Graph Theory Project

Steps:

1) Parse the regular expression from infix to postfix notation.

2) Build a series of small NFA’s for parts of the regular expression.

3) Use the smaller NFA’s to create the overall NFA.

4) Implement the matching algorithm using the NFA.

Step 1 –

To do this step I will need to learn how to implement the shunting yard algorithm, through videos provided and research online.

I will add a feature of user input. This will make it easier to test various strings.

Step 2 –

To do this step I needed to learn how to use the postfix notation to create small NFA’s, In the video on LearnOnline we were shown how to do this by using fragments.

Step 3 –

Once step two was finished it should be manageable to use the small NFA's to create a bigger one.

Step 4 –

Looking at the video provided on matching we need to be able to take in two strings. I will use user input to do this so I can again test various strings and regular expressions.

For user input I used the following,

```
regex = input("Enter regular expression:  ")
s = input("Enter String to compare:  ")
```

In another document called GraphTheoryProjectWithFileReader two files will be taken in. One file will have a regular expression and the other file with a string to compare. I used the following code.

```
with open('Test.txt', 'r') as RegularExpression:
      regex = RegularExpression.read()


with open('Test2.txt', 'r') as CompareString:
      s = CompareString.read()
```

I found this to be the best way to read in the files. There was various ways to read in the file, like using readLine(). I thought read() would work better.

I also added in the '+' and '?' operator. I looked on the website

Adding the + and ?

“https://www.debuggex.com/cheatsheet/regex/python” and it helped me to figure out how to implement them.

I used the following code to get the “+” operator to work.

```
        elif c == '+':
            frag = nfa_stack.pop()
            frag.accept.edges.append(frag.start)
            newfrag = Fragment(frag.start, frag.accept)
```

Seeing that this operator has to take in one or more I used the frag.start and frag.accept so it will be true if more than one is entered and false if there is an empty string.

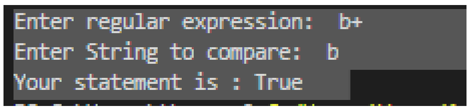
I added in the “?” which means 0 or. I used the following the code to add it.

```
elif c == '?':
            frag = nfa_stack.pop()
            accept = State()
            start = State(edges=[frag.start, accept])
            frag.accept.edges = ([accept])
            newfrag = Fragment(start, accept)
```

One frag gets taken of the nfa_stack as I t can either be 0 or 1.

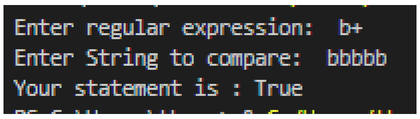
Accept state is equal to State() which is none so it can take in 0. Then it can take 1 as the start it equal to frag.start,accept.

Examples of the + operator working (1 or many)



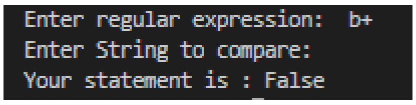
```
Enter regular expression:  b+
Enter String to compare:  b
Your statement is : True
```

True



```
Enter regular expression:  b+
Enter String to compare:  bbbbb
Your statement is : True
```

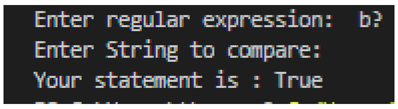
True



```
Enter regular expression:  b+
Enter String to compare:
Your statement is : False
```

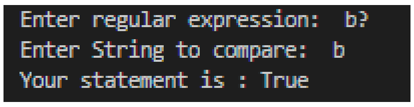
False

Examples of the ? operator working (0 or 1)



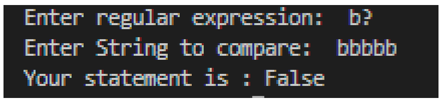
```
Enter regular expression:  b?
Enter String to compare:
Your statement is : True
```

True



```
Enter regular expression:  b?
Enter String to compare:  b
Your statement is : True
```

True



```
Enter regular expression:  b?
Enter String to compare:  bbbbb
Your statement is : False
```

False