AIM:

To convert Regular Expression to NFA.

ALGORITHM:

- Define a 2D array for storing the transition table.
- Initialize a counter variable.
- Define cases for each of the operations '|', '+', '*' with their respective transitions and the alphabets.
- Iterate through each of the symbols and update the transition table.
- Update the counter after each iteration.
- Finally, print the transition table after it has been filled with all the transitions and the start state and final state.

PYTHON CODE

```
inp = input("Enter: ")
"""
Give your input in the above variable
   a and b are the only terminals accepted by this script
   e denotes epsilon
   . is used for "and" operation Eg. ab = a.b
   + is used for "or" operation Eg. a|b = a+b
   * is the Kleene's Closure operator. You can give star operator after any closing brackets and terminals
"""



_="-"

start = 1 # denotes start of e-nfa table
end = 1   # denotes end of our table which is initially same as start
cur = 1   # denotes current position of our pointer
# this is intitial e-nfa table with only one state which is start and end both
table = [["state","epsilon","a","b"],
         [1,_,_,_]]

def print_t(table):
   """
   This function prints the e-nfa table
   """
   i = table[0]
   print(f'{i[0]: <10}'+f'| {i[1]: <10}'+f'| {i[2]: <10}'+f'| {i[3]: <10}')
   print("-"*46)
   for i in table[1:]:
      try:
```

```
        x = " ".join([str(j) for j in i[1]])
    except:
        x = ""
    try:
        y = " ".join([str(j) for j in i[2]])
    except:
        y = ""
    try:
        z = " ".join([str(j) for j in i[3]])
    except:
        z = ""
    print(f'{i[0]: <10}'+f'| {x: <10}'+f'| {y: <10}'+f'| {z: <10}')

def e_(cur,ed=end):
    """
    this fuction adds epsilon to the table
    """
    temp = table[cur]
    try:
        table[cur] = [cur,temp[1].append(cur+1),temp[2],temp[3]]
    except:
        table[cur] = [cur,[cur+1],temp[2],temp[3]]
    try:
        nv = table([cur+1])
    except:
        table.append([ed+1,_,_,_])
        ed+=1
    return ed


def a_(cur,ed=end):
    temp = table[cur]
    try:
        table[cur] = [cur,temp[1],temp[2].append(cur+1),temp[3]]
    except:
        table[cur] = [cur,temp[1],[cur+1],temp[3]]
    try:
        nv = table([cur+1])
    except:
        table.append([ed+1,_,_,_])
        ed+=1
    return ed

def b_(cur,ed=end):
    temp = table[cur]
    try:
        table[cur] = [cur,temp[1],temp[2],temp[3].append(cur+1)]
    except:
        table[cur] = [cur,temp[1],temp[2],[cur+1]]
    try:
        nv = table([cur+1])
```

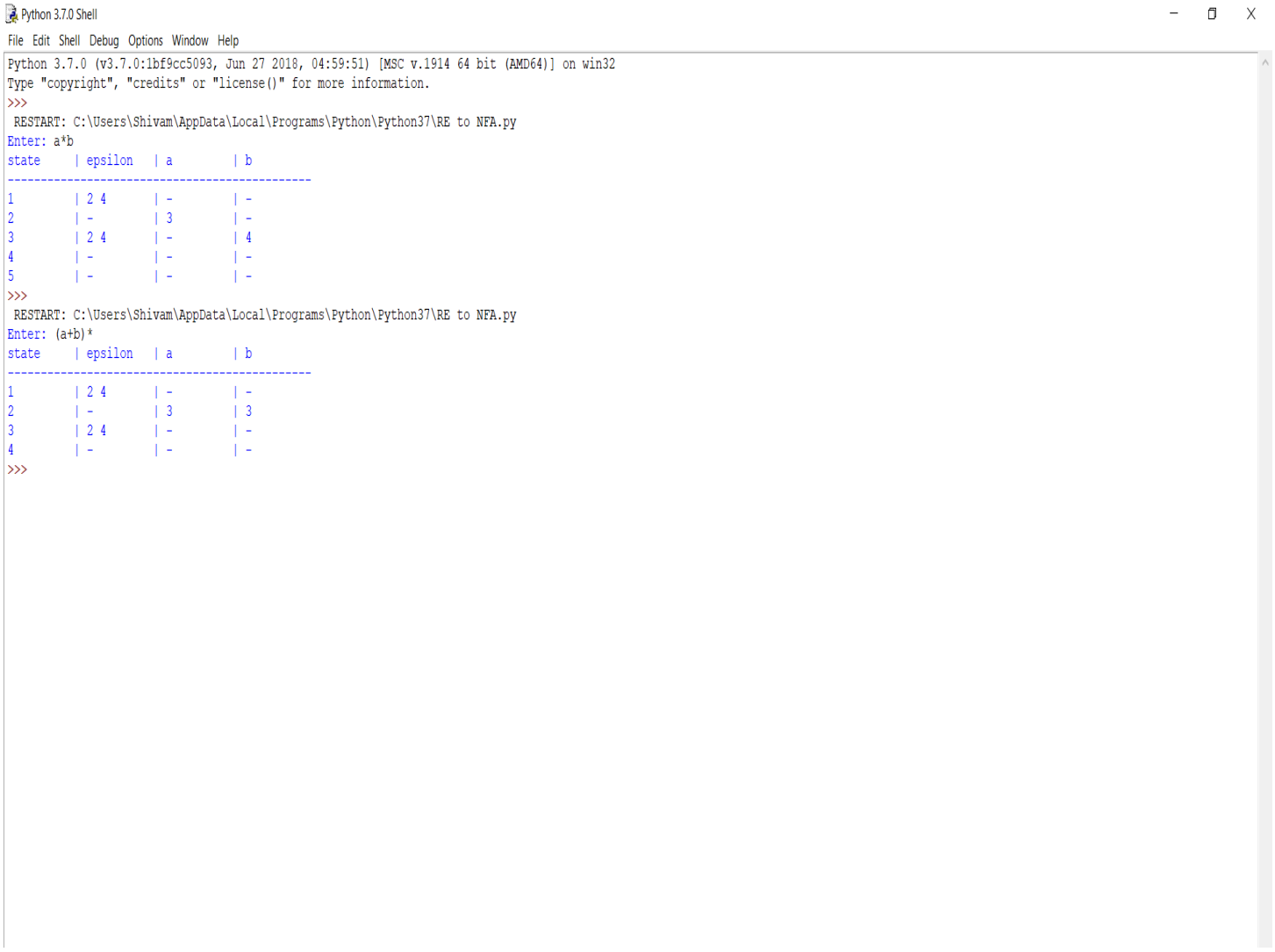
```
    except:
        table.append([ed+1,_,_,_])
        ed+=1
    return ed

def or_b(cur,ed=end):
    temp = table[cur]
    try:
        table[cur] = [cur,temp[1],temp[2],temp[3].append(cur+1)]
    except:
        table[cur] = [cur,temp[1],temp[2],[cur+1]]


def or_a(cur,ed=end):
    temp = table[cur]
    try:
        table[cur] = [cur,temp[1],temp[2].append(cur+1),temp[3]]
    except:
        table[cur] = [cur,temp[1],[cur+1],temp[3]]

def and_a(cur,ed=end):
    cur+=1
    temp = table[cur]
    try:
        table[cur] = [cur,temp[1],temp[2].append(cur+1),temp[3]]
    except:
        table[cur] = [cur,temp[1],[cur+1],temp[3]]
    try:
        nv = table([cur+1])
    except:
        table.append([cur+1,_,_,_])
        ed+=1
    return cur,ed

def and_b(cur,ed=end):
    cur+=1
    temp = table[cur]
    try:
        table[cur] = [cur,temp[1],temp[2],temp[3].append(cur+1)]
    except:
        table[cur] = [cur,temp[1],temp[2],[cur+1]]
    try:
        nv = table([cur+1])
    except:
        table.append([cur+1,_,_,_])
        ed+=1
    return cur,ed

def star(cur,ed=end):
    table.append([ed+1,_,_,_])
    table.append([ed+2,_,_,_])
```

```
    ed+=2
    for i in range(cur,ed):
        temp = [table[ed-i+cur][0]]+table[ed-i+cur-1][1:4]
        for j in [1,2,3]:
            try:
                temp[j] = [x+1 for x in table[ed-i+cur-1][j]]
            except:
                pass
        table[ed-i+cur] = temp
    table[cur]=[cur,_,_,_]

    temp = table[cur]
    try:
        table[cur] = [temp[0],temp[1]+[cur+1,ed],temp[2],temp[3]]
    except:
        table[cur] = [temp[0],[cur+1,ed],temp[2],temp[3]]

    temp = table[ed-1]
    try:
        table[ed-1] = [temp[0],temp[1]+[cur+1,ed],temp[2],temp[3]]
    except:
        table[ed-1] = [temp[0],[cur+1,ed],temp[2],temp[3]]

    return ed-1,ed


def mod_table(inp,start,cur,end,table):
    #print(inp)
    k = 0
    while k<len(inp):
        #print(start,cur,end,k,inp[k:],len(table)-1)
        if inp[k]=="a":
            end = a_(cur,end)
            #print("in a_")
        elif inp[k]=="b":
            end = b_(cur,end)
            #print("in b_")
        elif inp[k]=="e":
            end = e_(cur,end)
        elif inp[k]==".":
            k+=1
            if inp[k]=="a":
                #k-=1
                cur,end = and_a(cur,end)
            elif inp[k]=="b":
                cur,end = and_b(cur,end)
                #k-=1
            elif inp[k]=="(":
                li = ["("]
                l = k
                for i in inp[k+1:]:
```

```
            if i == "(":
                li.append("(")
            if i == ")":
                try:
                    del li[-1]
                except:
                    break
            if len(li)==0:
                break
            l+=1
        m = k
        k=l+1
        cur+=1
        start,cur,end,table = mod_table(inp[m+1:l+1],start,cur,end,table)

    elif inp[k]=="+":
        k+=1
        if inp[k]=="a":
            or_a(cur,end)
            #print("in or_a")
        elif inp[k]=="b":
            or_b(cur,end)
            #print("in or_b")
        else:
            print(f"ERROR at{k }Done:{inp[:k+1]}Rem{inp[k+1:]}")

    elif inp[k]=="*":
        #print("in star")
        cur,end = star(cur,end)
    elif inp[k]=="(":
        li = ["("]
        l = k
        for i in inp[k+1:]:
            if i == "(":
                li.append("(")
            if i == ")":
                try:
                    del li[-1]
                except:
                    break
            if len(li)==0:
                break

            l+=1
        m = k
        k=l+1
        try:
            if inp[k+1]=="*":
                cur_ = cur
        except:
            pass
```

```
            #print(inp[m+1:l+1])
            start,cur,end,table = mod_table(inp[m+1:l+1],start,cur,end,table)
            try:
                if inp[k+1]=="*":
                    cur = cur_
            except:
                pass
        else:
            print(f'error{k}{inp[k]}')
        k+=1
    return start,cur,end,table

start,cur,end,table = mod_table(inp,start,cur,end,table)

print_t(table)
```

IMPLEMENTATION



RESULT

Code was successfully implemented and the output was verified.