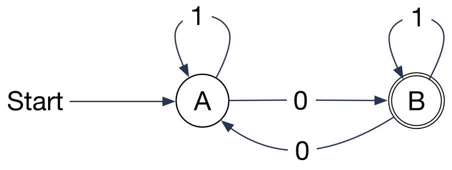
Algorithms: Programming Assignment — Regular Expressions Home-Made

Finite State Machines are easiest implemented in languages that allow the goto complex. The pattern is

```
State_A:   try:
               switch(get_input()):
               case "0":  goto State_B
               case "1":  goto State_A
           except inputError:  #there is no more string to process
               return "Not accepted"
State_B:   try:
               switch(get_input()):
               case "0":  goto State_A
               case "1":  goto State_B
           except inputError:  #there is no more string to process
               return "Accepted"
```

in order to implement the following simple DFA:



In languages without the goto (or languages that make the use of the goto difficult), we can maintain the set of states in which the machine can currently be. If we process a character from the input string, we go through all states in the set and calculate the set of states that can result from them. This is especially easy if we avoid machines with ϵ moves, but NFA are not more difficult to implement.

Create a program that reads in a file with only 0 and 1 characters. You can easily create such a file with the random module in Python. Your program has to decide whether the binary number corresponding to the file has remainder 1 modulo 3. You have to implement this with a DFA. Of course, the file is so large that casting it into an integer will be difficult even for Python, so **do not do that.**