**مقدمه**

این برنامه برای تبدیل ماشین متناهی نامعین (NFA) به ماشین متناهی معین (DFA) طراحی شده است . ساختار کلی این برنامه به این صورت است که ابتدا ماشین NFA از کاربر گرفته شده ، ذخیره شده و سپس مرحله به مرحله با استفاده از قضیه معادل بودن ماشین های NFA و DFA ، ماشین را به DFA تبدیل میکند که اثبات آن خارج از بحث ما است .

روش کلی تبدیل به صورت زیر است :

1. ابتدا بستار ستاره حالت (state) آغازین را پیدا میکنیم
2. سپس مشخص میکنیم حالت آغازین به کدام حالت ها میرود .
3. حالت های جدید به وجود آمده را نیز نوشته و این عمل را تا جایی که هیچ حالتی فاقد اطلاعات باشد انجام میدهیم
4. برای پیدا کردن حالت نهایی هر حالتی از ماشین DFA که شامل حالت های نهایی ماشین NFA باشد را حالت نهایی ماشین DFA حساب میکنیم

**ساختار کلی برنامه**

ساختار ذخیره اطلاعات در این برنامه به این صورت است که هر حالت به صورت آرایه ای از مقادیر بولین در یک بردار (vector ) ذخیره میشود و اندیس هر درایه نشان دهنده شماره آن مرحله است و مقدار آن اگر یک باشد یعنی شامل آن مرحله است . به طور مثال 0011 منظور مرحله 2و3 است و دلیل اینکار این است که در هنگام تبدیل به اگر به یک حالت دو بار اشاره شد مشکلی به وجود نیاید .

این برنامه شامل سه بخش کلی می باشد که به شرح زیر است :

1. ساختار داده (struct) برای ذخیره هر مرحله از ماشین NFA
2. ساختار داده (struct) برای ذخیره هر مرحله از ماشین DFA
3. کلاس مبدل به نام converter

**ساختار داده (struct) NFA**

این ساختار برای ذخیره هر حالت ماشین NFA نوشته شده است و با توجه به نزدیک بودن ساختار داده به کلاس ها میتوان مانند کلاس ها مجموعه ای از اشیا از ساختار داده ایجاد کرد و ماشین NFA را به صورت کامل ذخیره کرد .

این ساختار داده از سه بخش تشکیل شده است که به شرح زیر می باشد :

1. یک متغیر بولین برای ذخیره وضعیت اپسیلون به نام has\_epsilon
2. یک متغیر بردار در بردار برای ذخیره اطلاعات به نام state\_info
3. یک متغیر برداری برای ذخیره اطلاعات اپسیلون به نام epsilon
4. در این ساختار یک متغیر بولین برای تعیین اینکه مرحله ماشین از اپسیلون استفاده میکند یا نه استفاده شده است دلیل این کار این صرفه جویی در زمان است زیرا بررسی مقدار بولین ساده است و 1 بایت حافظه اشغال میکند
5. این متغیر مانند یک آرایه دو بعدی عمل میکند به طوری که اندیس اول شماره الفبا را مشخص میکند و مجموعه اندیس های بعد دوم مشخص میکند که حالت مذکور به کدام حالت میرود .
6. این متغیر برداری برای ذخیره اینکه حالت مذکور با اپسیلون به کدام حالت میرود ، تعریف شده است . دلیل تعریف جدای این بردار جلوگیری از پیچیدگی زمانی در کلاس انتقال (converter) است و میزان حافظه اتلافی(یک بردار خالی حدود 24 بایت حافظه اشغال میکند ) برای هر حالت در مقابل صرفه جویی شده در حافظه اتلافی نا چیز است .

**ساختار داده (struct) DFA**

این ساختار برای ذخیره هر مرحله ماشین DFA است و مانند ساختار داده قبلی با آرایه ای از اشیا میتوان ماشین DFA را به طور کامل ذخیره سازی کرد .

این ساختار داده از سه بخش تشکیل شده است که به شرح زیر می باشد :

1. یک متغیر بولین برای ذخیره وضعیت حالت نهایی به نام qacc
2. یک بردار برای ذخیره نام مرحله به نام ID
3. یک بردار در بردار برای ذخیره اطلاعات هر مرحله به نام info
4. این متغیر مشخص میکند که آیا این مرحله از ماشین حالت نهایی است یا خیر و یک بایت حافظه اشغال میکند
5. این بردار برای ذخیره نام مرحله است زیرا نام مراحل ماشین DFA زیر مجموعه ای از نام های ماشین NFA داده شده است به طور مثال اگر یک مرحله از ماشین DFA از دو مرحله از ماشین NFA تشکیل شده باشد میتوان با این بردار . مشخص کرد در واقع این بردار نشان دهنده این است که این مرحله از ماشین DFA از کدام مراحل ماشین NFA تشکیل شده است
6. این متغیر مانند یک آرایه دو بعدی عمل میکند به طوری که اندیس اول الفبا را مشخص میکند و مجموعه اندیس های بعد دوم مشخص میکند که حالت مذکور به کدام حالت میرود .

**کلاس مبدل**

در واقع این کلاس قسمت اصلی برنامه است ، این کلاس شامل دو بخش public و private تشکیل شده است که قسمت عمومی (public) از توابعی تشکیل شده است که انتقال را انجام میدهند و قسمت خصوصی (private) که اطلاعات مورد نیاز برای تبدیل را نگه داری میکند که به شرح زیر میباشد :

**قسمت عمومی(public)**

1. تابع get\_data()
2. تابع get\_table()
3. تابع convert\_qstart()
4. تابع convert\_machine()
5. تابع print()

**قسمت خصوصی (private)**

1. متغیر integer به نام qstart\_nfa
2. متغیر برداری از جنس بولین به نام qacc
3. متغیر integer به نام alphabet\_count
4. متغیر integer به نام nfa\_state\_count
5. متغیر integer به نام dfa\_state\_count
6. بردار از جنس ساختار داده nfa به نام nfa\_obj
7. بردار از جنس بولین به نام dfa\_qstart
8. بردار از جنس ساختار داده dfa به نام dfa\_obj
9. صف دو طرفه از جنس برداری از جنس بولین به نام dque

**قسمت خصوصی (private )**

**متغیر integer به نام qstart\_nfa :** در این متغیر حالت آغازین ماشین NFA ذخیره میشود چون حالت آغازین نمیتواند بیشتر از یک حالت باشد از integer استفاده شده است .

**متغیر برداری از جنس بولین به نام qacc :** در این متغیر حالت های پایانی ماشین ذخیره میشود و چون حالت های پایانی ماشین DFA زیر مجموعه ای از حالت های پایانی ماشین NFA است نیازی به تعریف دوباره این متعیر برای هر ماشین نیست

**متغیر integer به نام alphabet\_count:** در این متغیر تعداد حروف الفبا بدون حساب اپسیلون ذخیره میشود دلیل اینکه اپسیلون حساب نمیشود پیدا کردن عددی واحد برای تعداد حروف الفبا برای ماشینNFA و DFA است .

**متغیر integer به نام nfa\_state\_count :** در این متغیر تعداد حالت های ماشین NFA ذخیره میشود .

**متغیر integer به نام dfa\_state\_count :** در این متغیر تعداد حالت های ماشین DFA ذخریه میشود .

**بردار از جنس ساختار داده nfa به نام nfa\_obj :** در واقع این بردار آرایه ای از حالت های ماشین NFA است که در ساختار داده nfa ذخیره شده است ، و اندیس هر درایه از بردار مشخص کننده شماره آن حالت است ، یک حالت تهی با اندیس صفر بدون پر کردن اطلاعات در نظر گرفته شده است تا مشکلی برای استفاده صفر در اطلاعات حالت ها وجود نداشته باشد در غیر این صورت صفر در آرایه اطلاعات حالت ها به حالت q0 نیز اشاره کند ، برای همین حالت ها از یک شمارده میشوند ، به طور مثال q1,q2,q3 ,…

**بردار از جنس بولین به نام dfa\_qstart :** در این متغیر حالت آغازین برای ماشین DFA ذخیره میشود چون باید بستار ستاره حالت آغازین NFA ذخیره شود ممکن است بیشتر از یک حالت داشته باشیم و به همین دلیل از متغیر برداری استفاده شده است.

**بردار از جنس ساختار داده dfa به نام dfa\_obj :** در واقع در این بردار آرایه ای از حالت های ماشین DFA ذخیره میشود و چون حالت های این ماشین زیر مجموعه ای از حالت های ماشین NFA است اندیس هر درایه از بردار فقط مشخص کننده الویت پردازش در برنامه خواهد بود.

**صف دو طرفه از جنس برداری از جنس بولین به نام dque :**  دلیل تعریف این صف این است که حالت های جدیدی که در پردازش ایجاد میشود به صورت موقت در آن ذخیره شود و دلیل دو طرفه بودن آن این است که بتوان مراحلی که چندین بار ایجاد میشود را حذف کرد .

**قسمت عمومی (public)**

**تابع get\_data() :** وظیفه این تابع گرفتن اطلاعات اولیه از کاربر است این اطلاعات شامل تعداد حروف الفبا ، تعداد مراحل ماشین NFA ، حالت اولیه و حالت پایانی است ، هرچند که با توجه به روش ورودی اطلاعات میتوان از گرفتن تعداد حروف الفبا و تعداد حالت های ماشین NFA صرف نظر کرد .

**تابع get\_table() :** وظیفه این تابع گرفتن اطلاعات هر مرحله از ماشین NFA است ، به این صورت که برای هر مرحله به تعداد حروف الفبا از کاربر در مورد اینکه مرحله مورد نظر با الفبای انتخابی به کدام مرحله میرود و در پایان از کاربر سوال میشود که آیا مرحله با اپسیلون به جایی میرود یا خیر که با فشردن هر دکمه ای میتوان از این قسمت رد شد .

**تابع convert\_qstart() :** وظیفه این تابع تبدیل مرحله آغازین ماشین NFA به حالت آغازین DFA است ، در واقع این تابع بستار ستاره حالت آغازین ماشین NFA را حساب میکند .

**تابع convert\_machine() :** در واقع این تابع ،تابع اصلی تبدیل ماشین است و چندین بخش مختلف تشکیل شده است :در ابتدا حالت آغازین ماشین DFA که توسط تابع() convert\_qstart ساخته شده است را وارد صف دو طرفه میکنیم ، سپس یک بردار موقت برابر با مقدار جلویی صف ایجاد میکنیم دلیل این کار راحتی در کار با بردار موقت است تا خود مقداری که در صف است و در مقابل 24 بایت + 1 بایت به ازای هر مرحله استفاده از بردار موقت به صرفه به نظر میرسد ، سپس درایه های بردار موقت مورد بررسی قرار میگیرد و اگر برابر با مقدار یک بود ، برایر با ID تعریف شده در ساختار داده dfa قرار میدهیم و سپس اندیس بردار موقت به اضافه 1 (چون یک مرحله خالی برای جلوگیری از وجود دو معنی برای صفر ایجاد کرده بودیم ) برابر با حالتی است که ما در ماشین NFA داده شده نیاز داریم ، و اطلاعات آن مرحله را بررسی کرده و آن را با اطلاعات آن را با یک مرحله انتخابی از ساختار داده dfa جمع منطقی کرده (مقادیر ما بولین میباشد ) و اگر اندیس بردار موقت در **qacc** برابر یک بود آن حالت پایانی نیز میباشد و اگر هیچ کدام از مقادیر بردار موقت برابر یک نبود حالت تهی تشکیل میشود و حالت تهی با هر الفبایی به خود باز میگردد و سپس بستار ستاره آن را حساب میکنیم ، حال باید حالت های جدید اضافه شده را وارد صف دو طرفه کنیم برای این کار اطلاعات حالتی از ساختار dfa که انتخاب کرده بودیم را وارد صف دو طرفه میکنیم به شرطی که این حالت از قبل در ساختار داده dfa وجود نداشته باشد و در صف دو طرفه نیز وجود نداشته باشد و بعد از اضافه کردن مراحل مقدار dfa\_state\_count که برابر با یک قرار داده بودیم را به علاوه یک میکنیم و با توجه while تعریف شده در اول تابع این کار تا زمانی که صف خالی شود انجام میدهیم .

**تابع print() :** وظیفه این تابع نمایش اطلاعات به کاربر است و این قسمت کانلا به برنامه نویس مربوط میشود که از چه ابزاری برای نمایش استفاده کند ، من در برنامه به دلیل آموزش محور بودن و تجاری نبودن ساده ترین روش یعنی چاپ در کنسول و ذخیره آن در یک فایل csv بسنده کرده ام .

**سخن پایانی :**

این برنامه به زبان c++ نوشته شده است و سعی شده است که از لحاظ خوانا و خلاصه بودن بهینه باشد و به موضوع پیچیدگی حافظه توجه نشده است اما در بدترین حالت ممکن هم مصرف حافظه قابل قبولی خواهیم داشت .در صورت نیاز به کد این برنامه به صورت متن باز با مجوز MIT در سایت گیت هاب(github) نیز بارگذازی شده است .

هدف این برنامه آموزشی است و سعی شده است تا جای ممکن خوانا نوشته شود و به همین دلیل ممکن است در بعضی موارد از لحاظ پیچیدگی زمانی و حافظه بهینه نباشد و این برنامه به صورت متن باز در اختیار عموم قرار دارد بسیار ممنون میشوم اگر پیشنهادی برای بهتر شدن این برنامه دارید به بنده اطلاع دهید .