نام و نام خانوادگی: **امیرحسین علی‌بخشی**
شماره‌ی دانشجویی: **۹۷۳۱۰۹۶**

گزارش پروژه شماره ۱

استاد: **دکتر میبدی**
درس: **نظریه زبانها و ماشینها**

فهرست مطالب

تمرین اول[i]

در این تمرین ۴ کلاس ***State***، ***Path***، ***Acceptor*** و ***Main*** تعریف میشوند که در زیر به تحلیل و بررسی هرکدام میپردازیم:

کلاس حالت (State)

State
- title : string - isStarsState : boolean - IsFinalState : boolean

در این کلاس هر حالت را تعریف میکنیم. این کلاس فیلد هایی نظیر عنوان و دو فیلد از نوع بولین دارد که مشخص میکنند که آیا این حالت یک حالت نهایی/ابتدایی هست یا خیر.

کلاس مسیر (Path)

Path
- startState : string - inputAlphabet : string - endState : string

در این کلاس تلاش بر این داریم که یک مسیر را نمایش بدهیم. یک مسیر یعنی اینکه ما با گرفتن یک ورودی -که میان الفباهای تعریف شده است- از حالت فعلی به حالت بعدی برویم. در اینجا در فیلد اول و سوم به جای اینکه خود حالات را بعنوان فیلد انتخاب کنیم، عنوانشان را برگزیدیم (با این فرض که هیچ دو حالتی وجود ندارند که نامشان یکسان باشد!)

کلاس پذیرنده (Acceptor)

Acceptor
- accepterStates : ArrayList<State> - accepterAlphabet : ArrayList<String> - accepterPaths : ArrayList<Path>

یک پذیرنده در اصل همان ماشینی است که قصد طراحی آن را داریم. در واقع تمام کلاس های قبل را تعریف کردیم که بتوانیم این کلاس را طراحی کنیم. تمامی حالات، مسیر ها و الفباهای مجاز در این ماشین را داخل این کلاس قرار میدهیم.هنگام ساخته شدن این ماشین دو حالت وجود خواهد داشت(که از کاربر پرسیده میشود):

- ساخته شدن این ماشین را به عهده خود برنامه قرار دهیم. یعنی کاربر هیچ نقشی در ساخته شدن آن نخواهد داشت. این اطلاعات پیش‌فرض، همان مقادیری هستند که در صورت سوال به ما داده شده است.
- گرفتن اطلاعات از کاربر؛ یعنی کاربر در سطر اول الفباها، سطری بعد تمام حالات موجود در ماشین، سطر بعد حالت شروع سطر بعد حالت های نهایی و در خطوط بعدی را به اطلاعات هریک از مسیر های ماشین اختصاص دهد؛ بدین صورت که در هر سطر به ترتیب عنوان حالت شروع، الفبایی که موجب تغییر حالت میشود و در آخر عنوان حالت پایان را وارد میکند و میان هرکدام از این موارد از space استفاده کند. ورودی گرفتن تا زمانی صورت میگیرد که تعداد قسمت هایی که در هر سطر وارد شده و به وسیله ی space از هم جدا شده اند دقیقاً ۳ باشد. نه کمتر، نه بیشتر...

[i] تمام فایل های این پروژه به زبان java نوشته شده اند.

- ساخته شدن ماشین از داخل یک فایل متنی؛ برنامه دو گزینه پیش روی ما قرار خواهد داد: ۱) خواندن یک فایل داخل پوشه‌ی تمرین مورد نظر به نام DFA_Input_1.txt و ۲) خواندن فایل با یک نام که کاربر مشخص میکند.
حال اگر ورودی های داخل فایل مورد نظر صحیح بود ماشین به درستی کار میکند در غیر این صورت برنامه به ما خطا میدهد.

کلاس اصلی (Main)

این کلاس در اصل برای ارتباط برنامه با کاربر طراحی شده است. در ابتدای برنامه عبارت زیر ظاهر میشود:

```
CHOOSE ONE OF THIS OPTIONS:
 A - USE A DEFAULT DFA
 B - CREATE YOUR OWN DFA
 C - CREATE NFA FROM A TEXT FILE
 Q - EXIT THE PROGRAM
```

که واضح است هر کدام از قسمت ها مربوط به چه کاری هستند. اگر ما یکی از حالات A، B یا C را انتخاب کنیم بسته به مطالبی که در قسمت کلاس Acceptor توضیح داده شد، ماشین مورد نظرمان را میسازیم. بعد از ساخته شدن این ماشین پیام زیر ظاهر میگردد:

```
NOW YOU CAN ENTER STRINGS WITH THIS ALPHABETS:
[a, b]
ENTER " /quit " TO GO BACK.
ENTER " /status " TO GET THE DFA'S STATUS.
```

که در این مثال فرض شده است که الفباهای این ماشین a و b هستند. تا زمانی که کاربر عبارت `/quit` را وارد نکند، این حلقه تکرار میگردد (وگرنه از حلقه خارج شده و دوباره منوی ساخت ماشین ظاهر میشود). اگر عبارت `/status` وارد شود اطلاعات مربوط به ماشین برای ما نمایش داده میشود. بعنوان مثال هنگامی که ماشین ما همان ماشین پیشفرض صورت سوال باشد، با وارد کردن `/status` با پیام زیر روبرو میشویم:

```
>>> ALPHABETS:
   >>> a
   >>> b
--------------------------------------
>>> ALL OF THE STATES:
   >>> Q0
   >>> Q1
   >>> Q2
--------------------------------------
>>> START STATES:
   >>> Q0
--------------------------------------
>>> FINAL STATES:
   >>> Q1
--------------------------------------
>>> PATHS:
   >>> ( Q0 ) ----- a -----> ( Q1 )
   >>> ( Q0 ) ----- b -----> ( Q1 )
   >>> ( Q1 ) ----- a -----> ( Q2 )
   >>> ( Q1 ) ----- b -----> ( Q2 )
   >>> ( Q2 ) ----- a -----> ( Q2 )
   >>> ( Q2 ) ----- b -----> ( Q2 )
```

که واضح است هر کدام از خطوط بیانگر چیست.

اگر هر چیز دیگری به غیر از دو دستور گفته شده وارد شود، برنامه آن را یک رشته ی ورودی در نظر میگیرد و آن را تحلیل میکند. کاراکترها را از چپ به راست بررسی میکند و مشخص میکند که این ورودی مقبولِ این ماشین میباشد یا خیر.
بدیهی است که اگر یکی از ورودی ها نامعتبر باشد ماشین دست از کار میکشد و اعلام میکند که این ورودی نامعتبر بوده و منتظر ورودی های بعدی نمیماند. بعنوان مثال اگر ما ورودی "abadi" را به این پذیرنده بدهیم، ورودی های زیر را دریافت خواهیم کرد:

```
INPUT STRING: " abadi "
( 1 = SUCCESSFUL STEP , 0 = FAILED)
( Q0 ) ----- a -----> ( Q1 )

| a | b | a | d | i |
| 1 | - | - | - | - |

( Q1 ) ----- b -----> ( Q2 )

| a | b | a | d | i |
| 1 | 1 | - | - | - |

( Q2 ) ----- a -----> ( Q2 )

| a | b | a | d | i |
| 1 | 1 | 1 | - | - |

ERROR FOUND WHILE READING INPUT "d"!

| a | b | a | d | i |
| 1 | 1 | 1 | 0 | - |

THIS STRING IS NOT VALID!!!!
```

مشاهده میشود که در هر مرحله علاوه بر نمایش مسیر مورد نظر، زیر ورودی های کاربر عبارات زیر نمایش داده شده اند:

- **1:** یعنی این ورودی خاص توسط پذیرنده به درستی پذیرفته شده و مشکلی پیش نیامده است.
- **0:** یعنی با خطا روبرو شده ایم! خطا هم میتواند ملاحظه‌ی حرفی خارج از الفبای این ماشین و هم میتواند زمانی باشد که با الفبای مورد نظر مسیری به یک حالت دیگر وجود نداشته باشد.
- **-:** یعنی هنوز زمان بررسی این الفبا فرا نرسیده است.

منطقی است زمانی که در مرحله ی آخر عبارت زیر رشته‌ی ورودی کاربر بصورت 1..111 باشد. یعنی مشکلی رخ نداده است. حال باید به حالتی که دستگاه در حال حاضر در آن قرار دارد نگاه کنیم. اگر این حالت جزو حالات نهایی این دستگاه بود یعنی این رشته پذیرفته شده، وگرنه رشته‌ی ورودی معتبر نخواهد بود!

تمرین دوم

نحوه‌ی عملکرد این تمرین بسیار شبیه تمرین شماره یک است. با این تفاوت که در قسمت تابع main تمام درخواست های برنامه از کاربر برحسب NFA خواهد بود نه DFA. در این تمرین علاوه بر کلاس‌های موجود در تمرین اول دو کلاس دیگر تعریف میکنیم:

کلاس DFA که از NFA ساخته شده (DFAfromNFA)

DFAfromNFA
- accepterStates : List<List<State>> -
- accepterAlphabet : ArrayList<String>
- accepterPaths : ArrayList<PathForDFA>

این کلاس بسیار شبیه به کلاس Accepter میباشد. اما در این تمرین (برخلاف تمرین قبلی) کلاس Accepter برای NFA ورودی به کار میرود ولی این کلاس در اصلا همان DFA ای است که باید NFA را به آن تبدیل کنیم.

کلاس مسیر در DFA جدید (PathForDFA)

PathForDFA
- startState : List<State> - inputAlphabet : string - endState : List<State>

این کلاس بسیار شبیه Path میباشد با این تفاوت که میتواند مبدا و مقصد مسیر را به صورت مجموعه ای نمایش دهد. (List<State>) اما به طور کلی شبیه همان مسیر در تمرین قبل کار میکند.

حال به سراغ نحوه‌ی کارکردن این ماشین می‌رویم:

مانند تمرین قبل کاربر میتواند هم از NFA ورودی هم از نوع پیش‌فرض آن که در صورت سوال ذکر شده است استفاده نماید. پس از ساخته شدن این پذیرنده ی غیر قطعی آن را نمایش میدهیم(مانند تمرین قبل). لازم به ذکر است که اگر در این تمرین ورودی ها غلط باشند، برنامه پیغام WRONG INPUT میدهد و دوباره از کاربر ورودی میگیرد.

پس از ساخته شدن این NFA، اقدام میکنیم به ساختن DFA معادل آن. حالات این DFA در اصل مجموعه توانیِ حالت های NFA خواهد بود. یعنی به عنوان مثال در NFA پیش‌فرض که $Q = \{q0\ , q1\ , q2\}$ است، برای DFA خواهیم داشت:

$$Q' = 2^Q = \{[\ ], [q0], [q1], [q2], [q0, q1], [q1, q2], [q2, q0], [q0, q1, q2]\}$$

حالت اولیه همان q0 و حالت های نهایی تمام حالاتی هستند که در آنها حالت های نهایی NFA وجود دارد.(در این مثال [q1]،[q1,q2]،[q1,q0]و[q0,q1,q2])

به همین ترتیب مسیر هایی ممکن را به کمک اجتماع گیری پیدا میکنیم. از نمایش حالاتی که به [] میروند یا از [] شروع میشوند اجتناب میکنیم. همچنین مسیر هایی که با λ به حالتی غیر تهی میرسند را ساده میکنیم. در پایان خروجی بصورت زیر خواهد بود:

```
OUTPUT OF THE DFA:
0 1
[] [q0] [q0,q1] [q0,q1,q2] [q0,q2] [q1] [q1,q2] [q2]
[q0]
[q0,q1] [q0,q1,q2] [q1] [q1,q2]
[q0] 0 [q0,q2]
[q0] 1 [q1,q2]
[q0,q1] 0 [q0,q1,q2]
[q0,q1] 1 [q1,q2]
[q0,q1,q2] 0 [q0,q1,q2]
[q0,q1,q2] 1 [q1,q2]
[q0,q2] 0 [q1,q2]
[q0,q2] 1 [q1]
[q1] 0 [q0,q2]
[q1] 1 [q1,q2]
[q1,q2] 0 [q0,q2]
[q1,q2] 1 [q1,q2]
[q2] 0 [q2]
[q2] 1 [q1]
```

بعد از نمایش اطلاعات مربوط به این ماشین معادل، برنامه از ما میپرسد که آیا دوست داریم اطلاعات مربوط به ماشین جدید را ذخیره کنیم یا نه. اگر بنا بر ذخیره سازی بود همانند خواندن از فایل متنی از ما سوال میشود که قصد داریم از نام پیش‌فرض استفاده کنیم یا میخواهیم نام دلخواه خود را استفاده کنیم. حال کاربر میتواند دوباره انتخاب کند که ماشین جدیدی بسازد یا اینکه از برنامه خارج شود.