Floyd's Algorithm

Mirjam Skobe

University of Ljubljana, Faculty of Computer and Information Science

August, 2018

Abstract

This is a paper about Floyd's Algorithm for detecting collisions or cycles in a sequence. In the paper, we provide a description of the algorithm and its use in modern cryptography. We also look at some other cycle detecting algorithms that have been proven to work a lot faster than Floyd's Algorithm.

1 Introduction

In computer science, cycle detection or cycle finding is the algorithmic problem of detecting periodicity in sequences produced by repeated application of a given function. In other words, we are looking for collisions in a sequence. A collision is a pair of elements $x, y \in D$ where $x \neq y$ and $f(x) = f(y)$.

The sequence $\{y_i\}$ is defined as $y_i = f(y_{i-1})$ for all $i \geq 1$ with function $f : D \to D$ and an initial element $y_0 \in D$. If D is finite, this sequence must eventually become periodic. Then there exist unique μ and λ such that $y_0, \ldots, y_{\mu+\lambda-1}$ are all distinct, but $y_i = y_{i-\lambda}$ for all $i \geq \mu$. The index μ is defined as the smallest index such that the value y_μ reappears within the sequence $\{y_i\}$ and λ is the loop length. The elements $y_0, \ldots, y_{\mu-1}$ form the prefix of the sequence, and the elements $y_\mu, \ldots, y_{\mu+\lambda-1}$ constitute the elements on its cycle. The cycle detection problem asks to find a pair of elements $y_i = y_j$ for which $i \neq j$, and possibly also finding the cycle length λ [4].
Cycle detection arises in a number of situations [5]:

- in studying the behavior of random number generators,
- in searching for function collisions. A greedy way to find a collision is to repeatedly apply function f starting from some initial value x_0, and find the cycle length λ of the resulting sequence. Then, using the knowledge of λ, we can reconstruct the collision pair $x_{\mu-1}, x_{\mu+\lambda-1}$. But this approach is often inefficient. Finding collisions has several cryptanalytic applications.

- in order to detect when, say, a cellular automaton configuration has become periodic,
- in Pollard ρ-method for factorization and discrete logarithms.

One of the most well-known cycle detection algorithms is the Floyd's Algorithm.

The rest of this paper is organized as follows: In Section 2 we present the Floyd's Algorithm. In Section 3 we look at the use of the Floyd's Algorithm, mainly focusing on Pollard ρ-method and its modification using the algorithm and in Section 4, we look at some alternative cycle detection algorithms.

2 Floyd's Algorithm

Floyd's cycle finding algorithm [6, 3, 7] is defined as follows: given a sequence $\{y_i\}$, find the smallest index j for which:

$$y_j = y_{2j}, \quad 1 < j \leq \mu + \lambda$$

where μ is defined as the smallest index such that the value y_μ reappears within the sequence $\{y_i\}$ and λ is the loop length. This is done by calculating elements of both sequences $\{y_i\}$ and $\{y_{2i}\}$ at every step of the algorithm and comparing current elements y_i and y_{2i}. If the above relation is not valid for the current pair of elements (y_i, y_{2i}) we calculate the next pair (y_{i+1}, y_{2i+2}) with $y_{i+1} = f(y_i)$ and $y_{2i+2} = f(f(y_{2i}))$ and compare them using the equation. This procedure is repeated until we find a pair of elements for which the above equation is valid. The pseudocode of the Floyd's algorithm is presented in Algorithm 1. The number j is the result of Floyd's algorithm and is called Floyd's index.

A more informal definition of Floyd's collision finding algorithm [7] would be the following. Keep two pointers and run one of them at normal speed and the other one at double speed until they collide.

3 Use of Floyd's Algorithm

3.1 Pollard ρ-method

The Pollard ρ-method [6, 1] is currently the best and quickest method for solving the discrete logarithm problem on elliptic curves. Let D be a finite nonempty set. The Pollard ρ-method can be described as a method based on picking elements from set D until the collision is found. Let $\{y_i\}$ denote the sequence whose elements are stored in a table. At step j of this method, we pick an element y_j and check if this element is already in our table if it is then we have found a collision and we stop the procedure. Otherwise,

Algorithm 1 Floyd's algorithm

Data: Finite set D, seed $y_0 \in D$ and function $f : D \to D$ that recursively defines sequence $\{y_i\}$.
Result: The smallest index j, for which $y_j = y_{2j}$.

```
y := y_0
z := y_0
j := 0
while j ≠ z do
    j := j + 1
    y := f(y)
    z := f(f(z))
end while

return j
```

this element is added to the table. This is repeated until we find a collision. Because the set D is finite a collision always occurs. The numbers i and j are the indices of collision of sequence $\{y_i\}$, where $y_i = y_j$, and the number j is called Pollard's index. The sequence $\{y_i\}$ generated with the Pollard ρ-method is called Pollard's sequence. For this method to work, we have to define the first element of the method, $y_0 \in D$ and some function $f : D \to D$ to calculate elements of the Pollard's sequence, $y_{i+1} = f(y_i); i \geq 0$.
Because the elements of Pollard's sequences get stored in a table, the space complexity of this method grows with the length of the sequence. To reduce the space complexity we can use a cycle finding algorithm to find collisions in a sequence. If we use Floyd's algorithm to find a collision in Pollard ρ-method then we call this new method the Pollard-Floyd ρ-method.

3.2 Other Uses

Floyd's algorithm also has other very useful applications:

- it can be used to determine the strength of the pseudorandom number generator by calculating its cycle length,

- in cryptographic applications, the ability to find two distinct values $x_{\mu-1}$ and $x_{\lambda+\mu-1}$ mapped by some cryptographic function f to the same value x_μ may indicate a weakness in f,

- a cycle detection may be useful as a way of discovering infinite looping in certain types of computer programs.

4 Other Cycle Finding Algorithms

4.1 Brent's Algorithm

Richard P. Brent [6, 2] is an Australian mathematician and computer scientist. In 1980 he published an algorithm for detecting cycles, which may work a lot faster than Floyd's algorithm.

Let D be a finite nonempty set, $\{y_i\}$ a sequence and $e \geq 1$. In Brent's Algorithm we calculate the sequence $\{y_i\}$ with recursion $y_{i+1} = f(y_i)$ for $i \geq 1$, where $f : D \to D$ is an iterative function and $y_0 \in D$ is a seed. Brent's algorithm is defined as follows: find the smallest number j for which:

$$y_{2^{e-1}-1} = y_j, \quad j = 2^{e-1} + \ell\lambda - 1 \leq 2^e - 1,$$

where λ is the period of the sequence $\{y_i\}$, ℓ is the smallest number for which $\ell\lambda \geq 2^{e-2} + 1$ and $e > 1$. This is done by memorizing each element with index $2^e - 1$ and comparing the element y_{2^e-1} with elements y_{2^e+k-1} for all $2^{e-1} < k \leq 2^k$. If we found a collision then

$$y_{2^e-1} \neq y_{2^e+k-1}$$

for all $k \in \{2^{e-1} + 1, 2^{e-1} + 2, \ldots, 2^e\}$, and $y_{2^{e+1}-1}$ is the last calculated element. Now we do not need the element y_{2^e-1} and we replace it with the element $y_{2^{e+1}-1}$, which we use for comparison with elements $y_{2^{e+1}+k-1}$ for $2^e < k \leq 2^{e+1}$. The procedure is repeated until we find a collision. The pseudocode for the algorithm is presented in Algorithm 2. The returned number j of the algorithm is called Brent's index and the returned value i of the algorithm is some power of the number two reduced by one, which is the index of the element currently in memory when Bren's algorithm finishes.

Brent showed in his paper that his algorithm is always better than Floyd's, both on average and in worst case performance. In particular, Floyd's algorithm is expected to find a cycle after $\cong 3.0924$ function evaluations, whereas Brent's algorithm is expected to take only $\cong 1.9828$ function evaluations [4], so Brent's algorithm works on average 36% faster than Floyd's.

As with Floyd's Algorithm, Brent's Algorithm can also be used to find a collision in Pollard ρ-method.

Algorithm 2 Brent's algorithm

Data: Finite nonempty set D, seed $y_0 \in D$ and iterative function $f : D \to D$ that recursively defines sequence $\{y_i\}$.
Result: Numbers i and j, where $i \neq j$ and for which $y_i = y_j$.

```
y := y_0
fy := f(y_0)
power := i := 1
while y != fy do
  if power == i then
    y := fy
    power := power * 2
    i := 0
  end if
  fy := f(fy)
  i := i + 1
end while
j := 0
f := fy := y_0
while i > 0 do
  fy := f(fy)
  i = i - 1
end while
while y != fy do
  y := f(y)
  fy := f(fy)
  j = j + 1
end while

return i, j
```

4.2 Nivasch's Algorithm

In 2004 Gabriel Nivasch published a paper [5] in which he describes a new algorithm for finding a cycle by using a stack. We should keep in mind that a stack is an abstract data type, which is a collection of elements together with two major operations. The first operation is adding an element to the stack, and the second is removing the most recently added element still present in the stack.

The algorithm proceeds as follows. Record a stack of pairs (y_i, i) where both the y_i and i form is strictly increasing sequences at all times. This stack is initially empty, and for each step j, we remove from the stack all entries (y_i, i) where $y_i > y_j$. If $y_i = y_j$ is found, we are done, and we have recovered the cycle length, in that $\lambda = j - i$. Otherwise, move (y_j, j) to the

top of the stack, and perform the next step. The pseudocode for Nivasch's Algorithm is presented in Algorithm 3.

Algorithm 3 Nivasch's algorithm

Data: Finite nonempty set D, seed $y_0 \in D$ and iterative function $f : D \to D$ that recursively defines sequence $\{y_i\}$.
Result: numbers i and j, where $i \neq j$ and for which $y_i = y_j$.
create stack s which contains pairs $(element, index)$
$y := y_0$
$j := -1$
while true **do**
  if $j \geq 0$ **then**
    find smallest $t \leq j$ such that element$(s, t) \geq y$
  else
    $t := -1$
  end if
  if $t \neq -1$ **and** element$(s, t) = y$ **then**
    break
  end if
  $i := i + 1$
  $j := t + 1$
  if $j >$ size(s) **then**
    resize s
  end if
  put (y, i) into s
  $y := f(y)$
end while

return $i, j =$ index(s, t)

The Nivasch's Stack Algorithm runs in linear time, since the running time of each step is proportional to the number of elements removed from the stack at that step, and each element is removed at most once. It uses logarithmic space and is guaranteed to stop within the second repetition of the sequence's cycle, regardless of its size.

4.3 Compare and Adjust Algorithm

The last algorithm for detecting cycles we are going to look at is the Compare and Adjust Algorithm [6].

The algorithm works as follows. We have a finite noneempty set D, a Pollards sequence y_i where $y_k \in D$ and indeces μ and λ where μ is defined as the smallest index such that the value y_μ reappears infinitely often within the sequence and λ is the loop length. Then $y_{\sigma+\lambda} = y_\sigma$ is a collision in

the sequence y_i for $\sigma \in \mathbb{N}$ and $\sigma \geq \mu$. Let $v \in \mathbb{N}$ be the smallest number for which $\mu + \lambda \leq v\mu$. Then $\sigma + \lambda \leq v\sigma$ for every $\sigma \geq \mu$. Therefor, if we remember the element $y_\sigma \in y_i$ and we compare it to elements $y_{\sigma+1}, y_{\sigma+2}, \ldots$, then we find collision $y_{\sigma+\lambda} = y_\sigma$ where $\mu \leq \sigma < \sigma+\lambda \leq v\sigma$. The pseudocode of the algorithm is presented in Algorithm 4.

Algorithm 4 Compare and Adjust

Data: Finite nonempty set D, seed $y_0 \in D$, iterative function $f : D \to D$ that recursively defines sequence $\{y_i\}$ and empty table Z.
Result: Numbers i and j, where $i \neq j$ and for which $y_i = y_j$.

```
y := y_0
j := 0
i := 0
end := false
σ_1 := 0, ..., σ_t := 0          //t is the number of elements in the memory
Z[σ_1] := y_0, ..., Z[σ_t] := y_0
while end != true do
  j = j + 1
  y := f(y)
  if y = Z[σ_s] for some σ_s where s ∈ {1, ..., t} then
    i := σ_s
    end := true
  end if
  if j ≥ vσ_1 then
    Z[σ_1] := Z[σ_2], ..., Z[σ_{t-1}] := Z[σ_t]
    Z[σ_t] = y
    σ_1 := σ_2, ..., σ_{t-1} := σ_t
    σ_t := j
  end if
end while

return i, j
```

5 Conclusion

Even though it has been over 50 years since Floyd's algorithm was first published, the algorithm is still very useful in many scientific fields. It was a base for many modern cycles finding algorithms, which only improved its performance but did not drastically change the algorithm. Not much research has been done on the subject of finding cycles so there are not that many useful algorithms for performing this task. Most of the research focuses on using these algorithms and not developing them. I think that

for now, we are satisfied with the performance of algorithms such as Floyd's and Brent's cycle-finding algorithm and are now looking more into their application instead of improvement.

References

[1] S. BAI , R.P. BRENT *On the efficiency of Pollard's ρ method for discrete logarithms*. Proceedings of the fourteenth symposium on Computing: the Australasian theory, January 01-01, 2008, Wollongong, NSW, Australia

[2] R.P. BRENT *An improved Monte Carlo factorization algorithm*. BIT Numerical Mathematics, 1980, Volume 20, Number 2, p. 176.

[3] D. KNUTH. *The art of computer programming, Volume 2: Seminumerical algorithms*. Reading, Massachusetts, Addison-Wesley, 1969.

[4] J. MURPHY *Factorization and Collision Algorithms in Algebraic Cryptography*. A master's thesis in Mathematics. Wesleyan University, Connecticut. 2017.

[5] G. NIVASCH *Cycle Detection Using a Stack*. Information Processing Letters, 2004, 90(3), p. 135–140.

[6] E. SCHLEGEL. *Pollardova ρ-metoda*. A master's thesis. University of Ljubljana, Faculty of Mathematics and Physics. 2003.

[7] A. SHAMIR *Random graphs in cryptography*. Lecture at The Weizmann Institute Israel. June 28, 2010. The Onassis Foundation Science Lecture Series. Retrieved from: http://www.forth.gr/onassis/lectures/pdf/Random_Graphs_in_Cryptography_Onassis_Foundation.pdf
Viewed on: January 16, 2018.