علیرضا نعمتی ۹۹۳۱۰۸۱
فاز اول:

یک کلاس به نام `DFA` را پیاده‌سازی می‌کند که یک خودکار متناسب با ماشین پذیرش (Deterministic Finite Automaton) را نمایش می‌دهد. DFA شامل وضعیت‌ها، الفبا، تابع ترانزیشن، وضعیت شروع و مجموعه‌ای از وضعیت‌های قابل قبول است.

ویژگی‌های کلاس `DFA` شامل:
- `states`: یک مجموعه از وضعیت‌ها
- `alphabet`: یک مجموعه از الفبا
- `transitions`: یک دیکشنری که تابع ترانزیشن را نگهداری می‌کند (مشخص می‌کند که از هر وضعیت با ورودی خاص به کدام وضعیت باید رفت)
- `start_state`: وضعیت شروع
- `accept_states`: یک مجموعه از وضعیت‌های قابل قبول
- `current_state`: وضعیت فعلی

سازنده `DFA` دریافت مجموعه وضعیت‌ها، مجموعه الفبا، تابع ترانزیشن، وضعیت شروع و مجموعه وضعیت‌های قابل قبول را از ورودی می‌گیرد و آن‌ها را در متغیرهای متناظر ذخیره می‌کند. همچنین وضعیت فعلی را برابر با وضعیت شروع قرار می‌دهد.

تابع `run` یک رشته ورودی را به عنوان ورودی دریافت می‌کند و مشخص می‌کند که آیا این رشته توسط DFA قابل قبول است یا خیر. برای این منظور، با استفاده از تابع ترانزیشن و وضعیت فعلی، به طور ترتیب هر عنصر از رشته را بررسی می‌کند و با توجه به تابع ترانزیشن به وضعیت بعدی منتقل می‌شود. در نهایت، مشخص می‌کند که وضعیت فعلی در مجموعه وضعیت‌های قابل قبول قرار دارد یا خیر.

تابع `read_dfa_from_file` از یک فایل مشخص مشخصات یک DFA را خوانده و یک نمونه از کلاس `DFA` را برمی‌گرداند. فرمت فایل شامل اطلاعات الفبا، وضعیت‌ها، وضعیت شروع، وضعیت‌های قابل قبول و توابع ترانزیشن است.

در تابع `main`، یک نمونه از کلاس `DFA` از یک فایل ورودی ایجاد می‌شود و سپس رشته‌ای از کاربر گرفته می‌شود و با استفاده از تابع `run` بررسی می‌شود که آیا رشته ورودی توسط DFA قابل قبول است یا خیر. سپس نتیجه به کاربر نمایش داده می‌شود.

توابع `getAlphabet`، `getStates` و `getTransitions` نیز برای دسترسی به ویژگی‌های کلاس `DFA` ارائه می‌شوند.

فاز دوم:

یک کلاس به نام "DFA" و تعدادی تابع دیگر را پیاده‌سازی می‌کند که مربوط به تبدیل یک NFA (Automata نهایی غیرقطعی) به یک DFA (Automata متناهی قطعی) است.

کلاس `DFA` مدلی از یک Automata متناهی قطعی را نمایش می‌دهد و ویژگی‌های زیر را دارد:

- `states`: مجموعه وضعیت‌ها (states) در Automata
- `alphabets`: مجموعه حروف الفبا
- `transitions`: دیکشنری تابع انتقال‌ها که جفت مرتبی از وضعیت فعلی و حرف الفبا را به وضعیت بعدی نسبت می‌دهد
- `start`: وضعیت شروع (start state)
- `accept_states`: مجموعه وضعیت‌های قبول (accept states)
- `current_state`: وضعیت فعلی

همچنین کد، توابع دیگری را نیز پیاده‌سازی می‌کند:

- `lambda_compute(states_set, transactions_dict)`: این تابع براساس یک وضعیت و یک دیکشنری از تابع‌های انتقال، بسته‌ی lambda را محاسبه می‌کند. بسته‌ی lambda شامل تمام وضعیت‌های قابل دسترس (با استفاده از حروف الفبا λ) از وضعیت مبدأ است.
- `read_input(file_path)`: این تابع ورودی NFA را از یک فایل می‌خواند و آن را به شکل مناسب برمی‌گرداند. فایل شامل اطلاعات مانند حروف الفبا، مجموعه وضعیت‌ها، وضعیت شروع، وضعیت‌های قبول و توابع انتقال است.

- `nfa_to_dfa(file_path)`: این تابع NFA را به صورت Automata متناهی قطعی (DFA) تبدیل می‌کند. برای این کار از الگوریتم Subset Construction استفاده می‌کند.
- `write_output(file_path, dfa_object)`: این تابع DFA حاصل را به یک فایل خروجی می‌نویسد.

در نهایت، تابع `main` فراخوانی می‌شود که NFA را از یک فایل ورودی می‌خواند، آن را به یک DFA تبدیل می‌کند، و سپس DFA حاصل را در یک فایل خروجی ذخیره می‌کند.

بنابراین، این کد با استفاده از توابع و کلاس‌های تعریف شده، یک NFA را به یک DFA تبدیل می‌کند و خروجی را در یک فایل ذخیره می‌کند.

فاز سوم:
یک تبدیل‌کننده از عبارات منظم به NFA است. برای این کار، کدهایی برای تعریف کلاس `ObjectNFA` و توابع مختلفی نوشته شده است.

کلاس `ObjectNFA` شامل ویژگی‌هایی مانند وضعیت‌ها، الفبا، ترانزیشن‌ها، وضعیت شروع و وضعیت‌های پایانی است. این کلاس برای نمایش NFA استفاده می‌شود.

تابع `convert_unit_regex_to_nfa` یک عبارت منظم با طول یک واحد را به یک NFA تبدیل می‌کند. این تابع وضعیت‌ها، الفبا، ترانزیشن‌ها، وضعیت شروع و وضعیت پایانی را تعریف می‌کند و سپس یک نمونه از کلاس `ObjectNFA` را برمی‌گرداند.

تابع `concatenate_nfa` دو NFA را به هم می‌چسباند و یک NFA جدید را برمی‌گرداند. این تابع ابتدا الفبای جدید را با ترکیب الفباهای دو NFA تعریف می‌کند. سپس وضعیت‌ها، وضعیت شروع، وضعیت‌های پایانی و ترانزیشن‌های هر دو NFA را به NFA جدید اضافه می‌کند.

تابع `combine_nfa` دو NFA را با هم ترکیب می‌کند و یک NFA جدید را برمی‌گرداند. این تابع ابتدا الفبای جدید را با ترکیب الفباهای دو NFA تعریف می‌کند. سپس وضعیت‌ها، وضعیت شروع، وضعیت‌های پایانی و ترانزیشن‌های هر دو NFA را به NFA جدید اضافه می‌کند.

تابع `closure_nfa` یک NFA را با استفاده از عملگر بستار بسته می‌کند و یک NFA جدید را برمی‌گرداند. تابع closure_nfa وضعیت‌ها، الفبا، ترانزیشن‌ها، وضعیت شروع و وضعیت پایانی را تعریف می‌کند. سپس وضعیت‌های پایانی قبلی را به عنوان وضعیت شروع جدید اضافه می‌کند و ترانزیشن‌های جدید را به نحوی تعریف می‌کند که بتوانیم به وضعیت شروع قبلی با هر عنصر از الفبا برگردیم. در نهایت، وضعیت جدیدی را به عنوان وضعیت پایانی اضافه می‌کند. این نمونه جدید از کلاس ObjectNFA را برمی‌گرداند.

در کد پیاده‌سازی شما، تابع regex_to_nfa تعدادی پیمانه به عنوان ورودی می‌گیرد که هر یک به ترتیب به یکی از توابع convert_unit_regex_to_nfa، concatenate_nfa، combine_nfa یا closure_nfa منتقل می‌شود. این توابع به ترتیب وظایف تبدیل یک عبارت منظم با طول یک واحد به NFA، اتصال دو NFA، ترکیب دو NFA و بستار یک NFA را انجام می‌دهند.

به عنوان مثال، تابع regex_to_nfa('a') یک NFA را با تنها یک وضعیت، الفبای ['a']، وضعیت شروع و پایانی به ترتیب 0 و ترانزیشن {'0': {'0' :'a'}}} ایجاد می‌کند.