دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش پروژه اول درس نظریه زبان‌ها و ماشین‌ها

نگارش:

امیرحسین سرور - ۹۷۳۱۰۲۸

استاد درس:

دکتر محمدرضا میبدی

اردیبهشت ۱۳۹۹

- **سوال اول :**

```
# Reading The String
str = input()

# Opening dfa file and reading it
f = open("DFA_Input_1.txt", "r")
characters = f.readline().split()
states = f.readline().split()
state = f.readline().split()[0]
final_states = f.readline().split()
V = f.read().splitlines()
f.close()
```

در ابتدا رشته از کاربر گرفته می‌شود. سپس فایل DFA را باز کرده و اطلاعات مربوط به گراف DFA در برنامه ذخیره می‌شود.

```
# moving step by step with the dfa graph and characters of the input string
for char in str:
    if char in characters:
        q = state + " " + char
        for d in V:
            if q in d:
                state = d.split()[2]
                break
    else:
        print("not accepted")
        exit(0)

# Checking is the state we got is compatible with the final state or not
if state in final_states:
    print("accepted")
else:
    print("not accepted")
```

در مرحله بعد با اعمال for روی رشته وارد شده، کاراکترهای آن را به ترتیب می‌خوانیم و با توجه استیت فعلی و اطلاعاتی که از DFA داریم، یک رشته از استیت فعلی و کاراکتر خوانده شده تشکیل می‌دهیم و اگر این رشته جزو یکی از یال‌های DFA باشد، استیت فعلی را به قسمت سوم آن در فایل txt (یال) که در واقع استیت بعدی می‌باشد، تغییر می‌دهیم. همچنین چون در این سوال DFA داریم مطمئن هستیم یک یال با همچین مشخصاتی وجود دارد. این مراحل را تا زمانی که آخرین کاراکتر رشته موردنظر خوانده شود ادامه می‌دهیم و در آخر، اگر استیت فعلی جزو یکی از استیت های نهایی بود، عبارت accepted و در غیر اینصورت عبارت not accepted در خروجی چاپ می‌شود.

• سوال دوم :

```
def main():
    # opening nfa file
    f = codecs.open("NFA_Input_2.txt", "r", "utf-8-sig")
    # making nfa file into dfa file
    dfa(f)
    f.close()
```

بدنه اصلی که شامل تابع main است تنها از صدا زدن تابع dfa تشکیل شده است که وظیفه‌ی پیاده‌سازی الگوریتم تبدیل NFA بدون λ به DFA را دارد.

```

    fout = codecs.open("DFA_Output_2.txt", "w", "utf-8-sig")
    for char in characters:
        fout.write(char + " ")
    fout.write("\n")

    E = [{first_state}]
    Vout = []
    for state in E:
        for char in characters:
            next_state = []
            for q in state:
                next_state += next_states_without_l(q, char, Vin, characters)

            next_state = set(next_state)
            if next_state.__len__() == 0:
                next_state.add("∅")

            if next_state not in E:
                E.append(next_state)
            Vout.append(vtostring(state, next_state, char))

```

در تابع dfa ، بعد ازگرفتن و خواندن فایل NFA ، در فایل خروجی DFA ، ابتدا کاراکترهای استفاده شده نوشته می‌شوند. سپس آرایه از set ها تحت عنوان E برای نگه‌داری رئوس گراف تعریف می‌شود که اولین عنصر آن هم مشخص است. سپس روی این آرایه for زده تا رئوس بعدی و یال‌های مربوط نیز مشخص شوند. یال‌ها هم که در نهایت باید در فایل خروجی نوشته شوند، در آرایه‌ای تحت عنوان Vout نگه‌داری می‌شوند. در این for ، حالت بعدی هر استیت در DFA توسط اجتماع گیری از استیت‌های بعدی در گراف NFA بدون λ حالات موجود در استیت فعلی مشخص می‌شود. اگر حالت بعدی در رئوس فعلی موجود نبود، یک رأس جدید اضافه شده و اگر تهی باشد، به استیت ∅ هدایت می‌شود. در آخر یال جدید نیز به Vout اضافه می‌شود. قابل ذکر

است منظور از NFA بدون λ ، تبدیل‌یافته NFA با λ به NFA بدون λ توسط توابعی است که در زیر توضیح خواهیم داد. در واقع ما در ابتدا NFA با λ را به NFA بدون λ تبدیل میکنیم.

```
# this function returns δ'(state , char) with the formula we already knew
def next_states_without_l(state, char, V, characters):
    l_cl = l_closure(state, V, characters)
    qi = []
    for q in l_cl:
        qi += next_states_with_l(q, char, V, characters)
    qi = list(set(qi))
    qj = []
    for q in qi:
        qj += l_closure(q, V, characters)
    return qj
```

در تابع next_states_without_l ، در واقع تابع δ' برای بدست آوردن NFA بدون λ از طریق فرمول زیر پیاده‌سازی شده است:

$$\delta'(q_j, a) = \bigcup \lambda\_closure(\delta(q_i, a)) \quad , \forall q_i \in \lambda\_closure(q_j)$$

در اینجا به دو تابع next_states_with_l و l_closure احتیاج داریم که زیر مشاهده می‌شود:

```

# this function returns next state due to nfa txt file which may have 'λ'and the given state & input
def next_states_with_l(state, char, V, characters):
    states = []
    if char in characters + ["λ"]:
        for d in V:
            q = state + " " + char
            if q in d:
                states.append(d.split()[2])
    return states


# this function returns a list which is λ_closure of a state
def l_closure(state, V, characters):
    states = []
    states.append(state)
    for s in states:
        states += next_states_with_l(s, "λ", V, characters)
    return states

```

در تابع اول، مشخصاً استیت بعدی با توجه به کاراکتر ورودی و استیت فعلی، از روی NFA داده شده در فایل که ممکن است شامل λ هم باشد، مشخص می‌شود. در تابع دوم، در واقع λ_closure یک حالت با محاسبه حالت‌هایی که تنها با λ می‌توان به آن‌ها رسید با استفاده از تابع اول و توسط یک آرایه برگردانده می‌شود.

```
    for state in E:
        fout.write((etostring(state) + " "))
    fout.write("\n")

    fout.write(first_state + "\n")

    for x in Vin:
        if "λ" in x and first_state not in final_states:
            fout.write(first_state + " ")
            break
    for state in E:
        for final_state in final_states:
            if final_state in state:
                fout.write(etostring(state) + " ")
                break
    fout.write("\n")

    for v in Vout:
        fout.write(v + "\n")

    fout.close()
```

در آخر نیز استیت های بدست آمده، حالت‌های نهایی و یال‌ها در فایل DFA خروجی نوشته می‌شود. حالت‌های نهایی DFA هم با چک کردن وجود حداقل یکی از حالات نهایی NFA در تمام حالات DFA بدست می‌آید و اگر NFA اولیه λ را هم بپذیرد، حالت اولیه نیز به عنوان یکی از حالات نهایی پذیرفته می‌شود.

پایان