f, memo, c_set, arc):
    res = set()                # res: set集-拆分字符
    for s in c_set:            # s: 入口集
        if not s in memo:
            memo[s] = set()    # s为新集
            memo[s] = set([s]) # closure(s)
            if s in f:
                if arc in f[s]:
                    memo[s] |= ep_closure(f, memo, set(f[s][arc]), arc)  # 添加新closure结果
        res |= memo[s]         # s为旧集
    return res
```

Figure: closure function

Ideas:

Similar to the move function, use memo to store temporary data.

Read the starting data set, and generate the ending data set according to the guidance of "#" (use # to refer to ε) .

Note that the difference with move is that closure needs to be traversed multiple times along the path of ε to obtain the result.

The automaton generation module (s howMachine):

```
from graphviz import Digraph
```

```
def showMachine(fa, e_set, name):
    # 初始化
    if name == 'dfa':
        dot = Digraph(name="DFA", comment="the test DFA", format="png")
        dot.attr(label=r'\nDFA', fontsize='20')
        print("创建 NFA 流程图")
    elif name == 'nfa':
        dot = Digraph(name="NFA", comment="the test NFA", format="png")
        dot.attr(label=r'\nNFA', fontsize='20')
        print("创建 DFA 流程图")

    # 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(0, num_k):
        dot.node(fa["k"][i])            # 0~k元素, 设置每个节点

    # 特殊部分 指向首节点的箭头(隐藏方法)+doublecircle
    for i in range(0, len(fa["z"])):
        dot.node(fa["z"][i], shape='doublecircle')
    dot.node("fake", style='invisible')
    for i in range(0, len(fa["s"])):
        dot.edge("fake", fa["k"][i], style='bold')

    # 创建路径
    for i in range(0, num_k+1):         # 0~k个元素, 设置每条路径
        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"][str(i)]:    # 判断该起点下有什么路径
                    num_f = len(fa["f"][str(i)][n])                 # 获取需创建路径数目
                    for j in range(0, num_f):                       # 有 j 种终点可能
                        node_end = fa["f"][str(i)][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()
    if name == 'dfa':
        dot.render('DFA', view=True)
    elif name == 'nfa':
        dot.render('NFA', view=True)
```

Figure: automaton generation module (s howMachine)

Ideas:

①First call from graphviz import Digraph

②Initialize d ot

③Create nodes (for all elements in ″k″)

④Special processing (generating an arrow pointing to the initial state node and a double circle shape final state node)

⑤Create a path (traverse the s et set, combine the starting point-symbol-end point of each path)

⑥Image display and storage

【Experimental Results】

example 1:

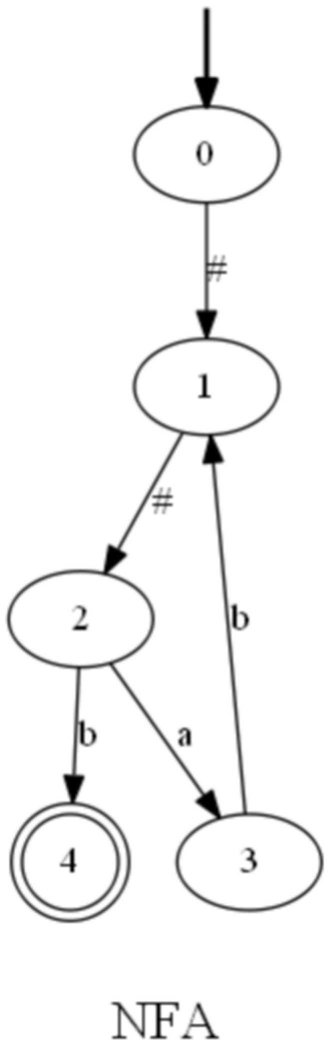

Figure: NFA generated by Graphviz (Example 1)

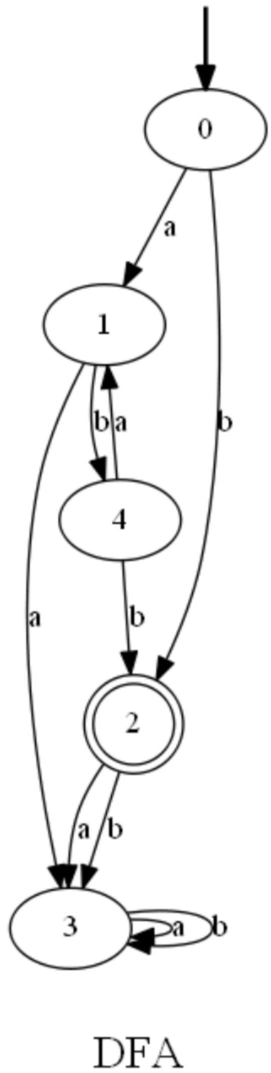

Figure: DFA generated by Graphviz (Example 1)

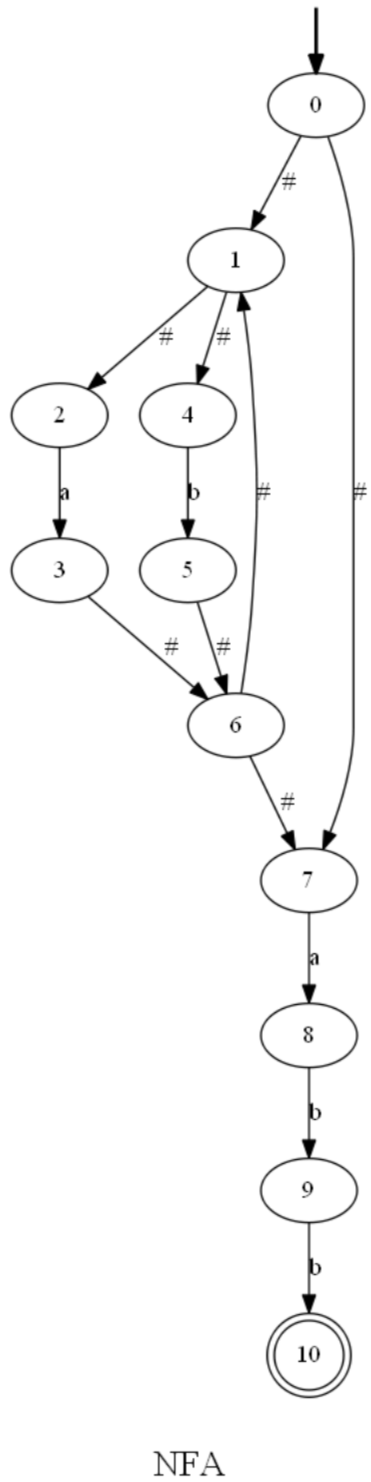	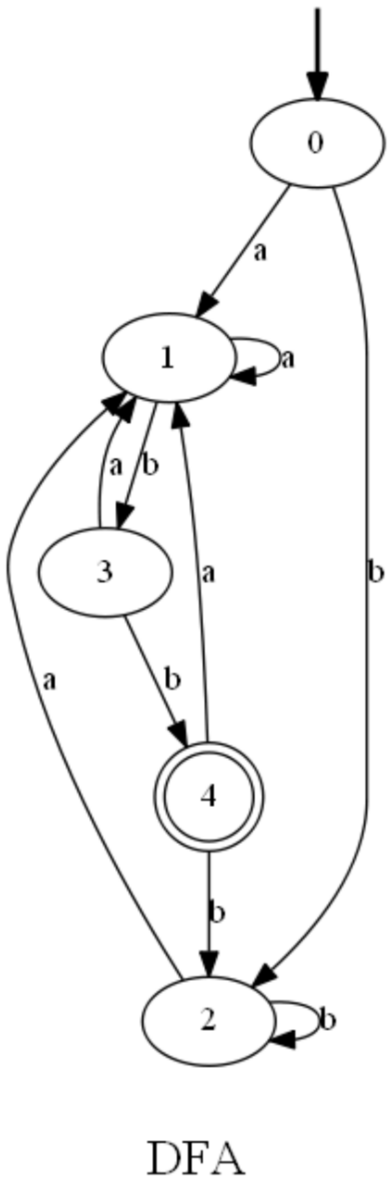
Figure: NFA generated by Graphviz (Example 2)	Figure: DFA generated by Graphviz (Example 2)

【Experiment Summary】

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

In realizing the generation of d fa , we first obtain the initial state node of n fa (that is, the entry), and based on this node, obtain the initial state node of d fa through the ε - closure() function, that is, the T 0 data set. Then based on the T 0 collection, continuously call the move () and ε - closure() functions to obtain the next T 1, T2, T3 and other data according to the symbol of the path arc (such as "a", "b") If there is a data set that is the same as the one that has been generated before, the set will be "abandoned" and will not be stored any further.

At the same time, in the work of generating dfa text , I also learned how to call Graphviz in python to automatically generate flowcharts in the form of code . First of all, on the official gve dit editor, I wrote the dot script according to the official document , and tried to run and analyze the generated codes of several directed and undirected flowcharts, and gained a preliminary understanding of Graphviz . However, the writing specification of the Graphviz library in python is different. For this reason, I further studied the syntax implementation of Graph viz in python , such as creating nodes through x xx.node( ), and then through x xx.edge( ) to create the path. I ended up writing a more robust function for automaton image generation.

Finally, through this experimental practice, I have improved my understanding of deterministic finite automata (DFA), uncertain finite automata (NFA) and their conversion methods, and better enhanced my knowledge. Code ability lays a solid foundation for the next experiment.

【Experiment code】

Compile_2.py

```
import json
from collections import deque
from graphviz import Digraph

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

def creat_memo(e_set):
memo = {}
for i in e_set: # Sub-path settings, such as path [a], [b]
memo[i] = {}
memo['#'] = {} # ep path setting
return memo

def move(f, memo, c_set, arc):
res = set() # res: set - split characters
for s in c_set: # s: entry set
if not s in memo:
memo[s] = set() # s is the new set
if s in f:
if arc in f[s]:
memo[s] = set(f[s][arc]) # path result
```

```
res |= memo[s] # s is the old set

return res


def ep_closure(f, memo, c_set, arc):

res = set() # res: set - split characters

for s in c_set: # s: entry set

if not s in memo:

memo[s] = set() # s is the new set

memo[s] = set([s]) # closure(s)

if s in f:

if arc in f[s]:

memo[s] |= ep_closure(f, memo, set(f[s][arc]), arc) # add new closure result

res |= memo[s] # s is the old set

return res


'''

def move(f, memo, s, arc):

  return closure(f, memo[arc], s, arc)


def ep_closure(f, memo, s):

  return closure(f, memo["#"], s, '#')


def closure(f, memo, c_set, arc):

  res = set()              # res: set 集-拆分字符

  for s in c_set:          # s: 入口集

    if not s in memo:

      memo[s] = set()      # s 为新集

      if arc == '#':

        #Attention here. Has to be a list
```

```
        memo[s] = set([s])  # closure(s)
      if s in f:
        if arc in f[s]:
          if arc == '#':
            memo[s] |= closure(f, memo, set(f[s][arc]), arc)  # 添加新 closure
结果
          else:
memo[s] = set(f[s][arc]) # path result
res |= memo[s] # s is the old set
return res
'''


def creat_dfa(e_set):
dfa = {}
dfa["k"] = []
dfa["e"] = list(e_set)
dfa["f"] = {}
dfa["s"] = []
dfa["z"] = []
return dfa


def process_dfa(k_set, e_set, f, s_set, z_set):
dfa = creat_dfa(e_set) # Initialize dfa, including path information list (a,b)
dfa_set = [] # set T0, T1, T2... storage
memo = creat_memo(e_set) # cache area-memo
ep = ep_closure(f, memo["#"], s_set, '#') # closure(starting point)
#Attention here. Has to be a list
queue = deque([ep]) # Create deque queue [ep], ie T0
```

```
dfa_set.append([ep]) # [ep] into the queue dfa_set, that is, T0 is stored
dfa["k"].append("0") # element set - add 0 (dfa starting point, element)
dfa["s"].append("0") # Start set - add 0 (starting point of dfa)
if not len(ep&z_set) == 0:
dfa["z"].append("0") # special case: as a starting point
i = 0

while queue:
T = queue.popleft() # take Ti
j = ""
index = str(i) # new symbol i (starting from 0)
i = i + 1
dfa["f"][index] = {}
for s in e_set:
t = ep_closure(f, memo["#"], move(f, memo[s], T, s), '#') # move first, then close,
result set t
try:
j = str(dfa_set. index(t))
except ValueError: # If the parameter is invalid
queue.append(t) # result set t-append-queue
j = str(len(dfa_set)) # Take the length of dfa_set and str to represent a new
element (expressed in numbers)
dfa_set.append(t) # result set t-append-dfa_set
dfa["k"].append(j) # "k"-append-new element
dfa["f"][index][s] = j # function-index-path()
if not len(t&s_set) == 0:
dfa["s"].append(j) # special case: as a starting point
if not len(t&z_set) == 0:
dfa["z"].append(j) # special case: as the end point
```

```
return dfa



def write_dfa(dfa, f):
f = open(f, "w")
f.write(json.dumps(dfa, indent=1)) # Write files based on json
f. close()

def showMachine(fa, e_set, name):
# initialization
if name == 'dfa':
dot = Digraph(name="DFA", comment="the test DFA", format="png")
dot.attr(label=r'\nDFA', fontsize='20')
print("Create NFA flowchart")
elif name == 'nfa':
dot = Digraph(name="NFA", comment="the test NFA", format="png")
dot.attr(label=r'\nNFA', fontsize='20')
print("Create DFA flowchart")

# 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(0, num_k):
dot.node(fa["k"][i]) # 0~k elements, set each node

# The special part points to the arrow of the first node (hidden method) +
doublecircle
for i in range(0, len(fa["z"])):
```

```
dot.node(fa["z"][i], shape='doublecircle')
dot.node("fake", style='invisible')
for i in range(0, len(fa["s"])):
dot.edge("fake", fa["k"][i], style='bold')


# create path
for i in range(0, num_k+1): # 0~k elements, set each path
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"][str(i)]: # Determine what path exists under the starting point
num_f = len(fa["f"][str(i)][n]) # Get the number of paths to be created
for j in range(0, num_f): # There are j possible end points
node_end = fa["f"][str(i)][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()
if name == 'dfa':
dot. render('DFA', view=True)
elif name == 'nfa':
dot. render('NFA', view=True)
else:
print("input wrong")


def main():
nfa_input = "NFA_1.json" # Example 1 NFA path
```

```
dfa_output = "DFA_1.json" # Example 1 DFA path

# nfa_input = "NFA_2.json" # Example 2 NFA path

# nfa_input = "DFA_2.json" # Example 2 DFA path

(k_set, e_set, f, s_set, z_set) = read(nfa_input) # five-tuple-input

nfa = json.load(open(nfa_input,"r"))

showMachine(nfa, e_set, 'nfa') #output image

dfa = process_dfa(k_set, e_set, f, s_set, z_set) # create-dfa

write_dfa(dfa, dfa_output) # output-dfa

showMachine(dfa, e_set, 'dfa') #output image


if __name__ == '__main__':

main()
```