به نام مهربان ترین

تبدیم ماشین غیر قطعی NFA_λ به ماشین قطعی DFA :

نکات ضروری راجب کد:

- در اخر فایل ورودی NFA باید enter بشد .
- λ در فایل ورودی باید به صورت l مشخص شود.

الگریتم تبدیل:

1. ساخت کلاس NFA و دریافت اطلاعات ماشین.
2. ساخت NFA.
3. ساخت مجموعه حالات های ماشین قطعی که همان مجموعه توانی حالان NFA میباشد.
4. ساخت جدولی از حالات DFA و الفبا
5. بدست اوردن تابع های انتقال با استفاده از جدول
6. حذف لاندا
   - پیدا کردن توابع انتقال حاوی لاندا (مثلن x λ y
   - اضافه کردن توابع انتقالی که از جمع دو تابع انتقال (تابعی که حالت پایانی ان x است و x λ y)
   - اضافه کردن توابع انتقالی که از جمع دو تابع انتقال (x λ y وتابعی که حالت اغازین ان y است)
7. بدست اوردن حالات پایانا:حالاتی که با لاندا به حالتی پایانی میرود حالاتی که در مجموعه ی خود حالتی از حالات پایانی NFA دارد.
8. حذف λ از الفبا
9. پایان

```
class NFA:
    def __init__(self, states, states_number,alphabet, transition_function
   , t_function, start_state, accept_states):
        """[initial NFA machine]

        Arguments:
            states {[dict]} -- [all states of machine]
            alphabet {[dict]} -- [all alphabet of machine]
            transition_function {[list]} -- [edge of machine graph]
            start_state {[string]} -- [start state]
            accept_states {[dict]} -- [all accept states of machine]
            states_number {[int]} -- [number of states]
            t_function {[list]} -
   - [transition functin store in another type]
        """
        self.states = states
        self.states_number=states_number
        self.alphabet = alphabet
        self.transition_function = transition_function
        self.t_function = t_function
        print(self.t_function)
        self.start_state = start_state
        self.accept_states = accept_states
        self.new_accept_states=accept_states
        self.current_state = start_state
        self.remove_lambda()
```

کلاس NFA

نکات:

باید حالات نهایی و توابع انتقال به صورت ساختان داده dictionary در زبان python تعریف شوند.در مرحله 7 الگریتم برای بررسی نهایی بودن حالت میتوان از دستون in accrpt_states استفاده کرد و برای مرحله 5 نیز میتوان با سرعت تابع انتقال مربوط به ان حالت و حرف را پیدا کرد و فهمید به چه حالتی میرود.

```
"""reading NFA from file"""
filepath = 'NFA_Input_2.txt'
with open(filepath,"r") as fp:
    l1 = fp.readline()
    ll1=l1[:len(l1)-1].split(" ")
    alphabet=list()
    for s in ll1:
        alphabet.append(s)
    alphabet.append('l')

    l2 = fp.readline()
    ll2=l2[:len(l2)-1].split(" ")
    states=dict()
    cn=0
    for a in ll2:
        states.update({a:cn})
        cn+=1

    l4 = fp.readline()
    start_state=states.get(l4[:len(l4)-1])

    l3 = fp.readline()
    ll3=l3[:len(l3)-1].split(" ")
    accept_states=dict()
    for a in ll3[:len(ll3)]:
        accept_states.update({states.get(a):states.get(a)})
    t_function=list()
    tf = dict()
    for  i in range (len(states)):
        for j in alphabet:
            tf.update({(i,j):list()})

    line = fp.readline()
    while line:
        l=line.split(' ')
        tf[(states.get(l[0]),l[1])].append (states.get(l[2][:len(l[2])-1]))
        t_function.append((states.get(l[0]),l[1],states.get(l[2][:len(l[2])-1])))
        line = fp.readline()
```

در اخر فایل ورودی NFA باید enter بشد .

حالات را به عدد تبدیل میکنیم.

در فایل ورودی باید به صورت l مشخص شود.

```
"""counstruct NFA object"""
n = NFA(states,len(states),alphabet, tf, t_function, start_state, accept_states)
```

تعداد حالاط برابر طول حالات است.

```
"""initial power set of NFA states as new DFA states"""
all_states=list()
stateList=range(n.states_number)
for i in range (2**(n.states_number)):
    cs=list()
    binaryn=decimalToBinary(i,n.states_number)
    for j in range(n.states_number):
        if binaryn[j] =='1':
            cs.append(stateList[j])
    all_states.append(cs)
```

مجموعه توانی حالات ماشین را به صورت یک لیست ذخیره میکنیم.(از روش bitmask برای یافتن مجموعه توانی استفاده میکنیم.)

```
"""make sigma table this table shows that in each DFA state with each alphabet
machine goes to which NFA states actually DFA state """
sigma_table=list()
for state_set in all_states:
    row=list()
    for a in n.alphabet:
        distance_set=list()
        for s in state_set:
            distance_set+= (n.transition_function.get((s,a)))
        row.append(distance_set)
    sigma_table.append(row)
```

	a	b
{Q1}	Q2	Q1
{Q2}	Q2,Q1	{}
{}	{}	{}
{Q1,Q2}	{}	Q1

جدول حاصل شبیه جدول بالا میشود.که Index ها هریک نشاندهنده ی یک حالت اند.

```
"""reading DFA transition functions from sigma table"""
DFA_transition_function=list()
scount=0
for i in sigma_table:
    alpha_counter=0
    for j in i:
        ds=['0']*n.states_number

        for k in j:


            ds[k]='1'

        if (n.alphabet[alpha_counter])!='l':
            DFA_transition_function.append(str(binaryToDecimal(decimalToBinary(sc
ount,n.states_number)))+' '+str(n.alphabet[alpha_counter])+' '+str(binaryToDecima
l(ds)))
        alpha_counter+=1
    scount+=1
```

برای حالات DFA یک لیست میسازیم و توابع انتقال را از جدول خوانده و در ان ذخیره میکنیم به عنوان مثال در جدول بالا توابع انتقال عبارت اند از

{Q1} a {Q2}

{Q1} b {Q1}

{Q2} a {Q1.Q2}

{Q2} b {}

{} a {}

{} b {}

{Q1,Q2} a {}

{Q1,Q2} b {Q1}

```
def remove_lambda(self):
        """[removin lambda from NFA machine]
        """
'''findin transition with lambda'''
        transitions_contains_lambda = list()
        for transition in self.transition_function:
            if transition[1]=='l':
                transitions_contains_lambda.append([transition[0],transition[1],s
elf.transition_function[transition]])
                for fnstate in  self.transition_function[transition]:
                    if fnstate in accept_states:
                         self.new_accept_states.update({transition[0]:transition[
0]})
        print(transitions_contains_lambda)
# x lambda y
        '''adding transitions that gose to a state that its lambda closer is not
null'''
        # for each k that k==x
        for transition in transitions_contains_lambda:
            for et in self.transition_function:
                for destination_of_tf in self.transition_function[et]:
                    if destination_of_tf == transition[0]:
                        self.transition_function[(et[0],et[1])]+=(transition[2])

        '''adding transitions that start with vertex that its lambda closer is no
t null'''
        #for each k that k==y
        for transition in transitions_contains_lambda:
            for et in self.transition_function:
                for distance_of_transition in self.transition_function[(transitio
n[0],transition[1])]:
                    if distance_of_transition == et[0]:
                        self.transition_function[(transition[0],et[1])]+=(self.tr
ansition_function[et])
```

ابتدا Tf ها یی را که لاندا دارند را پیدا میکند و در یک لیست ذخیره میکند .

سپس این لیست حرکت میکند. به ازای هر تاپل بررسی میکند اگر انتهای tf ای ابتدا ی این tf بود یالی جدید که جمع این دو است را اضافه میکند .

و اگر ابتدای یالی انتهای این tf بود هم یالی جدید که جمع این دو است را اضافه میکند .

به عنوان مثال :

X λ Y

Z a X

z b X

---

add Z b X

add Z a Y

```
"""finding DFA states that are a accept state"""
DFA_accept_states=list()
for i in all_states:
    for j in i:
        if j in n.new_accept_states :
            DFA_accept_states.append(i)

            break
```

حالاتی که با لاندا به یک حالت پایانی میروند خود حالت پایانی اند

تمام حالات مجموعه توانی که حد اقا یک حالت پایانی داشته باشند حالت پایانی اند

```
"""finding DFA alphabet actually it's same as NFA except it don't has lambda"""
DFA_alphabet=list()
for alpha in n.alphabet:
    if alpha!='l':
        DFA_alphabet.append(alpha)
```

حذف لاندا.

```
with open('DFA',"w") as fp:


    for i in all_states:
        ds=['0']*n.states_number
        for k in i:
            ds[k]='1'
        fp.write(str(binaryToDecimal(ds))+' ')
    fp.write('\n')

    for i in dalphabet:
        fp.write(str(i)+' ')
    fp.write('\n')
    fp.write(str(dstart_state)+'\n')

    for i in daccept_states:
        ds=['0']*n.states_number
        for k in i:
            ds[k]='1'
        fp.write(str(binaryToDecimal(ds))+' ')
    fp.write('\n')

    for i in dtransition_function:
        fp.write(str(i)+'\n')
```

ذخیره سازی در فایل