****
Simulating a concrete Postmachine using Markov Algorithms
****

Definition of a postmachine:
****

A Postmachine consists of a programm and an infinite FIFO Queue

The program can perform the following operations:
An operation is always of the form: line number followed by the instruction.
There are two types of instructions:

a) Append an alphabet symbol at the end of the queue and jump to a certain line number

e.g.

001 queue := queue || x; goto 010;

b) Remove the symbol at the beginning of the queue. Depending on this symbol jump to a certain line number

e.g.

010 case x: goto 011; case y: goto 100; case empty string: goto 101;

Simulation using a Markov algorithm
****

The alphabet of the markov algorithm consists of the symbols 0 and 1 to encode the line numbers ind binary representation and the alphabet symbols from the tag machine.

e.g.

A_Postmachine = {x,z}

A_Markov = {0,1,x,y}

The Markov algorithm needs a marker to move the appended symbol to the end of the queue.

M_Markov = {alpha}

Each instruction from the program of the postmachine is translated as follows:

Instruction of type a)

001 queue := queue || x; goto 010;
is translated into

```
0 0 1 -> 0 1 0 alpha x
```

We need additional rules at the beginning of the markov algorithm, which move the alpha together with the appended alphabet symbol towards the end of the queue:

```
alpha x x -> x alpha x
alpha y x -> y alpha x
```

and so on
The marker has to be removed once it has reached the end:

```
alpha -> epsilon
```

Instruction of type b)

010 case x: goto 011; case y: goto 100; case empty string: goto 101;
is translated into

```
0 1 0 x -> 0 1 1
0 1 0 y -> 1 0 0
0 1 0 -> 1 0 1
```

The order of these rules is important, as the match for the empty string would always be true and therefore has to be at the end of this set of rules

To avoid problems, the line numbering should be chosen in a way that no line number is a prefix of another one. This could be achieved for example by using line numbers of a fixed length.

Given a postmachine with an alphabet of size m and a program of length n.
That gives us a set of rules for the Markov algorithm simulating this postmachine of size O(m^3 + n).

View at the postmachine as an automaton
****

A postmachine can be considered as an automaton. The line numbers indicate the different states. A state transition is given by:

(qi,a) -> (qj,b)

where a is the symbol that is removed at the beginning and b is the symbol that is appended at the end of the queue.

An instruction of type a) would therefore be of the kind:

(qi,-) -> (qj,b)

An instruction of type b) would be:

(qi,x) -> (qj,-)
(qi,y) -> (qk,-)
(qi,epsiolon) -> (ql,-)

This version of the postmachine is able only either to remove a symbol or to append a symbol in a single step.

If we want a model where in a single step a symbol can be removed and a symbol appended, we have to use several commands from our postmachine together:

```
(qi,a) -> (qj,b)

is programmed as

qi case a: goto qk;
qk queue := queue || b; goto qj
```

Example of a simulation of a postmachine in the automat view using a markov algorithm:

****

Postmachine that decides the language L = {w | w = x^n y^n}

```
 A_Postmachine = {x,y,#}

        (000,-) -> (001,#)
        (001,x) -> (010,-)      // remove one x
        (001,#) -> (111,-)      // algorithm terminated, w accepted
        (010,x) -> (010,x)      // move x to the end of the queue
        (010,y) -> (011,-)      // remove one y
        (011,y) -> (011,y)      // move y to the end of the queue
        (011,#) -> (000,-)      // start a new iteration
```

Idea: The postmachines performs several iterations. In each iteration the number of x's and y's is reduced by one. If at the end, the # remains, the machine accepts and terminates in state 111, otherwise it rejects.

This algorithm can now be translated directly into the corresponding markov algorithm:

```
A_Markov = {0,1,x,y,#}
M_Markov = {alpha}


        alpha # x -> x alpha #
        alpha # y -> y alpha #
        alpha x x -> x alpha x
        alpha x y -> y alpha x
        alpha x # -> # alpha x
        alpha y x -> x alpha y
        alpha y y -> y alpha y
        alpha y # -> # alpha y

        alpha -> epsilon

        0 0 0 -> 0 0 1 alpha #
        0 0 1 x -> 0 1 0
        0 0 1 # -> 1 1 1
        0 1 0 x -> 0 1 0 alpha x
        0 1 0 y -> 0 1 1
        0 1 1 y -> 0 1 1 alpha y
        0 1 1 # -> 0 0 0
```

****

Conclusion:

This prooves that every algorithm that can be processed by a postmachine can equally be executed using a markov algorithm. Therefore the markov algorithm has the same computational power as a post machine.

A postmachine is equivalent to a turing machine. That gives the conclusion that the markov algorithm is as strong as a turing machine concerning its computational power.