Write a program to check whether input string is accepted by given DFA or not.

Intro:

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language. Example : (a+b)*aba(a+b)*

- The languages accepted by some regular expression are referred to as Regular languages.

- Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a finite number of states, the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.**

Code:

```
# Re = (a+b)*abb

def takeTT():
    totalSym = list(input("Symbols: ").split())
    totalStates = int(input("How many states: "))
    acceptedState = list(map(int,input("Accepted state numbers :").split()))
    temp = totalStates
    tt = []
    print(f'Enter T.T. values by rows for column {totalSym}')
    while temp > 0:
        tt.append(list(map(int,input().split())))
        temp -= 1
    print(tt)
    return tt, totalSym, acceptedState

def takeAndCheck_Strings(TT,symbols,accStates):
    #take strings to check
    strList = []
    userString = ''
    print("RE: (a+b)*abb")
```

```
    while userString != 'null':
        userString = input()
        strList.append(userString)

    #check using TT

    for string in strList[:-1]:
        state = 0
        for ch in string:
            prevState = state
            symIndex = symbols.index(ch)
            state = TT[state][symIndex]
            print(prevState,"->",state)

        if state in accStates:
            print(string,"is Accepted")
        else:
            print(string,"is not Accepted")

transTable, Symbols, accState = takeTT()
takeAndCheck_Strings(transTable,Symbols,accState)
```

Output:

```
Symbols: a b
How many states: 4
Accepted state numbers :3
Enter T.T. values by rows for column ['a', 'b']
1 0
1 2
1 3
1 0
[[1, 0], [1, 2], [1, 3], [1, 0]]
RE: (a+b)*abb
abb
baa
abbbbaaabb
null
0 -> 1
1 -> 2
2 -> 3
abb is Accepted
0 -> 0
0 -> 1
1 -> 1
baa is not Accepted
0 -> 1
1 -> 2
2 -> 3
3 -> 0
0 -> 0
0 -> 1
1 -> 1
1 -> 1
1 -> 2
2 -> 3
abbbbaaabb is Accepted
```