Finding the Period of a Periodic Sequence

Klaus Pommerening

February 2009 – english version February 2012

This text presents some general results and algorithms on how to determine the period of a periodic sequence in a finite set.

1 Periods of state changes

Let M be a finite set with $m = \#M$. We may think of the elements of M as “states”. Consider a map (“state transition”)

$$g : M \longrightarrow M.$$

For each element (“initial state”) $x_0 \in M$ we define a sequence $(x_i)_{i\in\mathbb{N}}$ in M by the recursion formula $x_i = g(x_{i-1})$ for $i \geq 1$. After a preperiod of length μ this sequence becomes periodic with a period of ν, see Figure 1.

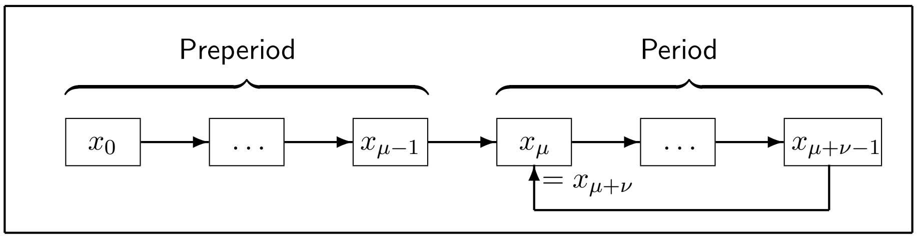

Figure 1: Period and preperiod

Since M is finite there are smallest integers $\mu \geq 0$ and $\nu \geq 1$ such that $x_{\mu+\nu} = x_\mu$: Take for μ the smallest index such that the element x_μ reappears somewhere in the sequence, and for $\mu + \nu$ the index where the first repetition occurs. Then also

$$x_{i+\nu} = x_i \quad \text{for } i \geq \mu.$$

Obviously $0 \leq \mu \leq m-1$, $1 \leq \nu \leq m$, $\mu + \nu \leq m$. The values $x_0, \ldots, x_{\mu+\nu-1}$ are all distinct, and the values $x_0, \ldots, x_{\mu-1}$ never reappear in the sequence.

Definition: μ is called (length of the) **preperiod**, ν is called (length of the) **period**. The **state-transition graph** is the directed graph whose nodes are the elements of M, and whose arrows $\boxed{x} \to \boxed{y}$ correspond to the relations $y = g(x)$.

Example: $M = \mathbb{F}_2^2$, $g(x_1, x_2) = (x_1x_2, x_1)$, yields the state-transition graph of Figure 2.

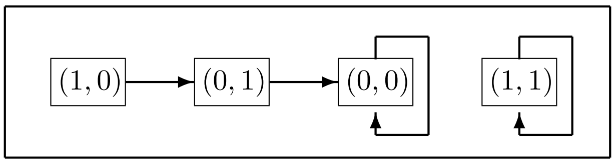

Figure 2: A state-transition graph

2 The case of a known state-transition function

We may determine the length of the period by "naively" searching through the sequence. There is another algorithm by R. W. FLOYD which needs extremely few storage cells [1, Exercise 3.1.6]. It is based on the following consideration:

Lemma 1 *Let $j > i$. Then*

$$x_j = x_i \iff i \geq \mu \quad \textit{and} \quad j \equiv i \pmod{\nu}.$$

In particular $x_{2i} = x_i$ only, if $\nu | i$.

Proof. We only have to show the implication "$\Longrightarrow$". The minimality of μ gives $i \geq \mu$. Let $j - i = k\nu + r$ with $0 \leq r < \nu$ be the division with remainder. Then $i + r = j - k\nu$, hence

$$x_{i+r} = x_j = x_i,$$

and successively going to the preceding element we also get $x_\mu = x_{\mu+r}$. The minimality of ν gives $r = 0$, hence $j - i$ is divisible by ν.

The addendum is trivial, if $i = 0$, and otherwise it follows with $j = 2i$. $\diamond$

Let us call FLOYD **index** the smallest index $\rho > 0$ such that $x_{2\rho} = x_\rho$. We may calculate it as follows:

Proposition 1 [FLOYD] *Let M be a finite set and $g : M \longrightarrow M$. From the initial value $x_0 \in M$ generate a sequence $(x_i)_{i \in \mathbb{N}}$ with $x_i = g(x_{i-1})$ for $i \geq 1$. Let μ be the preperiod, ν the period, and ρ the* FLOYD *index of this sequence. Then one of the following statements is true:*

(i) *$\mu = 0$ and $\rho = \nu$.*

(ii) *$\mu \geq 1$, $\mu \leq \rho < \mu + \nu$, and $\rho = \nu \cdot \lceil \frac{\mu}{\nu} \rceil$.*

In any case ρ is a multiple of ν, namely

$$\rho = \begin{cases} \mu, & \textit{if } \nu | \mu \textit{ and } \mu \geq 1, \\ \mu + \nu - (\mu \bmod \nu) & \textit{otherwise.} \end{cases}$$

Proof. The assertion is trivial for $\mu = 0$. Now let $\mu > 0$. The first inequality $\rho \geq \mu$ follows from the minimality of μ.

Between μ (inclusive) and $\mu + \nu$ (exclusive) there is a multiple $k\nu$ of ν: Simply choose the integer k with $(k-1)\nu < \mu \leq k\nu$. Since $\mu > 0$ we have $k > 0$ and

$$x_{k\nu} = x_{k\nu+\nu} = \cdots = x_{k\nu+k\nu} = x_{2k\nu}.$$

Therefore there is at least one index i with $\mu \leq i < \mu + \nu$ and $x_{2i} = x_i$. Hence this inequality is a forteriori true for the index ρ.

That ρ is a multiple of ν is in the lemma above. The formula for ρ follows because the right hand side is the unique multiple of ν between μ (inclusive) and $\mu + \nu$ (exclusive). $\diamond$

The FLOYD algorithm derives from Proposition 1 and consists of the following steps:

1. Determine the FLOYD index ρ: Start at $i = 1$, recursively generate $x_i = g(x_{i-1})$ and $x_{2i} = g(g(x_{2i-2}))$, until these two elements coincide. This takes up only 2 storage cells, but 3ρ evaluations of g, that together make less then $3 \cdot (\mu + \nu)$; furthermore we have to do ρ comparisions.

2. Determine the preperiod μ: Since ρ is a multiple of the period ν, and a repetition can occur at x_μ at the earliest, μ is the smallest index i with $x_{i+\rho} = x_i$. Start at x_0 and x_ρ, kept from step 1, and increase both indices succesively by 1, until both elements coincide. Here too we need only 2 storage cells; the number of necessary evaluations of g is 2μ, the number of comparisions is μ.

3. Determine the period ν: ν is the smallest index $i \geq 1$ with $x_\mu = x_{\mu+i}$, and again we determine it using 2 storage cells, and ν additional evaluations of g, and ν comparisions.

The entire costs are less then $5\mu + 4\nu$ evaluations of g and less then $2\mu + 2\nu$ comparisions, or roughly $\mathrm{O}(\mu + \nu)$, and alltogether the procedure needs only 2 storage cells for elements of M, plus a third one, that keeps x_0.

The naive algorithm would determine *and store* all the x_i successively, would need $\mu + \nu$ storage cells and evaluations of g, but a quadratic number of comparisions.

3 The case of an unknown state-transition function

Consider the situation where the function g, that generates the sequence, is unknown, but a partial sequence $x_0, \ldots, x_{n-1}$ is given; this situation may occur for example when one tries to predict a pseudorandom generator without knowing its parameters.

The algorithm proceeds in a similar way, but instead of calling the function g one simply takes the next element from the given partial sequence. In addition one has to catch the case where the end of the given partial sequence is reached without detecting periodicity.

Here is the naive algorithm in Mathematica (note that Mathematica numbers the elements of a list beginning with 1):

```
periodNaive[inlist_] := Module[{n, i, j},
  n = Length[inlist];
  For[j = 2, j <= n, j++,
   For[i = 1, i < j, i++,
    If[inlist[[i]] == inlist[[j]],
     Return[{i - 1, j - i}] (* repetition found *)
     ] (* End If *)
    ] (* End For i *)
   ]; (* End For j *)
  Return[{0, 0}] (* no repetition found *)
  ] (* End PeriodNaive *)
```

The Floyd algorithm for this case looks like this:

```
periodFloyd[inlist_] := Module[{n, i, rho = 0, mu = 0, nu = 0},
  n = Length[inlist];
  For[i = 2, 2 i - 1 <= n, i++,
   If[inlist[[i]] == inlist[[2 i - 1]],
    rho = i - 1;
    Break[] (* FLOYD index rho found *)
    ] (* End If *)
   ]; (* End For *)
  If[rho == 0, Return[{0, 0}]]; (* algorithm failed *)
  For[i = 1, i <= n - rho, i++,
   If[inlist[[i]] == inlist[[i + rho]],
    mu = i - 1;
    Break[] (* preperiod found *)
    ] (* End If *)
   ]; (* End For *)
  For[i = 2, i <= n - mu, i++,
   If[inlist[[mu + 1]] == inlist[[i + mu]],
    nu = i - 1;
    Break[] (* period found *)
    ] (* End If *)
   ]; (* End For *)
  Return[{mu, nu}]
  ] (* End PeriodFloyd *)
```

In assessing the costs of these algorithms the number of storage cells is no longer relevant, because the sequence $(x_0, \ldots, x_{n-1})$ is given as input. The costs are dominated by the comparisions, whose number is linear in n for the Floyd algorithm, and quadratic for the naive algorithm. More exactly:

1. The naive algorithm needs the elements $x_0, \ldots, x_{\mu+\nu}$ of the sequence and does
$$\frac{(\mu+\nu)(\mu+\nu+1)}{2}$$
comparisions. The worst case is $\mu+\nu \geq n$: Then the algorithm doesn't find a repetition, and does $\frac{n(n-1)}{2}$ comparisions (in vain).

2. The Floyd algorithm uses the elements of the sequence up to $x_{2\rho}$, that amounts to $2(\mu+\nu)-1$ elements in the worst case (or $2\nu+1$ in the case $\mu=0$), and $2\mu+1$ elements in the best case. If it is given less elements, it doesn't detect a repetition. The number of necessary comparisions is $2\rho+\mu+\nu$, less than $3(\mu+\nu)$ (or $=2\nu+1$ in the case $\mu=0$). The worst case is $2\rho \geq n$: Then the algorithm does't find a repetition—even when there is one in the given partial sequence—and it executes $\lfloor\frac{n-1}{2}\rfloor$ comparisions.

So the Floyd algorithm has significantly smaller asymptotic expenses, but it doesn't succeed in all cases where the naive algorithm finds the period. Therefore we may recommend:

- For an application where the length n is rather small (for example in predicting pseudorandom sequences) the argument of efficiency is rather meaningless. Prefer the naive algorithm for its better success probability.

- If efficiency matters for an application, then first try the Floyd algorithm, and only after this failed, switch to the naive algorithm.

4 Sequences of unknown origin

Suppose we are given a sequence $x_0, \ldots, x_{n-1}$ in M and don't know whether it is generated by a recursion as above.

Then we want to know whether there is such a recursion that generates our sequence, and to learn enough on the generating (state change) map as to predict the sequence from the index n on.

If there is no repetition in the given sequence, then the solutions are exactly the maps $g : M \longrightarrow M$ with
$$g(x) = \begin{cases} x_i & \text{if there is an } i \text{ with } 1 \leq i \leq n-1 \text{ and } x = x_{i-1}, \\ \text{any value} & \text{otherwise,} \end{cases}$$
in particular the immediate continuation x_n of the sequence is completely arbitrary.

If there is a repetition in the given sequence, then the algorithms above will find it—the naive one in any case, the Floyd algorithm, if the sequence is long enough—, and they provide two candidate numbers μ and ν for the preperiod and the period. Then we have to check the **consistency condition**
$$x_{i+\nu} = x_i \quad \text{for } \mu < i < n - \nu.$$

If and only if this is true, then the given sequence may be generated by a recursion. Consistency is checked by the following algorithm in Mathematica; the return value `crash` gives the index where the first inconsistency is found, and 0 in case of success:

```
checkPeriod[inlist_, p_Integer, q_Integer] :=
 Module[{l, i, crash = 0},
  If[p < 0, Return[0],]; (* nonsense input *)
  If[q <= 0, Return[0],]; (* nonsense input or supposed nonperiodicity *)
  n = Length[inlist];
  For[i = p + 1, i <= n - q, i++,
   If[inlist[[i + q]] == inlist[[i]], ,
    crash = i + q; Break[]
    ] (* End If*)
   ]; (* End For i *)
  Return[crash]
  ] (* End checkPeriod*)
```

All possible generating maps g are given by the formula above. The continuation of the sequence is unique and simply relies on the detected periodicity.

If the consistency condition is violated, then there is no recursion formula of the considered kind that generates the sequence, and any further prediction is impossible by this approach.

References

[1] D. Knuth: *The Art of Computer Programming*, Vol. 2. Addison-Wesley, Reading 1981 (2nd Ed.).