ID 914855947
student: Hsuan-Chih, Chen

CS311 programming assignment 1

DFA Analyzer Writeup

I choose python to do this DFA project. Firstly, I saw DFA can be written and displayed by a form when you was teaching how to covert a NFA to a DFA.
A example is shown below:

State	a	b
q0	q0	q1
q1	q1	q2
q2	q2	q2

I realize that I have to create a program can memorize this form for the DFA analyzer after I read this assignment. It's like a matrix. However, a big problem appeared. I need to input several DFAs so the number of the columns and rows will be different. I may have to create a dynamic form by dynamic arrays(or dynamic vectors) for dealing with different cases. After I searched for information on websites, I found the data type dictionary of python can easily solve this problem, and it is one of main reasons I choose python to do this assignment.
The above form can be transferred and stored by the dictionaries shown below :
{q0:{a: q0, b: q1}}, q1: {a:q1, b:q2}, q2:{a:q2,b:q2}}
The individual state(q0,q1..) of dictionary(source) has another dictionary in the value part(destination). The destination is another dictionary for mapping exact destination state by a certain character. (destination = dic[source state][charater])
Finally, I realize I just need a dictionary variable to process the DFA in my program no matter how many states and number of alphabet elements are read from a file. I don't need to worry about setting dynamic memory in my program anymore. Dictionary data type is very useful for keeping mapping data. We can use it kind of plate by plate(dictionary of dictionary), and finally we can map the certain data we want to connect. In my test file, it is only a dictionary data, but it actually contain many dictionaries in a dictionary variable. It's already like a data structure.

This assignment another part annoyed me is reading the all information of a DFA from a file. Initially, I didn't know how to read the file because it's with different long of dictionaries and lists. After discussing with some classmates, I knew Json is a very powerful tool. When I was writing the code for loading a file of Json, I felt it's really easy and simple. I don't even need to code a loop. I just need to set keywords to my certain data in my test file. For example, "Accepting state" : ["q1","q3]. It will directly find the list of "Accepting state" and load it in.

After I solved the problems about storing data and reading data, I start to write the code for processing a string. I set a one condition in my code to recognize whether each character of a string is in the certain alphabet list or not. If my program read a character not a element in the alphabet list, it will directly skip the rest characters and return a False message to my main function. By this way, I can reject the invalid string as soon as they are encountered. Furthermore, one important thing I concern is there are a significant number of corner cases such as empty string. I thought empty string is not in the alphabet list so the program is supposed to show up "invalid input". I added one more condition for checking empty string case. However, finally I found the empty string can automatically pass the alphabet condition. I don't have to set the special case for empty string. I think maybe python already considerate the empty string case in its machine process.

The program basically generates the DFA transitions form I mentioned so it should already contain all situations in the DFA. When I tested my program, I chose some normal conditions and make sure the outputs are all right at first. After that, I input empty string. If it's a empty string, the machine will stop at the start state. If start state is a accepting state, it will be accepted. If start state is not a accepting state , it will be rejected. After that, I input some characters not in the alphabet list and make sure they all reject as soon as they are encounter.

Testing result :

DFA machine 1:   This DFA accepts binary multiples of 3.

```
Input string:    0000
input( 0 ) q0    ->    q0
input( 0 ) q0    ->    q0
input( 0 ) q0    ->    q0
input( 0 ) q0    ->    q0

Final state:    q0

Accepted state(s):
q1
```

ID 914855947
student: Hsuan-Chih, Chen

```
  0000    is rejected.

Input string:    0011
input( 0 ) q0    ->    q0
input( 0 ) q0    ->    q0
input( 1 ) q0    ->    q2
input( 1 ) q2    ->    q1

Final state:    q1

Accepted state(s):
q1

  0011    is accepted.

Input string:    0110
input( 0 ) q0    ->    q0
input( 1 ) q0    ->    q2
input( 1 ) q2    ->    q1
input( 0 ) q1    ->    q1

Final state:    q1

Accepted state(s):
q1

  0110    is accepted.

Input string:    010
input( 0 ) q0    ->    q0
input( 1 ) q0    ->    q2
input( 0 ) q2    ->    q3

Final state:    q3

Accepted state(s):
q1
```

ID 914855947
student: Hsuan-Chih, Chen

```
  010    is rejected.

Input string:    1100011
input( 1 ) q0    ->    q2
input( 1 ) q2    ->    q1
input( 0 ) q1    ->    q1
input( 0 ) q1    ->    q1
input( 0 ) q1    ->    q1
input( 1 ) q1    ->    q2
input( 1 ) q2    ->    q1

Final state:    q1

Accepted state(s):
q1

  1100011    is accepted.

Input string:    113avb
input( 1 ) q0    ->    q2
input( 1 ) q2    ->    q1

invaild input!

Input string:

Final state:    q0

Accepted state(s):
q1

      is rejected.

User Input: 1100

Input string:    1100
input( 1 ) q0    ->    q2
```

ID 914855947
student: Hsuan-Chih, Chen

```
input( 1 ) q2    ->    q1
input( 0 ) q1    ->    q1
input( 0 ) q1    ->    q1

Final state:    q1

Accepted state(s):
q1

  1100    is accepted.
>>>
```

DFA machine 2:   This DFA accepts strings that contain the substring 'abb'.

```
Input string:    aaaaab
input( a ) q0    ->    q1
input( a ) q1    ->    q1
input( a ) q1    ->    q1
input( a ) q1    ->    q1
input( a ) q1    ->    q1
input( b ) q1    ->    q2

Final state:    q2

Accepted state(s):
q3

  aaaaab    is rejected.

Input string:    bbbaabbb
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( a ) q0    ->    q1
input( a ) q1    ->    q1
input( b ) q1    ->    q2
input( b ) q2    ->    q3
input( b ) q3    ->    q3
```

ID 914855947
student: Hsuan-Chih, Chen

```
Final state:    q3

Accepted state(s):
q3

  bbbaabbb    is accepted.

Input string:    bbbbaaaa
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( a ) q0    ->    q1
input( a ) q1    ->    q1
input( a ) q1    ->    q1
input( a ) q1    ->    q1

Final state:    q1

Accepted state(s):
q3

  bbbbaaaa    is rejected.

Input string:

Final state:    q0

Accepted state(s):
q3

      is rejected.

Input string:    aacc
input( a ) q0    ->    q1
input( a ) q1    ->    q1
```

ID 914855947
student: Hsuan-Chih, Chen

```
invaild input!

Input string:    %$#

invaild input!

Input string:

invaild input!

User Input: bbbbba

Input string:    bbbbba
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( b ) q0    ->    q0
input( a ) q0    ->    q1

Final state:    q1

Accepted state(s):
q3

  bbbbba    is rejected.
>>>
```