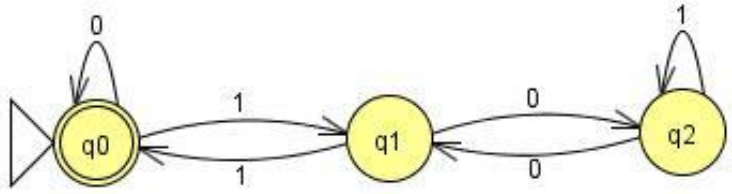
DFA based division

Deterministic Finite Automaton (DFA) can be used to check whether a number "num" is divisible by "k" or not. If the number is not divisible, remainder can also be obtained using DFA.

We consider the binary representation of 'num' and build a DFA with k states. The DFA has transition function for both 0 and 1. Once the DFA is built, we process 'num' over the DFA to get remainder.

Let us walk through an example. Suppose we want to check whether a given number 'num' is divisible by 3 or not. Any number can be written in the form: num = 3*a + b where 'a' is the quotient and 'b' is the remainder.
For 3, there can be 3 states in DFA, each corresponding to remainder 0, 1 and 2. And each state can have two transitions corresponding 0 and 1 (considering the binary representation of given 'num').
The transition function F(p, x) = q tells that on reading alphabet x, we move from state p to state q. Let us name the states as 0, 1 and 2. The initial state will always be 0. The final state indicates the remainder. If the final state is 0, the number is divisible.



In the above diagram, double circled state is final state.

1. When we are at state 0 and read 0, we remain at state 0.
2. When we are at state 0 and read 1, we move to state 1, why? The number so formed(1) in decimal gives remainder 1.
3. When we are at state 1 and read 0, we move to state 2, why? The number so formed(10) in decimal gives remainder 2.
4. When we are at state 1 and read 1, we move to state 0, why? The number so formed(11) in decimal gives remainder 0.

5. When we are at state 2 and read 0, we move to state 1, why? The number so formed(100) in decimal gives remainder 1.
6. When we are at state 2 and read 1, we remain at state 2, why? The number so formed(101) in decimal gves remainder 2.

The transition table looks like following:

```
state   0   1

_____________
 0      0   1
 1      2   0
 2      1   2
```

Let us check whether 6 is divisible by 3?
Binary representation of 6 is 110
state = 0
1. state=0, we read 1, new state=1
2. state=1, we read 1, new state=0
3. state=0, we read 0, new state=0
Since the final state is 0, the number is divisible by 3.

Let us take another example number as 4
state=0
1. state=0, we read 1, new state=1
2. state=1, we read 0, new state=2
3. state=2, we read 0, new state=1
Since, the final state is not 0, the number is not divisible by 3. The remainder is 1.

Note that the final state gives the remainder.

We can extend the above solution for any value of k. For a value k, the states would be 0, 1, …. , k-1. How to calculate the transition if the decimal equivalent of the binary bits seen so far, crosses the range k? If we are at state p, we have read p (in decimal). Now we read 0, new read number becomes 2*p. If we read 1, new read number becomes 2*p+1. The new state can be obtained

by subtracting k from these values (2p or 2p+1) where 0 <= p < k.

Based on the above approach, following is the working code:

```
#include <stdio.h>
#include <stdlib.h>

// Function to build DFA for divisor k
void preprocess(int k, int Table[][2])
{
    int trans0, trans1;

    // The following loop calculates the two transitions
    // starting from state 0
    for (int state=0; state<k; ++state)
    {
        // Calculate next state for bit 0
        trans0 = state<<1;
        Table[state][0] = (trans0 < k)? trans0: trans0-k

        // Calculate next state for bit 1
        trans1 = (state<<1) + 1;
        Table[state][1] = (trans1 < k)? trans1: trans1-k
    }
}

// A recursive utility function that takes a 'num' and DI
// table) as input and process 'num' bit by bit over DFA
void isDivisibleUtil(int num, int* state, int Table[][2]
{
    // process "num" bit by bit from MSB to LSB
    if (num != 0)
    {
        isDivisibleUtil(num>>1, state, Table);
        *state = Table[*state][num&1];
    }
}

// The main function that divides 'num' by k and returns
int isDivisible (int num, int k)
{
    // Allocate memory for transition table. The table w
    int (*Table)[2] = (int (*)[2])malloc(k*sizeof(*Table

    // Fill the transition table
    preprocess(k, Table);
```

```
    // Process 'num' over DFA and get the remainder
    int state = 0;
    isDivisibleUtil(num, &state, Table);

    // Note that the final value of state is the remaind
    return state;
}

// Driver program to test above functions
int main()
{
    int num = 47; // Number to be divided
    int k = 5; // Divisor

    int remainder = isDivisible (num, k);

    if (remainder == 0)
        printf("Divisible\n");
    else
        printf("Not Divisible: Remainder is %d\n", remai

    return 0;
}
```

Output:

```
Not Divisible: Remainder is 2
```

DFA based division can be useful if we have a binary stream as input and we want to check for divisibility of the decimal value of stream at any time.